Sécurité matérielle des objets connectés

Alexandre Menu

N°d'ordre NNT : 2021LYSEM032

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Ecole des Mines de Saint-Etienne

Ecole Doctorale N° 488
Sciences, Ingénierie, Santé

Spécialité de doctorat : Microélectronique
Discipline : Sécurité matérielle

Soutenue publiquement le 29/11/2021, par :
Alexandre Menu

Sécurité matérielle des objets connectés

Devant le jury composé de :

Feillet, Dominique	Professeur des universités, Mines Saint-Étienne	Président du Jury
Guilley, Sylvain	Professeur invité, Télécom Paris	Rapporteur
Maurine, Philippe	Maître de conférence HDR, Université de Montpellier	Rapporteur
Potet, Marie-Laure	Professeure des universités, Université Grenoble-Alpes	Examinatrice
Di Natale, Giorgio	Directeur de recherche CNRS, Université Grenoble-Alpes	Examinateur
Dutertre, Jean-Max	Professeur des universités, Mines Saint-Étienne	Co-directeur de thèse
Danger, Jean-Luc	Directeur d'étude, Télécom Paris	Co-directeur de thèse
Rigaud, Jean-Baptiste	Maître assistant HDR, Mines Saint-Étienne	Co-encadrant

Spécialités doctorales	Responsables :	Spécialités doctorales	Responsables
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche	MATHEMATIQUES APPLIQUEES	M. Batton-Hubert
MECANIQUE ET INGENIERIE	S. Drapier, professeur	INFORMATIQUE	O. Boissier, Professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche	SCIENCES DES IMAGES ET DES FORMES	JC. Pinoli, Professeur
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche	GENIE INDUSTRIEL	N. Absi, Maitre de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	V.Laforest, Directeur de recherche	MICROELECTRONIQUE	Ph. Lalevée, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

ABSI	Nabil	MR	Génie industriel	CMP
AUGUSTO	Vincent	MR	Génie industriel	CIS
AVRIL	Stéphane	PR	Mécanique et ingénierie	CIS
BADEL	Pierre	PR	Mécanique et ingénierie	CIS
BALBO	Flavien	PR	Informatique	FAYOL
BASSEREAU	Jean-François	PR	Sciences et génie des matériaux	SMS
BATTON-HUBERT	Mireille	PR	Mathématiques appliquées	FAYOL
BEIGBEDER	Michel	MA	Informatique	FAYOL
BILAL	Blayac	DR	Sciences et génie de l'environnement	SPIN
BLAYAC	Sylvain	PR	Microélectronique	CMP
BOISSIER	Olivier	PR	Informatique	FAYOL
BONNEFOY	Olivier	PR	Génie des Procédés	SPIN
BORBELY	Andras	DR	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR	Génie Industriel	FAYOL
BRUCHON	Julien	PR	Mécanique et ingénierie	SMS
CAMEIRAO	Ana	PR	Génie des Procédés	SPIN
CHRISTIEN	Frédéric	PR	Science et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR	Génie Industriel	CMP
DEBAYLE	Johan	MR	Sciences des Images et des Formes	SPIN
DEGEORGE	Jean-Michel	MA	Génie industriel	Fayol
DELAFOSSE	David	PR	Sciences et génie des matériaux	SMS
DELORME	Xavier	PR	Génie industriel	FAYOL
DESRAYAUD	Christophe	PR	Mécanique et ingénierie	SMS
DJENIZIAN	Thierry	PR	Science et génie des matériaux	CMP
BERGER-DOUCE	Sandrine	PR	Sciences de gestion	FAYOL
DRAPIER	Sylvain	PR	Mécanique et ingénierie	SMS
DUTERTRE	Jean-Max	PR	Microélectronique	CMP
EL MRABET	Nadia	MA	Microélectronique	CMP
FAUCHEU	Jenny	MA	Sciences et génie des matériaux	SMS
FAVERGEON	Loïc	MR	Génie des Procédés	SPIN
FEILLET	Dominique	PR	Génie Industriel	CMP
FOREST	Valérie	PR	Génie des Procédés	CIS
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GAVET	Yann	MA	Sciences des Images et des Formes	SPIN
GERINGER	Jean	MA	Sciences et génie des matériaux	CIS
GONDRAN	Natacha	MA	Sciences et génie de l'environnement	FAYOL
GONZALEZ FELIU	Jesus	MA	Sciences économiques	FAYOL
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GRIMAUD	Frederic	EC	Génie mathématiques et industriel	FAYOL
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR	Génie des Procédés	SPIN
HAN	Woo-Suck	MR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR	Génie des Procédés	SPIN
ISMAILOVA	Esma	MC	Microélectronique	CMP
KERMOUCHE	Guillaume	PR	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	DR	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	DR	Mécanique et ingénierie	FAYOL
LIOTIER	Pierre-Jacques	MA	Mécanique et ingénierie	SMS
MEDINI	Khaled	EC	Sciences et génie de l'environnement	FAYOL
MOLIMARD	Jérôme	PR	Mécanique et ingénierie	CIS
MOULIN	Nicolas	MA	Mécanique et ingénierie	SMS
MOUTTE	Jacques	MR	Génie des Procédés	SPIN
NAVARRO	Laurent	MR	Mécanique et ingénierie	CIS
NEUBERT	Gilles	PR	Génie industriel	FAYOL
NIKOLOVSKI	Jean-Pierre	Ingénieur de recherche	Mécanique et ingénierie	CMP
O CONNOR	Rodney Philip	PR	Microélectronique	CMP
PICARD	Gauthier	PR	Informatique	FAYOL
PINOLI	Jean Charles	PR	Sciences des Images et des Formes	SPIN
POURCHEZ	Jérémy	DR	Génie des Procédés	CIS
ROUSSY	Agnès	MA	Microélectronique	CMP
SANAUR	Sébastien	MA	Microélectronique	CMP
SERRIS	Eric	IRD	Génie des Procédés	FAYOL
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
VALDIVIESO	François	PR	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	DR	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR	Génie industriel	CIS
YUGMA	Gallian	MR	Génie industriel	CMP

Mise à jour : 04/02/2021

Remerciements

Les travaux de recherche doctorale présentés dans ce manuscrit ont été réalisés au sein du laboratoire SAS en co-direction avec l'école des Mines de Saint-Étienne et Télécom Paris. Ils n'auraient pas vu le jour sans le soutien, la présence, l'écoute et les conseils de nombreuses personnes que je tiens à remercier dans ce chapitre.

Je remercie d'abord mes directeurs de thèse, Jean-Max Dutertre et Jean-Luc Danger, ainsi que mon encadrant de thèse Jean-Baptiste Rigaud. Je vous suis reconnaissant pour la confiance que vous m'avez accordée depuis notre rencontre, les opportunités que vous m'avez offertes et l'investissement qui a été le vôtre dans mon encadrement, bien au-delà de mes attentes.

Je tiens à remercier Dominique Feillet, Marie-Laure Potet, Sylvain Guilley, Philippe Maurine et Giorgio di Natale pour l'intérêt qu'ils portent à mes travaux de recherche doctorale. Je suis honoré qu'ils aient accepté de les évaluer en tant que membres du jury. Je remercie tout particulièrement Sylvain Guilley et Philippe Maurine d'être les rapporteurs de ces travaux.

J'ai eu l'opportunité d'élargir mon champ de recherche au sein du laboratoire Temasek de l'université technologique de Nanyang à Singapour. Je remercie Shivam Bhasin de m'avoir accueilli et encadré sur place au sein de l'équipe PACE. Je remercie également l'école des Mines de Saint-Étienne, et plus particulièrement Christophe Desrayaud, d'avoir contribué à la réalisation financière de cette collaboration.

Ces travaux n'auraient pas été possibles sans la collaboration de MicroPackS et l'aide précieuse de mes collègues. Je remercie tout particulièrement Anne-Lise Ribotta, Simon Pontié, Clément Gaine, Mounia Kharbouche-Harrari, Elias Kharbouche et Benjamin Lac pour leur aide et leurs recommandations qui m'ont fait gagner un temps précieux.

J'exprime toute ma gratitude à mes relecteurs de talent, Elodie Hotton et Laurent De La Roy. Pour votre travail rapide et précis, un grand merci.

Je remercie mes co-auteurs Brice Colombier, Pierre-Alain Moëllic et Olivier Potin pour leurs encouragements, leurs conseils formateurs et les bons moments partagés. Je remercie également mes collègues de l'équipe commune Mines Saint-Étienne/CEA, ainsi que du département administratif et de la direction des systèmes d'information qui ont partagé mon quotidien, mes questionnements ou juste un sourire rassurant tout au long de cette aventure.

Mes derniers remerciements vont à mes proches qui m'ont épaulé et soutenu pendant ces travaux : ma petite sœur, mes parents, mon parrain, les loulous de l'école d'ingé, les copains de longue date, et ma plus belle rencontre.

Sommaire

Table des figures

Liste des tableaux

Introduction générale

Les progrès réalisés dans la fabrication de composants microélectroniques depuis les années 1960 permettent aujourd'hui de concevoir des circuits intégrés à bas coût de production, avec une forte puissance de calcul et une faible consommation, capables de collecter et d'exploiter des données de manière intelligente. Cette intelligence repose, pour une grande part, sur la connexion de ces circuits à des infrastructures spécialisées dans l'exploitation d'ensembles massifs de données – ou *Big Data* – à travers un réseau mondial de machines, sur le principe d'internet. Le potentiel de cette infrastructure se matérialise dans l'internet des objets (*Internet of Things* – IoT) : un réseau massif d'*objets connectés* à l'interface entre le monde numérique et le monde physique. La croissance à deux chiffres du nombre d'objets connectés déployés dans des secteurs comme la domotique, le transport, le bâtiment, les applications gouvernementales ou la gestion de l'énergie témoigne ainsi de la confiance investie dans les technologies de l'IoT [1].

Si les objets connectés sont porteurs de nombreuses applications, ils sont aussi porteurs de risques qu'il convient de maîtriser pour garantir la sécurité des biens, des infrastructures et des personnes. Les questions propres au respect de la vie privée se doivent également d'être posées lors d'une intrusion de ces dispositifs dans la sphère personnelle, comme c'est par exemple le cas pour des capteurs biométriques ou un assistant personnel. Le déploiement d'objets connectés dans un environnement accessible à un attaquant pose enfin la question de la protection des logiciels couverts par la propriété intellectuelle des concepteurs d'objets.

Les contraintes de marché imposées aux concepteurs d'objets connectés s'expriment pourtant au détriment de l'investissement dans la sécurité des dispositifs commercialisés. De surcroît, le déploiement massif d'objets connectés favorise l'existence de vulnérabilités partagées, souvent issues de mauvaises pratiques de sécurité, qui peuvent être exploitées de manière systématique pour prendre le contrôle de ces objets. L'agrégation de machines corrompues au sein d'un réseau de machines zombies, ou *botnet*, permet alors de collecter de l'information ou d'exploiter la puissance de calcul du réseau pour compromettre une infrastructure grâce à une attaque par déni de service distribué (*Distributed Denial of Service* – DDoS) [7]. Les objets connectés non-sécurisés

1. Cette croissance est particulièrement marquée dans les secteurs de l'automobile, de la santé et de la domotique avec une augmentation de plus de 30 % par ans des objets connectés en circulation entre 2018 et 2020 pour un total de 710 millions d'unités en circulation dans le monde en 2018 [77]. Cependant, cette tendance est fortement infléchie par la crise des semi-conducteurs dans le contexte pandémique actuelle où la demande excède nettement les capacités de production des fabricants de circuits intégrés [171].

constituent donc une menace à l'intérieur des réseaux auxquels ils sont connectés.

Pour empêcher l'exploitation de ces vulnérabilités, il convient de mettre en œuvre des mécanismes de sécurité utilisant des algorithmes cryptographiques. Ces mécanismes ont pour objectif de garantir l'intégrité, la confidentialité et l'authenticité des logiciels exécutés par un circuit, des informations sensibles mémorisées par ce dernier et des échanges réalisés avec un tiers. Ces garanties sont notamment formulées grâce à des problèmes mathématiques complexes dont la résolution n'a encore jamais été démontrée en un temps limité avec une capacité de calcul finie.

Cependant, les algorithmes cryptographiques ont aussi leurs points faibles, déjà bien connus de l'industrie des circuits sécurisés. Un attaquant qui dispose d'un accès physique à un circuit peut en effet tirer parti de l'implémentation matérielle du circuit pour extraire des informations sensibles ou contourner une vérification grâce à des techniques d'observation et de perturbation. Parmi ces techniques, les attaques passives par analyse des canaux cachés (*Side Channel Analysis* – SCA) et les attaques actives par injection de fautes ont été particulièrement étudiées dans la littérature scientifique. Elles nécessitent des dispositifs spécialisés accessibles à partir de quelques centaines d'euros, pour les dispositifs les plus simples, jusqu'à plusieurs centaines de milliers d'euros, pour les dispositifs les plus avancés.

Les travaux de RONEN, SHAMIR, WEINGARTEN et al. [144] en 2016 ont démontré qu'une vulnérabilité matérielle peut permettre de lancer une cyber-attaque à grande échelle contre les objects connectés d'une même série. L'attaque proposée par les auteurs exploitent une vulnérabilité du protocole de communication *ZigBee* implémenté dans les lampes connectées d'un fabricant reconnu pour propager un ver informatique d'un réseau à un autre en abusant du mécanisme de mise à jour à distance de ces lampes. Or ce mécanisme avait été sécurisé par le fabricant grâce à une unique clé secrète de 128 bits utilisée pour chiffrer et authentifier les mises à jour. En analysant l'activité électromagnétique d'un circuit embarqué dans une lampe connectée pendant une opération de chiffrement grâce à une SCA, les auteurs ont finalement réussi à extraire les 128 bits de la clé afin de réaliser leur attaque en pratique.

Ces travaux s'inscrivent dans un contexte où des dispositifs d'injection de fautes par impulsion électromagnétique (EM) et laser peuvent être assemblés à partir de composants disponibles dans le commerce pour quelques milliers d'euros [1], [89]. Par ailleurs, plusieurs démonstrations ont été réalisées de l'utilisation des attaques en faute pour compromettre la sécurité des microcontrôleurs non-sécurisés utilisés dans des applications embarquées [34], [127], [129], [153]. Cependant, les modèles de fautes obtenus avec ces techniques d'injection ne sont pas analysés en détail et les conditions d'obtention de ces modèles restent encore mal comprises.

L'objectif de ces travaux est donc de proposer une analyse détaillée des modèles de fautes matériels et logiciels obtenus avec un dispositif d'injection de fautes par impulsion EM et un dispositif d'injection de fautes par illumination laser sur plusieurs microcontrôleurs non-sécurisés similaires aux circuits utilisés dans les objets connectés. Ces modèles de fautes sont évalués au travers de plusieurs scénarios d'attaques.

Ces travaux ont été réalisés au sein du laboratoire SAS de l'école des Mines de Saint-Étienne et au sein du laboratoire Temasek de la Nanyang Technical University de Singapour, en collaboration avec le LTCI de l'école Télécom Paris.

Le chapitre 1 présente le cadre théorique et expérimental de nos travaux sur l'injection de fautes EM et laser. Il détaille notamment l'état de l'art des modèles de fautes obtenus sur un microcontrôleur non-sécurisé grâce à ces techniques d'injection.

Le chapitre 2 propose une analyse détaillée des fautes matérielles obtenues lors du chargement d'une mémoire tampon dans la mémoire Flash d'un microcontrôleur 32 bits. Le modèle de faute étudié permet d'améliorer la compréhension des sauts d'instructions obtenus grâce à l'injection d'une faute pendant la lecture des instructions dans la mémoire Flash d'un microcontrôleur 8 bits.

Le chapitre 3 présente une analyse d'un mécanisme original d'injection de fautes sur les mots lus dans la mémoire NOR Flash embarquée de deux microcontrôleurs 32 bits. Cette analyse articule la compréhension du mécanisme physique d'une injection de fautes, l'analyse des modèles de fautes matériels et l'analyse des modèles de fautes logiciels obtenus grâce à une impulsion laser sur les deux mémoires.

Le chapitre 4 décrit enfin plusieurs scénarios d'attaques qui ont été réalisés en pratique pour compromettre la sécurité d'un algorithme de vérification d'un code PIN et celle d'un algorithme de chiffrement AES-128 implémentés de manière logicielle. Nous proposons également plusieurs applications théoriques des modèles de fautes étudiés dans ces travaux pour contourner un mécanisme d'amorçage sécurisé, ou *secure boot*.

Chapitre 1

État de l'art des injections de fautes EM et laser sur des microcontrôleurs

L'étude des attaques matérielles répond d'abord au besoin de garantir un niveau de sécurité élevé pour les circuits intégrés de type *smart cards* destinés à des applications sensibles. Cependant, ces attaques représentent aujourd'hui une menace pour les circuits de l'internet des objets dont la sécurité repose, en dernier ressort, sur des algorithmes exécutés par des microcontrôleurs qui ne sont pas sécurisés contre les attaques matérielles. En particulier, les attaques locales par injection de fautes laser et électromagnétique permettent d'exploiter une grande diversité d'effets pour contourner un test de sécurité ou extraire une clé de chiffrement. Afin de comprendre au mieux ces attaques, il convient donc au préalable d'en décrire les mécanismes physiques et les effets possibles. Ce chapitre propose ainsi des clés de lecture pour permettre au lecteur de comprendre le cadre théorique et expérimental de nos travaux.

1.1 Sécurité des objets connectés

1.1.1 Contexte de l'internet des objets

La fabrication à bas coût de circuits intégrés sécurisés est l'un des défis que doivent relever les fabricants de circuits intégrés destinés à l'IoT. Ce défi est non seulement technique, car les fonctionnalités de ces circuits, leur capacité de calcul, leur mémoire et leur consommation énergétique sont limitées, mais aussi économique, car la production en masse de circuits intégrés a un effet multiplicatif sur le coût de chacune des fonctionnalités d'un circuit [151].

De surcroît, les délais de lancement très courts imposés par le marché des objets connectés sont difficilement compatibles avec les délais des processus de certifications qui s'appliquent aux circuits sécurisés [176]. Ces difficultés mobilisent la communauté industrielle et académique sur l'évaluation des menaces et la conception de protections modulaires adaptées aux contraintes de l'IoT.

1.1.2 Définition d'un objet connecté

Dans le cadre de ces travaux, un object connecté désigne un circuit intégré sur silicium utilisé pour implémenter une application de l'IoT. Ces circuits sont des systèmes embarqués, généralement des MCU ou des systèmes sur puce (*System on Chip* – SoC).

Un microcontrôleur (*Microcontroller Unit* – MCU) est un circuit intégré sur silicium qui implémente un micro-ordinateur. Il est composé d'un cœur de calcul et de plusieurs mémoires embarquées, dont une SRAM dédiée à la mémorisation des données pendant le fonctionnement du circuit et une mémoire non-volatile NOR Flash qui mémorise les instructions et les données en l'absence d'alimentation électrique. Un MCU comporte également divers périphériques, dont des interfaces de communication et des broches générales d'entrée-sortie (*General Purpose Input Output* – GPIO), qui sont utilisées pour interfacer le circuit avec des composants matériels de bas niveau. Les composants d'un MCU sont connectés entre eux par un ensemble de bus de communication.

Un système sur puce (*System on Chip* – SoC) intègre des fonctionnalités hétérogènes sur une puce unique. Il est généralement composé d'un processeur multi-cœurs qui interagit avec une hiérarchie de mémoires, dont plusieurs niveaux de mémoires cache et une SRAM, par l'intermédiaire d'une unité de gestion de mémoires (*Memory Management Unit* – MMU). La MMU réalise notamment la traduction entre les plages d'adresses physiques des mémoires et les plages d'adresses virtuelles utilisées pour isoler les processus d'un système d'exploitation. Un SoC intègre également des modules et des co-processeurs dédiés, comme un module de sécurité, un module de communication sans-fil ou un co-processeur graphique (*Graphics Processing Unit* – GPU), qui lui permettent d'implémenter des applications avancées. À la différence des microcontrôleurs, les SoC chargent généralement les programmes qu'ils exécutent à partir d'une mémoire non-volatile externe de type NAND Flash.

1.1.3 Objectifs de sécurité

La notion de sécurité dans le contexte des systèmes d'informations recouvre principalement trois propriétés : (i) l'intégrité, (ii) la confidentialité, (iii) et l'authenticité des informations. Une information est intègre si elle n'a pas été modifiée malicieusement après sa création, confidentielle si les parties dans le secret sont les seuls à avoir accès à son contenu et authentique si son origine est identifiée et certifiée. Une quatrième propriété, la non-répudiation, permet en outre de garantir que les parties d'un contrat s'engagent par leur signature. Cette dernière est essentielle dans l'implémentation de transactions commerciales par la voie numérique.

Dans le contexte de l'IoT, il convient d'abord de garantir l'intégrité, la confidentialité et l'authenticité des données et des programmes mémorisés dans un objet connecté. Il convient ensuite de garantir l'authenticité de l'objet et des tierces parties avec lesquelles il communique. Il convient enfin de garantir l'intégrité et la confidentialité des échanges entre un objet et une tierce partie. Pour atteindre ces objectifs, des fonctions de sécurité doivent être implémentées sur les circuits intégrés destinés à l'IoT. Ces fonctions

utilisent généralement des primitives cryptographiques.

1.1.4 Modèle d'attaquant

Les fonctions de sécurité d'un objet connecté sont mises à l'épreuve à travers différents scénarios d'attaques. Ces attaques et les méthodes qu'elles emploient peuvent être classées en deux catégories : les attaques logicielles et les attaques matérielles.

— **Les attaques logicielles** exploitent les failles logiques dans l'implémentation logicielle d'un algorithme. Elles sont généralement réalisées grâce à un équipement informatique standard et peuvent être réalisées à distance dans le cadre d'une cyber-attaque si la cible des attaques est accessible sur un réseau.

— **Les attaques matérielles** exploitent les failles de sécurité dans l'implémentation physique d'un composant microélectronique. Elles sont réalisées grâce à des équipements spécialisés et requièrent généralement un accès physique à la cible des attaques.

1.1.5 Familles d'attaques matérielles

Les propriétés de sécurité implémentées par un circuit intégré peuvent être garanties mathématiquement grâce à des algorithmes cryptographiques. Ces garanties reposent sur la difficulté à résoudre certains problèmes mathématiques complexes en un temps limité avec une capacité de calcul finie. Cependant, elles ne s'étendent pas au fonctionnement des circuits qui exécutent ces algorithmes. L'exploitation des fuites d'informations dans les canaux auxiliaires d'un circuit permet alors d'extraire les informations secrètes manipulées par un algorithme cryptographique.

L'étude des attaques matérielles est un domaine de recherche récent dont les méthodes les plus utilisées ont été introduites dans la littérature scientifique à la fin des années 1990. Ces attaques sont divisées principalement en deux familles : les attaques passives par observation, aussi appelées SCA, et les attaques actives par perturbation, aussi appelées attaques en faute (*Fault Attack* – FA). Nous évoquerons également les attaques microarchitecturales, les attaques en rétroconception, les attaques en sondage et l'utilisation de chevaux de Troie matériels.

1.1.5.A Attaques par observation

Les attaques par canaux auxiliaires exploitent les interactions entre l'implémentation matérielle d'un circuit et les canaux d'observation auxiliaires, comme le temps d'exécution d'un algorithme, la consommation énergétique d'un circuit ou ses émissions électromagnétiques. En 1996, KOCHER a démontré l'existence de corrélations entre l'information contenue par ces canaux et les données secrètes manipulées par un algorithme de sécurité. Plusieurs schémas d'attaque, comme une analyse simple de la consommation (*Simple Power Analysis* – SPA) ou une analyse différentielle de la consommation (*Differential Power Analysis* – DPA), permettent alors à un attaquant d'exploiter ces corrélations pour extraire la clé secrète d'un algorithme de chiffrement [97], [98].

Les attaques microarchitecturales sont des attaques par SCA qui exploitent des vulnérabilités microarchitecturales. Elles constituent une sous-catégorie spéciale parmi les attaques matérielles. En effet, ces attaques peuvent être réalisées par la voie logicielle sans équipement spécialisé. Les méthodes et les techniques propres à la réalisation pratique de ces attaques se rapprochent donc, de fait, des cyber-attaques. Le lecteur en quête d'informations additionnelles sur ces attaques est renvoyé vers [150].

1.1.5.B Attaques par perturbation

Les attaques en faute exploitent l'action d'une perturbation physique ou d'une perturbation des conditions nominales de fonctionnement d'un circuit pour modifier les fonctionnalités de ce dernier. Ces attaques étaient connues des communautés de *hackers* et de l'industrie de la télévision payante bien avant leur introduction dans la littérature scientifique [6]. La possibilité d'exploiter les fautes dans le calcul d'un algorithme de chiffrement pour extraire la clé secrète de l'algorithme a été introduite en 1996, puis publiée en 2001, par BONEH, DEMILLO et LIPTON [32]. Ces résultats pionniers sont à l'origine d'un domaine de recherche actif consacré à l'étude des techniques permettant d'obtenir des fautes et des schémas d'attaque exploitant ces fautes.

Les chevaux de Troie matériels sont des circuits malveillants dont l'activation suivant un schéma opératoire prédéfini permet de déclencher l'apparition d'une vulnérabilité dans un système électronique. Ce modèle de menace se présente notamment lorsque la chaîne d'approvisionnement d'un système électronique n'est pas fiable. Un attaquant situé dans cette chaîne peut alors introduire un cheval de Troie dans ce dernier en modifiant ses plans de conception. L'activation de ce cheval de Troie, lors d'une cyber-attaque par exemple, permet alors d'extraire des informations ou de prendre le contrôle du système.

1.1.5.C Outils d'édition et de sondage

Les outils d'édition et de sondage sont utilisés dans l'industrie de la microélectronique pour tester et analyser les défaillances d'un circuit après sa fabrication. Ces outils nécessitent généralement un accès à l'une des faces d'un circuit intégré.

Une sonde ionique focalisée (*Focused Ion Beam* – FIB) est un faisceau d'ions à haute vélocité qui permet d'ajouter ou de soustraire de la matière à un circuit intégré au un niveau atomique. Elle est notamment utilisée dans l'industrie microélectronique pour modifier un circuit intégré après sa production. Une FIB permet ainsi de corriger une fonctionnalité sur un circuit intégré, de modifier le routage d'un composant ou de créer un plot de connexion pour observer un signal interne d'un circuit. Si une FIB permet à un attaquant de réaliser un grand nombre de modifications, son utilisation nécessite néanmoins une forte expertise. De surcroît, une station FIB représente un investissement conséquent [6].

Le micro-sondage permet de positionner des pointes conductrices, appelées micro-sonde, sur les pistes d'un circuit ou sur des plots de connexion afin d'observer d'observer

les signaux qui transitent par ces pistes ou de forcer leur état. Cette technique requiert ainsi un accès aux pistes métalliques du circuit et la possibilité d'établir une connexion électrique avec les signaux internes du circuit.

1.1.5.D Rétroconception

La rétroconception désigne un ensemble de techniques utilisées pour extraire des informations sur les plans de conception d'un circuit. Une microscopie électronique à balayage (*Scanning Electron Microscopy* – SEM) des différentes couches d'un circuit intégré permet par exemple de reconstruire la liste des portes logiques d'un circuit et des interconnexions entre ces portes afin d'extraire les fonctions implémentées et le détail de leur implémentation physique. Cette technique peut être utilisée par ailleurs pour extraire la valeur des informations mémorisées physiquement dans une mémoire Flash ou une EEPROM [53].

1.1.6 Critères d'évaluation des menaces

Protéger un système nécessite d'adapter son niveau de sécurité aux risques auxquels il est exposé. Le guide méthodologique des Critères Communs (*Commun Criteria* – CC) définit en particulier cinq critères pour évaluer la sécurité d'un circuit intégré sur silicium [52] :

1. le temps nécessaire pour préparer et réaliser l'attaque ;
2. le niveau d'expertise générale de l'attaquant ;
3. la connaissance de l'attaquant sur le système ;
4. la complexité et le coût de l'équipement nécessaire pour réaliser l'attaque ;
5. le nombre d'échantillons nécessaire au perfectionnement d'une attaque sur un circuit.

Les attaques qui minimisent ces critères sont particulièrement attrayantes pour un attaquant.

1.2 Attaques par perturbation

1.2.1 Techniques d'injection de fautes

Une technique d'injection de fautes permet de modifier le fonctionnement d'un circuit en générant une perturbation maîtrisée. Les paragraphes qui suivent décrivent les vecteurs de perturbation recensés dans la littérature scientifique.

1.2.1.A Modification des conditions nominales de fonctionnement

Un circuit intégré est conçu pour opérer dans des conditions nominales de fonctionnement. Ces conditions définissent une plage de valeurs acceptables pour la tension d'alimentation, la température et la fréquence de fonctionnement du circuit.

Modification statique. Une augmentation de la fréquence d'horloge d'un circuit (*overclocking*) ou une diminution de sa tension d'alimentation (*underpowering*) augmentent les délais de propagation des signaux dans un circuit intégré. Si ces délais sont trop importants, les contraintes temporelles du circuit ne sont plus respectées et des fautes de mémorisation apparaissent. Une augmentation globale de la température d'un circuit intégré peut aussi être utilisée en conjonction avec une diminution de la tension d'alimentation pour faciliter l'injection de fautes [4], [101].

En ajustant précisément la tension d'alimentation grâce à un générateur de tension, des fautes peuvent être injectées sur un petit nombre de bascules dans les chemins critiques d'un circuit [23], [158]. De nombreuses applications récentes de ce principe exploitent des vulnérabilités matérielles dans la gestion de l'horloge et de l'alimentation des circuits à travers d'une vulnérabilité logicielle [91], [122], [140], [165].

Modification dynamique. La précision temporelle d'une attaque par perturbation du signal d'horloge peut être améliorée grâce à un générateur de *glitch* d'horloge. Ce générateur utilise plusieurs signaux déphasés pour moduler une période d'horloge choisie par l'attaquant avec une grande précision [2], [21]. Des résultats similaires peuvent être obtenus en modifiant temporairement la tension d'alimentation grâce à un générateur d'impulsions de tension [184], un générateur de signaux [34] ou un court-circuit temporaire de l'alimentation [128].

Ces attaques nécessitent un accès direct au signal d'horloge ou à la tension d'alimentation d'un circuit intégré. Leur mise en œuvre sur des circuits non-sécurisés nécessite donc peu d'étapes de préparation. De surcroît, les dispositifs d'injection de fautes peuvent être assemblés facilement à partir de composants disponibles dans le commerce pour un prix modéré. Pour ces raisons, les techniques d'injection de fautes par perturbation du signal d'horloge ou de la tension d'alimentation ont été extensivement étudiées [128]. Des capteurs qui détectent l'effet global de ces techniques ont aussi été proposés [182].

1.2.1.B Modification de la polarisation du substrat.

En 2012, TOBICH, MAURINE, ORDAS et al. [167] propose une technique d'injection de fautes originale qui exploite le couplage entre le substrat d'un circuit intégré et son réseau d'alimentation électrique. Le dispositif proposé par les auteurs est composé d'un générateur d'impulsions de tension et d'une micro-sonde conductrice qui est positionnée au contact du substrat, sur la face arrière d'un circuit. Une impulsion de tension modifie alors la polarisation du substrat de manière transitoire et locale [28]. Par couplage, ces perturbations de la polarisation du substrat modifient de manière transitoire et locale la polarisation des lignes d'alimentation et de la masses d'un circuit.

Une validation pratique de cette technique est réalisée avec une attaque Bellcore sur un algorithme RSA-CRT avec un dispositif de laboratoire [166] et grâce à un dispositif assemblé pour quelques centaines d'euros [130]. Un capteur pour la détection des attaques par impulsion EM et par injection de fautes par polarisation direct du substrat (*Forward Body Biaising Injection* – FBBI) est introduit dans EL-BAZE, RIGAUD et

MAURINE [72].

1.2.1.C Perturbation du champ électromagnétique

La possibilité d'injecter des fautes dans un circuit intégré grâce au phénomène d'induction EM est évoquée dès 2002 par SAMYDE, SKOROBOGATOV, ANDERSON et al. [147]. Une démonstration expérimentale de ce principe est proposée cinq ans plus tard par SCHMIDT et HUTTER [155] en exposant un microcontrôleur dans son boîtier à un arc électrique. Des dispositifs permettant d'atteindre une forte résolution spatiale et temporelle sont introduits en 2011 par POUCHERET, CHUSSEAU, ROBISSON et al. [136] et en 2012 par DEHBAOUI, DUTERTRE, ROBISSON et al. [57]. Ils peuvent aujourd'hui être construits grâce à des composants disponibles dans le commerce pour quelques centaines d'euros [1], [20], [54], [129].

Ces dispositifs utilisent une antenne, aussi appelée sonde d'injection, qui est excitée par un générateur de signaux. Deux familles de techniques sont distinguées en fonction de la nature de l'excitation. Les attaques par injection de fautes électromagnétique (*Electromagnetic Fault Injection* – EMFI) utilisent une impulsion électromagnétique (*Electromagnetic Pulse* – EMP) pour corrompre le fonctionnement d'un circuit intégré de manière transitoire [57], [168]. Les attaques par interférence électromagnétique (*Electromagnetic Interference* – EMI) utilisent quant à elles une excitation harmonique pour perturber un signal d'horloge [137] ou une source d'entropie [26], [114]. Remarquons cependant que ces domaines d'applications ne sont pas restrictifs. Une salve d'impulsion électromagnétique (*Electromagnetic Pulse* – EMP) peut être utilisée pour perturber une source d'entropie [106] et une excitation harmonique transitoire peut être utilisée pour corrompre l'exécution d'une instruction [46].

1.2.1.D Exposition à un faisceau de photons

Les interactions entre un faisceau de photons et un matériau semi-conducteur sont à l'origine de nombreux phénomènes qui perturbent le fonctionnement d'un circuit intégré. Les techniques qui exploitent ces phénomènes, à l'exception des rayons X, nécessitent préalablement d'aménager un accès au circuit intégré, soit par la face avant, soit par la face arrière.

Faisceau laser. Les effets des rayonnements cosmiques sur les circuits intégrés sont analysées grâce aux techniques d'injection laser depuis le milieu du XX^{e} siècle afin d'améliorer la fiabilité des systèmes électroniques en fonctionnement dans l'espace [85]. Ces techniques sont introduites dans le domaine de la sécurité matérielle en 2002 par SKOROBOGATOV et ANDERSON [163]. Un dispositif d'injection laser dans les infrarouges proches (*Near Infrared* – NIR) à l'état de l'art permet aujourd'hui de focaliser un faisceau de photons à l'intérieur d'un ou plusieurs spots de moins d'un micromètre de diamètre en générant des impulsions de l'ordre de la picoseconde ou de la nanoseconde sur la face avant ou la face arrière d'un circuit intégré. Des performances inférieures, mais néanmoins suffisantes pour perturber précisément le fonctionnement d'un circuit intégré, peuvent être obtenues grâce à des dispositifs d'injection fabriqués avec des composants

disponibles dans le commerce pour un coût de 500 € – 2 000 € [84], [89]. Les techniques d'injection laser peuvent être utilisées pour corrompre non seulement les calculs et les mémoires volatiles d'un circuit intégré par injection d'un photocourant [163], mais aussi le contenu d'une cellule mémoire non-volatile par effet thermique [94], [160] ou par un phénomène d'ionisation à deux photons [43].

Rayonnement UV. Le mécanisme de photoémission ultraviolet (UV) était historiquement utilisé pour effacer les transistors à grille flottante des mémoires mortes programmable électriquement [29]. Ce mécanisme est aujourd'hui remplacé par des mécanismes électriques dans les mémoires fabriquées avec la technologie EEPROM ou Flash. Cependant, les transistors à grille flottante de ces mémoires restent sensibles au mécanisme de photoémission UV. L'exposition d'une EEPROM ou d'une mémoire Flash à un faisceau UV permet ainsi d'effacer les tables de correspondances d'un algorithme de chiffrement enregistrées dans la mémoire Flash d'un circuit [154] ou de modifier l'état d'un fusible de sécurité [127]. Cependant, elle ne permet pas d'atteindre une résolution spatiale aussi forte que les techniques d'injection de fautes laser.

Faisceau de rayons X. Un accélérateur de particules permet de produire un faisceau de rayons X cohérent composé de photons à haute énergie qui peuvent être focalisés dans un spot de moins d'une centaine de nanomètres de diamètre. ANCEAU, BLEUET, CLÉDIÈRE et al. [5] ont démontré que ce dispositif pouvait être utilisé pour modifier l'état d'une cellule SRAM ou pour effacer un transistor individuel dans une mémore NOR Flash. Cependant, cette technique nécessite l'accès à des installations avancées et une forte expertise des phénomènes physiques en jeu.

1.2.2 Modélisation des fautes

Définition d'une faute. La conception d'un circuit intégré s'articule autour de spécifications matérielles. Ces dernières définissent la structure, les fonctions et les caractéristiques d'un circuit intégré. Une violation de ces spécifications est appelée une faute.

Définition d'un modèle de faute. Les relations répétables qu'entretiennent les fautes avec les conditions expérimentales de leur observation définissent un modèle de faute. Ces derniers permettent alors de décrire les effets d'une perturbation sur un circuit intégré pour un ensemble de paramètres fixés.

Abstraction d'un modèle de faute. De nombreux niveaux d'abstractions sont utilisés dans la littérature scientifique pour décrire un modèle de faute. Ces niveaux sont répartis entre la description du phénomène physique à l'origine d'une faute et ses effets au niveau applicatif [174]. Dans une analyse inductive des fautes, ces modèles sont construits progressivement du niveau physique au niveau applicatif [145]. Nous adoptons une approche similaire, bien que simplifiée, pour analyser l'effet d'une perturbation laser ou électromagnétique sur un MCU.

Nous articulons en particulier deux niveaux d'analyses, à savoir :

— **le niveau matériel**, car les perturbations étudiées sont des phénomènes physiques qui agissent sur l'implémentation matérielle d'un MCU ;

— et **le niveau logiciel**, car les actions de ces phénomènes se traduisent par des corruptions de l'état d'un MCU qui peuvent être analysées au niveau de son architecture de jeu d'instructions (*Instruction Set Architecture* – ISA) [21], [120], [138].

Les paragraphes suivants présentent les modèles de fautes matériels et logiciels utilisés dans ces travaux.

1.2.2.A Modèles de fautes matériels

Nous distinguons, d'un côté, les fautes dites permanentes ou destructives qui altèrent définitivement les fonctionnalités d'un circuit et, de l'autre, les fautes dites transitoires – aussi appelées *soft errors* – dont les effets sont temporaires. Remarquons que les fautes dites persistantes, introduites par ZHANG, LOU, ZHAO et al. [180], correspondent à un cas particulier de fautes transitoires. Dans le modèle proposé par les auteurs, ces fautes persistent pendant plusieurs opérations de chiffrement dans une mémoire volatile du circuit et disparaissent lors de sa remise à zéro. Nos travaux se sont portés sur les fautes transitoires dans l'objectif de conserver les fonctionnalités des MCU pendant la phase d'exploitation des fautes.

L'étude systématique des défaillances d'un circuit intégré causées par le passage d'un faisceau de particules est à l'origine d'une classification standardisée des fautes matérielles [74]. Les catégories de cette classification utilisées pour décrire les fautes transitoires sont détaillées ci-après.

Un SET (*Single Event Transient*) correspond à l'injection d'un photocourant transitoire lors du passage d'une particule au travers d'un circuit intégré. Ce photocourant peut se propager dans le circuit sans affecter le fonctionnement électrique du circuit (masquage électrique), l'état des sorties d'un circuit logique (masquage logique) ou les données échantillonnées de manière synchrone lors d'un front d'horloge (masquage temporel). Dans le cas contraire, une faute est observée.

Un SEU (*Single Event Upset*) correspond au basculement de l'état logique d'un élément de mémorisation bistable, ou bascule, sous l'effet d'un SET. La corruption de l'état de plusieurs bascules correspond à un MEU (*Multiple Event Upset*).

Un SEFI (*Single Event Functional Interrupt*) correspond à l'injection d'une *soft error* à l'origine d'une remise à zéro, d'un verrouillage ou d'une défaillance temporaire d'un circuit intégré. Les fonctionnalités défaillantes du circuit sont rétablies après une remise à zéro ou une mise hors tension de ce dernier.

L'injection d'une faute est qualifiée de statique lorsqu'elle n'est pas corrélée à l'état du signal d'horloge et de dynamique ou computationelle lorsque la faute apparaît suite à la corruption de l'opération d'échantillonnage d'une bascule synchrone [95].

Remarquons qu'une faute transitoire observable se traduit systématiquement par la modification de l'état logique d'une bascule dans un circuit numérique. Dans la suite de ces travaux, nous utiliserons donc indifféremment les termes faute et *soft error*.

En notant Q_{t-1} et Q_t les états d'une bascule avant et après un front d'horloge en conditions nominales de fonctionnement et en notant $\widetilde{Q_t}$ une faute injectée sur l'état Q_t de la bascule, plusieurs types de fautes peuvent être distingués.

— Une faute de type *bit flip* ($\widetilde{Q_t} = \overline{Q_t}$) correspond au cas de figure où l'état d'une bascule est inversé après l'injection d'une faute ;

— une faute de type *bit set* ($\widetilde{Q_t} = \mathbf{1}$) correspond au cas de figure où l'état d'une bascule est forcé à un après l'injection d'une faute ;

— une faute de type *bit reset* ($\widetilde{Q_t} = \mathbf{0}$) correspond au cas de figure où l'état d'une bascule est forcé à zéro après l'injection d'une faute ;

— une faute de type *no sampling* ($\widetilde{Q_t} = Q_{t-1}$) correspond au cas de figure où une bascule conserve son état initial après l'injection d'une faute.

Par définition, les fautes de type *bit flip* ne dépendent pas de l'état des bascules, contrairement aux fautes de type *bit set* et de type *bit reset*. Les fautes de type *no sampling*, quant à elles, dépendent de l'état initial d'une bascule. Par conséquent, plusieurs expériences doivent être réalisées afin d'identifier le type d'une faute. Ces expériences sont détaillées dans les sections méthodologiques du chapitre 2 et du chapitre 3.

En fonction de la localité d'une perturbation, une faute peut être observée sur un bit, un semi-octet – aussi appelé *nibble* –, un octet ou un mot [83]. Nous désignons cette caractéristique par la précision spatiale d'un modèle de faute.

Dans certains cas, la variabilité des conditions expérimentales nécessite, en outre, d'adopter un modèle probabiliste pour décrire la nature d'une faute [63]. La répétabilité p_F d'un modèle de faute F est alors définie, pour un état donné du dispositif d'injection, par l'équation 1.1, où n_F désigne le nombre de fautes associées au modèle F et N le nombre total de tentatives réalisées afin d'injecter une faute :

$$p_F = \frac{n_F}{N} \tag{1.1}$$

Par extension, un modèle de corruption de données probabiliste peut être défini par une probabilité discrète $\mathbb{P}$ sur un ensemble de valeurs $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$. La distribution des fautes est généralement représentée sous forme d'un tableau qui associe dans la ligne d'indice i la valeur ω_i et l'estimation $\mathbb{P}(\{\omega_i\})$ de sa probabilité d'observation calculée grâce à l'équation 1.1. Cette distribution peut être estimée pour un bit, un semi-octet, un octet, un mot ou un ensemble de mots, en fonction de la granularité de l'analyse et de l'extension du modèle de faute étudié.

1.2.2.B Modèles de fautes logiciels

Une faute décrite au niveau matériel sur les instructions ou les données d'un algorithme se traduit par une corruption de l'algorithme au niveau logiciel. Ces corruptions s'analysent grâce à l'architecture de jeu d'instructions (*Instruction Set Architecture* – ISA) d'un

cœur de calcul qui définit le format des instructions et les conventions de représentation des données implémentées par le cœur de calcul. Les paragraphes suivants présentent les principaux modèles de fautes logiciels.

Une corruption d'instruction décrit la modification du code binaire d'une instruction suite à l'injection d'une faute. Ce code binaire définit notamment :

— l'opcode de l'instruction, qui détermine l'opération exécutée par le cœur de calcul ;

— les opérandes de l'instruction, qui déterminent les données sur lesquelles le cœur de calcul opère et l'adresse à laquelle le résultat de l'opération est mémorisé ;

— et des conditions optionnelles qui déterminent si l'instruction est exécutée ou non par le cœur de calcul en fonction du résultat des tests conditionnels réalisés par le cœur de calcul avant l'exécution de l'instruction.

Par conséquent, une corruption d'instruction permet non seulement de modifier l'opcode et les opérandes d'une instruction [35], [104], mais aussi d'inverser un test conditionnel [146]. Le modèle du remplacement d'instruction est parfois employé pour désigner une corruption d'instruction répétable [120] et maîtrisée [146]. Ce cas de figure correspond au modèle de faute le plus observé dans la littérature scientifique [21], [120], [139].

Si l'instruction corrompue n'est pas valide, l'effet observé dépend de l'architecture du MCU. Les architectures 8 bits AVR, par exemple, n'implémentent pas de mécanisme d'exceptions pour gérer ces erreurs. Les instructions invalides sont donc exécutées comme des instructions `"nop"`, ou *no operation*, qui n'ont pas d'effet sur l'exécution d'un programme. Les architectures Arm, au contraire, lèvent une exception `"UsageFault"` lorsqu'une instruction invalide est décodée. L'exécution du programme est alors redirigée vers la routine de gestion des exceptions [13], [14].

Un saut d'instruction modélise une faute qui empêche l'exécution d'une instruction ou produit un effet équivalent. Un saut d'instruction peut être utilisé pour empêcher l'exécution d'une instruction de branchement [69], la mise à jour d'une variable ou le chargement d'un registre [99]. Par conséquent, ce modèle de faute permet de modifier facilement le flot de contrôle [156] ou l'état intermédiaire d'un programme [57].

Une corruption d'instruction correspond à un saut d'instruction si une instruction est remplacée soit par une instruction équivalente à une instruction `"nop"`, soit par une instruction qui est exécutée hors du contexte d'un programme. Le premier cas de figure correspond par exemple à l'injection d'une faute qui force le code machine d'une instruction de 16 bits à zéro. Le code machine `"0x0000"` correspond en effet à l'instruction `"movs r0, r0"` dans le jeu d'instructions Thumb et à l'instruction `"nop"` dans le jeu d'instructions AVR. Cette corruption d'instruction entraîne donc un saut d'instruction sur une architecture Arm qui supporte le jeu d'instructions Thumb ou sur une architecture AVR. Le deuxième cas de figure, quant à lui, correspond par exemple au remplacement d'une instruction par une instruction qui modifie le contenu d'une

cellule mémoire dans une plage d'adresses qui n'est pas utilisée par le programme. Ce cas de figure explique la plupart des sauts d'instruction observés en pratique [49].

Un rejeu d'instructions correspond au remplacement d'un bloc d'instructions par le bloc d'instructions qui le précède. Ce dernier est alors exécuté deux fois. Au niveau matériel, ce phénomène peut être interprété par l'injection d'une faute de type *no sampling* lors de la lecture des instructions en mémoire. Les instructions conservées dans la mémoire tampon dédiée aux instructions [21] ou dans un cache d'instructions [143] sont alors réexécutées. Notons que certaines instructions, dites idempotentes, peuvent être exécutées plusieurs fois consécutives sans modifier l'état final d'un programme [119]. Le rejeu d'une instruction idempotente permet ainsi de réaliser un saut d'instruction sur l'instruction qui la suit directement.

Une corruption de données décrit une modification des données manipulées par un algorithme suite à l'injection d'une faute. Notons que, dans le cas d'une implémentation logicielle d'un algorithme, ces fautes peuvent également être obtenues suite à une corruption ou un saut d'instruction. Depuis les travaux pionniers de BONEH, DEMILLO et LIPTON [32], les corruptions de données sont utilisées pour construire de nombreuses attaques par injection de fautes dont l'objectif est d'extraire la clé secrète d'un algorithme de chiffrement [83]. Les fautes de ce type peuvent aussi être utilisées pour contourner un mécanisme de protection [173] ou modifier le flot de contrôle d'un programme [70].

Force est de constater que les modèles de fautes logiciels ne capturent pas toute la variété des fautes possibles [139], [177]. Ils forment néanmoins une base de travail indispensable pour concevoir des attaques contre des algorithmes de sécurité.

1.2.3 Schémas d'attaques en faute

Un schéma d'attaque est un algorithme qui exploite les propriétés d'un modèle de faute pour diminuer le niveau de sécurité d'un algorithme cryptographique. Il est généralement utilisé pour extraire la clé d'un algorithme de chiffrement ou pour réduire l'espace de recherche associé à cette clé.

L'attaque Bellcore proposée par BONEH, DEMILLO et LIPTON [32] dans l'équipe éponyme constitue la première démonstration de ce principe. Elle permet de factoriser les deux nombres premiers qui constituent la clé privée d'un algorithme de chiffrement asymétrique RSA (*Rivest-Shamir-Adleman*) en injectant une faute pendant la signature d'un message. Suite à ces travaux, de nombreux schémas d'attaques ont été développés pour compromettre la sécurité d'un algorithme de chiffrement. Les paragraphes qui suivent proposent un aperçu synthétique de ces schémas.

Les DFA (*Differential Fault Analysis*) introduites par BIHAM et SHAMIR [30] permettent d'extraire la clé secrète d'un algorithme de chiffrement par blocs en analysant la propagation des fautes dans les calculs de l'algorithme. En supposant qu'une faute de type

bit flip soit injectée sur un bit de l'état de l'algorithme de chiffrement pendant une des trois dernières rondes, les auteurs proposent une méthode qui permet d'extraire la dernière clé de ronde de l'algorithme DES (*Data Encryption Standard*) et, par suite, la clé secrète de l'algorithme. Ce schéma d'attaque a été adapté pour extraire la clé secrète de l'algorithme de chiffrement AES (*Advanced Encryption Standard*) sous différentes hypothèses sur le modèle de faute [82], [135].

Les analyses *safe error* exploitent les fautes qui n'ont pas d'effet sur le résultat d'une opération de chiffrement. En particulier, les SEA (*Safe Error Analysis*) [178] exploitent les fautes dont la propagation dans le calcul d'un algorithme de chiffrement dépend de la valeur d'un bit de la clé secrète, tandis que les IFA (*Ineffective Fault Analysis*) [47] exploitent les fautes dont l'effet dépend de la valeur du bit sur lesquelles elles sont injectées, comme les fautes de type *bit set* ou *bit reset* par exemple. L'injection de fautes ciblées sur des bits choisis de la clé secrète [105] ou sur l'état d'un algorithme de chiffrement après l'addition de la clé secrète [31] permet alors d'extraire cette dernière en intégralité. Ce type d'attaque est pertinent lorsqu'un attaquant peut injecter des fautes de type *bit set* ou *bit reset* avec une forte précision spatiale (en général un bit) et une forte répétabilité.

Les analyses de sensibilité exploitent au contraire une augmentation progressive des effets d'une injection de fautes avec l'augmentation du stress appliqué sur un circuit intégré pour extraire la clé secrète utilisée par une implémentation matérielle d'un algorithme de chiffrement. Une FSA (*Fault Sensitivity Analysis*) [103] exploite à cet effet les corrélations entre les données manipulées par l'algorithme et les modèles de fautes obtenus progressivement par violation de contraintes temporelles. Une DFIA (*Differential Fault Intensity Analysis*) [79] exploite, quant à elle, les corrélations entre l'intensité d'une perturbation et le biais statistique introduit dans la distribution des résultats d'une opération de chiffrement. Elle se rapproche, en ce sens, des analyses statistiques décrites dans le paragraphe suivant.

Les analyses statistiques , comme les attaques par SFA (*Statistical Fault Analysis*) [75], par SIFA (*Statistical Ineffective Fault Analysis*) [59] et par PFA (*Persistent Fault Analysis*) [180], [181], exploitent la connaissance des textes chiffrés collectés après l'injection d'une faute afin d'extraire la clé secrète d'un algorithme de chiffrement. Pour ce faire, ces schémas analyse les biais introduits dans la distribution des textes chiffrés lorsqu'une faute est injectée, soit de manière transitoire dans un octet de l'état de l'algorithme de chiffrement [59], [75], soit de manière persistante dans la table de correspondances de l'algorithme [180], [181]. Ils permettent ainsi de dépasser les limitations des schémas d'attaques précédents qui supposent pour la plupart qu'un attaquant peut répéter plusieurs fois le chiffrement d'un même texte clair.

Cet aperçu des différents schémas d'attaques illustre l'importance de maîtriser les dispositifs expérimentaux pour l'injection de fautes et de comprendre les modèles de fautes qui peuvent être obtenus de manière réaliste pour évaluer les risques associés à un schéma d'attaques.

1.2.4 Protections

Historiquement, les attaques par perturbation ont été étudiées pour protéger les circuits intégrés destinés à des applications qui nécessitent un haut niveau de sécurité, comme les applications bancaires ou la télévision payante [6], [176]. Un aperçu synthétique des principales stratégies pour la protection de ces circuits est proposé dans les paragraphes suivants.

Le durcissement matériel désigne l'ensemble des techniques de conception matérielle dont l'objectif est d'augmenter la fiabilité d'un système électronique soumis à des perturbations. Ces techniques adaptent notamment le processus de fabrication, le dimensionnement et le placement des structures microélectroniques [93], ainsi que leur architecture [39], pour diminuer l'effet d'une perturbation sur un circuit intégré.

Le principe de redondance permet d'améliorer la fiabilité d'un système en répétant plusieurs fois une même opération. Dans le cadre des attaques en faute, ce principe se fonde ainsi sur l'hypothèse qu'un attaquant ne peut pas reproduire plusieurs fois la même faute. La répétition peut être temporelle, lorsqu'une opération est réalisée plusieurs fois par le même bloc matériel, ou spatiale, lorsqu'elle est réalisée par plusieurs blocs matériels différents [22]. La comparaison des résultats obtenus lors des différentes itérations permet alors de détecter ou de corriger une faute. Le principe de redondance permet également de protéger les instructions et les données manipulées par un programme en modifiant leur représentation [108]. Les codes correcteurs d'erreurs (*Error Correcting Code* – ECC) sont une application classique de ce principe.

Les mécanismes de CFI (*Control Flow Integrity*) garantissent l'intégrité des branchements et des retours de fonctions réalisés pendant l'exécution d'un programme. Ils empêchent ainsi un attaquant d'exploiter une vulnérabilité matérielle ou logicielle pour exécuter une suite d'instructions non-autorisées. La vérification des adresses de branchements et de retours peut être implémentée de manière logicielle [102], [138] ou matérielle [48], [56].

Les techniques d'obfuscation ont pour objectif de masquer le fonctionnement interne d'un système électronique. À titre d'exemple, l'exécution d'opérations factices (*dummy*) [81], le ré-ordonnancement des opérations (*shuffling*) et l'utilisation d'une horloge interne irrégulière [118] introduisent une composante aléatoire dans le profil temporel de l'exécution d'un algorithme. L'application d'algorithmes de *scrambling* sur les mémoires et les bus de données d'un circuit permet en outre de décorréler, dans une certaine mesure, les chemins de données et l'organisation physique des informations de leurs représentations [142]. Des modifications intentionnelles peuvent également être insérées dans le processus de fabrication d'un circuit et dans le placement et le routage des composants microélectroniques afin de camoufler les fonctions logiques du circuit et leur implémentation matérielle [175].

La détection d'une intrusion ou d'une faute repose sur l'existence de capteurs matériels capables de détecter la présence de lumière [44], les variations du signal d'horloge et de la tension d'alimentation [58], la présence d'une interférence EM [86] ou d'une EMP [72], ou encore une violation globale des contraintes temporelles d'un circuit [184]. Les composants d'un circuit intégré peuvent également être protégés sous un bouclier fabriqué dans les niveaux de métallisation supérieurs du circuit. L'intégration du bouclier dans un dispositif actif permet alors de détecter les tentatives de modification du bouclier par une mesure des constantes propres à ce dernier [78].

La mise en œuvre de ces contre-mesures nécessite de comprendre les capacités d'un attaquant au regard des techniques d'injection de fautes utilisées par ce dernier. Les techniques d'injection de fautes locales, en particulier, permettent de ne perturber qu'une partie d'un circuit. Cette caractéristique est à l'origine d'une double difficulté pour les concepteurs de protections matérielles. Elle permet en effet d'obtenir de nombreux modèles de fautes matériels dont la qualification exhaustive est délicate, tout en permettant à un attaquant d'échapper aux dispositifs de détection implémentés dans un circuit en localisant une perturbation hors du champ de détection de ces dispositifs. Dans le cadre de ces travaux, nous nous sommes intéressés à l'utilisation de deux techniques d'injection de fautes locales : l'injection de fautes EM et l'injection de fautes laser. L'état de l'art dans l'utilisation de chacune de ces techniques pour perturber le fonctionnement d'un MCU est présenté dans la section 1.3 et dans la section 1.4.

1.3 Injection de fautes par impulsion EM sur un MCU

Malgré l'intérêt porté par la communauté des attaques en faute pour les attaques par EMFI, une description complète des mécanismes de fautes en jeu reste encore hors de portée. Les paragraphes qui suivent introduisent donc, en premier lieu, les principaux mécanismes de fautes proposés dans la littérature scientifique afin de présenter, en second lieu, l'état de l'art de la modélisation d'une EMFI sur un microcontrôleur.

1.3.1 Fonctionnement d'un circuit synchrone

Un circuit synchrone est un circuit numérique dont les opérations sont cadencées par un signal périodique, appelé signal d'horloge. La synchronisation des différents composants d'un circuit synchrone s'effectue généralement lors d'une transition du signal d'horloge – aussi appelée front montant – de l'état bas à l'état haut.

Une description RTL (*Register Transfer Level*) permet d'analyser l'évolution temporelle des signaux logiques dans un circuit synchrone. Dans cette description, les éléments d'un circuit sont divisés en deux catégories : les éléments combinatoires et les éléments séquentiels. Les premiers sont des circuits logiques dont les sorties correspondent au résultat d'une opération booléenne sur les entrées. Les seconds, aussi appelés registres, sont des circuits de mémorisation dont les entrées sont copiées sur les sorties à chaque front montant du signal d'horloge. Les registres permettent ainsi de mémoriser le résultat d'une opération entre deux fronts montants du signal d'horloge.

1.3.1.A La bascule D

Les registres sont généralement implémentés par des bascules à verrouillage D (*D flip-flop* – DFF). Ces circuits sont composés de deux circuits de mémorisation asynchrones, ou verrous, connectés en série : le verrou maître et le verrou esclave. Un signal de *set* ou de *reset* asynchrone permet de fixer l'état mémorisé par chacun des verrous après la mise sous tension du circuit.

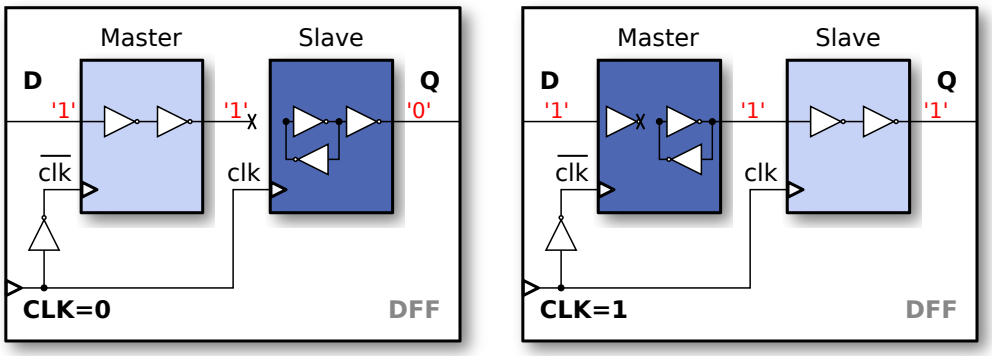
Un verrou possède deux modes de fonctionnement : un mode ouvert et un mode fermé. Dans le mode ouvert, le verrou recopie son signal d'entrée sur sa sortie. L'état du signal d'entrée est alors mémorisé sur la sortie du verrou. Dans le mode fermé, le verrou maintient l'état de son signal de sortie, indépendamment des variations de son signal d'entrée. L'état de la sortie du verrou correspond alors à l'état mémorisé par le verrou. Un signal d'activation permet au verrou d'adopter l'un ou l'autre des modes de fonctionnement.

Dans une architecture de bascule déclenchée sur front montant – ou *edge triggered* DFF – les verrous maître et esclave fonctionnent en alternance sur les deux demi-cycles d'une période d'horloge. Ce fonctionnement est illustré sur la Figure 1.1.

— Lorsque le signal d'horloge est à l'état bas, le verrou maître est ouvert ($\overline{\text{clk}} = 1$) et le verrou esclave est fermé (clk = 0). Le signal d'entrée de la DFF est donc recopié par le verrou maître, tandis que l'état du signal de sortie de la DFF est maintenu constant par le verrou esclave. Ce cas de figure est illustré sur la Figure 1.1a.

— Lorsque le signal d'horloge est à l'état haut, le verrou maître est fermé ($\overline{\text{clk}} = 0$) et le verrou esclave est ouvert (clk = 1). Le signal d'entrée de la DFF est alors déconnecté du verrou maître qui maintient l'état de son signal de sortie. Le signal de sortie du verrou maître est recopié sur la sortie de la DFF par le verrou esclave. Ce cas de figure est illustré sur la Figure 1.1b.

L'état du signal de sortie d'une DFF est ainsi maintenu constant entre deux fronts d'horloge, indépendamment des variations du signal d'entrée de la DFF. La mise à jour de la sortie de la DFF avec l'état du signal d'entrée à chaque front d'horloge correspond à un échantillonnage.



(a) Recopie du signal d'entrée.

(b) Mise à jour du verrou esclave.

FIGURE 1.1 – Fonctionnement d'une bascule D déclenchée par un front montant du signal d'horloge.

La valeur mémorisée par une DFF est le produit d'un arbitrage entre les variations du signal d'horloge et les variations du signal d'entrée de la DFF réalisé par le verrou maître pendant un échantillonnage [87]. Pour éviter les phénomènes de métastabilité et les erreurs de mémorisation, le signal d'entrée d'un registre doit être stable avant le front d'horloge pendant un intervalle de temps noté t_{su}, qui correspond au temps de *setup* du registre, et après le front d'horloge pendant un intervalle de temps noté t_{ho}, qui correspond au temps de *hold* du registre.

1.3.1.B Contraintes temporelles de fonctionnement

Les éléments combinatoires et séquentiels d'un circuit synchrone présentent des délais de fonctionnement qui doivent être soigneusement étudiés lors de la conception du circuit. Nous illustrons les contraintes temporelles d'un circuit synchrone sur le circuit séquentiel de la Figure 1.2. Ce circuit est constitué d'un circuit combinatoire, noté Σ, et de deux registres implémentés par des DFF. Le signal d'horloge – ou *clock* – est représenté par l'acronyme « clk ».

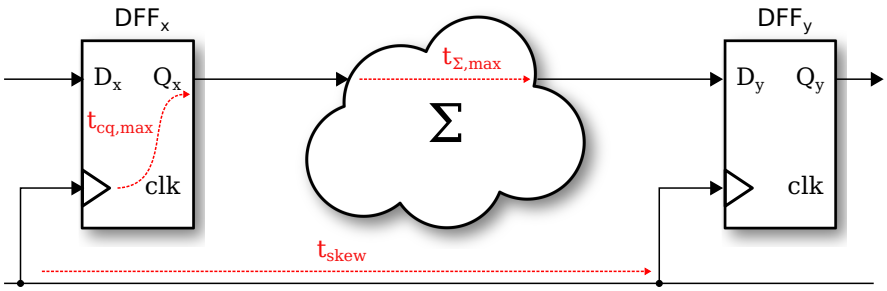


FIGURE 1.2 – Délais de propagation dans un circuit séquentiel.

Lors d'un front d'horloge, l'état Q_x de la sortie du premier registre est d'abord mis à jour avec l'état D_x de l'entrée du registre. Le signal de sortie du registre commence à changer après un délai $t_{cq,min}$ et se stabilise après un délai $t_{cq,max}$ qui correspond au temps de propagation du signal dans le registre après le front d'horloge. À sa suite, le signal de sortie D_y du circuit combinatoire commence à changer après un délai $t_{\Sigma,min}$ et se stabilise après un délai $t_{\Sigma,max}$ qui correspond au temps de propagation du signal dans le circuit combinatoire. Le signal de sortie du circuit combinatoire est finalement échantillonné par le deuxième registre lors du front d'horloge suivant avec un retard t_{skew} égal au décalage de phase du signal d'horloge entre les deux registres. L'état Q_y de la sortie du deuxième registre est alors mis à jour avec l'état D_y de la sortie du circuit combinatoire.

Pour éviter les erreurs de mémorisation, la propagation des signaux doit respecter les temps de *setup* et de *hold* des registres. En notant t_{clk} la période d'horloge, les délais de fonctionnement doivent ainsi vérifier l'inégalité 1.2 (contrainte de *setup*) et l'inégalité 1.3 (contrainte de *hold*) suivantes [124] :

$$t_{clk} + t_{skew} \geq t_{cq,max} + t_{\Sigma,max} + t_{su} \tag{1.2}$$

$$t_{ho} + t_{skew} \leq t_{cq,min} + t_{\Sigma,min} \tag{1.3}$$

Ces contraintes définissent respectivement la borne supérieure (inégalité 1.2) et la borne inférieure (inégalité 1.3) du temps de propagation d'un signal entre deux registres.

1.3.2 Injection d'une tension par induction électromagnétique

Une EMFI exploite le phénomène d'induction électromagnétique pour perturber le fonctionnement d'un circuit intégré. Ce phénomène explique l'apparition d'une force électromotrice $e_{f.e.m.}$ dans un chemin fermé $\mathscr{C}$ de pistes métalliques lorsque le flux magnétique Φ à travers la surface S délimitée par ce chemin varie dans le temps. Au niveau macroscopique, ce phénomène est décrit par l'équation de Lenz-Faraday :

$$e_{f.e.m.} = -\frac{d\Phi}{dt} \tag{1.4}$$

où le flux magnétique Φ est obtenu en intégrant la densité de flux magnétique $\overrightarrow{\mathrm{B}}$ sur la surface S délimitée par le circuit fermé $\mathscr{C}$ suivant l'équation :

$$\Phi = \oiint_{\mathrm{S}} \overrightarrow{\mathrm{B}} \,.\, \overrightarrow{d\mathrm{S}} \tag{1.5}$$

L'équation 1.4 explique ainsi l'apparition d'une tension parasite sous l'effet d'une perturbation électromagnétique dans chacun des grands réseaux de pistes métalliques d'un circuit intégré, à savoir :

— le réseau d'alimentation électrique (*Power Ground Network* – PGN) [57], [62] ;

— l'arbre d'horloge [46], [80] ;

— les lignes de remise à un (*set*) et de remise à zéro (*reset*) des DFF [132].

À titre d'exemple, les chemins fermés de pistes métalliques dans un réseau d'alimentation sont illustrés sur la Figure 1.3.

En 2012, DEHBAOUI, DUTERTRE, ROBISSON et al. [57] proposent d'utiliser une sonde d'injection constituée d'un solénoïde enroulé autour d'un noyau de ferrite pour perturber un circuit intégré grâce à une impulsion électromagnétique. L'existence de ce champ magnétique est une conséquence directe de la loi de Maxwell-Ampère. En notant $\nabla\wedge$ l'opérateur rotationnel, cette loi relie, en régime quasi-stationnaire, la densité de courant $\overrightarrow{\mathrm{J}}$ en un point du solénoïde et le champ magnétique $\overrightarrow{\mathrm{H}}$ au voisinage de ce point par l'équation :

$$\nabla \wedge \overrightarrow{\mathrm{H}} = \overrightarrow{\mathrm{J}} \tag{1.6}$$

La densité de flux magnétique $\overrightarrow{\mathrm{B}}$ dans la ferrite s'obtient alors grâce à la relation constitutive de la ferrite :

$$\overrightarrow{\mathrm{B}} = \mu \overrightarrow{\mathrm{H}} \tag{1.7}$$

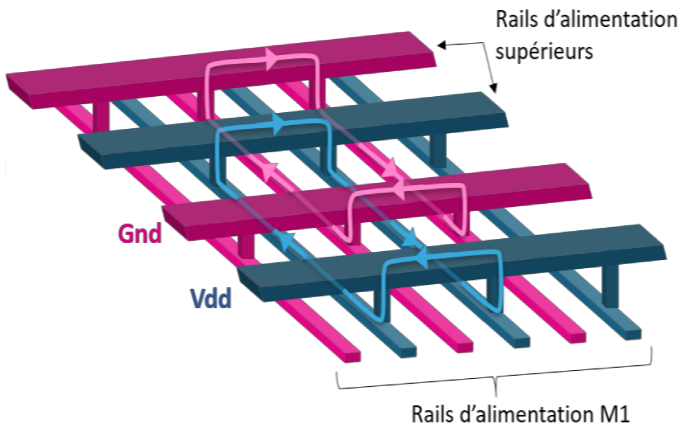


FIGURE 1.3 – Chemins fermés de pistes métalliques dans le réseau d'alimentation d'un circuit intégré [61].

où la quantité μ désigne la perméabilité magnétique du matériau, supposé uniforme et isotrope. Elle est de l'ordre de 2 000 H/m pour un barreau en ferrite 78. Les propriétés ferromagnétiques d'une ferrite sont ainsi utilisées pour amplifier les effets d'une variation de la densité de courant $\vec{J}$ dans le solénoïde sur la densité de flux magnétique $\vec{B}$ et, du fait de l'équation 1.4, sur la tension induite $e_{f.e.m.}$ dans les réseaux de pistes métalliques d'un circuit intégré.

En utilisant des ferrites de quelques centaines de micromètres de diamètre, ce procédé permet de générer une EMP avec une grande précision spatiale au voisinage du solénoïde. L'excitation d'un solenoïde avec une impulsion de tension de quelques nanosecondes de largeur grâce à un générateur d'impulsions de tension permet également d'obtenir une forte précision temporelle.

La modélisation des effets d'une EMP sur l'ensemble des réseaux métalliques d'un circuit intégré représente un défi de taille. L'influence prédominante des conditions expérimentales sur les effets d'une perturbation électromagnétique accentue, de surcroît, la difficulté d'une telle entreprise. Il convient donc de considérer plusieurs mécanismes de fautes dans la modélisation physique des effets d'une EMP sur un MCU.

1.3.3 Discussion des mécanismes de fautes EM

Une EMP génère une tension transitoire sur le PGN d'un circuit intégré. Or, les caractéristiques d'un circuit intégré sont sensibles aux variations du processus de fabrication du circuit, de sa température de fonctionnement et de sa tension d'alimentation. En particulier, une augmentation de la température du circuit ou une diminution de sa tension d'alimentation provoquent une augmentation des temps de propagation du circuit. Le non-respect du temps de *setup* d'un registre entraîne alors l'apparition d'une faute de mémorisation [158].

Une augmentation du délai de propagation des signaux peut également être observée lorsqu'un PGN est excité par une impulsion de tension. Ce phénomène s'explique notamment par une diminution temporaire de la tension d'alimentation sous l'effet d'une

impulsion de polarité positive ou négative [184]. Au regard de ces considérations, le mécanisme de violation des contraintes temporelles d'un circuit synchrone s'applique, en théorie, à l'injection d'une faute causée par une EMP.

Cette hypothèse a été vérifiée en 2012 par DEHBAOUI, DUTERTRE, ROBISSON et al. [57] sur une implémentation matrice prédiffusée programmable par l'utilisateur (*Field Programmable Gate Array* – FPGA) d'un algorithme de chiffrement AES. Trois caractéristiques d'un mécanisme de violation de contraintes temporelles ont été identifiées par les auteurs :

— les fautes sont localisées à proximité des chemins de l'AES avec la marge temporelle la plus faible, aussi appelés chemins critiques de l'AES ;

— une augmentation progressive de l'intensité de la perturbation électromagnétique fait apparaître un effet de seuil où la nature des fautes change progressivement d'aléatoire (mémorisation métastable) à déterministe (mémorisation précoce) ;

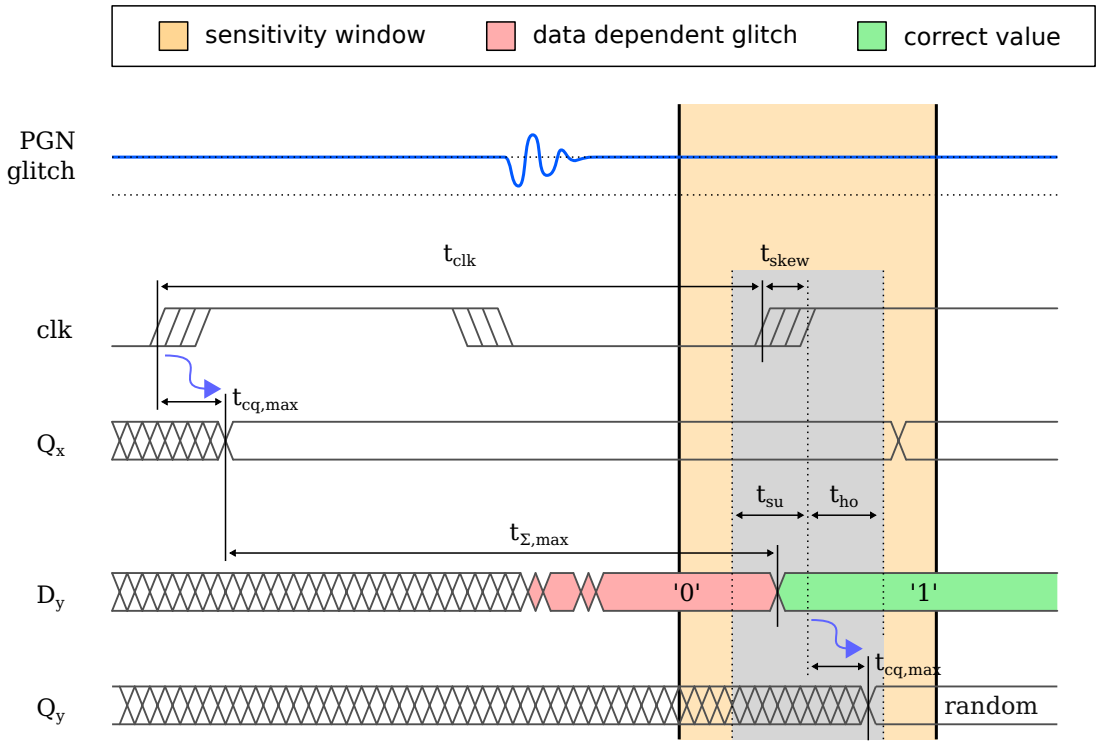
— les fautes observées dépendent des données d'entrée de l'AES.

Ces caractéristiques sont illustrées sur la Figure 1.4 qui décrit l'effet d'une perturbation de faible intensité (Figure 1.4a) et de forte intensité (Figure 1.4b) sur la mémorisation de l'état logique d'un signal dans le cas du circuit séquentiel présenté sur la Figure 1.2.

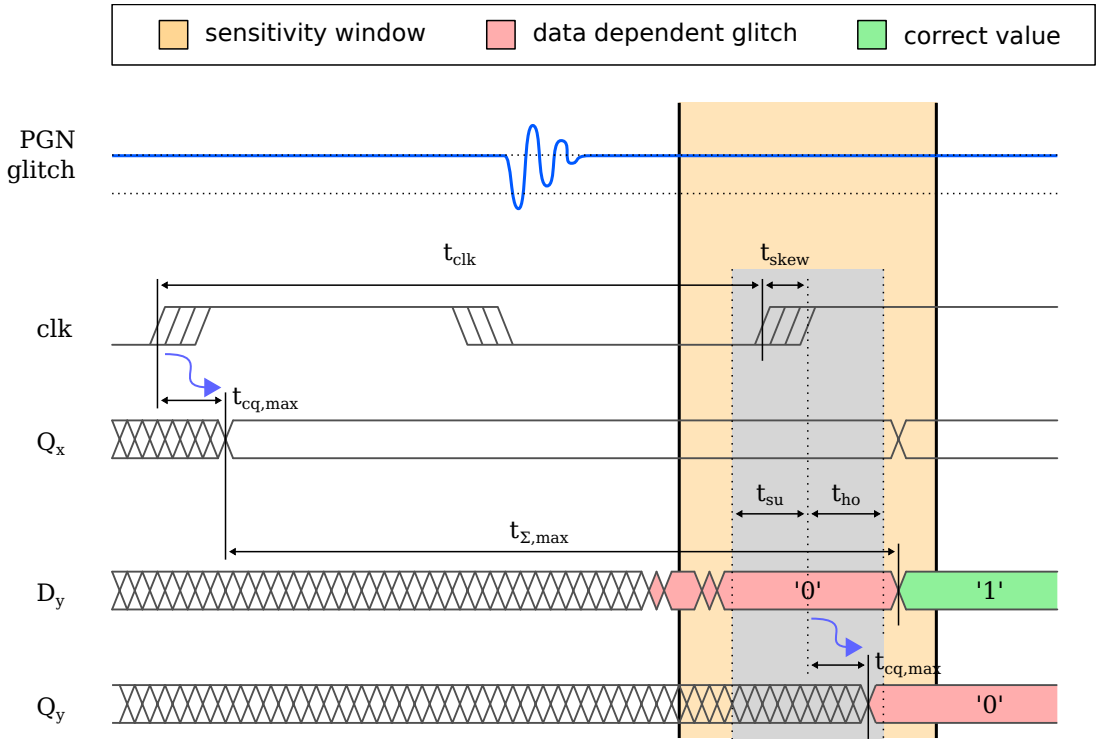
Cependant, les résultats obtenus lors d'expérimentations sur l'injection de fautes EM par ORDAS, GUILLAUME-SAGE, TOBICH et al. en 2014 sortent du cadre d'interprétation du mécanisme de violation de contraintes temporelles. Les auteurs ont en effet réussi à injecter des fautes de type *bit set* et de type *bit reset* dans les DFF d'un circuit intégré spécifique (*Application Specific Integrated Circuit* – ASIC) de manière statique en déconnectant l'horloge du circuit [133]. Ces résultats surprenants ont été attribués à l'existence d'un phénomène de couplage entre la sonde d'injection EM et les signaux de *set* et de *reset* asynchrones des DFF. Ils apportent la preuve expérimentale que plusieurs mécanismes de fautes coexistent lors d'une injection de fautes EM.

L'année suivante, ORDAS, GUILLAUME-SAGE et MAURINE [134] montrent sur un circuit FPGA que la probabilité d'injecter une faute de type *bit set* ou de type *bit reset* varie avec la même période que l'horloge du circuit. Dans leurs expériences, les auteurs ont constaté l'existence de fenêtres temporelles à l'intérieur desquelles la probabilité d'injecter une faute de type *bit set* ou de type *bit reset* est proche de 100 % et à l'extérieur desquelles la même probabilité est quasi-nulle. Ces fenêtres, aussi appelées *sensitivity window*, sont illustrées sur la Figure 1.5. D'autres expériences réalisées à différentes fréquences sur le même circuit ont montré en outre que la largeur de ces fenêtres ne dépend pas de la fréquence de fonctionnement du circuit. Ces résultats ont été reproduits expérimentalement sur plusieurs circuits FPGA par EL-BAZE [71], dont les travaux confirment les prédictions faites par ORDAS, GUILLAUME-SAGE et MAURINE dans différents scénarios d'expérimentations.

Pour rendre compte de ces observations, ORDAS, GUILLAUME-SAGE et MAURINE [134] ont introduit les fautes d'échantillonnage – ou *sampling faults* – qui traduisent les effets d'une perturbation sur l'opération d'échantillonnage d'une DFF autour d'un front montant du signal d'horloge. Une explication théorique des fautes d'échantillonnage est



(a) Faute aléatoire causée par un phénomène de métastabilité de la bascule DFF_y.



(b) Faute déterministe causée par une mémorisation précoce du signal d'entrée de la bascule DFF_y.

FIGURE 1.4 – Effets d'une augmentation progressive de l'intensité d'une perturbation sur la mémorisation d'un signal.

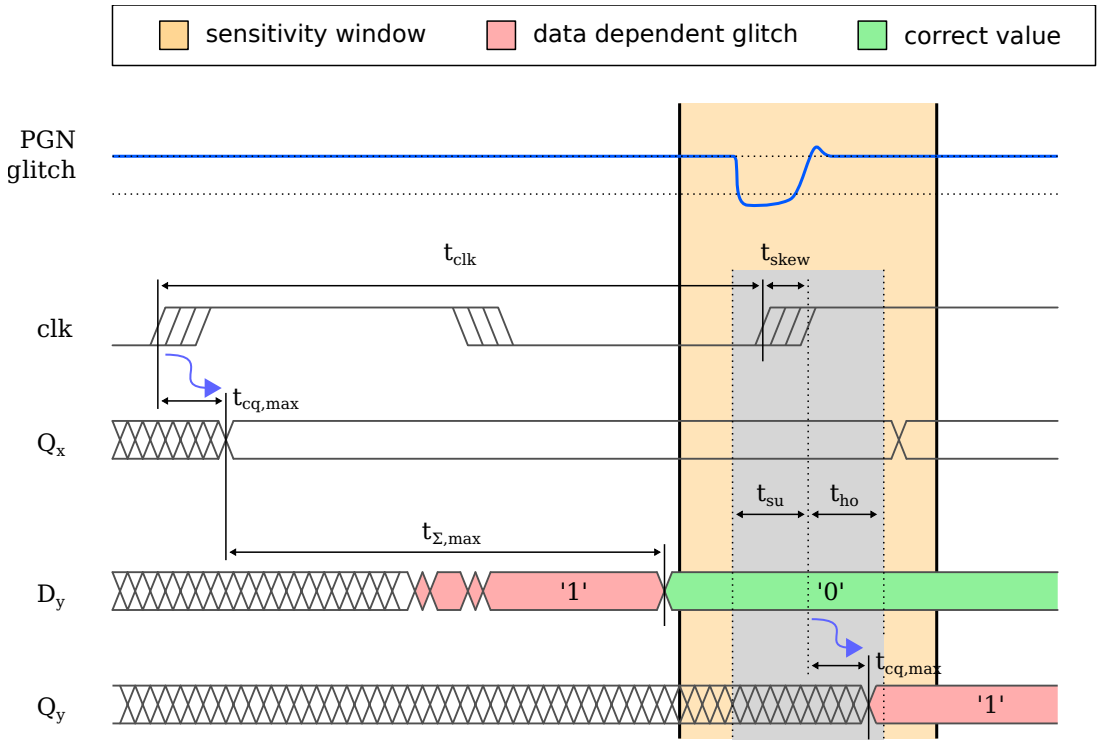


FIGURE 1.5 – Faute de type *bit set* causée par une faute d'échantillonnage.

proposée quatre ans plus tard par DUMONT, LISART et MAURINE [62] grâce à une modélisation électrique des effets d'une EMP sur le réseau d'alimentation électrique d'un modèle générique de circuit intégré. Dans ce modèle, une inversion temporaire et locale de la polarité de l'alimentation d'une DFF se produit sous l'effet d'une perturbation électromagnétique. Le retour de la DFF à ses conditions nominales de fonctionnement autour d'un front d'horloge explique alors l'apparition de fautes de type *bit set* ou de type *bit reset* en fonction de la polarité de l'impulsion. Ce mécanisme est illustré sur la Figure 1.5.

Remarquons cependant que la description théorique des fautes injectées grâce à une EMP reste incomplète. En effet, si la largeur des fenêtres de sensibilité varie peu avec la période d'horloge d'un circuit [132], le seuil d'injection d'une faute dépend, lui, de la fréquence de fonctionnement du circuit [104]. Ces éléments suggèrent ainsi l'existence d'un mécanisme plus général, dont l'étude est hors du cadre de ces travaux.

1.3.4 Modèles de fautes logiciels

Les premières études sur l'EMFI ont montré qu'une EMP localisée spatialement et temporellement permet d'obtenir un saut d'instruction pendant l'exécution d'un programme sur un MCU [57], [121]. Ces observations ont été vérifiées sur une grande variété de circuits, dont un MCU 8 bits AVR [57], un MCU 32 bits Arm Cortex-M [120], un SoC 32 bits Arm Cortex-A9 [139] et un SoC 64 bits Arm Cortex-A53 [76]. Un saut d'instruction causé par une EMP est un modèle de faute versatile qui permet par exemple de réaliser une attaque par DFA [57], de contourner une vérification de sécurité dans une pile de protocoles afin d'extraire le contenu d'une mémoire protégée [129] ou de forcer

l'authentification sur le circuit d'un utilisateur non-autorisé [76].

Au début de ces travaux, l'hypothèse selon laquelle un attaquant ne peut pas injecter un saut de plus de deux instructions avait déjà été remise en cause par les travaux de RIVIÈRE, NAJM, RAUZY et al. [143] en 2015 et de YUCE, GHALATY, SANTAPURI et al. [179] en 2016. Les premiers ont montré qu'une faute injectée pendant la mise à jour du cache d'instructions d'un MCU permet de remplacer quatre instructions de 32 bits alignées sur un bloc de 128 bits par les quatre instructions précédentes. Ce modèle de faute décrit un rejeu de quatre instructions causé par l'injection d'une faute d'échantillonnage sur la mémoire tampon de préchargement des instructions du MCU [143]. Les seconds ont montré que les paramètres d'un *glitch* d'horloge peuvent être ajustés de sorte à corrompre plusieurs instructions à différents étages du *pipeline* d'un processeur à jeu d'instructions réduit (*Reduced Instruction Set Computer* – RISC). Une protection logicielle contre les attaques en faute par duplication d'instructions [24], [121] peut ainsi être contournée avec une seule faute [179].

Un saut affectant jusqu'à six instructions consécutives avec une faible répétabilité a été obtenu par ELMOHR, LIAO et GEBOTYS [73] sur un cœur de calcul Sifive E31 en ajustant l'amplitude de l'impulsion de tension à l'origine d'une perturbation électromagnétique. Remarquons cependant que le cœur de calcul, cadencé à 320 MHz, est composé d'un *pipeline* à cinq étages et embarque plusieurs niveaux de caches. Or, l'influence de ces fonctionnalités matérielles n'est pas analysée par les auteurs. La question se pose alors de savoir si les sauts de plusieurs instructions reposent intégralement sur les fonctionnalités microarchitecturales d'un MCU que sont les caches et le *pipeline* d'un cœur de calcul.

1.3.5 Modèles de fautes matériels

Des analyses exhaustives des modèles de fautes obtenus grâce à une EMP sur un MCU [120] et sur un SoC [139], [170] ont révélé que les sauts d'instruction et les rejeux d'instruction ne constituent qu'une partie des modèles de fautes observables.

En 2013, MORO, DEHBAOUI, HEYDEMANN et al. [120] ont en effet montré que la plupart des fautes injectées sur un MCU 32 bits Arm Cortex-M peuvent être expliquées par des corruptions d'instructions ou de données. Les auteurs ont ainsi observé qu'une instruction spécifique peut être remplacée par une autre de manière répétable en ajustant l'instant d'injection d'une perturbation électromagnétique. Des fautes de type *bit set* ont aussi été injectées sur les données en transfert entre la mémoire et le cœur de calcul. Une corrélation a été observée entre le nombre de bits forcés à `'1'` et l'amplitude de l'impulsion de tension utilisée pour générer une perturbation électromagnétique. Pour rendre compte de ces observations, les auteurs ont introduit un modèle de faute matériel dans lequel ils supposent qu'une perturbation électromagnétique provoque une faute de *timing* dans le chemin de préchargement des instructions et des données d'un MCU déjà identifié comme un chemin critique dans la littérature scientifique [23].

Six ans plus tard, LIAO et GEBOTYS [104] ont proposé une analyse détaillée des bits corrompus dans une instruction sous l'effet d'une EMP sur la face arrière d'un MCU 16

bits PIC16F687. Dans cette analyse, les auteurs ont mis en évidence l'injection de fautes de type *bit reset* et, exceptionnellement, d'une faute de type *bit set* sur des blocs de bits groupés à l'intérieur des instructions transférées entre la mémoire Flash et le cœur de calcul d'un MCU 16 bits PIC16F687. Si ces résultats peuvent être attribués à la localité d'une injection EM, l'influence de la position de l'objectif sur les fautes obtenues n'est toutefois pas analysée.

La même année, BECKERS, BALASCH, GIERLICHS et al. [27] ont proposé une analyse des bits corrompus lors d'un chargement de données par une EMFI réalisée sur la face avant d'un MCU 8 bits AVR au niveau de la mémoire Flash embarquée. Les auteurs ont constaté, d'une part, que les instants d'injection d'une faute sur une instruction ou une donnée sont regroupés dans des fenêtres temporelles telles que décrites par le mécanisme de fautes d'échantillonnage et, d'autre part, que seules des fautes de type *bit set* peuvent être injectées sur les données. Si des travaux précédents ont mis en lumière le rôle de la polarité d'une impulsion de tension dans l'injection de fautes de type *bit set* ou de type *bit reset* liées au mécanisme de *sampling* sur un circuit FPGA [133], une analyse similaire pour un MCU n'est cependant proposée ni dans [104], ni dans [27].

L'analyse des modèles de fautes matériels obtenus sur un SoC en conditions réelles est particulièrement délicate du fait de la diversité des effets engagés par une EMFI sur les différents composants d'un circuit intégré [107]. Si d'importantes contributions méthodologiques ont été apportées pour analyser les fautes obtenues au niveau de l'ISA d'un SoC [139] et pour isoler les blocs microarchitecturaux à l'origine d'une défaillance matérielle [170], la compréhension de l'influence des paramètres expérimentaux sur les modèles de fautes observés reste néanmoins lacunaire.

1.4 Injection de fautes par impulsion laser sur un MCU

Une impulsion laser permet à un attaquant de perturber temporairement et localement un circuit intégré en foctionnement avec une grande précision temporelle et spatiale. De surcroît, une injection de fautes laser (*Laser Fault Injection* – LFI) sur différents composants d'un MCU engage des effets très divers qui peuvent être exploités pour compromettre la sécurité d'un circuit intégré. Dans l'objectif de comprendre au mieux ces effets, cette section présente l'état de l'art dans la modélisation des fautes obtenues sur un MCU grâce à une impulsion laser.

1.4.1 Mécanisme d'injection de fautes laser

Dans les paragraphes suivants, nous présentons tout d'abord la structure des transistors fabriqué dans la technologie CMOS qui domine historiquement les technologies de fabrication de circuits intégrés [45]. Nous détaillons, ensuite, les conditions d'application du mécanisme photoélectrique sur un circuit intégré. Nous introduisons, enfin, le modèle électrique qui est utilisé dans ces travaux pour comprendre les effets d'une impulsion laser sur un MCU.

1.4.1.A Le transistor CMOS

Les transistors CMOS sont des structures microélectroniques planaires constituées de deux zones de diffusion – la source et le drain – implantées dans un substrat semi-conducteur en silicium. L'ensemble est surmonté d'une grille de contrôle en polysilicium isolée électriquement du matériau semi-conducteur par une couche d'oxyde de silicium [110].

Les transistors NMOS et PMOS se distinguent par la nature du dopage des zones de diffusion et du substrat. Ainsi, le transistor NMOS est constitué de deux zones de diffusion enrichies en électrons (dopage $n+$) implantées dans un substrat appauvri en électrons (dopage p) qui est généralement polarisé à la masse. Le transistor PMOS est quant à lui constitué de deux zones de diffusion appauvries en électrons (dopage $p+$) implantées dans un puits N enrichi en électrons (dopage n) qui est généralement polarisé à la tension d'alimentation. Ce puits est implanté à l'intérieur du même substrat que le transistor NMOS [141]. Une vue en coupe des deux dispositifs est présentée sur la Figure 1.6. Il Y figurent la source **(1)**, la grille **(2)**, le drain **(3)** et l'oxyde de grille **(4)** d'un transistor, ainsi que les contacts de polarisation du substrat P **(5)**, ou *p-substrate*, et du puits N **(6)**, ou *n-well*.

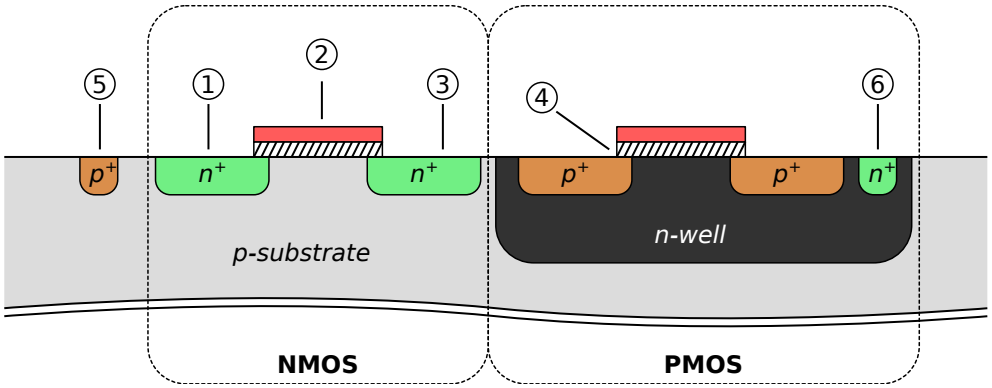


FIGURE 1.6 – Vue en coupe d'un dispositif NMOS (à gauche) et d'un dispositif PMOS (à droite).

Ces dispositifs sont modélisés par des composants électriques à quatre terminaux, à savoir la grille G, la source S, le drain D et le substrat B, ou *bulk*. Ces dispositifs opèrent principalement suivant deux modes : le mode bloqué et le mode passant.

Transistor bloqué. En l'absence de polarisation, les différences de dopage entre les zones de diffusion et le substrat sont telles que le drain et la source du dispositif sont isolés électriquement. Le dispositif est alors dit bloqué.

Transistor passant. En revanche, lorsqu'une tension supérieure à la tension de seuil du dispositif est appliquée sur la grille de contrôle, un canal de conduction se forme entre le drain et la source du transistor à l'interface entre le substrat et l'oxyde de grille. Le dispositif est alors dit passant.

Les transistors CMOS se comportent ainsi comme des sources de courant contrôlées par la tension appliquée sur la grille de contrôle. La Figure 1.7 introduit les schémas électriques des transistors NMOS et PMOS.

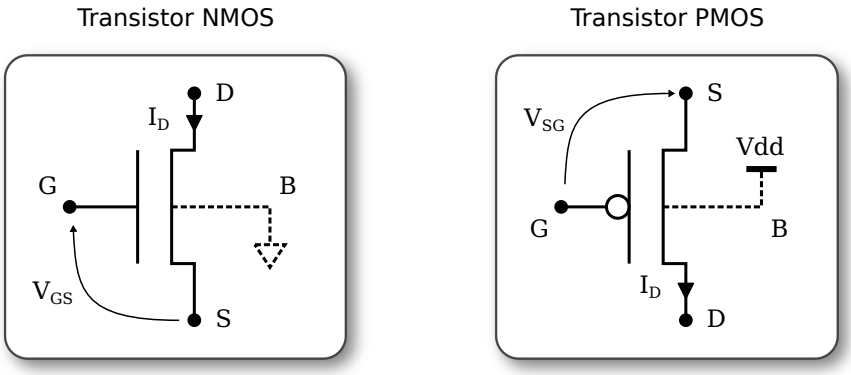


FIGURE 1.7 – Schéma électrique d'un transistor CMOS.

1.4.1.B Phénomène photoélectrique

Le phénomène photoélectrique est un mécanisme emblématique de la révolution quantique. Sa description par Albert Einstein en 1905 introduit le concept de photon en tant que particule élémentaire de la lumière pour décrire les échanges d'énergie entre un faisceau de lumière et les électrons d'un matériau conducteur. Ce phénomène s'applique également aux échanges d'énergie entre un faisceau laser et les électrons d'un matériau semi-conducteur.

Les niveaux d'énergie des électrons dans un matériau semi-conducteur sont distribués à l'intérieur de deux bandes d'énergie, à savoir la bande de valence et la bande de conduction. Ces bandes d'énergie sont séparées par une bande d'énergie interdite, ou *band gap*. Or, pour participer au phénomène de conduction, un électron doit préalablement passer de la bande de valence à la bande de conduction. Ces électrons sont alors dits libres. En l'absence d'excitation, un matériau semi-conducteur se comporte donc comme un isolant.

L'exposition d'un matériau semi-conducteur à un faisceau laser provoque un transfert d'énergie entre les photons du faisceau et le matériau. L'énergie E_{ph} d'un photon s'exprime en fonction de sa longueur d'onde λ, de la vitesse de la lumière c et de la constante de Planck h par l'équation 1.8 suivante :

$$E_{ph} = \frac{hc}{\lambda} \tag{1.8}$$

Si cette énergie est supérieure à l'énergie de *gap* du silicium (1.12 eV), ce photon peut être absorbé lors d'une interaction photoélectrique avec le matériau au cours de laquelle un électron passe de la bande de valence à la bande de conduction. Par conservation de la charge électrique du matériau, cette transition génère un piège à électrons, ou trou, assimilé à un porteur de charge positive dans la bande de valence. L'effet photoélectrique explique ainsi la création de paires électrons-trous dans un matériau semi-conducteur en silicium lorsque ce dernier est exposé à un faisceau laser de longueur d'onde adéquate. Ce processus est illustré sur la Figure 1.8.

En l'absence de champ électrique, les paires électrons-trous générées dans le substrat

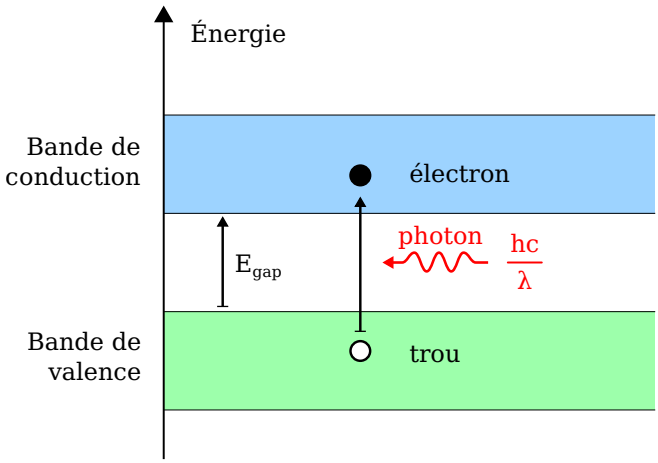


FIGURE 1.8 – Absorption d'un photon par effet photoélectrique.

s'annihilent naturellement lors d'un processus de recombinaison radiative. En revanche, les électrons et les trous générées au niveau des jonctions PN polarisée en inverse sont dissociées sous l'effet du champ électrique de la jonction. Un courant photoélectrique, ou photocourant, se forme alors [85]. Dans un circuit CMOS, ces zones correspondent généralement aux zone de charge d'espace (ZCE) situées à l'interface entre le drain et le substrat des transistors bloqués.

La Figure 1.9 illustre les trois phases de la formation d'un photocourant [25]. Dans la première phase (*laser shot*), le passage d'un faisceau laser dans un matériau semi-conducteur génère des paires électrons-trous dans son sillon. Dans la deuxième phase (*drift current*), ces paires sont dissociées et un fort courant de conduction apparaît au niveau des jonctions PN polarisées en inverse. Les électrons sont collectés au niveau des ZCE et les trous au niveau des plots de contact du substrat. Dans la troisième et dernière phase (*diffusion current*), la collection des électrons et des trous résulte d'un phénomène de diffusion des porteurs de charge dans le substrat. La forme générale du photocourant ainsi produit est présentée sur la Figure 1.10. Le photocourant atteint sa valeur maximale $I_{ph,peak}$ pendant la phase de collection de charges dominée par le phénomène de conduction.

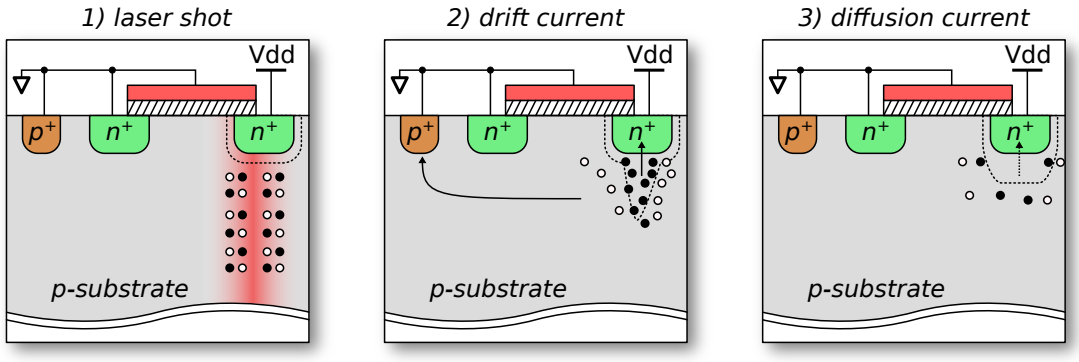


FIGURE 1.9 – Mécanisme de formation d'un photocourant.

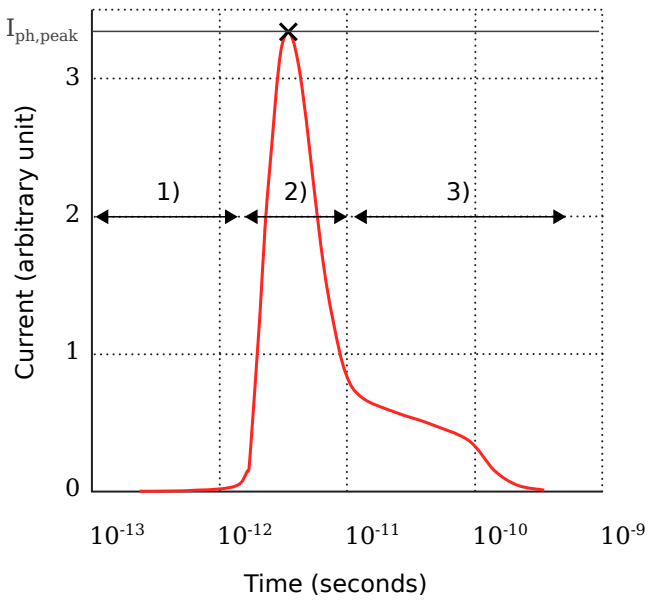


FIGURE 1.10 – Profil d'un courant photoélectrique créé par une impulsion courte [25].

1.4.1.C Modélisation électrique d'un photocourant

Pour des impulsions d'une durée supérieure à la nanoseconde, les effets d'une injection de fautes laser peuvent être modélisés par une source de courant entre le drain et le substrat d'un transistor dans la zone d'action du faisceau laser [60]. Le schéma électrique de ce modèle est présenté sur la Figure 1.11. Le réseau constitué de la capacité C_1 et des résistances R_2 et R_3 modélise les chemins de conduction des charges dans le substrat semi-conducteur. Nous négligerons son effet dans nos analyses qualitatives d'une injection de fautes laser.

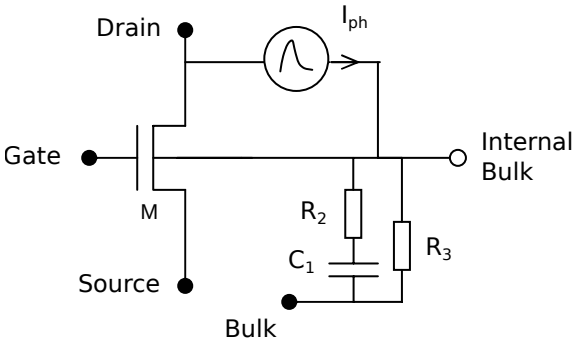


FIGURE 1.11 – Modèle électrique d'un transistor NMOS sous illumination laser [60].

En notant V_{DB} la tension appliquée sur la jonction drain-substrat du transistor et S la surface de la jonction, la valeur maximale du photocourant $I_{ph,peak}$ peut être estimée

par l'équation 1.9 suivante [148] :

$$I_{ph,peak} = (a \times V_{DB} + b) \times \alpha_{gauss}(x, y) \times S \tag{1.9}$$

Dans cette équation, les facteurs a et b sont des polynômes de la puissance de la source laser. La fonction α_{gauss}, quant-à elle, traduit le profil gaussien de l'intensité du faisceau laser dans le plan orthogonal à la direction de propagation du faisceau. Elle prend comme paramètres la position (x, y) de l'objectif laser dans le plan du circuit.

1.4.2 Modélisation des fautes sur la SRAM et les registres

Les données, le contexte et le résultat de l'exécution d'un programme sont mémorisés dans la SRAM et les registres d'un MCU. Ces mémoires représentent donc des cibles privilégiées pour un attaquant. Or, elles sont implémentées avec des circuits bistables dont l'architecture est vulnérable au mécanisme d'injection de fautes laser. Les paragraphes suivants illustrent ce mécanisme avec l'exemple d'une cellule SRAM standard et décrivent les exploitations qui en sont faites.

Un circuit bistable est un système à deux points d'équilibre stables qui codent respectivement pour l'état `'1'` et pour l'état `'0'`. Une stimulation électrique permet alors au système de basculer d'un état à l'autre. En revanche, en l'absence de stimulation, le circuit maintient son état d'équilibre. Chaque circuit bistable mémorise ainsi un bit d'information.

Une cellule SRAM standard implémentée avec six transistors est présentée sur la Figure 1.12. La cellule est composée de deux inverseurs ($M_1 - M_2$ et $M_3 - M_4$) connectés tête-bêche. Deux transistors de sélection (M_5 et M_6) activés par le signal « word » permettent alors d'accéder à l'état de la cellule « bit » et à son complémentaire « $\overline{\text{bit}}$ ».

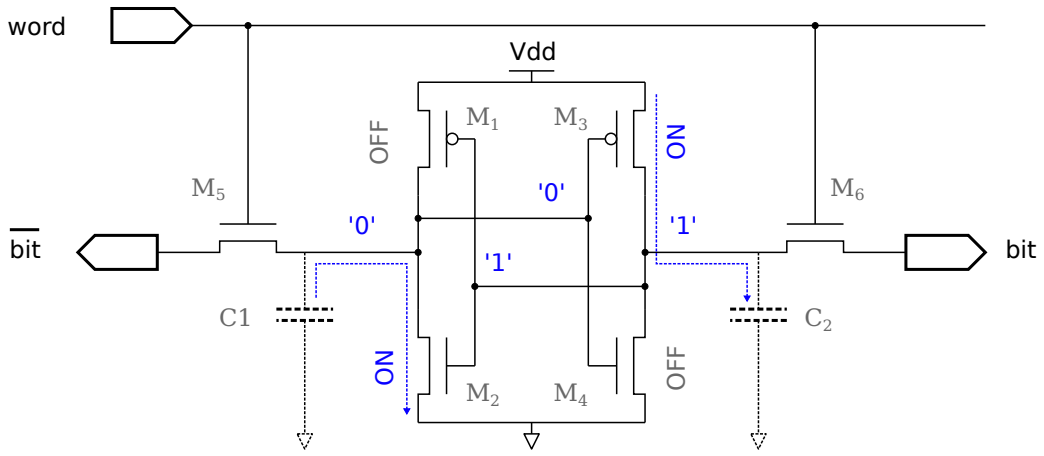


FIGURE 1.12 – Implémentation CMOS d'une cellule SRAM à six transistors [124]. Les chemins de conduction sont illustrés par des flèches bleues en pointillé.

Lorsque l'entrée de l'inverseur $M_1 - M_2$ est polarisée à la tension d'alimentation (état `'1'`), la capacité de sortie de l'inverseur est maintenue à la masse (état `'0'`) au travers

du transistor M_2. Parallèlement, lorsque l'entrée de l'inverseur $M_3 - M_4$ est polarisée à la masse (état `'0'`), la capacité de sortie de l'inverseur est maintenue à la tension d'alimentation (état `'1'`) au travers du transistor M_3. Par rétroaction, chacun des inverseurs agit donc sur l'entrée de son homologue pour maintenir l'état du système. Dans l'exemple illustré sur la Figure 1.12, l'état `'1'` est ainsi mémorisé.

Le mécanisme d'injection d'un photocourant par illumination laser décrit dans les paragraphes précédents s'applique aux jonctions drains-substrats des transistors bloqués de la cellule SRAM. Ces transistors correspondent aux transistors M_1 et M_4 sur le schéma de la Figure 1.12 lorsque la cellules mémorise l'état `'1'`. Ils correspondent aux transistors M_2 et M_3 lorsqu'elle mémorise l'état `'0'`.

Lorsque la cellule mémorise l'état `'1'`, l'injection d'un photocourant dans le transistor M_4 décharge la capacité C_2. Si ce photocourant dépasse un seuil critique, l'état de la cellule bascule et une faute de type *bit reset* est injectée. Ce cas de figure est illustré sur la Figure 1.13. Remarquons qu'une faute similaire peut être injectée en illuminant le transistor M_1 dont le substrat est connecté à la tension d'alimentation. L'injection d'un photocourant sur le transistor M_1 provoque en effet une faute de type *bit reset* en chargeant la capacité C_1. Par analogie, l'injection d'un photocourant sur le transistor M_2 ou le transistor M_4 permet d'injecter une faute de type *bit set* lorsque la cellule mémorise l'état `'0'`.

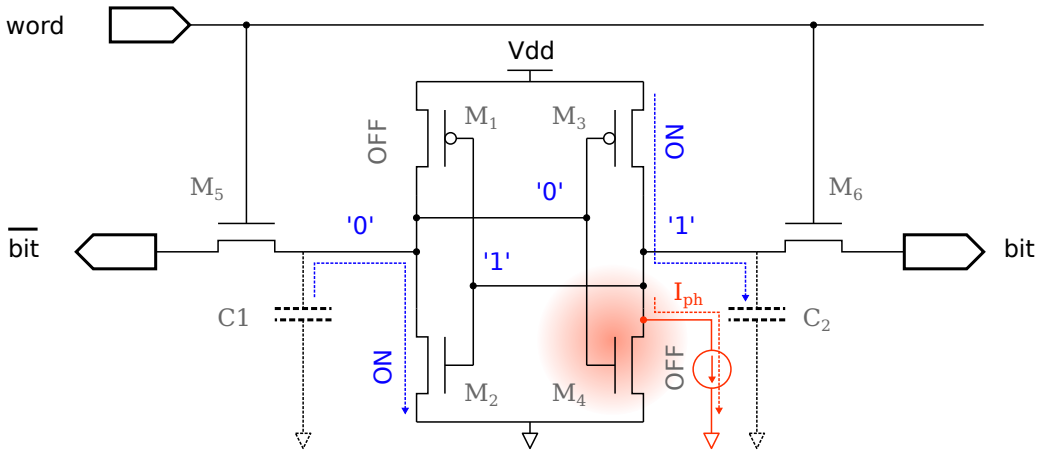


FIGURE 1.13 – Injection d'une faute de type *bit reset* par illumination laser d'une cellule SRAM. La zone d'effet du spot laser est colorée en rouge.

L'illumination laser d'un circuit bistable se traduit ainsi par une faute de type *bit reset* ou de type *bit set*, en fonction de la position de l'objectif laser [149]. À l'état de l'art, la possibilité d'injecter une faute de type *bit reset* ou de type *bit set* sur un seul bit dans un circuit intégré a été validée expérimentalement sur les bascules d'un ASIC fabriqué au nœud technologique CMOS 28 nm [66].

Une injection de fautes laser permet ainsi de modifier très précisément les variables, les constantes et les instructions d'un algorithme mémorisé dans la SRAM d'un MCU afin d'extraire une clé de chiffrement [3], [70], [181] ou de modifier l'exécution d'un pro-

gramme [33], [70]. Or, les fautes injectées dans la SRAM persistent jusqu'à la réécriture de cette dernière, ou jusqu'à une remise à zéro du circuit. Par conséquent, l'injection d'une faute par illumination laser sur la SRAM peut être soit synchronisée avec l'exécution d'un algorithme [3], [70], soit réalisée avant l'exécution de l'algorithme [33], [181].

Une injection de fautes laser sur les registres généraux d'un MCU permet enfin de modifier l'état ou le flot de contrôle d'un algorithme [35], [90], [169]. En utilisant cette technique, VASSELLE, THIEBEAULD, MAOUHOUB et al. [173] ont par exemple désactivé avec succès le mécanisme d'amorçage sécurisé d'un SoC fabriqué dans la technologie CMOS 32 nm.

1.4.3 Modélisation des fautes sur les bus et la logique

Un inverseur CMOS est un circuit élémentaire qui inverse, à sa sortie, l'état de son entrée. Il est composé de deux transistors, à savoir un transistor PMOS et un transistor NMOS, qui fonctionnent de manière complémentaire pour charger ou décharger la capacité de sortie du circuit. Les circuits inverseurs trouvent ainsi de nombreuses applications dans la conception de circuits logiques, de tampons (*buffers*) et de sources de courant pilotées en tension (*drivers*). Le schéma électrique d'un bus de données est présenté à titre d'exemple sur la Figure 1.14.

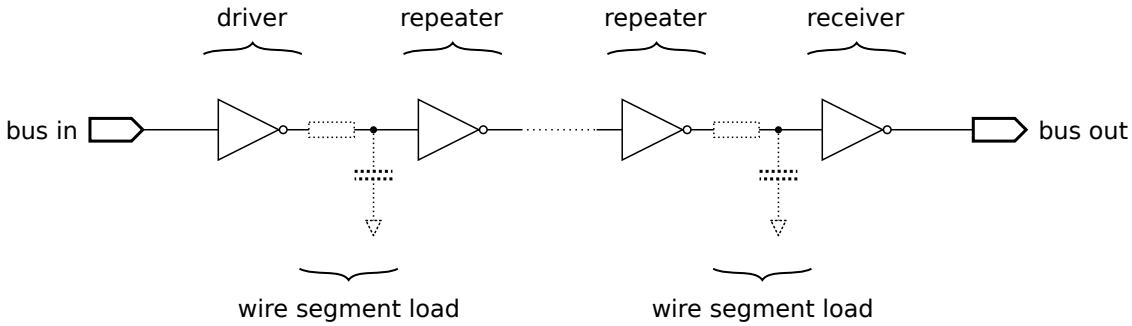
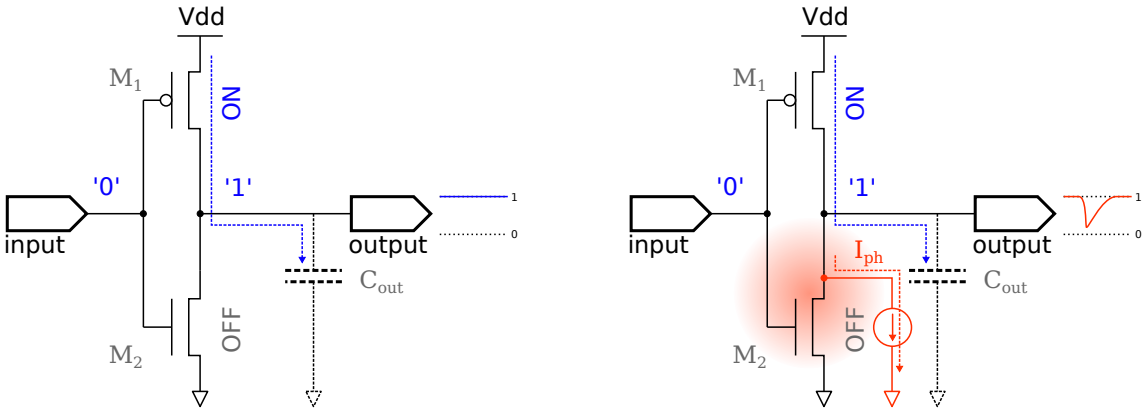


FIGURE 1.14 – Schéma électrique d'un bus de données [124].

Le fonctionnement d'un inverseur CMOS en conditions nominales et sous illumination laser est illustré sur la Figure 1.15. L'inverseur est constitué de deux transistors, M_1 et M_2, connectés par leur grille à l'entrée du circuit et par leur drain à sa sortie. Lorsque l'entrée du circuit est polarisée à la masse (état `'0'`), le transistor M_2 se bloque et la capacité de sortie C_{out} du circuit se charge au travers du transistor M_1 (Figure 1.15a). Le potentiel de la sortie du circuit augmente alors jusqu'à atteindre la tension d'alimentation (état `'1'`).

Dans cette configuration, un photocourant transitoire peut être injecté par illumination laser à l'interface entre le drain et le substrat du transistor M_2. Le cas échéant, la capacité de sortie C_{out} se décharge temporairement dans le substrat du transistor NMOS avant d'être chargée à nouveau par le transistor PMOS (Figure 1.15b). Ce phénomène se traduit donc par l'apparition d'un *glitch* de tension sur la sortie de l'inverseur. Une



(a) Fonctionnement en conditions nominales.

(b) Fonctionnement sous illumination laser.

FIGURE 1.15 – Injection d'un *glitch* de tension par illumination laser sur un inverseur CMOS. La zone d'effet du spot laser est colorée en rouge.

impulsion laser permet ainsi d'injecter une erreur de calcul transitoire dans les portes logiques qui composent un circuit combinatoire, ou de modifier temporairement l'état d'un bus de données.

Une erreur de calcul transitoire se transforme en une faute de mémorisation si elle est échantillonnée par un registre lors d'un front d'horloge. L'illumination laser d'un circuit logique provoque alors une violation des contraintes temporelles du circuit (voir section 1.3.1). Or, la largeur d'un *glitch* de tension augmente avec la durée de l'impulsion laser à l'origine de ce *glitch*. Par conséquent, une augmentation progressive de la durée de l'impulsion laser induit une violation progressive des contraintes temporelles du circuit qui se traduit par un changement de la nature des fautes, d'aléatoire à déterministe. Ce mécanisme d'injection de fautes a été exploité par SCHELLENBERG, FINKELDEY, GERHARDT et al. [152] pour réaliser une FSA [103] sur une implémentation matérielle d'un algorithme de chiffrement.

Les fautes déterministes injectées par illumination laser sur la logique d'un MCU ou sur le transfert des instructions et des données peuvent également être exploitées au niveau logiciel. Des corruptions et des sauts d'instruction répétables sont ainsi identifiés en testant systématiquement les effets d'une impulsion laser sur des programmes de tests lors d'un balayage spatial des positions de l'objectif laser. Ces modèles de fautes permettent de réaliser une attaque Bellcore [169] ou une DFA [84], de modifier le flot de contrôle d'un programme [35], ou encore d'inhiber une protection contre les attaques en faute [89], [169]. Ces attaques ont des conséquences immédiates sur la sécurité des clés de chiffrement mémorisées dans le circuit.

De surcroît, ces attaques peuvent être combinées si un attaquant est capable d'injecter plusieurs fautes sur un même algorithme. TRICHINA et KORKIKYAN [169] ont ainsi réalisé avec succès une attaque Bellcore sur un algorithme RSA protégé contre les attaques en faute en injectant deux sauts d'instruction à deux instants différents pendant l'exécution

de l'algorithme sur un MCU 32 bits. La localisation des fautes injectées par la face avant du microcontrôleur, à proximité de la mémoire Flash, a amené les auteurs à conclure que les fautes étaient probablement injectées sur un bus qui connecte la mémoire Flash et le cœur de calcul du MCU.

1.4.4 Modélisation des fautes sur la mémoire Flash embarquée

Les MCU embarquent des mémoires non-volatiles pour mémoriser les instructions et les données d'un programme en l'absence d'alimentation électrique. Ces mémoires sont donc des cibles privilégiées pour corrompre un algorithme de sécurité ou extraire des informations confidentielles par une attaque en faute. Les paragraphes suivants présentent, dans un premier temps, le fonctionnement des mémoires NOR Flash embarquées, utilisées dans la plupart des MCU et détaillent, dans un second temps, les modèles de fautes obtenus par LFI sur les différents composants de ces mémoires.

1.4.4.A Présentation de la technologie NOR Flash

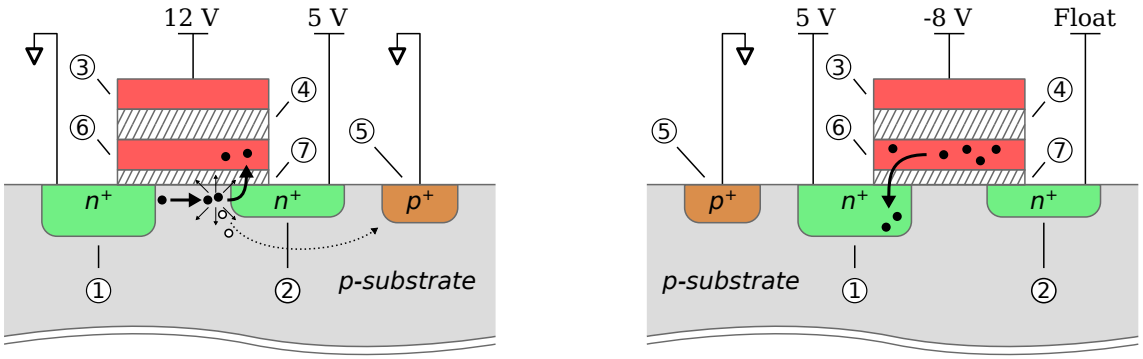
Les mémoires non-volatiles dédiées à la mémorisation des programmes et des données implémentent généralement l'architecture NOR Flash. Cette architecture, introduite en 1984 par MASUOKA, ASANO, IWAHASHI et al. [109] reprend le principe de la technologie EEPROM – qui permet d'effacer et de reprogrammer simplement la mémoire de manière électrique après la distribution du circuit – en améliorant ses performances et sa densité [1]. Elle permet notamment aux instructions et aux données d'un programme d'être chargées mot par mot de la mémoire Flash aux registres d'un cœur de calcul avec un faible temps de latence. En revanche, la programmation de la mémoire est une opération lente. Elle est réalisée avec une granularité d'un bloc de mots, ou page, dont la taille varie en fonction des architectures. La technologie EEPROM est donc préférée pour mémoriser, lire et modifier des paramètres de configuration, des clés de chiffrement ou des variables de sécurité avec une granularité d'un mot ou d'un octet.

Le principe de fonctionnement des deux technologies est cependant identique : un bit d'information est enregistré, de manière analogique, par de porteurs de charge, en général des électrons, maintenus captifs dans une structure de transistor (). Ces structures, inventées au sein des laboratoires Bell à la fin des années 1960, sont largement inspirées des transistors MOS à effet de champ (*Metal Oxide Semiconductor Field Effect Transitor* – MOSFET), inventés au sein du même laboratoire par Mohammed Attala et Dawon Kahng quelques années plus tôt [88].

Le transistor à grille flottante est constitué de deux zones de diffusion implantées dans un substrat semi-conducteur et d'une grille de contrôle qui surmonte le dispositif. Cependant, à la différence des transistors CMOS, une grille flottante en polysilicium est insérée entre la grille de contrôle et le matériau semi-conducteur des transistors à grille flottante. Elle est isolée électriquement du dispositif par un oxyde de grille du côté de la grille de contrôle et par un oxyde tunnel du côté du matériau semi-conducteur. Une vue en coupe d'un dispositif à grilles empilées – ou *stacked gate* – est présentée sur la

1. Une mémoire Flash se caractérise notamment par une structure de cellule mémoire à un transistor, au lieu de deux pour une EEPROM [109].

Figure 1.16. Il y figure la source **(1)**, le drain **(2)**, la grille **(3)**, l'oxyde de grille **(4)**, la grille flottante **(6)** et l'oxyde tunnel **(7)** d'un transistor, ainsi que les contacts de polarisation du substrat P **(5)**. Les électrons sont transférés dans et hors de la grille flottante grâce à deux mécanismes distincts détaillés dans le paragraphe suivant.



(a) Programmation (CHEI).

(b) Effacement (Fowler-Nordheim).

FIGURE 1.16 – Vue en coupe d'un transistor à grille flottante. Les mécanismes de programmation et d'effacement du dispositif sont illustrés respectivement sur la figure de gauche et la figure de droite.

1.4.4.B Écriture d'un transistor à grille flottante

La programmation d'un transistor à grille flottante utilise généralement le mécanisme CHEI (*Channel Hot-Electron Injection*). Ce mécanisme explique l'apparition d'un transfert d'électrons entre le drain et la grille flottante d'un transistor à grille flottante sous l'effet d'un phénomène d'ionisation (schématisé sur la Figure 1.16a). Ce phénomène résulte de collisions entre les électrons du canal de conduction, accélérés sous l'effet d'un fort champ électrique et les atomes du matériau semi-conducteur au voisinage du drain d'un transistor à grille flottante. La majorité des porteurs de charge générés par ces collisions sont ensuite collectés, soit par le drain, soit par le substrat. Cependant, certains électrons très énergétiques franchissent l'oxyde tunnel et chargent alors la grille flottante du transistor. Du fait des conventions de conception matérielle, ce transistor – dit programmé – enregistre ainsi la valeur `'0'` [40].

L'effacement d'un transistor à grille flottante utilise généralement le mécanisme Fowler-Nordheim (FN). Ce mécanisme explique l'apparition d'un courant très faible par effet tunnel (schématisé sur la Figure 1.16b) entre la grille flottante et la source d'un transistor à grille flottante lorsque la grille de contrôle et la source du transistor sont soumises à une forte différence de potentiels. Ce courant permet alors de décharger simultanément les grilles flottantes de nombreux transistors. Du fait des conventions de conception matérielle, ces transistors – dits effacés – enregistrent ainsi la valeur `'1'` [40].

1.4.4.C Lecture d'un transistor à grille flottante

Pour lire un transistor à grille flottante, son drain doit être polarisé à une tension V_{DS} autour de 1 V et sa grille à une tension V_{GS} autour de 5 V. Sa source, quant à elle, doit être connectée à la masse.

Deux régimes de fonctionnement peuvent alors être distingués en fonction de la valeur de la tension de seuil V_T d'un transistor à grille flottante :

— le régime bloqué définit par l'équation : $V_{GS} \geq V_T$;

— le régime linéaire, ou régime triode, définit par l'équation : $V_{DS} \leq \alpha_G(V_{GS} - V_T + fV_{DS})$.

où α_G et f sont deux facteurs adimensionnels qui s'expriment en fonction des capacités introduites par la grille flottante dans le dispositif [40]. L'équation du régime triode peut être reformulée pour isoler la tension de drain du transistor :

$$\left(\frac{1}{\alpha_G} - f\right) V_{DS} \leq V_{GS} - V_T \tag{1.10}$$

Au facteur $\frac{1}{\alpha_G} - f$ près, l'équation 1.10 correspond ainsi à l'équation du régime triode d'un transistor MOSFET.

Le courant de drain I_D d'un transistor à grille flottante en régime triode vérifie l'équation [40] :

$$I_D = \beta\left((V_{GS} - V_T)V_{DS} - \left(f - \frac{1}{2\alpha_G}\right)V_{DS}^2\right) \tag{1.11}$$

où la quantité β est un facteur des paramètres géométriques et physiques du transistor et la quantité V_T désigne la tension de seuil du transistor.

La charge Q stockée sur la grille flottante d'un transistor programmé augmente la tension de seuil du dispositif vu de la grille de contrôle. L'augmentation ΔV_T de la tension de seuil est décrite par la relation suivante :

$$\Delta V_T = -\frac{Q}{C_{FG}} \tag{1.12}$$

où C_{FG} désigne la capacité entre la grille flottante et la grille de contrôle. Ainsi, pour une même tension de lecture V_{GS}, le courant de drain d'un transistor effacé est supérieur à celui d'un transistor programmé.

Cette propriété est illustrée par la Figure 1.17 qui présente les caractéristiques $I_D - V_{GS}$ d'un transistor à grille flottante programmé et d'un transistor à grille flottante effacé. Un amplificateur de détection (*Sense Amplifier* – SA) permet de détecter l'état binaire mémorisé par un transistor à grille flottante, à savoir `'0'` ou `'1'`, en comparant le courant de drain du transistor à un courant de référence (en pointillé noir sur la Figure 1.17).

1.4.4.D Organisation d'une mémoire Flash

Les mémoires NOR Flash sont des circuits complexes qui permettent de lire, de programmer et d'effacer des transistors à grille flottante de manière autonome. Ces mémoires sont composées :

1. de circuits analogiques permettant de générer les tensions et les courants de référence utilisés pour opérer le circuit, ainsi que les tensions nécessaires pour programmer les transistors à grille flottante ;

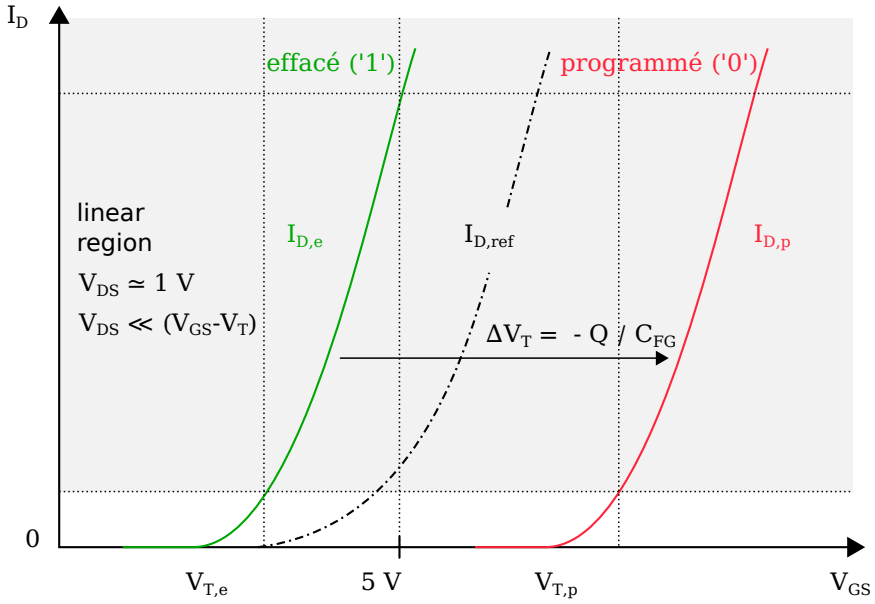


FIGURE 1.17 – Courant de drain $I_{D,e}$ et $I_{D,p}$ d'un transistor à grille flottante effacé et programmé respectivement, en fonction de la tension de lecture V_{GS} appliquée sur la grille de contrôle du transistor. La position des courants par rapport à un courant de référence $I_{D,ref}$ permet de déterminer l'état binaire mémorisé par le transistor.

2. d'un décodeur de lignes (*row decoder*) et d'un décodeur de colonnes (*column decoder*) permettant d'adresser un ensemble de transistors à grille flottante dans un tableau de cellules mémoire ;
3. de plusieurs SA permettant de traduire la charge captive d'un transistor à grille flottante en un bit d'information ;
4. d'un circuit de contrôle logique qui orchestre les opérations de lecture et d'écriture dans la mémoire, par exemple grâce à une machine d'états ;
5. et de mémoires tampon (*buffers*), à la jonction entre le bloc de contrôle de la mémoire et l'interface du bus de données, qui mémorisent le données lues ou écrites en mémoire [124].

La Figure 1.18 illustre l'agencement de ces composants au sein d'une mémoire Flash embarquée.

1.4.4.E Fautes sur la logique de contrôle

Les travaux de SKOROBOGATOV en 2010 ont montré qu'une LFI sur la logique de contrôle d'une mémoire Flash permet de mettre à zéro les données transférées hors d'une mémoire NOR Flash [161] ou d'empêcher sélectivement l'écriture d'un mot dans la mémoire [162]. Ces expériences sont réalisées par la face arrière de plusieurs MCU fabriqués dans les technologies CMOS 1.2 µm, 0.9 µm et 0.5 µm. Dans les deux cas, les phénomènes sont parfaitement répétables pour toutes les adresses testées une fois que des paramètres d'injection appropriés sont trouvés. De surcroît, ces phénomènes persistent tant que la source laser est activée. Un bloc de données de taille arbitraire peut ainsi être mis à zéro en ajustant précisément l'instant d'injection et la durée de

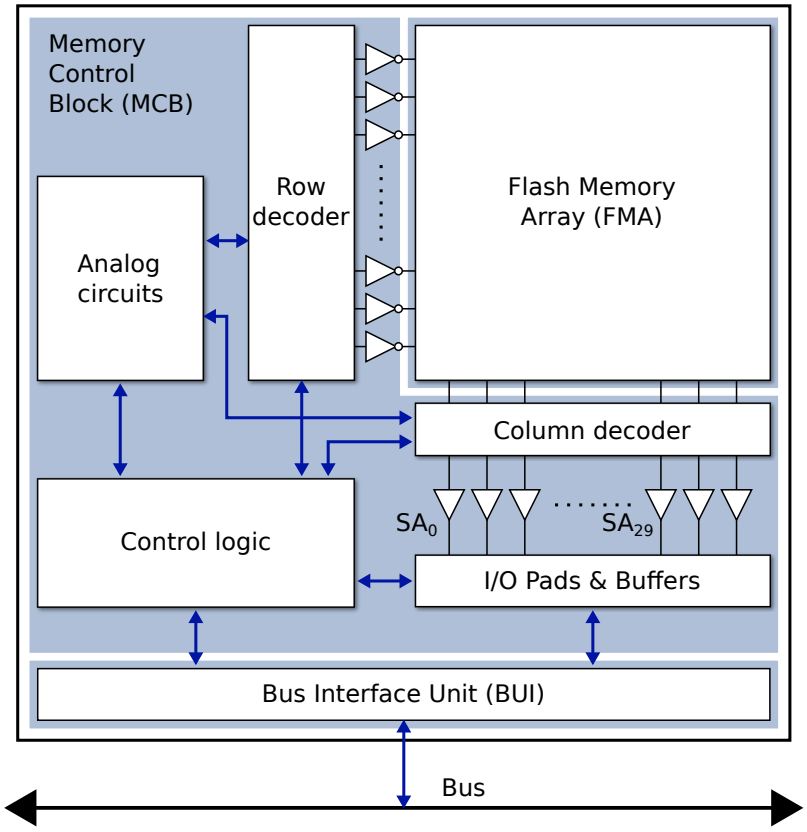


FIGURE 1.18 – Organisation générale d'une mémoire Flash embarquée.

l'impulsion laser.

Les fautes injectées sur les données lues en mémoire peuvent aussi être limitées à quelques bits si la source laser est activée à un instant précis lors de l'échantillonnage d'une donnée [161]. Par ailleurs, l'illumination du *driver* d'une ligne de mots dans le décodeur de lignes de la mémoire permet d'empêcher la sélection d'une ligne dans le tableau de mémoire et de bloquer ainsi la modification d'un bit lors d'une opération d'écriture [162]. L'auteur remarque enfin qu'un tir laser sur un transistor à grille flottante permet d'empêcher le transistor d'être programmé. En revanche, les mêmes paramètres d'injection n'ont pas d'effet sur l'effacement du transistor [162].

La mise à zéro d'une instruction de 16 bits se traduit par un saut d'instruction sur les architectures 8 bits AVR et les architectures 32 bits Arm qui supportent le jeu d'instructions Thumb (voir section 1.2.2.B). Les résultats rapportés dans [161] suggèrent ainsi la possibilité d'empêcher l'exécution de sections entières d'un programme avec le même mode opératoire que celui utilisé pour mettre à zéro les données lues en mémoire Flash. Cette hypothèse est confirmée par des études récentes sur l'injection de sauts d'instruction grâce à une LFI.

Dans [36], les auteurs ont analysé l'influence des paramètres expérimentaux sur l'injection de sauts d'instruction sur un MCU 8 bits AVR. Des sauts d'instruction parfaitement répétables ont été obtenus en synchronisant précisément l'injection laser avec l'exécution des programmes ciblés. En ajustant la durée de l'impulsion, les auteurs ont

démontré en pratique qu'un saut d'instructions peut affecter toutes les instructions d'une fonction. Un saut de la fonction d'addition des clés de ronde d'un AES à la fin de la dernière ronde permet alors de monter une analyse différentielle triviale pour extraire la clé de chiffrement de l'algorithme [36].

Ces résultats sont étendus par [69] à un bloc arbitraire d'instructions. À l'instar des résultats obtenus dans [161], la position et la taille du bloc d'instructions non-exécutées peuvent être choisies en ajustant l'instant d'injection et la durée de l'impulsion laser. Les auteurs ont montré, par ailleurs, que plusieurs impulsions rapprochées parfaitement synchronisées pouvaient être produites en modulant l'intensité d'un faisceau laser grâce à un signal de synchronisation généré par une carte externe. Ce dispositif a été utilisé pour corrompre plusieurs tests conditionnels répétés à intervalles réguliers lors de la vérification d'un code PIN en temps constant.

1.4.4.F Fautes sur les amplificateurs de détection

Une LFI sur l'interface de lecture d'une mémoire Flash permet à un attaquant de corrompre un bit à l'intérieur d'une instruction avec une grande précision. SKOROBOGATOV [159] a montré en effet sur deux MCU 8 bits fabriqués dans les technologies CMOS 0.9 µm et 0.35 µm que l'injection d'un photocourant dans le décodeur de colonnes de la mémoire EEPROM ou dans les entrées d'un SA permet de forcer le résultat de la comparaison, donc un bit d'information, à `'0'`.

Ce mécanisme a été analysé de manière systématique par SAKAMOTO, FUJIMOTO et MATSUMOTO [146] sur la mémoire Flash embarquée d'un MCU 32 bits fabriqué au nœud technologique CMOS 130 nm ou inférieur. Pour chacun des amplificateurs de détection de la mémoire, les auteurs ont réussi à injecter un photocourant soit sur l'entrée positive de l'amplificateur, soit sur l'entrée négative. Ce phénomène se traduit par l'existence de deux zones différentes associées à l'injection de fautes de type *bit set* et de type *bit reset* respectivement pour chacun des bits d'un mot de 32 bits lu en mémoire Flash. Un attaquant peut ainsi modifier un bit d'indice choisi à l'intérieur d'une ou plusieurs instructions consécutives en ajustant, d'une part, la position de l'objectif laser sur l'amplificateur de détection de son choix et en synchronisant, d'autre part, l'instant d'injection et la durée de l'impulsion laser sur la lecture des instructions ciblées [146].

1.4.4.G Fautes sur les références de courant

Si ces résultats désignent l'interface de lecture d'une mémoire Flash comme une cible privilégiée pour les LFI, des travaux récents sur un MCU 8 bits fabriqué dans la technologie CMOS 0.35 µm ont cependant démontré que des fautes de type *bit reset* peuvent également être obtenues en positionnant l'objectif laser à l'intérieur du tableau de cellules mémoire [100]. Pour chaque bit d'un mot lu en mémoire Flash, une cartographie des positions de l'objectif pour lesquelles une faute de type *bit reset* est injectée fait apparaître une zone de sensibilité rectangulaire qui suit l'organisation en colonnes de la mémoire. Les auteurs attribuent ces observations à l'injection d'une faute sur les références de courant de la mémoire.

1.4.4.H Fautes sur les transistors à grille flottante

Une LFI permet enfin à un attaquant d'effacer plusieurs transistors à grille flottante. Deux mécanismes expliquent ce phénomène. Le premier relie l'augmentation de la température des transistors à grille flottante avec la diminution de la rétention des charges maintenues captives dans ces transistors. Une augmentation locale de la température des transistors à grille flottante grâce à une impulsion laser de plusieurs secondes permet ainsi d'effacer les transistors au voisinage du spot laser [160]. Le second décrit un transfert de charges entre la grille flottante et le substrat causé par un phénomène d'ionisation à deux photons au niveau de la grille flottante et du matériau semi-conducteur grâce à une impulsion laser femtoseconde [43]. Cependant, du fait de la densité des mémoires NOR Flash fabriquées dans des technologies submicrométriques, la précision des LFI ne permet pas de modifier un unique bit.

1.5 Conclusion

1.5.1 Objectif de ces travaux

L'objectif de ces travaux est d'évaluer la possibilité d'utiliser les attaques en faute comme point d'entrée pour compromettre la sécurité d'un objet connecté. Pour ce faire, ces travaux proposent une analyse approfondie des modèles de fautes matériels et logiciels obtenus grâce à deux techniques d'injection de fautes locales, à savoir les LFI et les EMFI. Cette analyse est réalisée « en boîte grise » sur des MCU non-sécurisés, malgré les limitations de leur instrumentation et l'absence de documents publiques sur leur conception. Ces travaux doivent permettre d'identifier les propriétés de modèles de fautes originaux et d'évaluer, sur cette base, différents scénarios d'exploitations possibles.

1.5.2 Contributions

Ces travaux ont fait l'objet de plusieurs publications internationales dans des revues avec comité de relecture. Ces publications rassemblent notamment :

— une analyse détaillée d'un modèle de faute original *bit set* mono-bit par injection laser sur la mémoire Flash d'un microcontrôleur 32 bits fabriqué dans la technologie CMOS 80 nm [51] ;

— une analyse détaillée d'un mécanisme d'injection de fautes original par injection d'un photocourant sur un transistor à grille flottante et une confirmation de ce mécanisme sur deux architectures différentes de mémoire NOR Flash, fabriquées dans les technologies CMOS 65 nm et 80 nm [113] ;

— une analyse détaillée d'un modèle de faute original *bit set* et *bit reset* local, contrôlable et répétable d'une injection de fautes électromagnétique sur les transferts de données entre la mémoire Flash et le *buffer* de préchargement des données de la mémoire [111] ;

— une étude des paramètres d'injection permettant d'obtenir un saut sur une à six instructions consécutives avec une forte répétabilité grâce à une EMP [112].

Ces travaux sont détaillés dans les chapitres suivants.

Chapitre 2

Modèles de fautes d'une impulsion EM sur une mémoire NOR Flash

Dans ce chapitre, nous analysons, tout d'abord, les modèles de fautes matériels obtenus sur la mémoire tampon, ou *buffer*, de préchargement des données d'un MCU 32 bits Arm Cortex-M3. Nous étudions, en particulier, les fautes de type *bit set* et de type *bit reset* qui peuvent être utilisées pour modifier les instructions et les données manipulées par un MCU avec une grande simplicité. Nous analysons ensuite un cas particulier de corruptions d'instructions dans le *buffer* de préchargement des instructions d'un MCU 8 bits AVR. Ce cas se traduit par des sauts d'une ou plusieurs instructions. Nous étudions les propriétés et les limites de ces sauts d'instructions au regard des paramètres d'une EMFI.

2.1 Motivations

Des travaux précédents ont décrit les effets d'une EMP sur les mémoires tampon, ou *buffers*, et les mémoires cache dédiées au préchargement des instructions et des données d'un programme initialement mémorisé dans la mémoire Flash d'un MCU [27], [104], [120], [143]. En particulier, deux de ces travaux décrivent l'injection de fautes de type *bit set* sur le transfert des données entre la mémoire Flash et le cœur de calcul d'un MCU 32 bits [120] et d'un MCU 8 bits [27]. Cependant, ces travaux ne proposent qu'une analyse partielle des corruptions de données au regard des paramètres d'une injection de fautes.

Ce chapitre a ainsi pour objectif de répondre aux questions suivantes :

— Quels sont les modèles de fautes induits par une EMP pendant les transferts d'instruction et de données entre la mémoire Flash et le cœur de calcul d'un MCU ?

— Quelle est la précision et la répétabilité de ces modèles de fautes ?

— Peut-on identifier les composants matériels à l'origine des fautes observées ?

— Quelle est la maîtrise d'un attaquant sur les modèles de fautes obtenus ?

Pour ce faire, nous identifions, en premier lieu, plusieurs modèles de fautes matériels sur une mémoire de préchargement des données d'un MCU 32 bits, dont nous analysons les caractéristiques spatiales et temporelles ainsi que la répétabilité. En second lieu, nous étudions les caractéristiques principales d'un modèle de faute logiciel de saut d'instruction sur un MCU 8 bits. Si l'analyse proposée sur la mémoire de préchargement des données n'est pas directement transposable à celle des fautes injectées dans une mémoire de préchargement des instructions, les mécanismes étudiés permettent néanmoins d'améliorer la maîtrise expérimentale des fautes injectées.

2.2 Dispositifs et méthodologie

Les travaux décrits dans ce chapitre ont nécessité de nombreux essais expérimentaux sur plusieurs circuits intégrés avec un dispositif d'EMFI. Cette section décrit, d'une part, l'ensemble des dispositifs utilisés et, d'autre part, les méthodes mises en œuvre pour automatiser l'exploration des paramètres d'injection.

2.2.1 Dispositif d'injection de fautes EM

Les paragraphes qui suivent décrivent l'architecture générale du banc d'injection de fautes par perturbation électromagnétique utilisé dans ces travaux, ainsi que les paramètres expérimentaux qui y sont associés. Une vue d'ensemble du dispositif est proposée sur la Figure 2.1.

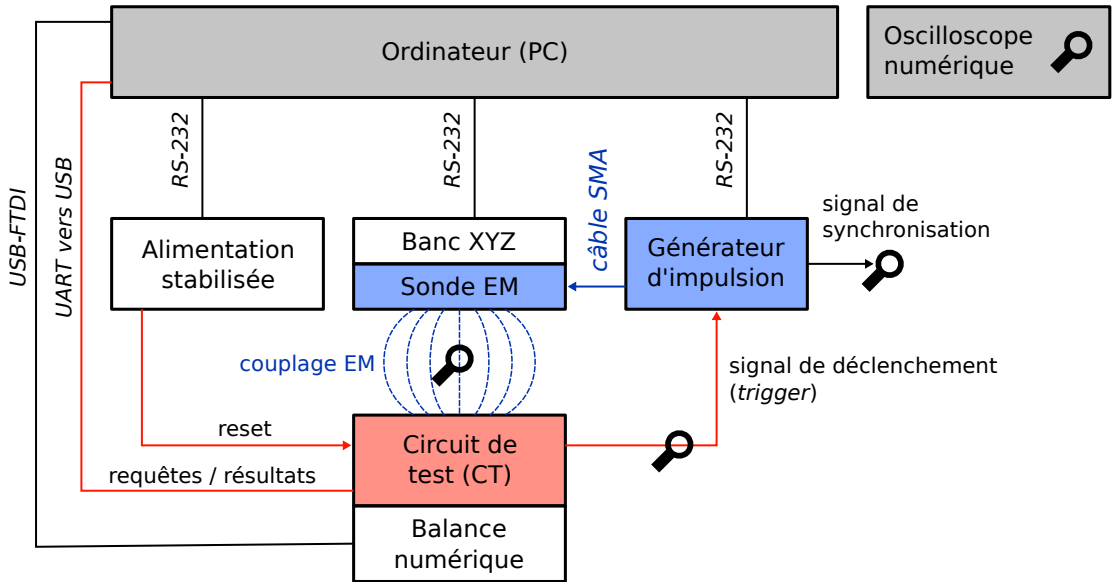


FIGURE 2.1 – Vue schématique de notre banc d'injection EM.

2.2.1.A Description du banc d'injection EM

Le générateur d'impulsions de tension est capable de générer une impulsion de tension d'une amplitude de plusieurs centaines de volts avec une grande précision. Les caractéristiques des deux générateurs utilisés dans nos expériences sont résumées dans

Modèle	AVL-5-B-TR (400 V) [19]	AVIR-4-B (200 V) [18]
Impédance de sortie	50 Ω	50 Ω
Amplitude maximale	400 V	200 V
Polarité	pos. ou nég.	pos. ou nég.
Largeur d'impulsion	8 ns – 100 ns	4 ns – 200 ns
Délai programmable	0 s – 1 s	0 s – 1 s
Temps de monté	≤ 2 ns	≤ 2 ns
Temps de descente	≤ 5 ns	≤ 2.5 ns
Temps de propagation	≤ 350 ns	≤ 100 ns
Gigue de déclenchement	≤ 100 ps	≤ 100 ps

TABLE 2.1 – Caractéristiques des générateurs d'impulsions de tension.

la Table 2.1. Les paramètres d'une impulsion – à savoir son amplitude, sa largeur et la temporisation insérée avant la génération d'une impulsion – peuvent être modifiés grâce à l'interface de programmation de l'équipement. Ce dernier possède en outre un mode de déclenchement externe, ou *Trig In*, qui est utilisé dans nos expériences pour synchroniser la génération d'une perturbation avec le fonctionnement des circuits de tests. Dans ce mode de déclenchement, un signal externe, appelé signal de déclenchement, ou *trigger*, est fourni au générateur d'impulsions. Le basculement de ce signal à l'état haut provoque la génération d'un signal intermédiaire, appelé signal de synchronisation, et d'une impulsion de tension dont les caractéristiques sont illustrées sur la Figure 2.2. La sortie du générateur d'impulsions de tension est reliée à une sonde d'injection EM par l'intermédiaire d'un câble coaxial.

La sonde d'injection EM est composée d'un fil de cuivre verni de section 0.03 mm^2 enroulé sous forme de solénoïde autour d'un noyau cylindrique en ferrite 78 d'un diamètre de 500 µm. Nous avons fabriqué deux sondes d'injection EM : la première, notée 3N/∅500, est composée d'un solénoïde à trois tours et la seconde, notée 6N/∅500, d'un solénoïde à six tours. Les deux extrémités du fil de cuivre sont reliées, d'un côté, à l'âme d'un connecteur SMA et, de l'autre, à sa masse, comme illustré sur la Figure 2.3. L'excitation de la sonde d'injection avec une impulsion de tension produit une forte variation du champ magnétique au voisinage du solénoïde [57].

Le banc XYZ est une plateforme de positionnement micrométrique à trois degrés de liberté. La sonde d'injection, solidaire de la plateforme, est positionnée au-dessus du circuit de tests dans un repère à trois dimensions X, Y et Z.

La balance numérique de précision est placée sous le circuit de tests pour détecter les pressions exercées par la sonde d'injection sur la surface de ce dernier. Elle permet ainsi de positionner la sonde d'injection au plus proche de la surface du circuit, sans pour autant rentrer en contact avec lui, afin de maximiser l'effet d'une perturbation électromagnétique [71].

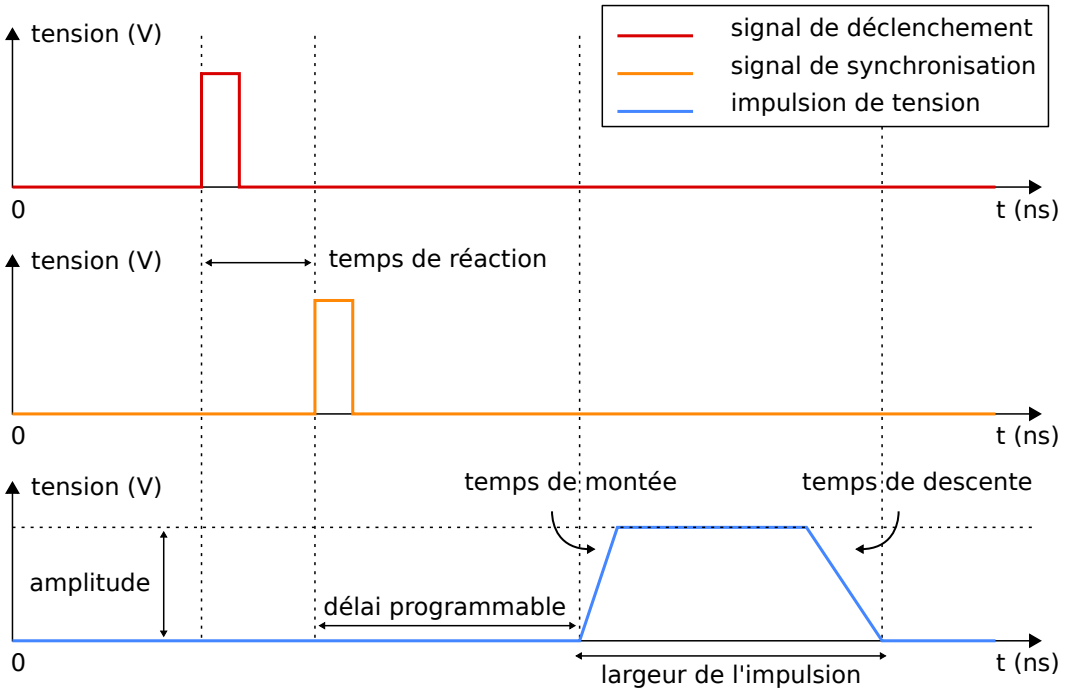


FIGURE 2.2 – Caractéristiques d'une impulsion de tension de polarité positive.

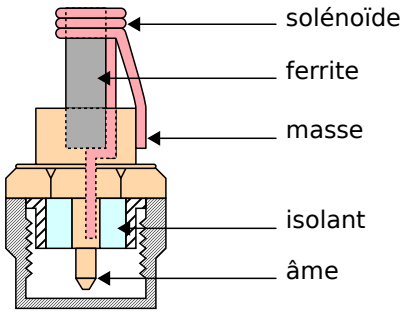


FIGURE 2.3 – Sonde d'injection EM fabriquée à partir d'un connecteur SMA.

L'alimentation stabilisée est utilisée pour réaliser une remise à zéro du circuit, ou *reset*, dans l'éventualité où une anomalie est détectée dans le fonctionnement du circuit après l'injection d'une faute.

L'oscilloscope numérique permet de contrôler le déroulement d'une injection de fautes. Il mesure notamment :

— le niveau du signal de *trigger* produit par le circuit de tests à destination du générateur d'impulsions ;

— le signal de synchronisation produit par le générateur d'impulsions sur réception d'un signal de *trigger* ;

— la présence d'une perturbation électromagnétique grâce à une antenne placée dans le chemin de la perturbation.

L'ordinateur contrôle enfin l'ensemble des équipements électroniques. Il orchestre l'exploration automatique des paramètres d'une injection de fautes grâce à un ensemble de scripts écrits en langage Python.

2.2.2 Circuits de tests

Un MCU est dit sécurisé s'il est conforme à des standards de sécurité évalués par des organismes gouvernementaux ou indépendants. C'est le cas notamment des cartes à puce, ou *smart cards*, qui doivent être certifiées Critères Communs (CC) avant d'être commercialisées. Par opposition à ces circuits intégrés, la grande majorité des MCU utilisés dans les objets connectés sont dits non-sécurisés. En particulier, ils ne sont pas protégés contre les attaques en fautes.

Notre étude des modèles de fautes produits par des impulsions EM sur des MCU s'est concentrée sur deux circuits non-sécurisés destinés à des applications embarquées : le MCU 32 bits SAM3X8E et le MCU 8 bits ATmega328P. Ces circuits constituent les circuits de tests (CT) de nos expériences.

Notre choix s'est orienté vers des MCU intégrés à des cartes de prototypage dont les plans de conception sont disponibles en accès libre sur internet. Ce choix est motivé, d'un côté, par une prise en main rapide de ces circuits et, d'un autre côté, par la possibilité de concevoir facilement de nouvelles cartes de prototypages adaptées aux contraintes mécaniques de notre dispositif d'injection. Les paragraphes suivants décrivent les caractéristiques techniques de nos circuits de tests.

2.2.2.A MCU SAM3X8E

Le MCU SAM3X8E [16] est un MCU 32 bits fabriqué par la société Microchip Technology Inc. La technologie de fabrication du circuit est estimée à 0.25 µm en mesurant la surface de la mémoire Flash sur les images par microscopie de la face avant d'un circuit décapsulé. La méthode utilisée pour cette estimation est détaillée dans l'Annexe A. Dans nos expériences, le circuit est monté sur une carte de prototypage Arduino Due [8].

Le MCU embarque :

— un cœur de calcul Arm Cortex-M3 [11] ;

— une SRAM de 96 ko ;

— une mémoire NOR Flash de 512 ko divisée en deux bancs de 256 ko.

En revanche, il n'embarque pas de mémoire cache.

Le cœur de calcul est un RISC qui implémente un *pipeline* à trois étages : l'étage de préchargement des instructions, ou *fetch*, l'étage de décodage des instructions, ou *decode*, et l'étage d'exécution des instructions, ou *execute*. Il permet d'exécuter des programmes binaires écrits en assembleur dans le jeu d'instructions Thumb supporté par l'architecture ARMv7-M [14]. Les instructions de ces programmes sont codées sur 16 bits ou 32 bits.

Le MCU dispose en outre de seize GPR, de plusieurs GPIO et d'un émetteur-récepteur asynchrone universel (*Universal Asynchronous Receiver-Transmitter* – UART) qui sont utilisés par nos programmes de tests. Dans nos expériences, le MCU est cadencé à une fréquence de 84 MHz.

La mémoire Flash du MCU opère à une fréquence maximale de 23 MHz. Afin de compenser la latence de cette dernière, le constructeur préconise de configurer son contrôleur pour imposer quatre cycles de latence sur le bus de données à chaque accès en lecture [16]. Le contrôleur de la mémoire Flash du MCU embarque par ailleurs trois mémoires tampon de 128 bits dont la fréquence de fonctionnement est égale à celle du cœur de calcul : deux sont dédiées au bus d'instructions du MCU et la dernière au bus de données. Grâce à ces mémoires tampon, la lecture de quatre mots consécutifs de 32 bits alignés sur un bloc de 128 bits ne nécessite qu'une seule opération de lecture dans le tableau de la mémoire Flash. Ces mémoires tampon permettent ainsi de masquer au cœur de calcul la latence des accès à la mémoire Flash.

La Figure 2.4 présente une image par microscopie de la face avant du MCU SAM3X8E. Les positions de la mémoire Flash et de la SRAM du circuit y sont encadrées en rouge.

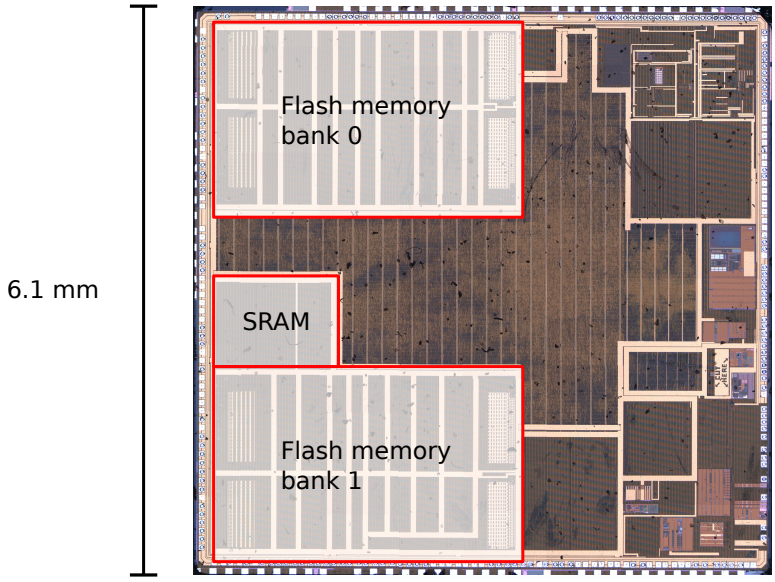


FIGURE 2.4 – Image par microscopie de la face avant du MCU SAM3X8E.

2.2.2.B MCU ATmega328P

Le MCU ATmega328P [15] est un MCU 8 bits fabriqué dans la technologie CMOS 0.35 µm par la société Microchip Technology Inc. Dans nos expériences, le circuit est monté sur une carte de prototypage Arduino Uno [9].

Le MCU embarque :

— un cœur de calcul AVR ;

— une SRAM de 2 ko ;

— une mémoire NOR Flash de 32 ko ;

— une EEPROM de 1 ko.

En revanche, il n'embarque pas de mémoire cache.

Le cœur de calcul est un RISC qui implémente un *pipeline* à deux étages : l'étage de préchargement des instructions, ou *fetch*, et l'étage d'exécution des instructions, ou *execute*. Il permet d'exécuter des programmes binaires écrits en assembleur dans le jeu d'instructions 16 bits AVR [17].

Le MCU dispose en outre de trente-deux GPR, de plusieurs GPIO et d'une interface de communication UART qui sont utilisés par nos programmes de tests. Dans nos expériences, le MCU est cadencé à une fréquence de 16 MHz.

La Figure 2.5 présente une image par microscopie de la face avant du MCU ATmega-328P. Les positions de la mémoire Flash, de la SRAM et de la EEPROM du circuit y sont encadrées en rouge. Le repère utilisé pendant nos expériences y figure en rouge, en haut à droite de la mémoire Flash. Les sauts d'instructions causés par une EMP sont obtenus au niveau de la zone indiquée par une étoile bleue. À titre de comparaison, des résultats similaires par illumination laser du circuit sont obtenus par DUTERTRE, RIOM, POTIN et al. [69] au niveau de la zone indiquée par une étoile rouge.

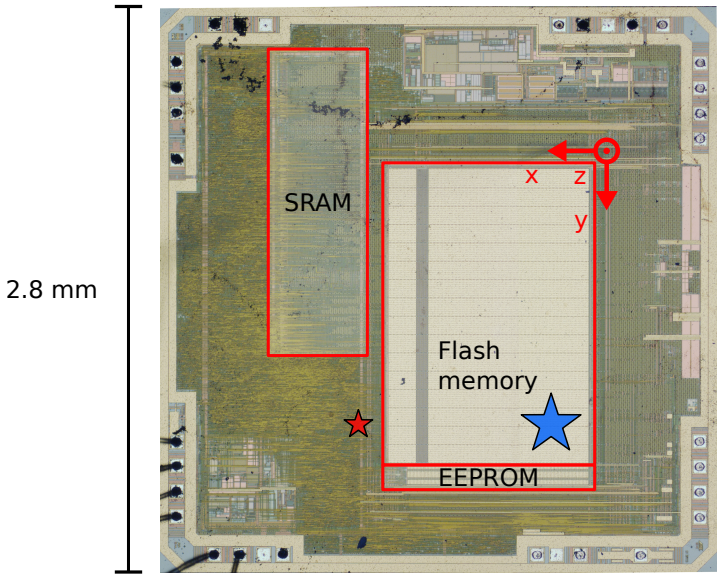


FIGURE 2.5 – Image par microscopie de la face avant du MCU ATmega328P.

2.2.3 Programmes de tests

Nous avons choisi d'analyser les effets d'une perturbation électromagnétique sur des MCU disponibles dans le commerce. Or ces MCU ne sont pas instrumentés pour analyser les effets d'une perturbation physique. Nos travaux se limitent donc à l'analyse des fautes logiques, ou *soft errors*, qui résultent de l'échantillonnage des effets d'une perturbation par un circuit synchrone dans un MCU.

Pour mener à bien cette analyse, nous avons étudié les effets d'une perturbation électromagnétique sur des programmes de tests (PT) écrits principalement en langage assembleur. Ces programmes interagissent directement avec le cœur de calcul des MCU à un niveau proche du matériel, à savoir le niveau de l'ISA. Cette approche a été utilisée avec succès dans la littérature scientifique pour modéliser le comportement d'un MCU sous l'effet d'une injection de fautes [21], [65], [120].

2.2.3.A Modèles de fautes

Lors de nos expérimentations sur l'EMFI, nous n'avons pas observé de corruption des informations « au repos » enregistrées dans la SRAM et la mémoire Flash des MCU SAM3X8E et ATmega328P. Nous nous sommes donc intéressés aux fautes injectées sur les informations « en mouvement » entre les mémoires et le cœur de calcul d'un circuit de tests.

Le MCU SAM3X8E permet l'exécution d'instructions à partir de la mémoire Flash ou de la SRAM. Lors de nos expériences, nous avons constaté que les programmes chargés à partir de la mémoire SRAM du MCU s'exécutaient sans trace de corruption. Nous avons donc choisi de poursuivre notre analyse en isolant les corruptions matérielles induites par une EMFI sur le chargement d'une donnée entre la mémoire Flash et le cœur de calcul du MCU SAM3X8E grâce à un programme de tests exécuté à partir de la SRAM du MCU [65]. Les fautes injectées pendant les transferts de données traduisent ainsi les corruptions matérielles d'une ou plusieurs DFF dans le chemin des données entre la mémoire Flash et le cœur de calcul du MCU SAM3X8E. Ces corruptions peuvent être modélisées par des fautes de type *bit flip*, *bit set*, *bit reset* et *no sampling* qui peuvent être identifiées grâce aux procédures décrites ci-dessous.

Identification d'une faute de type *bit set*. Dans une première étape, les DFF étudiés sont initialisés à l'état `'0'`. Ces circuits sont ensuite mis à jour avec l'état `'0'` et une faute est injectée pendant cette mise à jour. Si l'état final d'un circuit à l'issue d'une injection de fautes est l'état `'1'`, nous considérons qu'une faute de type *bit set* a été injectée.

Identification d'une faute de type *bit reset*. Dans une seconde étape, les DFF étudiés sont initialisés à l'état `'1'`. Ces circuits sont ensuite mis à jour avec l'état `'1'`. Si l'état final d'un circuit à l'issue d'une injection de fautes est l'état `'0'`, nous considérons qu'une faute de type *bit reset* a été injectée.

Identification d'une faute de type *bit flip*. Lorsqu'une faute de type *bit set* et une faute de type *bit reset* ont été identifiées, nous considérons que le modèle de faute correspond à celui d'une faute de type *bit flip*.

Cas d'une faute de type *no sampling*. Puisque notre procédure met à jour l'état final des circuits avec leur état initial, une faute de type *no sampling* correspond à une exécution sans faute. Par conséquent, l'observation d'une faute de type *bit set*, de type *bit reset* ou de type *bit flip* est incompatible avec l'observation d'une faute de type *no sampling*. Notre procédure exclut ainsi, dans sa conception, l'identification des rejeux d'instructions, déjà analysés par RIVIÈRE, NAJM, RAUZY et al. [143].

2.2.3.B Analyse des corruptions de données

L'implémentation en langage assembleur de l'identification d'une faute de type *bit set* sur les transferts de données entre la mémoire Flash et le cœur de calcul du MCU SAM3X8E est présentée sur le Code source 2.1. Ce programme de tests s'étend sans difficulté à l'identification d'une faute de type *bit reset* selon la procédure décrite dans les paragraphes précédents. Son fonctionnement est détaillé ci-dessous.

Initialisation (lignes 5–6). Une instruction de chargement de registre (*LoaD Register* – LDR) charge tout d'abord le registre `"r0"` avec l'adresse du label `".payload"` (ligne 5). Ce label est placé dans la mémoire Flash au début d'un tableau de douze mots de 32 bits. Une instruction de chargement multiple (*LoaD Multiple* – LDM) initialise ensuite les registres `"r1"` à `"r12"` avec les douze mots du tableau à l'adresse contenue dans le registre `"r0"` (ligne 6).

Basculement du *trigger* à l'état haut (lignes 9–11). Le registre `"r5"` est chargé avec l'adresse de base du contrôleur d'un bloc de GPIO et l'état de la GPIO définie comme *trigger* est basculé à l'état haut en modifiant un bit dans un registre de contrôle.

Temporisation (lignes 14–16). La macro `".rept <N> ... .endr"` délimite un bloc d'instructions qui est répété N fois dans le code compilé. Cette macro est utilisée pour insérer un bloc de seize instructions NOP (*No OPeration*) avant la section de code soumise à une EMP. Ce procédé permet d'isoler temporellement les instructions testées des instructions de contrôle du programme de tests. Il facilite par ailleurs la synchronisation d'une perturbation sur l'exécution du programme.

Instructions de test (lignes 17–19). Trois instructions de chargement multiple avec post incrément chargent les registres `"r1"` à `"r12"` par bloc de quatre registres avec douze mots de 32 bits mémorisés dans un tableau en mémoire Flash à l'adresse du label `".payload"`. Après chaque instruction, l'adresse mémorisée par le registre `"r0"` est incrémentée pour pointer sur les quatre mots suivants.

Mémorisation des résultats (lignes 25–26). Après un délai fixé par une deuxième séquence d'instructions `"nop"`, le contenu des registres `"r1"` à `"r12"` est finalement

```
; -- prolog --
    push    {r4-r12}

; -- init --
    ldr     r0, =.payload
    ldm     r0, {r1-r12}

; -- set trigger --
    ldr    r5, =0x400E1200
    ldr    r6, =0x20000000
    str    r6, [r5, 0x30]

; -- test code --
.rept 16
    nop
.endr
    ldm     r0!, {r1-r4}
    ldm     r0!, {r5-r8}
    ldm     r0!, {r9-r12}
.rept 16
    nop
.endr

; -- write result --
    ldr     r0, =sram_array
    stm     r0, {r1-r12}

; -- clear trigger --
    ldr    r5, =0x400E1200
    ldr    r6, =0x20000000
    str    r6, [r5, 0x34]

; -- epilog --
    pop     {r4-r12}
    bx      lr
```

CODE SOURCE 2.1 – Programme en langage assembleur pour la détection des fautes de type *bit set* sur le chargement d'une séquence de mots de 32 bits.

transféré dans un tableau mémorisé en mémoire SRAM grâce à une instruction de mémorisation de registre (*STore Register* – STR). Ce tableau est envoyé ultérieurement à l'ordinateur de contrôle par l'interface UART du MCU.

Basculement du *trigger* à l'état bas (lignes 29–31). Le registre `"r5"` est chargé avec l'adresse de base du contrôleur d'un bloc de GPIO et l'état de la GPIO définie comme *trigger* est basculé à l'état bas en modifiant un bit dans un registre de contrôle.

La définition d'un tableau de douze mots initialisés à zéro dans la mémoire Flash est réalisée en langage assembleur. L'exemple d'un tableau initialisé à zéro est présenté sur le Code source 2.2. Le tableau est constitué de douze mots de 32 bits déclarés par la directive `".word"` à l'intérieur d'une macro de répétition. Ce tableau est placé dans la section des variables en lecture seule `".rodata"` grâce à la directive `".section"`. Ces variables sont enregistrées en mémoire Flash et chargées à partir de la mémoire Flash à chaque utilisation. La directive `".align 16"` aligne l'adresse du tableau sur un bloc de 128 bits.

```
1 .align      16
2 .section    .rodata
3 .payload:
4 .rept 12
5 .word 0x00000000
6 .endr
```

CODE SOURCE 2.2 – Définition d'un tableau de douze mots de 32 bits dans la mémoire Flash.

2.2.3.C Analyse des sauts d'instructions

Notre objectif est d'identifier et d'analyser les sauts d'instructions par le biais de leurs effets sur des programmes de tests écrits principalement en assembleur. Pour chaque série de tests, deux signaux de synchronisation temporelle sont générés par le circuit de tests :

— le premier est généré en avance par rapport à l'exécution du programme de tests afin de palier la latence du banc d'injection utilisé (~ 300 ns) ;

— le second nous permet de localiser temporellement l'exécution des instructions de test.

Le programme de tests utilisé pour calibrer les paramètres expérimentaux lors de nos expérimentations sur le MCU ATmega328P est décrit dans le Code source 2.3.

La section de code ciblée est constituée de dix instructions de chargement avec post-incrément `"ld rn, Z+"` (lignes 28–37). Chaque instruction transfère un mot de 8 bits de la mémoire SRAM au registre de destination `"rn"`. L'adresse du mot transféré est déterminée par la valeur du registre `"Z"` qui pointe initialement sur le premier élément d'un tableau de dix octets en mémoire SRAM. À l'issue de l'opération, le registre `"Z"` est automatiquement incrémenté. Il pointe alors sur l'élément suivant.

```
.section    .text.test_program_ld
.global     test_program_ld
test_program_ld:
; -- prolog --
;   push r2-r17, r28, r29

; -- trigger 1 up --
    sbi     0x05, 0x03

; -- init --
    ldi     r16, 0x55
    ldi     r17, 0x55
;    ...
    ldi     r25, 0x55

; -- load address of target array --
    ldi     ZL, lo8(sram_load_array)
    ldi     ZH, hi8(sram_load_array)

; -- trigger 2 up --
    sbi     0x05, 0x02

; -- test code --
.rept 6
    nop
.endr

    ld      r16, Z+
    ld      r17, Z+
    ld      r18, Z+
    ld      r19, Z+
    ld      r20, Z+
    ld      r21, Z+
    ld      r22, Z+
    ld      r23, Z+
    ld      r24, Z+
    ld      r25, Z+

.rept 4
    nop
.endr

; -- trigger 2 down --
    cbi     0x05, 0x02

; -- write results --
    ldi     ZL, lo8(sram_store_array)
    ldi     ZH, hi8(sram_store_array)
    st      Z+, r16
    st      Z+, r17
;   ...
    st      Z+, r25

; -- trigger 1 down --
    cbi     0x05, 0x03

; -- epilog --
;   pop r2-r17, r28, r29
    ret
```

CODE SOURCE 2.3 – Programme en langage assembleur pour la détection de sauts d'instructions.

Les registres de destination sont préalablement initialisés à la valeur `0x55` en notation hexadécimale (lignes 11–14). Cette valeur est choisie de manière à observer simultanément la mise à `'1'` et la mise à `'0'` des bits dans les mots transférés.

Le tableau mémorisé en SRAM contient les octets de `0x39` à `0x30`. Dans le cas d'une exécution normale, la relecture des registres `"r16"` à `"r25"` après la procédure d'injection renvoie donc dix octets de `0x39` à `0x30`. En revanche, lorsqu'une des dix instructions n'est pas exécutée, la valeur d'initialisation `0x55` apparaît pour le registre correspondant. De plus, le registre `"Z"` n'est pas incrémenté pour cette instruction. Les registres de destination suivants prennent alors la valeur destinée aux registres précédents. Ces réalisations sont illustrées sur la Table 2.2.

TABLE 2.2 – Valeurs de relecture des registres `"r16"` à `"r25"`, pour une exécution normale (en haut) et pour un saut de l'instruction `"ld r19, Z+"` (en bas, surligné en rouge).

Registre	`"r16"`	`"r17"`	`"r18"`	`"r19"`	`"r20"`	`"r21"`	`"r22"`	`"r23"`	`"r24"`	`"r25"`
Éxecution normale	`0x39`	`0x38`	`0x37`	`0x36`	`0x35`	`0x34`	`0x33`	`0x32`	`0x31`	`0x30`

Registre	`"r16"`	`"r17"`	`"r18"`	`"r19"`	`"r20"`	`"r21"`	`"r22"`	`"r23"`	`"r24"`	`"r25"`
Saut d'instruction	`0x39`	`0x38`	`0x37`	`0x55`	`0x36`	`0x35`	`0x34`	`0x33`	`0x32`	`0x31`

2.2.4 Déroulement d'une expérience

Nous définissons une expérience par l'association d'un circuit de tests, d'un programme de tests et d'un espace de paramètres expérimentaux à explorer. Chaque expérience constitue une campagne de mesures dont les résultats sont mémorisés dans un fichier texte au format CSV (*Comma-Separated Values*). Les paragraphes suivant décrivent le déroulement de ces campagnes.

2.2.4.A Automatisation des mesures

L'étude d'un modèle de faute consiste principalement à reproduire une procédure d'injection de fautes dans plusieurs configurations en faisant varier, d'un côté, les paramètres du banc d'injection et, de l'autre, les programmes de tests exécutés par le circuit de tests. L'explosion combinatoire des configurations possibles nécessite alors d'automatiser le déroulement des expériences. Pour répondre à cette problématique, un programme d'automatisation des mesures a été développé dans le langage Python.

2.2.4.B Paramètres expérimentaux

Nous avons choisi d'explorer les paramètres suivants :

— **la coordonnée X** de la sonde d'injection EM ;

— **la coordonnée Y** de la sonde d'injection EM ;

— **la coordonnée Z** de la sonde d'injection EM ;

— **l'amplitude** de l'impulsion de tension ;

— **la largeur** de l'impulsion de tension ;

— **le délai** entre le signal de synchronisation et le premier front d'une impulsion de tension.

À chaque mesure, le programme d'automatisation génère d'abord une nouvelle configuration pour les paramètres susmentionnés. L'ordinateur de contrôle utilise ensuite l'interface programmable des équipements du banc d'injection pour mettre à jour l'état des équipements avec la nouvelle configuration.

2.2.4.C Communication avec le circuit de tests

Un protocole de communication implémenté en langage C grâce à la librairie `"simple-serial"` de la distribution logicielle ChipWhisperer [126] permet au programme d'automatisation des mesures de communiquer avec le circuit de tests en associant un caractère ASCII à chaque programme de tests. Ce protocole est téléchargé avec les programmes de tests dans la mémoire Flash du circuit de tests au début de chaque campagne de mesures.

2.2.4.D Synchronisation d'une injection de fautes

À chaque mesure, le programme d'automatisation envoie une requête au circuit de tests sous la forme d'un caractère ASCII. Le circuit de tests charge alors l'adresse du programme de tests associé et l'exécute. Les premières instructions du programme de tests modifient ensuite l'état logique d'une GPIO du circuit de tests qui sert de signal de déclenchement au générateur d'impulsions. Ce dernier génère alors une impulsion de tension en parallèle de l'exécution du programme de tests. En modifiant le délai programmable du générateur d'impulsions de tension, une faute peut ainsi être injectée à différents instants de l'exécution d'un programme de tests.

Ce procédé est illustré par la Figure 2.6. Dans cet exemple, le programme de tests, nommé `"progtest()"`, est associé à la lettre `'A'` et le programme de relecture des registres, nommé `"send()"` à la lettre `'Z'`.

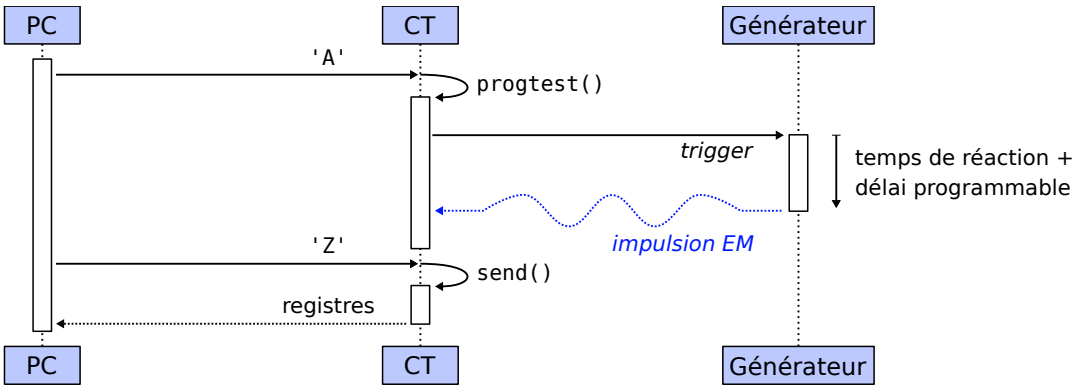


FIGURE 2.6 – Diagramme séquentiel d'une injection de fautes.

2.2.4.E Sauvegarde des résultats

À la fin de l'exécution d'un programme de tests, l'état des GPR du circuit de tests est sauvegardé dans la SRAM. L'envoi d'un caractère spécifique permet alors de relire

l'état des registres. Cette information est finalement sauvegardée avec les paramètres d'injection par l'ordinateur de contrôle dans un fichier de mesures au format CSV.

Les analyses des mesures collectées pendant nos travaux sur l'injection de fautes EM sont détaillées dans les sections suivantes.

2.3 Analyse temporelle des modèles de fautes matériels

Une EMFI sur un circuit intégré doit être précisément synchronisée avec le fonctionnement du circuit pour corrompre l'état d'une DFF. Les fautes injectées dépendent alors de l'instant choisi pour générer une EMP. Cette section détaille l'analyse des résultats obtenus sur le MCU SAM3X8E avec la sonde 3N/∅500 et le générateur d'impulsions de tension AVIR-4-B (200 V) pour différents instants d'injection.

2.3.1 Identification des zones d'intérêt

Afin d'analyser l'influence de l'instant d'injection sur les modèles de fautes obtenues grâce à une EMFI, il convient d'identifier au préalable une position de la sonde d'injection et un instant d'injection pour lesquels une faute est injectée sur un programme de tests. Un balayage spatial de la surface du boîtier du MCU SAM3X8E a été réalisé à cet effet avec la sonde d'injection pour plusieurs valeurs du délai d'injection, de l'amplitude et de la largeur de l'impulsion de tension à l'origine d'une EMP. Cette expérience a pour objectif d'identifier des paramètres pour lesquels une faute est injectée pendant le transfert de 128 bits de la mémoire Flash aux GPR du circuit de tests.

Les résultats d'un balayage spatial avec un pas d'exploration de 500 µm pour un délai de 471 ns, une amplitude de 70 V et une largeur d'impulsion de 6 ns sont reportés sur la Figure 2.8. Pour chaque position de la sonde d'injection, la procédure d'identification des fautes de type *bit set* a été répétée dix fois et le nombre de bits corrompus en moyenne dans un chargement de 128 bits a été consigné. Nos expériences ont été réalisées sur un MCU non-décapsulé. L'origine du repère a donc été choisi sur un des coins du boîtier, hors du périmètre du dé du circuit intégré. Le périmètre d'exploration des coordonnées x et y est encadré en noir sur la Figure 2.7.

La zone sensible aux injections EM est située à l'intérieur d'un spot de 3 mm de diamètre environ, localisé au niveau du banc 0 de la mémoire Flash du circuit. Ce banc correspond au banc de la mémoire Flash où les données du programme de tests sont enregistrées. Le centre du spot ($x = 8$ mm, $y = 8.5$ mm) est une zone de forte sensibilité où les 128 bits transférés de la mémoire Flash aux GPR sont systématiquement mis à `'1'` sous l'effet d'une EMP. Cette position est conservée dans les paragraphes suivants pour analyser l'influence de l'instant d'injection sur les fautes injectées.

2.3.2 Sélection du modèle de faute

Afin d'analyser l'influence de l'instant d'injection sur les propriétés des fautes injectées, un balayage du délai d'injection a été réalisé pour plusieurs valeurs de l'amplitude de l'impulsion de tension, à savoir 40 V, 50 V et 100 V. Des travaux précédents ont mis en

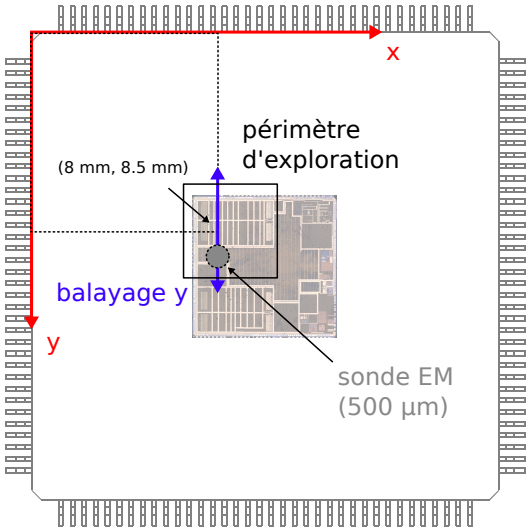


FIGURE 2.7 – Périmètre d'exploration et zone de balayage choisis pour la coordonnée y de la sonde d'injection au-dessus du banc 0 de la mémoire Flash du microcontrôleur SAM3X8E.

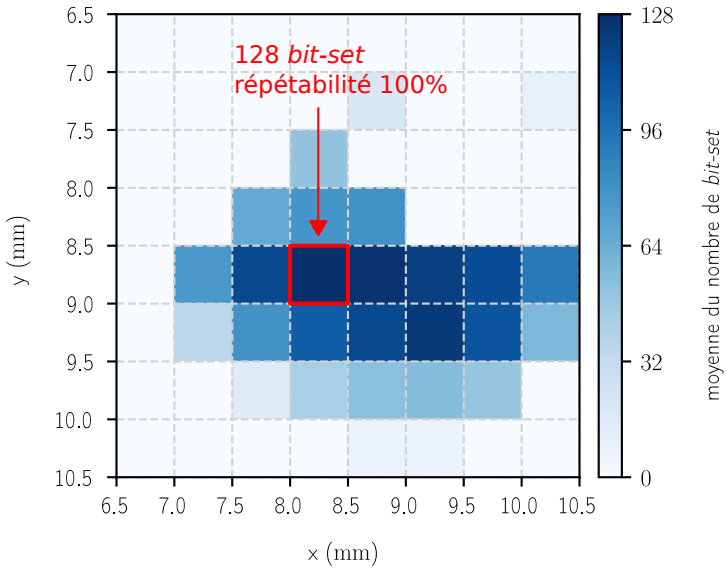


FIGURE 2.8 – Cartographie des zones d'intérêt pour la corruption d'un transfert de 128 bits entre la mémoire Flash et les GPR d'un microcontrôleur (amplitude : 70 V ; délai : 471 ns ; largeur d'impulsion : 6 ns ; pas d'exploration : 0.5 mm ; répétitions : 10).

évidence l'influence de la polarité d'une impulsion de tension sur le type de fautes observées, à savoir *bit set* ou *bit reset*. L'expérience a donc été réalisée sur la procédure d'identification des fautes de type *bit set* et sur celle des fautes de type *bit reset* avec des impulsions de tension de polarités positive et négative. Pour chaque configuration des paramètres d'injection, les deux procédures ont été répétées vingt fois. Le nombre de bits corrompus en moyenne lors d'un chargement de 128 bits dans les registres `"r1"` à `"r4"` (instruction `"ldm r0!, {r1-r4}"` du Code source 2.1) est reporté sur la Figure 2.9.

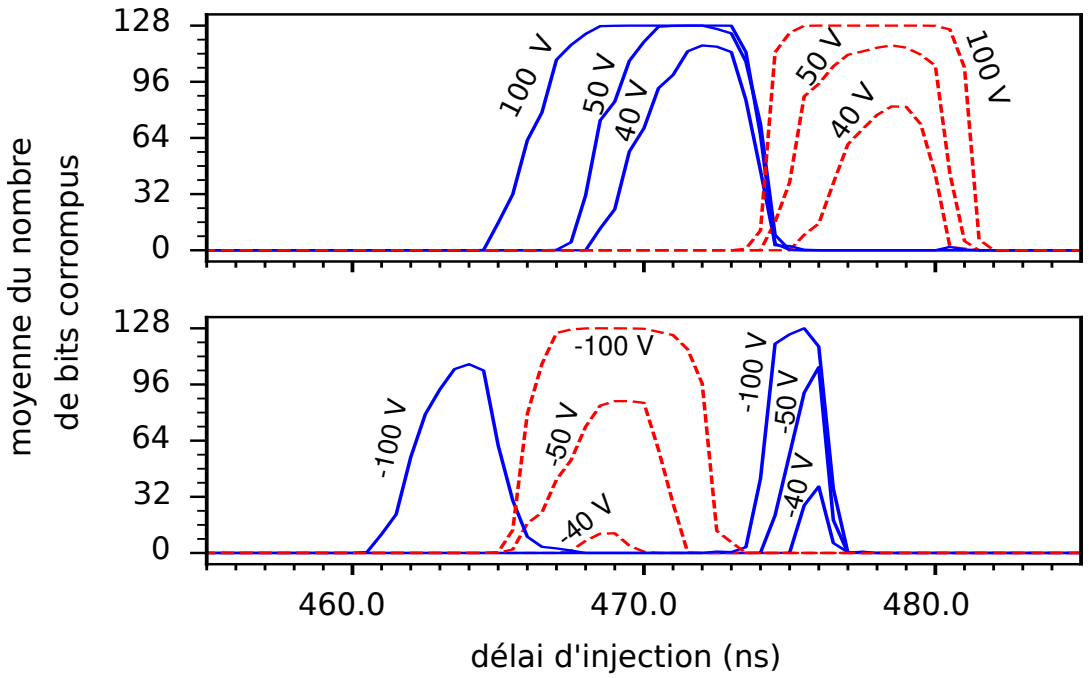


FIGURE 2.9 – Influence du délai d'injection sur le modèle de faute obtenu, *bit set* (en bleu) ou *bit reset* (en rouge), pour des impulsions de polarités positive (figure du haut) et négative (figure du bas) lors du transfert de 128 bits entre la mémoire Flash et les GPR (coordonnée x : 8 mm ; coordonnée y : 8.5 mm ; largeur d'impulsion : 6 ns ; répétitions : 20).

Pour une impulsion d'amplitude 100 V et de polarité positive, l'intégralité des 128 bits transférés dans les registres `"r1"` à `"r4"` peuvent être mis à `'1'` pour un délai d'injection compris dans l'intervalle 469 ns – 473 ns et à `'0'` pour un délai d'injection compris dans l'intervalle 476 ns – 480 ns. Ces résultats sont obtenus avec une répétabilité de 100 % sur vingt répétitions. Ils démontrent ainsi qu'il existe deux fenêtres temporelles de sensibilité distinctes pour les fautes de type *bit set* et de type *bit reset*.

En revanche, pour une impulsion d'amplitude 100 V et de polarité négative, l'intégralité des 128 bits transférés dans les registres `"r1"` à `"r4"` peuvent être mis à `'0'` pour un délai d'injection compris dans l'intervalle 468 ns – 471 ns et à `'1'` pour un délai d'injection de 476 ns. Ces résultats sont obtenus avec une répétabilité de 100 % sur 20 répétitions. Le type de fautes obtenues dans une fenêtre temporelle de sensibilité dépend ainsi de la polarité de l'impulsion de tension.

Le nombre moyen de bits corrompus diminue rapidement en dehors des fenêtres temporelles de sensibilité conformément au mécanisme d'injection de fautes étendu présenté dans la section 1.3.3. Une diminution de l'amplitude de l'impulsion de tension

diminue également le nombre moyen de bits corrompus. Cette dernière observation s'explique par la dépendance entre l'amplitude de l'impulsion de tension et la zone d'effet d'une EMP. Cette propriété est étudiée plus en détails dans l'analyse spatiale des paramètres d'injection (section 2.4).

Ces résultats démontrent qu'un attaquant peut corrompre l'intégralité d'un transfert de 128 bits de données selon un modèle de faute *bit set* ou *bit reset* avec une forte répétabilité en ajustant l'instant d'injection et la polarité d'une impulsion de tension.

Des résultats identiques ont été obtenus en chargeant les registres `"r1"` à `"r4"` avec différentes valeurs de 128 bits comme `0xAA...AA` et `0x55...55` par exemple. Ces expériences et les précédentes confirment que les fautes injectées sont bien des fautes de type *bit set* et de type *bit reset*, et non des fautes de type *bit flip*.

Dans les paragraphes suivants, nous choisissons une polarité positive pour l'impulsion de tension. Ce choix est motivé par la largeur et la régularité des fenêtres temporelles de sensibilité mises en évidence sur la Figure 2.9 pour des impulsions de polarité positive.

2.3.3 Identification des composants vulnérables

2.3.3.A Hypothèses de travail

Lors de nos expérimentations sur l'EMFI, nous avons tout d'abord considéré la possibilité d'injecter des fautes de manière statique sur la SRAM et la mémoire Flash du MCU SAM3X8E. Cependant, comme précisé dans la section 2.2.3.A, nous n'avons pas observé de corruption des informations « au repos » enregistrées dans les mémoires du MCU. Par ailleurs, plusieurs analyses détaillées ont souligné la senibilité des DFF dans les chemins de préchargement des instructions et des données d'un MCU et d'un SoC à une injection de fautes EM [120], [143], [170]. Nous nous sommes donc intéressé aux fautes injectées sur les informations « en mouvement » entre les mémoires et le cœur de calcul du MCU.

Le MCU SAM3X8E permet d'exécuter des programmes à partir de la SRAM du circuit. Or, nous avons constaté que les programmes chargés à partir de la SRAM s'exécutaient sans traces de corruption lors de plusieurs campagnes d'injections de fautes avec notre dispositif. En suivant la méthodologie proposée par DUREUIL, POTET, CHOUDENS et al. [65], nous avons donc choisi de poursuivre notre analyse en isolant les corruptions matérielles induites par une EMFI sur le chargement d'une donnée entre la mémoire Flash et le cœur de calcul du MCU SAM3X8E grâce à un programme de tests exécuté à partir de la SRAM du MCU. Le principe de cette méthode est illustré schématiquement sur la Figure 2.10.

Le programme de tests et les données chargées de la mémoire Flash aux GPR du MCU pendant une injection de fautes sont initialement programmés dans la mémoire Flash du circuit. Lors d'une remise à zéro du MCU, un code dit *bootstrap*, programmé lui aussi dans la mémoire Flash du MCU, est exécuté. Ce code copie alors le programme de tests dans la SRAM et exécute ce dernier **(étape 1)**. Les instructions du programme de tests sont alors transférées de la SRAM au cœur de calcul avant d'être exécutées

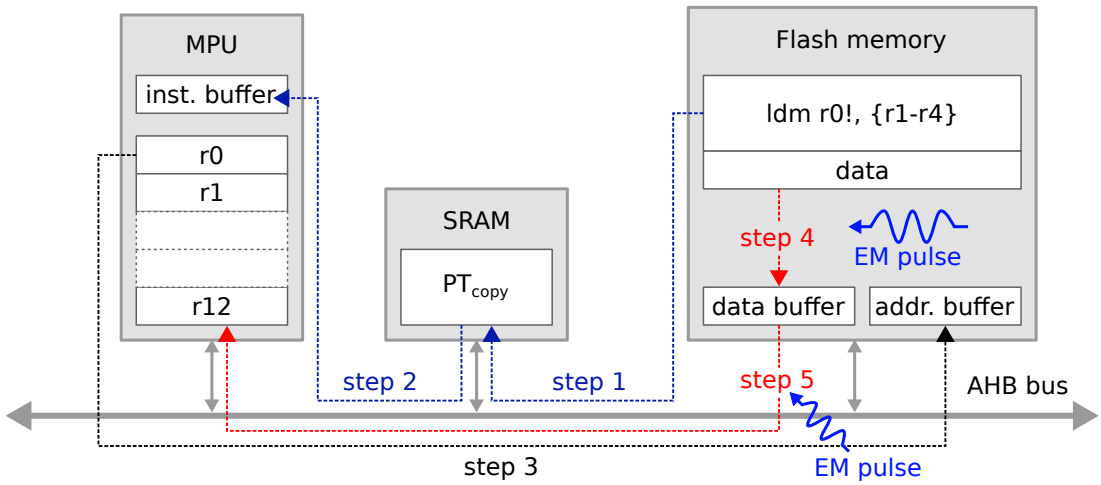


FIGURE 2.10 – Analyse matérielle des fautes sur les transferts de données entre la mémoire Flash et les GPR grâce un programme de tests exécuté à partir de la SRAM d'un MCU.

séquentiellement **(étape 2)**. L'exécution d'une instruction de chargement multiple `"ldm r0!, {r1-r4}"` provoque alors la lecture de 128 bits mémorisés dans la mémoire Flash **(étape 3)**. Ces bits sont lus simultanément dans le tableau de la mémoire et mémorisés dans le *buffer* de préchargement des données de 128 bits **(étape 4)**. À chaque cycle d'horloge, un mot de 32 bits est finalement transféré du *buffer* à l'un des GPR du circuit de tests **(étape 5)**. Une analyse temporelle des fautes doit ainsi permettre de déterminer si l'injection d'une faute se produit :

1. pendant la lecture et l'échantillonnage de 128 bits dans le *buffer* de préchargement des données ;
2. ou lors du transfert d'un mot de 32 bits entre le *buffer* de préchargement des données et un registre du cœur de calcul.

2.3.3.B Analyse théorique

Le *buffer* de préchargement des données de 128 bits améliore le temps d'exécution des programmes lors d'accès séquentiels à la mémoire Flash. Pour ce faire, quatre mots de 32 bits alignés sur un bloc de 128 bits sont préchargés simultanément dans le *buffer* lors du premier accès en lecture au bloc de 128 bits. Ce mécanisme améliore significativement la bande passante de la mémoire Flash dont la fréquence de fonctionnement est inférieure à celle du cœur de calcul du MCU étudié.

Le contrôleur de la mémoire Flash peut être configuré par ailleurs pour insérer des cycles de latence, ou *stall*, sur le bus de données. Ces cycles de latence gèle l'exécution du programme jusqu'à ce que les données soient disponibles. Le nombre de cycles de latence dépend de la fréquence de fonctionnement du cœur de calcul. Le constructeur Microchip Technology Inc. préconise d'insérer quatre cycles de latence lorsque le MCU SAM3X8E fonctionne à une fréquence de 84 MHz. Le diagramme temporel de l'exécution consécutive de deux instructions `"ldm"` dans un *pipeline* à trois étages (*fetch-decode-execute*) est reconstitué sur la Figure 2.11.

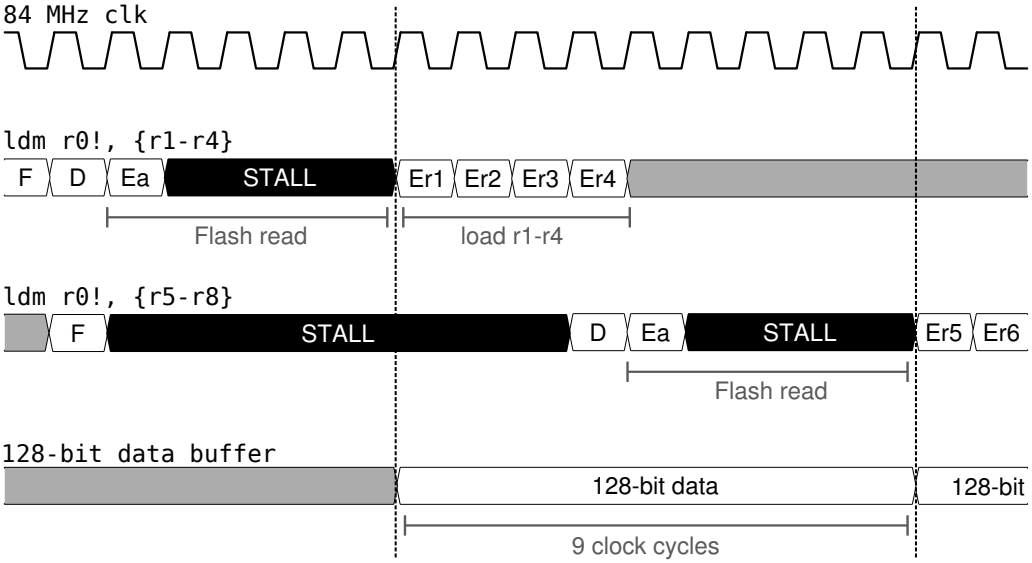


FIGURE 2.11 – Exécution de deux instructions `"ldm"` consécutives dans un *pipeline* à trois étages.

Lors du premier cycle d'horloge, l'instruction `"ldm r0!, {r1-r4}"` est préchargée par le cœur de calcul. Cette phase est désignée par l'acronyme **F**, ou *fetch* dans la littérature anglo-saxonne. Elle s'exécute en un cycle d'horloge.

Lors du second cycle d'horloge, l'instruction `"ldm r0!, {r1-r4}"` est décodée par le cœur de calcul. Cette phase est désignée par l'acronyme **D**, ou *decode*. Elle s'exécute en un cycle d'horloge. En parallèle, l'instruction suivante `"ldm r0!, {r5-r8}"` est préchargée par le cœur de calcul.

À partir du troisième cycle d'horloge, l'instruction `"ldm r0!, {r1-r4}"` est exécutée par le cœur de calcul. L'exécution d'une instruction de chargement indirect se décompose en deux phases.

— Lors de la première phase, le contrôleur de la mémoire Flash est mis à jour avec l'adresse de lecture contenue dans le registre `"r0"`. Cette phase est désignée par l'acronyme **Ea**, ou *Execute address*. Elle s'exécute en un cycle d'horloge. Le contenu du *buffer* de préchargement des données de 128 bits est disponible à l'issue des quatre cycles de latence imposés par le contrôleur de la mémoire. La lecture de 128 bits nécessite donc cinq cycles d'horloge.

— Lors de la seconde phase, le résultat de l'opération de lecture est chargé dans les registres correspondants. Cette phase est désignée par l'acronyme **ErX**, ou *Execute register X*. Chaque chargement de registre nécessite un cycle d'horloge. Le chargement des quatre registres nécessite donc quatre cycles d'horloge. Le décodage de l'instruction suivante ne peut avoir lieu qu'à l'issue de l'avant-dernier chargement.

L'exécution du programme de tests provoque donc un accès en lecture à la mémoire Flash suivi de l'échantillonnage de 128 bits dans le *buffer* de préchargement des données tous les neuf cycles d'horloge.

2.3.3.C Résultats expérimentaux

Pour distinguer les fautes sur l'échantillonnage de 128 bits dans le *buffer* de préchargement des données des fautes sur le transfert d'un mot de 32 bits entre le *buffer* de préchargement des données et un GPR du circuit de tests, trois instructions de chargement multiple sont exécutées séquentiellement. Le programme de tests est détaillé sur le Code source 2.1. Ces instructions chargent les douze mots de 32 bits d'un tableau enregistré dans la mémoire Flash du circuit de tests dans les registres `"r1"` à `"r12"` du circuit de tests. La première instruction charge les registres `"r1"` à `"r4"`, la seconde les registres `"r5"` à `"r8"` et la dernière les registres `"r9"` à `"r12"`. Une EMP est générée pour différentes valeurs du délai d'injection afin de corrompre le chargement du *buffer* de préchargement des données ou des registres.

L'indice des bits corrompus lors du chargement des registres `"r1"` à `"r4"`, `"r5"` à `"r8"` et `"r9"` à `"r12"` est reportée sur la Figure 2.12 en fonction du délai d'injection pour une amplitude de 150 V et une largeur de 6 ns de l'impulsion de tension à l'origine d'une EMP aux coordonnées (8 mm, 8.5 mm). La procédure est répétée vingt fois à chaque valeur du délai d'injection avec, dans un premier temps, des mots dont tous les bits sont à `'0'` et, dans un second temps, des mots dont tous les bits sont à `'1'`. Les fautes de type *bit set* obtenues dans le premier cas sont colorées en bleu, les fautes de type *bit reset* obtenues dans le second cas sont colorées en rouge.

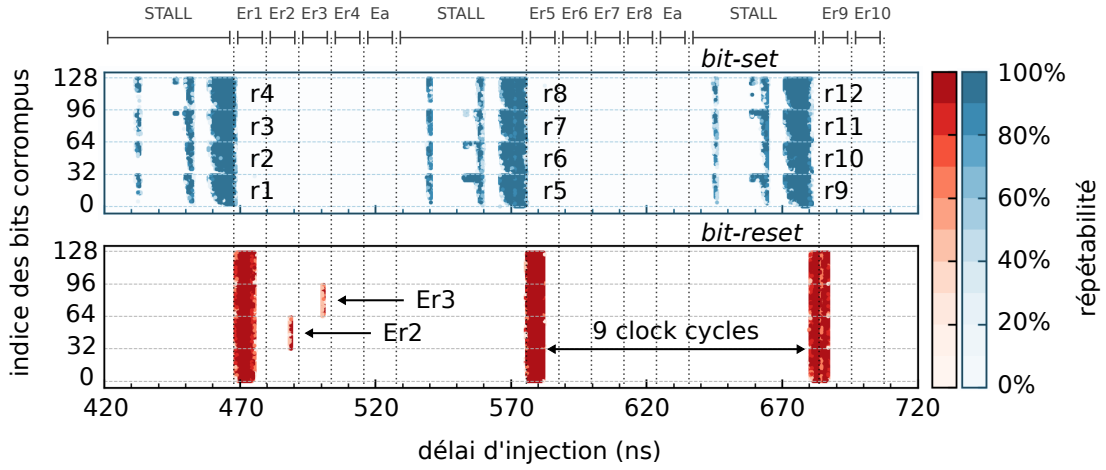


FIGURE 2.12 – Indice des bits sur lesquels une faute de type *bit set* (en bleu) ou de type *bit reset* (en rouge) est identifiée lors du chargement des registres `"r1"` à `"r12"` en fonction du délai d'injection (coordonnée x : 8 mm ; coordonnée y : 8.5 mm ; amplitude : 150 V ; largeur d'impulsion : 6 ns ; répétitions : 20).

Les transferts de douze mots consécutifs dans les registres `"r1"` à `"r12"` peuvent être corrompus par groupe de quatre mots avec une période d'environ 105 ns, ou neuf cycles d'horloge du circuit de tests cadencé à une fréquence de 84 MHz. Les 128 bits transférés dans les registres `"r1"` à `"r4"` sont ainsi mis à `'1'` lors de l'exécution de la première instruction `"ldm"` pour un délai d'injection de 465 ns. Ceux transférés dans les registres `"r5"` à `"r8"` sont ensuite mis à `'1'` lors de l'exécution de la deuxième instruction `"ldm"` pour un délai d'injection de 570 ns. Ceux transférés dans les registres `"r9"` à `"r12"` sont enfin mis à `'1'` lors de l'exécution de la troisième instruction `"ldm"` pour un délai

d'injection de 675 ns. Des résultats similaires sont obtenus pour les fautes de type *bit reset* à des délais d'injection de 475 ns, 580 ns et 685 ns pour les registres `"r1"` à `"r4"`, `"r5"` à `"r8"` et `"r9"` à `"r12"` respectivement.

Des fautes de type *bit reset* ont également été obtenues sur le transfert d'un mot de 32 bits du *buffer* de 128 bits vers le registre `"r2"` pour un délai d'injection de 490 ns et sur le transfert d'un mot de 32 bits du *buffer* de 128 bits vers le registre `"r3"` pour un délai d'injection de 502 ns. Ces résultats correspondent aux résultats obtenus par MORO, DEHBAOUI, HEYDEMANN et al. [120] suite à l'injection de fautes par impulsion EM sur le bus de données d'un MCU 32 bits.

Notons que ce mécanisme d'injection de fautes n'a pas été observé lors des expérimentations réalisées en fixant à 100 V l'amplitude de l'impulsion de tension à l'origine d'une EMP. Ce phénomène peut s'expliquer par les différents temps de propagation des différents segments du chemin de chargement des données. Une EMP de faible intensité provoque en effet une faute sur le chemin avec le temps de propagation les plus élevées, à savoir le transfert d'un mot de la mémoire Flash au *buffer* de préchargement des données, conformément au mécanisme d'injection de fautes étendu présenté dans la section 1.3.3. Une augmentation progressive de l'intensité d'une EMP permet alors de fauter le transfert d'un mot de la mémoire au *buffer* de préchargement des données, ainsi que le transfert d'un mot du *buffer* de préchargement des données aux registres généraux. Puisque le temps de chargement des registres n'est pas le même pour tous les registes, seuls certains registres peuvent être fautés pour les paramètres d'injection utilisés.

Les fautes de type *bit reset* obtenues sur le transfert d'un mot de 32 bits entre le *buffer* de 128 bits et les registres `"r2"` et `"r3"` servent de points de référence pour aligner les résultats expérimentaux avec le chronogramme issues de l'analyse théorique réalisée dans les paragraphes précédents. Nous constatons ainsi que l'injection de fautes de type *bit set* et de type *bit reset* coïncide avec l'opération de lecture des données dans la mémoire Flash. Au regard de ces éléments, nous concluons que les 128 bits du *buffer* de préchargement des données peuvent être simultanément mis à `'1'` ou à `'0'` en synchronisant la production d'une EMP pendant la lecture et l'échantillonnage des données enregistrées dans la mémoire Flash.

2.4 Analyse spatiale des modèles de fautes matériels

L'EMFI a un effet local [57], [183] qui peut être classé entre l'effet local des techniques d'injection laser et l'effet global des techniques d'injection par perturbation des signaux d'horloge et d'alimentation. Par conséquent, la position de la sonde d'injection a un effet significatif sur la nature des fautes injectées. Cette section détaille l'analyse des résultats obtenus sur le MCU SAM3X8E avec la sonde 3N/∅500 et le générateur d'impulsions de tension AVIR-4-B (200 V) pour différentes positions de la sonde d'injection.

Dans les sections précédentes, nous avons établi qu'une faute de type *bit set* est injectée sur les 128 bits du *buffer* de préchargement des données lorsque l'amplitude de l'impulsion de tension est fixée à 70 V (polarité positive), la largeur de l'impulsion de tension à

6 ns, le délai d'injection à 471 ns, la coordonnée x de la sonde à 8 mm et la coordonnée y de la sonde à 8.5 mm. Nous avons donc choisi de conserver ces paramètres pour analyser l'influence de la coordonnée y de la sonde d'injection sur la corruption d'un bit parmi les 128 bits transférés entre le tableau de la mémoire Flash et le *buffer* de préchargement des données lors d'une opération de lecture dans la mémoire Flash. L'axe d'exploration de la coordonnée y est illustré en bleu sur la Figure 2.7

Un balayage de la coordonnée y a été réalisé sur la surface du boîtier du MCU SAM3X8E par pas de 50 µm et une EMP est générée pendant la lecture de quatre mots de 32 bits en mémoire Flash à chaque position. L'opération a été répétée cinquante fois par position. Les distributions des fautes de type *bit set* obtenues sur chacun des bits d'indice 0 à 127 sont reportées en fonction de la coordonnée y de la sonde d'injection sur la Figure 2.13.

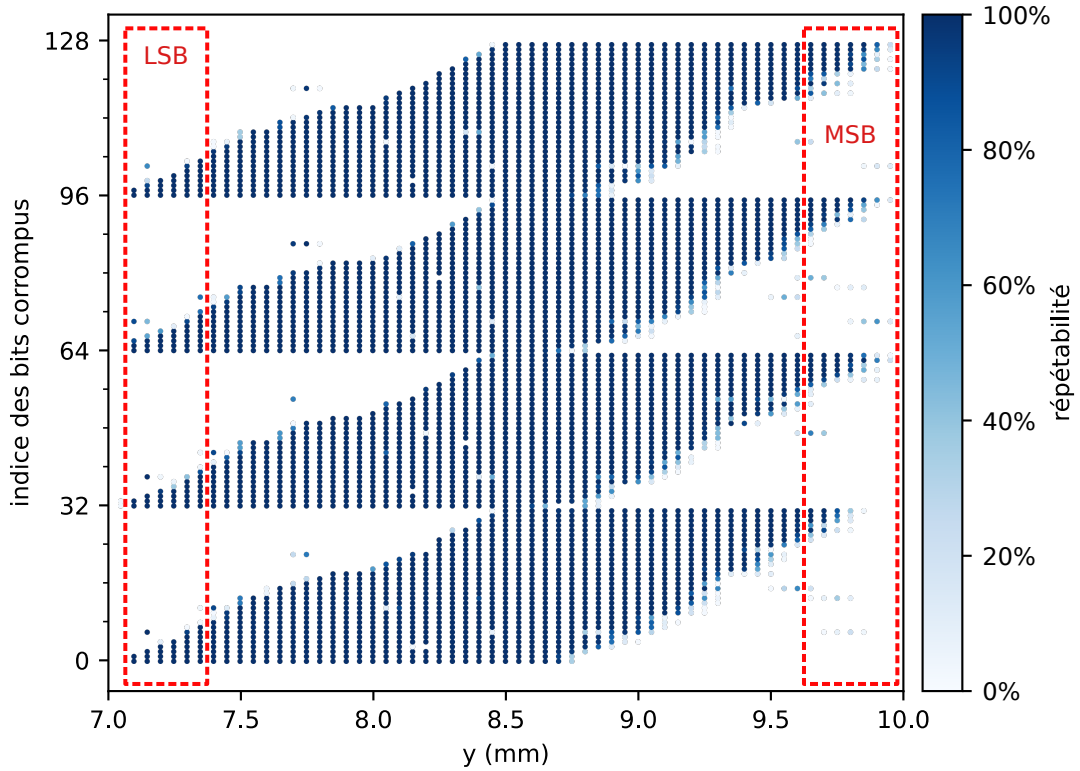


FIGURE 2.13 – Indice des bits mis à '1' dans le *buffer* de préchargement des données en fonction de la position de la sonde (coordonnée x : 8 mm ; amplitude : 70 V ; délai : 471 ns ; largeur d'impulsion : 6 ns ; pas d'exploration : 50 µm ; répétitions : 50).

Nous observons, en premier lieu, que la distribution spatiale des fautes sur un bit d'indice donné est regoupée à l'intérieur d'une fenêtre de sensibilité spatiale d'environ 1.5 mm où les fautes sont injectées avec une répétabilité proche de 100 %. Par exemple, pour le bit d'indice 0, cette fenêtre recouvre l'intervalle 7.1 mm – 8.7 mm. Nous observons, en second lieu, que la position de ces fenêtres dépend de l'indice du bit sur lequel une faute de type *bit set* est injectée. En effet, nous constatons que les bits de même indice modulo 32 peuvent être corrompus simultanément pour les mêmes valeurs de la coordonnée y de la sonde d'injection. Ainsi, les fautes de type *bit set* sont injectées uniquement sur les bits de poids faible des quatre mots de 32 bits chargés dans le *buffer* lorsque la valeur de la coordonnée y de la sonde d'injection est inférieure à 7.4 mm. De manière analogue, les fautes de type *bit set* sont injectées uniquement sur les bits

de poids fort lorsqu'elle est supérieure à 9.6 mm. Les fenêtres spatiales correspondant à ces deux scénarios sont encadrés en pointillé rouge sur la Figure 2.13. Des résultats similaires ont été observés pour les fautes de type *bit reset* avec un délai d'injection de 479 ns.

Des expériences complémentaires, qui ne sont pas détaillées dans ce manuscrit, nous ont permis d'observer une réduction de la largeur des fenêtres de sensibilité spatiale avec une réduction de l'amplitude de l'impulsion de tension à l'origine d'une EMP. Ces expériences confirment donc une dépendance entre la taille de la zone d'effet d'une EMP et l'intensité de la perturbation. Notons que cette dépendance avait déjà été observée par LIAO et GEBOTYS [104] par exemple. La résolution spatiale d'une EMP permet ainsi de modifier l'intégralité ou une partie du *buffer* de préchargement des données avec une forte répétabilité.

2.5 Fautes sur les données et les instructions

Dans les sections précédentes, nous avons démontré que les 128 bits du *buffer* de préchargement des données peuvent être facilement mis à `'1'` ou à `'0'` sans modifier l'exécution du programme lorsque ce dernier est exécuté à partir de la mémoire SRAM. Dans cette section, nous démontrons que c'est toujours le cas lorsque le programme est exécuté à partir de la mémoire NOR Flash du MCU.

Nous étudions plus particulièrement la possibilité d'injecter des fautes pendant le transfert d'une clé de 128 bits entre la mémoire Flash et les registres de calcul au début de la fonction de dérivation de clé d'un algorithme de chiffrement AES-128. Le code étudié est extrait de l'implémentation publique proposée par SCHWABE et STOFFELEN [157]. Cette dernière est optimisée en assembleur pour les MCU Cortex-M3 et Cortex-M4. Les premières instructions de la fonction de dérivation de clé sont données dans le Code source 2.4.

L'adresse de la clé est un pointeur sur un tableau de 16 octets mémorisés en mémoire Flash. Ce pointeur est passé en premier argument à la fonction de dérivation de la clé `"AES_128_keyschedule"`. Selon la convention d'appel ARM, cet argument est mémorisé dans le registre `"r0"` au début de l'exécution de la fonction.

Le résultat de la fonction de dérivation de la clé est mémorisé dans un tableau de cent soixante octets en mémoire SRAM. L'adresse de ce tableau est passée à la fonction en second argument.

L'instruction `"ldm r0, {r4-r7}"` (ligne 13) transfère la clé de 128 bits dans les registres `"r4"` à `"r7"`. Ce chargement de données est la cible des injections de fautes.

Nous avons réussi à fixer la valeur de tous les bits de la clé à `'1'` pour deux délais d'injection différents (635 ns et 636 ns) et à `'0'` pour deux autres délais d'injection (639 ns et 640 ns) avec une répétabilité de 100 % sur cent répétitions, sans modifier l'exécution du reste du code, en fixant la position de la sonde d'injection aux coordonnées (9 mm, 9 mm), la largeur de l'impulsion de tension à 6 ns et l'amplitude de l'impulsion de tension à 65 V. Ces essais mettent en évidence la possibilité de modifier des chargements

```
1  .syntax unified
2  .thumb
3
4  @ void AES_128_keyschedule(const uint8_t *key, uint8_t *rk)
5  .section    .text.AES_128_keyschedule
6  .global     AES_128_keyschedule
7  .type       AES_128_keyschedule, %function
8  AES_128_keyschedule:
9  ; -- prolog --
10     push    {r4-r11}
11
12 ; -- load 128-bit key --
13     ldm     r0, {r4-r7}
14
15 ; -- load T-table address --
16     adr     r3, AES_Te0
17
18 ; -- beginning of round 1 --
19     uxtb    r8, r7, ror #8
20     uxtb    r9, r7, ror #16
21     uxtb    r10, r7, ror #24
22     uxtb    r11, r7
```

CODE SOURCE 2.4 – Implémentation en assembleur de la fonction de dérivation de clé de l'AES-128.

de données alors que les instructions et les données sont chargées à partir de la mémoire Flash. Les mêmes paramètres sont utilisés dans la section 4.2.2 pour corrompre le chargement d'une SBOX en mémoire SRAM lors d'une remise à zéro du MCU SAM3X8E.

2.6 Étude expérimentale du saut d'instruction

L'objectif de nos recherches consistait à reproduire le modèle du saut d'instruction sur un MCU grâce à une EMFI afin d'en étudier la précision et la répétabilité. Des travaux précédents démontrent par ailleurs que plusieurs instructions consécutives peuvent être affectées par un saut. Un rejeu de quatre instructions peut par exemple être obtenu en perturbant l'opération de lecture d'un cache d'instructions [143]. Par conséquent, les quatre instructions suivantes ne sont pas exécutées. Une unique perturbation du signal d'horloge d'un circuit permet également d'empêcher l'exécution de plusieurs instructions en injectant simultanément des fautes à différents étages d'un *pipeline* complexe [179]. Un saut d'un nombre de six instructions consécutives a été réalisé avec une faible répétabilité par ELMOHR, LIAO et GEBOTYS [73] sur une architecture avec un *pipeline* à cinq étages et plusieurs niveaux de caches. Cependant, les auteurs n'ont pas investigué les causes microarchitecturales de cet effet. La question demeure donc de savoir si un saut de plusieurs instructions peut être obtenu en modulant seulement les caractéristiques temporelles d'une perturbation.

Pour répondre à cette question, nous avons choisi d'étudier les sauts d'instructions injectés sur un MCU sans cache et avec un *pipeline* élémentaire à deux étages. Cette section détaille notre analyse expérimentale des sauts d'instructions causés par une

EMP sur le MCU ATmega328P avec la sonde 6N/∅500 et le générateur d'impulsions de tension AVL-5-B-TR (400 V).

2.6.1 Identification des zones d'intérêt

L'objectif de ces travaux était d'obtenir un effet similaire à celui démontré par DUTERTRE, RIOM, POTIN et al. [69] pour une injection de fautes laser. De nombreux paramètres ont été explorés à cet effet : la position de la sonde d'injection, la synchronisation de la perturbation avec le code de test et l'amplitude de l'impulsion de tension. Ce processus est particulièrement chronophage, en particulier concernant la position de la sonde d'injection. Une semaine entière de manipulations a ainsi été nécessaire pour trouver les paramètres d'injection adéquats. La position de la sonde d'injection est indiquée au niveau de la mémoire Flash MCU ATmega328P sur l'image par microscopie de la face avant du circuit présentée sur la Figure 2.5.

Un exemple de fautes qui peut être observé est illustré sur la Figure 2.14. Deux signaux produits par le circuit de tests sont représentés en haut de la figure :

— le premier est généré en avance de l'exécution du programme de tests afin de palier la latence du banc d'injection utilisé (~ 300 ns) ;

— le second nous permet de localiser temporellement l'exécution des instructions de test.

Il y figure aussi, en bas, une image de la perturbation électromagnétique par une antenne couplée avec la sortie du générateur. Les signaux produits par le circuit de tests sont issus d'une exécution en conditions normales (en rouge) et d'une exécution perturbée sous l'effet d'une EMFI à l'origine d'une corruption du chargement d'une valeur dans le registre `"r19"` par l'instruction `"ld"` (en bleu).

Par ailleurs, une instruction `"ld"` est exécutée en deux cycles d'horloge, alors qu'une instruction `"nop"` est exécutée en un cycle d'horloge. Chaque transformation de l'une en l'autre réduit donc la durée de l'exécution d'un programme d'un cycle d'horloge. Le saut d'une instruction `"ld"` s'observe ainsi par une réduction d'un cycle d'horloge de la largeur des signaux de synchronisation générés par le circuit de tests. La Figure 2.14 illustre ce phénomène pour une amplitude de –200 V et une largeur de 100 ns de l'impulsion de tension. Les valeurs relues après l'injection d'une faute correspondent ainsi à l'exemple détaillé dans le tableau 2.2. Avec ces paramètres, un taux de succès de 100 % a été atteint sur un millier d'itérations.

Cette expérience confirme la possibilité de réaliser un saut d'instruction pendant l'exécution d'un code sur un MCU. La configuration expérimentale associée sert de point de départ dans la suite de nos expériences.

2.6.2 Influence de l'instant d'injection

Nous étendons les résultats précédents en faisant varier le délai entre le signal de commande du générateur d'impulsions et la génération d'une impulsion de tension par pas de 1 ns. Pour chaque valeur explorée, une injection de fautes est réalisée. Les

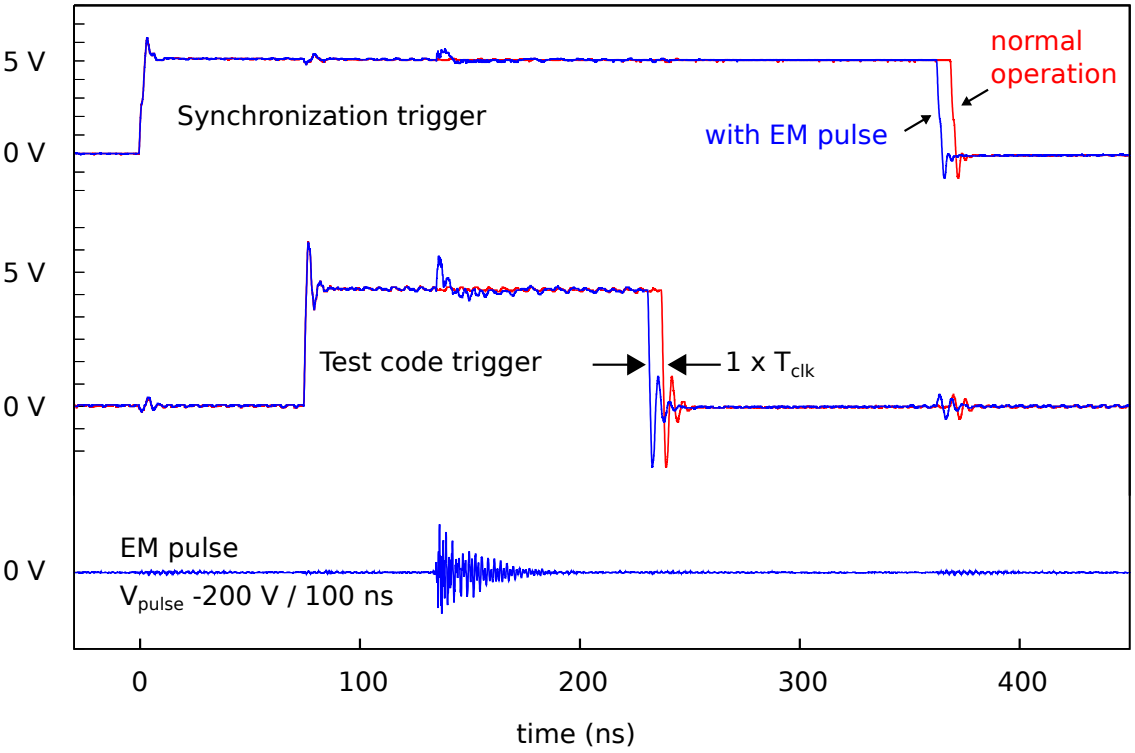


FIGURE 2.14 – Effets de l'injection d'un saut d'instruction unique sur les signaux de synchronisation.

registres corrompus parmi les registres `"r16"` à `"r25"` sont alors consignés. Les résultats de cette expérience sont présentés sur la Figure 2.15.

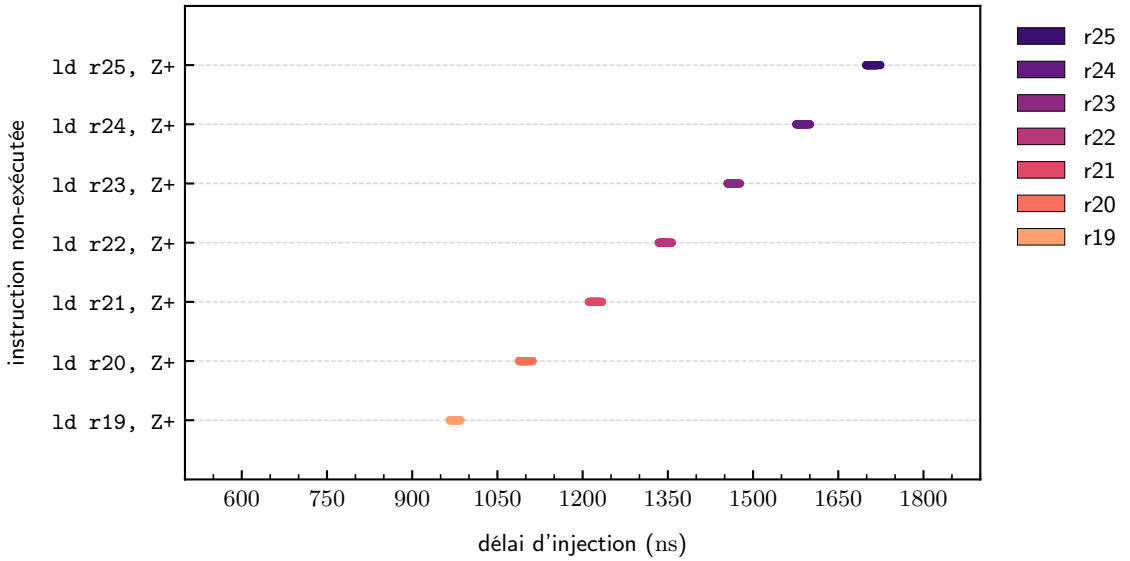


FIGURE 2.15 – Position de l'instruction transformée en `"nop"` en fonction de l'instant d'injection (x : 900 µm ; y : 0 µm ; amplitude : –200 V ; largeur d'impulsion : 100 ns).

Pour chacune des dix instructions ciblées, nous avons identifié une fenêtre temporelle de 25 ns où l'instruction peut être transformée en `"nop"` avec une répétabilité de 100 %. Ces fenêtres sont régulièrement espacées d'un intervalle de 135 ns qui correspond à la durée d'exécution d'une instruction `"ld"`, à savoir deux périodes d'horloge, sur le circuit de tests cadencé à 16 MHz.

Cette expérience démontre qu'un attaquant peut sélectionner les instructions qui sont transformées en `"nop"` pendant l'exécution d'un programme avec une forte résolution temporelle. En effet, pour chaque instruction, nous avons trouvé un délai d'injection permettant d'injecter un saut avec un taux de succès de 100 %.

2.6.3 Influence de l'amplitude de l'impulsion de tension

Nous avons réitéré l'expérience précédente en augmentant l'amplitude de l'impulsion de tension à –250 V. Les résultats obtenus sont présentés sur la Figure 2.16. Ils démontrent qu'une augmentation de l'amplitude de l'impulsion de tension permet de réaliser un saut de deux instructions avec une seule perturbation. L'observation d'une augmentation du nombre d'instructions affectées par un saut avec une augmentation de l'amplitude de l'impulsion de tension est en accord avec les résultats obtenus par ELMOHR, LIAO et GEBOTYS [73].

La résolution temporelle de l'injection d'une faute reste telle qu'un attaquant peut choisir les deux instructions sur lesquelles un saut est injecté. Pour un délai de 980 ns par exemple – représenté en bleu sur la Figure 2.16 – une seule EMP permet d'injecter un saut d'instructions sur les instructions `"ld r19, Z+"` et `"ld r20, Z+"`.

L'expérience est répétée avec des amplitudes de –200 V à –400 V et de 200 V à 400 V. Nous n'avons pas observé de saut de plus de deux instructions consécutives au cours de ces expériences. Ces éléments expérimentaux suggèrent que l'amplitude d'une impulsion de tension possède un effet limité sur l'extension temporelle d'une perturbation induite par une EMP sur un circuit intégré.

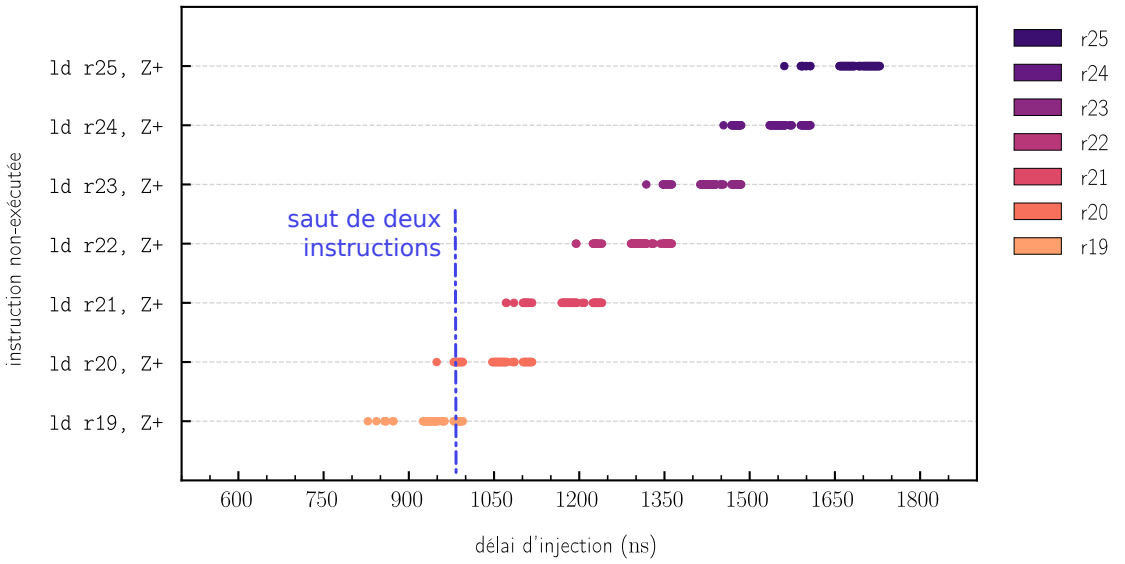


FIGURE 2.16 – Position des instructions transformées en `"nop"` en fonction de l'instant d'injection (amplitude : –250 V; llargeur d'impulsion : 100 ns).

2.6.4 Influence de la largeur de l'impulsion de tension

Lors d'une série de tests, nous avons découvert que la largeur de l'impulsion de tension a un effet sur le nombre d'instructions consécutives affectées par un saut d'instructions.

Pour quantifier ce phénomène, un balayage de la largeur d'une impulsion de tension sur l'intervalle 10 ns – 100 ns a été réalisé pour plusieurs valeurs de l'amplitude de l'impulsion de tension et de l'instant d'injection, ainsi que pour plusieurs positions de la sonde d'injection au-dessus de la mémoire Flash. Les résultats obtenus pour une amplitude de l'impulsion de tension de –400 V et un délai de 976 ns aux coordonnées (1 250 µm, 100 µm) sont reportés sur la figure de gauche de la Figure 2.17, et ceux obtenus pour une amplitude de l'impulsion de tension de –300 V et un délai de 975 ns aux coordonnées (1 250 µm, 250 µm) sur la figure de droite de la Figure 2.17. Les instructions affectées par un saut d'instructions sont indiquées sous forme abrégée par les registres qu'elles modifient.

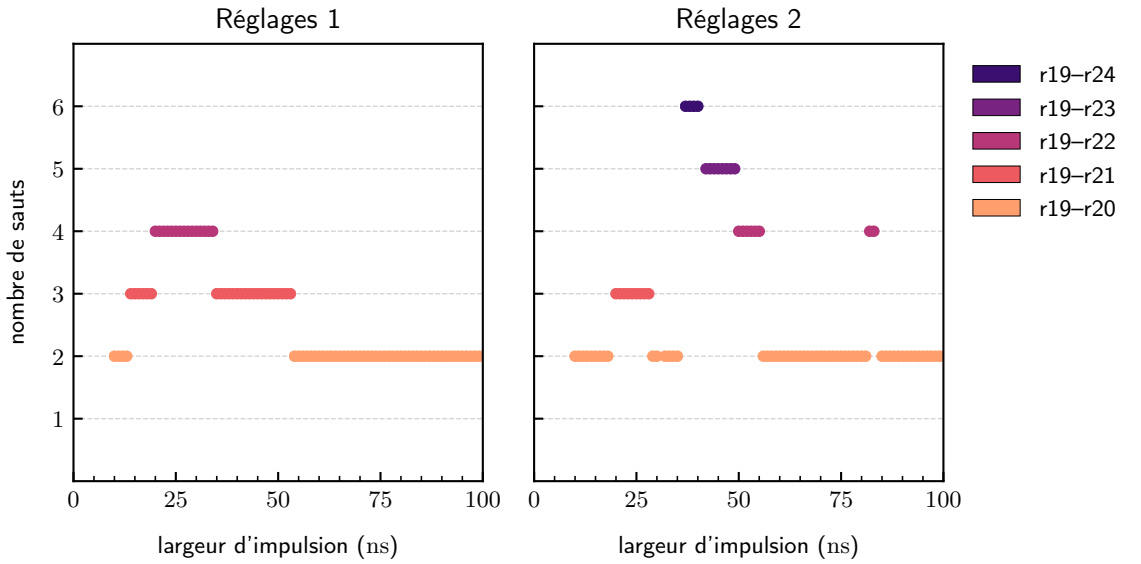


FIGURE 2.17 – Effets de la largeur d'impulsion sur le nombre d'instructions consécutives transformées en "nop" pour le **(réglages 1)** (amplitude : –400 V ; x : 1 250 µm ; y : 100 µm ; délai : 976 ns) et pour le **(réglages 2)** (amplitude : –300 V ; x : 1 250 µm ; y : 250 µm ; délai : 975 ns).

En ajustant la largeur d'une impulsion de tension sur 25 ns, 45 ns ou 75 ns avec le premier réglage, un attaquant peut injecter un saut d'instructions respectivement sur deux, trois ou quatre instructions consécutives avec une seule EMP. En ajustant la largeur d'une impulsion de tension sur 40 ns avec le deuxième réglage, un saut de six instructions répétables a été obtenu.

Nous remarquons cependant que la relation entre la largeur de l'impulsion de tension et le nombre d'instructions affectées par un saut n'est pas linéaire. Nous identifions en effet un maximum global pour une largeur d'impulsion comprise entre 20 ns et 35 ns sur le graphique de gauche de la Figure 2.17 et trois maximums locaux pour une largeur d'impulsion d'environ 25 ns, 40 ns et 87 ns sur le graphique de droite de la Figure 2.17.

Pour confirmer que le réglage de la largeur d'une impulsion de tension ne compromet pas la possibilité de choisir les instructions ciblées, un balayage des valeurs du délai d'injection a été réalisé avec une largeur d'impulsion fixée. La Figure 2.18 synthétise les résultats obtenus pour une amplitude de l'impulsion de tension de –300 V et une largeur de l'impulsion de tension de 26 ns aux coordonnées (1 250 µm, 250 µm).

Ces résultats mettent en évidence deux fenêtres temporelles de sensibilité pour chaque

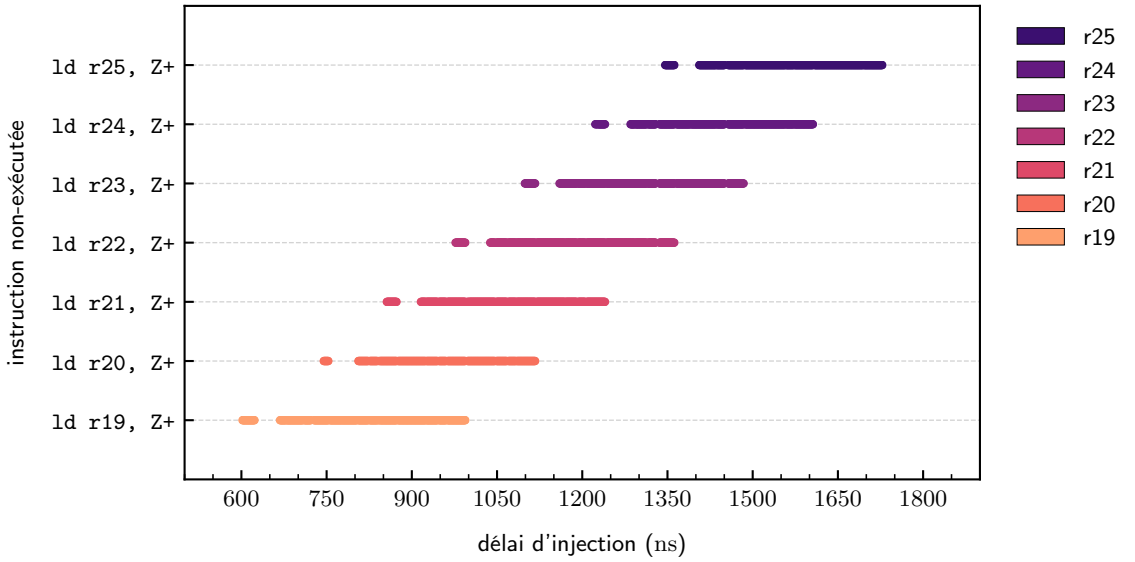


FIGURE 2.18 – Position des instructions transformées en `"nop"` en fonction de l'instant d'injection pour le **(réglage 1)** (largeur d'impulsion : 26 ns ; amplitude : –400 V ; x : 1 250 µm ; y : 100 µm).

instruction – la première, d'une largeur d'environ 25 ns et, la seconde, d'une largeur d'environ 350 ns – dans une fenêtre globale d'une largeur d'environ 400 ns. Pour l'instruction `"ld r19, Z+"`, la première fenêtre dans l'intervalle 600 ns – 625 ns et la seconde dans l'intervalle 650 ns – 1 045 ns recouvrent la quasi-totalité de l'intervalle 600 ns – 1 045 ns. Or, les fenêtres associées aux dix instructions de chargement se répètent avec une période de deux cycles d'horloge, soit 125 ns. Plusieurs fenêtres se chevauchent donc pour des délais d'injection spécifiques. Pour un délai de 976 ns par exemple, un saut d'instructions affecte quatre instructions consécutives, de `"ld r19, Z+"` à `"ld r22, Z+"`.

2.6.5 Discussion du mécanisme d'injection de fautes

Une EMP induit une impulsion de tension sur les grands réseaux métalliques d'un circuit intégré. Ces réseaux comprennent notamment le PGN d'un circuit intégré [62], mais aussi l'arbre d'horloge [80] et les réseaux de *set* et de *reset* des DFF [132]. De nombreux mécanismes physiques ont été proposés dans la littérature scientifique pour décrire les effets d'une EMP sur ces réseaux [57], [80], [104], [121], [134].

Le mécanisme d'injection de fautes étendu considéré dans ces travaux prend en compte l'existence de fenêtres temporelles de sensibilité à l'intérieur desquels la probabilité d'injecter une faute sur une DFF est très élevée. Conformément aux prédictions faites par ce mécanisme, les fenêtres de sensibilité observées dans nos travaux se répètent périodiquement avec une période d'horloge identique aux opérations synchrones affectées par une perturbation électromagnétique. Une modélisation électrique de ce mécanisme a été proposée par DUMONT, LISART et MAURINE [62] pour expliquer l'apparition de fautes d'échantillonnage autour d'un front montant du signal d'horloge sous l'effet d'une impulsion de tension amortie.

Cependant, nos travaux ont mis en évidence une dépendance entre la largeur de ces

fenêtres et la largeur d'une impulsion de tension, dont l'influence était jusqu'alors négligée dans la littérature scientifique [62], [120]. De surcroît, nous avons constaté l'existence d'un ou plusieurs maximums locaux en fonction de la position de la sonde d'injection et de la largeur de l'impulsion de tension. Cette dépendance non-linéaire suggère l'existence d'un mécanisme d'injection de fautes plus complexe que le mécanisme d'injection de fautes étendu.

Ces résultats peuvent être comparés à l'analyse réalisée par ZUSSA, DEHBAOUI, TOBICH et al. [183] du mécanisme d'injection de fautes à l'origine des fautes obtenues grâce à une impulsion de tension sur le signal d'alimentation d'un circuit intégré. Les auteurs ont mesuré les perturbations du signal d'alimentation d'un circuit FPGA grâce à des voltmètres numériques embarqués dans le circuit. Les mesures réalisées ont mis en évidence deux trains d'oscillations amortis en opposition de phase induits respectivement par le front montant et le front descendant d'une impulsion de tension. Des interférences constructives et destructives entre les deux trains d'ondes ont également été obtenues en ajustant précisément la largeur de l'impulsion de tension. Ces interférences ont été utilisées pour augmenter ou diminuer l'extension temporelle de la perturbation à l'origine d'une violation des contraintes temporelles d'une implémentation matérielle d'un AES.

2.7 Conclusion

Dans un premier temps, nous avons analysé les fautes causées par une EMP sur les transferts de données entre la mémoire NOR Flash et le *buffer* de préchargement d'un MCU 32 bits fabriqué dans la technologie CMOS 0.25 µm. Nous avons mis en évidence la possibilité d'injecter des fautes de type *bit set* ou de type *bit reset* sur les transferts de données sans effet visible sur l'exécution du code, en ajustant l'instant d'injection et la polarité d'une EMP. Nous avons étudié les propriétés du modèle de faute, à savoir :

— sa précision spatiale, ou la possibilité de modifier tout ou une partie d'un *buffer* de 128 bits avec une résolution spatiale à l'octet et une forte répétabilité en modifiant la position de la sonde d'injection et l'amplitude de l'impulsion ;

— sa précision temporelle, ou la possibilité de ne fauter qu'un seul chargement de données avec 100 % de répétabilité si les prérequis de synchronisation sont satisfaits.

Nous avons également injecté des fautes sur le transfert d'un mot de 32 bits entre le *buffer* de préchargement des données et l'un des registres `"r2"` ou `"r3"` en augmentant l'amplitude de l'impulsion de tension à l'origine d'une EMP. Les caractéristiques de ce modèle de faute sont résumées sur la Table 2.3.

Dans un second temps, nous avons étudié, en pratique, le modèle logiciel du saut d'instruction causé par une EMP sur un MCU 8 bits ATmega328P. À la différence des modèles de fautes similaires reportés dans la littérature scientifique, les sauts de plusieurs instructions injectés sur le MCU ATmega328P ne s'expliquent ni par une corruption simultanée des instructions dans le *pipeline* du cœur de calcul [179], ni par une corruption d'un cache ou d'un *buffer* d'instruction [143]. Par conséquent, le modèle de faute décrit

PROPRIÉTÉS	PARAMÈTRES
Type de fautes	- **Délai** d'injection - **Polarité** de l'impulsion de tension - **Amplitude** de l'impulsion de tension
Indice des bits mis à `'1'`	- Coordonnées **Y** de l'objectif
Précision spatiale	- Coordonnées **Y** de l'objectif - **Amplitude** de l'impulsion de tension
Précision temporelle	- **Délai** d'injection - **Largeur** de l'impulsion de tension - **Amplitude** de l'impulsion de tension
Répétabilité	- Coordonnées **X** et **Y** de l'objectif - **Délai** d'injection - **Amplitude** de l'impulsion de tension

TABLE 2.3 – Influence des paramètres expérimentaux sur les modèles de fautes par injection EM étudiés dans ces travaux.

résulte principalement des effets temporels de la perturbation. Nous avons mis en évidence le rôle de la largeur de l'impulsion de tension, jusqu'alors négligé dans la littérature scientifique, dans le nombre d'instructions affectées par un saut. Nous avons étudié les propriétés du modèle de faute, à savoir :

— son extension, ou la possibilité de réaliser un saut d'instructions comprenant une à six instructions ;

— sa précision, ou la possibilité de fauter un bloc d'instructions choisi avec 100 % de répétabilité si les prérequis de synchronisation sont satisfaits.

Plusieurs vulnérabilités résultant de ces modèles de fautes sont étudiées dans le chapitre 4.

Ces résultats ont fait l'objet de deux publications :

— A. MENU, S. BHASIN, J. DUTERTRE et al., « Precise Spatio-Temporal Electromagnetic Fault Injections on Data Transfers, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, 2019, p. 1-8 ;

— A. MENU, J. DUTERTRE, O. POTIN et al., « Experimental Analysis of the Electromagnetic Instruction Skip Fault Model, » in *Proceedings of the IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Marrakech, Morocco, 2020, p. 1-7.

Chapitre 3

Modèles de fautes d'une impulsion laser sur une mémoire NOR Flash

De nombreux modèles de fautes décrivent l'effet d'une impulsion laser sur le cœur de calcul ou la SRAM d'un MCU, ainsi que sur la logique ou l'interface de lecture de la mémoire NOR Flash du circuit. Une illumination laser permet également d'effacer le contenu d'un transistor en augmentant localement la température du circuit. Lors de nos expérimentations, nous avons constaté que des fautes pouvaient être injectées sur les transferts d'instructions et de données entre la mémoire Flash et le cœur de calcul d'un MCU en positionnant l'objectif à l'intérieur du tableau de transistors à grille flottante de la mémoire. Or, nous n'avons pas observé de modification statique du contenu des mémoires Flash. Les fautes obtenues résultent donc de l'échantillonnage par un circuit synchrone d'une perturbation transitoire générée au niveau des transistors à grille flottante.

Ce chapitre présente notre analyse de ce mécanisme original d'injection de fautes sur les transistors à grille flottante de deux mémoires NOR Flash embarquées implémentées dans des technologies et des architectures différentes. Nous détaillons tout d'abord les dispositifs et les méthodes utilisés. Nous présentons ensuite plusieurs analyses spatiales et temporelles des modèles de fautes matériels obtenus sur les mots lus en mémoire Flash et nous étudions les limites de notre dispositif d'injection sur les deux circuits de tests. Les conclusions formulées sur les modèles de fautes matériels sont étendues à l'analyse des modèles de fautes logiciels. Une description du mécanisme physique d'injection de fautes sur les transistors à grille flottante des mémoires NOR Flash est enfin proposée.

3.1 Dispositifs et méthodologie

Les travaux décrits dans ce chapitre ont nécessité de nombreux essais expérimentaux sur plusieurs circuits intégrés avec un dispositif d'injection de fautes par impulsion laser. Cette section décrit, d'une part, l'ensemble des dispositifs utilisés et, d'autre part, les méthodes mises en œuvre pour automatiser l'exploration des paramètres d'injection.

3.1.1 Dispositif d'injection de fautes laser

Les paragraphes qui suivent décrivent l'architecture générale du banc d'injection de fautes par perturbation laser utilisé dans ces travaux, ainsi que les paramètres expérimentaux qui y sont associés.

Le banc d'expérience utilisé pour générer des perturbations laser est constitué d'une source laser et d'un système optique, reliés entre eux par une fibre optique mono-mode. Une vue schématique du dispositif est donnée sur la Figure 3.1.

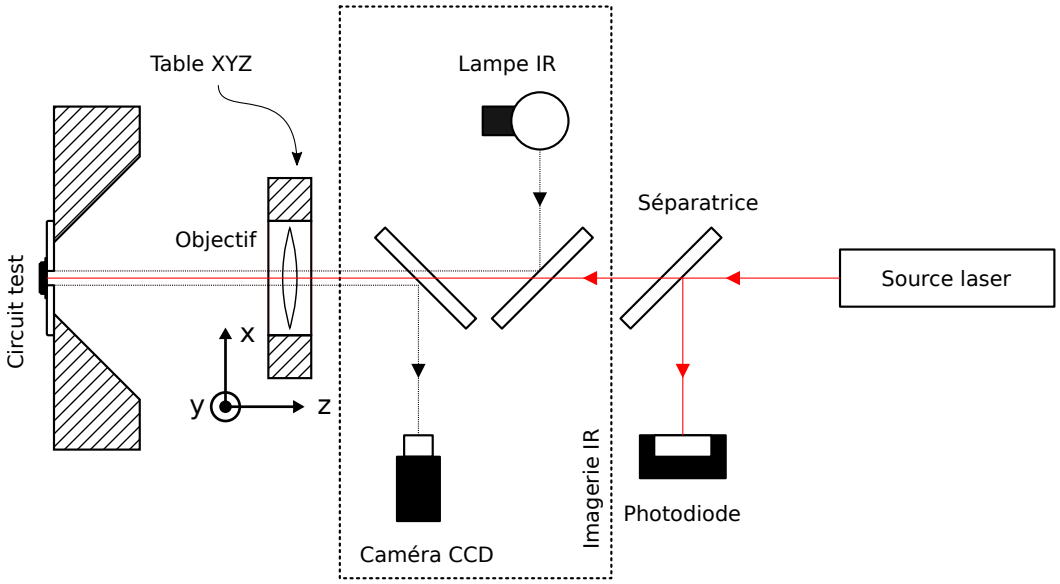


FIGURE 3.1 – Schéma optique d'un banc d'injection laser.

Le banc laser est conçu autour d'une architecture client-serveur qui permet à l'expérimentateur de piloter la source laser et la plateforme de positionnement de manière logicielle.

3.1.1.A Source laser

Notre source laser est capable de générer des impulsions laser nanosecondes dont la durée peut être ajustée entre 50 ns et 1 s, et la puissance entre 0 W et 3 W. Elle est constituée d'une diode laser qui produit un faisceau continu de photons et d'un interrupteur optique à commande électrique qui module l'intensité du faisceau pour former une impulsion. Le contrôleur de la source laser accepte le mode de commande « TrigIn » qui permet d'insérer une temporisation entre la réception d'un signal de déclenchement, aussi appelé *trigger*, et l'ouverture de l'interrupteur. L'utilisation du banc d'injection laser dans le cadre d'une campagne d'injection est donc similaire à celle du banc d'injection EM décrit dans la section 2.2.1.

Les injections de fautes par la face arrière des circuits de tests sont réalisées avec une source laser de longueur d'onde 1 064 nm. La distance de pénétration des photons dans le silicium pour cette longueur d'onde, d'environ 650 µm, permet au faisceau de photons de traverser le substrat, d'une épaisseur de plusieurs centaines de micromètres, jusqu'aux transistors, sans être pour autant intégralement absorbé [38]. Les résultats

des expériences sur l'injection laser décrits dans ce manuscrit ont été obtenus sans amincir le substrat des circuits intégrés.

3.1.1.B Objectifs optiques

Un objectif permet de focaliser le faisceau laser sur le circuit de tests. L'intensité I du faisceau dans le plan focal de l'objectif suit une distribution radiale gaussienne de la forme [38] :

$$\forall r \in \mathbb{R}_+, \quad \mathrm{I}(r) = \mathrm{I}_0 e^{-\frac{2r^2}{\omega_0^2}} \tag{3.1}$$

où le paramètre ω_0 détermine la dispersion de la distribution gaussienne dans le plan focal de l'objectif. Le paramètre est lié par ailleurs à la longueur d'onde λ du laser et à l'ouverture numérique NA de l'objectif :

$$\omega_0 = \frac{2\lambda}{\pi \cdot \mathrm{NA}} \tag{3.2}$$

La taille du spot FWHM (*full-width-at-half-maximum*) est définie comme le diamètre du périmètre circulaire où l'intensité du faisceau est égale à la moitié de l'intensité maximale. La relation entre la taille du spot d_0 et le paramètre ω_0 est ainsi donnée par l'équation :

$$d_0 = \omega_0 \sqrt{\frac{\ln(2)}{2}} \tag{3.3}$$

Les caractéristiques des différents objectifs à notre disposition sont résumées dans la Table 3.1.

Objectif	×5/0.04	×20/0.16	×100/0.8
Grossissement	×5	×20	×100
Ouverture numérique (NA)	0.04	0.16	0.8
Taille de spot (d_0)	20 µm	5 µm	1 µm

TABLE 3.1 – Caractéristiques du jeu d'objectifs.

L'expérimentateur peut ainsi choisir une taille de spot laser de 1 µm, 5 µm ou 20 µm en modifiant manuellement l'objectif utilisé.

3.1.1.C Plateforme XYZ

L'objectif est solidaire d'une plateforme de positionnement micrométrique XYZ qui permet à l'expérimentateur d'ajuster précisément le positionnement du circuit de tests par rapport au système optique dans un espace à trois dimensions. Les coordonnées x et y sont choisies par convention dans le plan du circuit, tandis que la coordonnée z modélise la distance algébrique entre le circuit et le plan focal de l'objectif.

3.1.1.D Système d'imagerie

Le dispositif d'imagerie est constitué d'une lampe infrarouge (IR), de longueur d'onde 1 064 nm, dont le rayonnement est focalisé par le système optique du dispositif d'injection sur la face arrière d'un circuit de tests. Une partie de ce rayonnement est réfléchie sur les niveaux de métallisation du circuit et capturée par une caméra à capteurs photographiques CCD. L'image produite permet alors d'identifier les différents composants d'un circuit, comme le cœur de calcul et les mémoires. Une mise au point de l'image permet d'ajuster le plan focal de l'objectif afin de focaliser le faisceau laser au niveau des zones actives des transistors.

3.1.1.E Paramètres d'injection

Le succès d'une injection de fautes laser dépend de nombreux paramètres utilisés pour configurer le banc d'injection laser, à savoir :

— **le type d'objectif** utilisé (×5, ×20, ×100) ;

— **la coordonnée X** de l'objectif ;

— **la coordonnée Y** de l'objectif ;

— **la coordonnée Z** de l'objectif ;

— **la puissance** de la source laser ;

— **la durée** de l'impulsion laser ;

— **le délai** entre le signal de commande et la génération d'une impulsion laser.

3.1.2 Circuits de tests

Nos travaux sur l'injection de fautes par impulsion laser nous ont permis de mettre en évidence l'existence d'un mécanisme d'injection de fautes original sur la mémoire NOR Flash d'un microcontrôleur 32 bits. Afin de confirmer les hypothèses que nous avons formulées sur l'origine matérielle des fautes observées, des expériences similaires ont été réalisées sur deux microcontrôleurs 32 bits non-sécurisés fabriqués par deux constructeurs différents : le microcontrôleur STM32F100RB fabriqué par la société ST Microelectronics et le microcontrôleur SAMD21G18A fabriqué par la société Microchip Technology Inc. Ces circuits constituent les circuits de tests (CT) de nos expériences.

À l'instar des circuits de tests utilisés dans nos expériences sur l'injection de fautes par EMP, les microcontrôleurs STM32F100RB et SAMD21G18A sont montés sur des cartes de prototypage dont les plans de conception sont disponibles en accès libre sur internet. Les paragraphes suivants décrivent les caractéristiques techniques de nos circuits de tests.

3.1.2.A Préparation des circuits de tests

Décapsulation. Puisque la résine époxy qui compose les boîtiers des circuits intégrés n'est pas transparente à la lumière laser, un accès optique doit être aménagé au travers de ces derniers. Pour ce faire, le boîtier est dissout localement par un jet acide composé

d'acide nitrique et d'acide chlorhydrique. Un équipement semi-automatisé contrôle, d'une part, la stœchiométrie, le débit et la température du mélange et évacue, d'autre part, les produits de la réaction chimique tout en protégeant l'opérateur. Ce procédé de décapsulation plus complexe, mais aussi plus performant, qu'une ouverture mécanique par abrasion, permet de réaliser des petites séries d'ouvertures de manière semi-automatique.

Choix de la face illuminée. Le procédé de décapsulation sus-mentionné permet d'accéder au circuit intégré, soit par sa face avant, soit par sa face arrière. Or, les faisceaux laser dirigés sur la face avant d'un circuit intégré sont partiellement réfléchis par les niveaux de métallisation du circuit avant d'atteindre les transistors. Par conséquent, ce phénomène rend difficile l'évaluation des vulnérabilités pour les circuits fabriqués dans des technologies récentes [169]. Dans ces travaux, nous choisissons donc de réaliser les LFI par la face arrière des circuits de tests.

Accès au substrat. Les circuits intégrés des microcontrôleurs étudiés sont montés à l'intérieur d'un boîtier et connectés par l'intermédiaire de fils de pontage, ou *bonding*, à un support métallique, ou *lead frame*, qui améliore la robustesse du circuit et réalise l'interface avec l'environnement extérieur. Afin d'exposer la face arrière du circuit intégré, il convient donc de retirer la partie du support métallique qui est solidaire du circuit. Puisque cette partie appartient généralement à un plan de masse, une série d'incisions permet d'accéder à la face arrière sans endommager le fonctionnement électrique du circuit.

3.1.2.B Microcontrôleur STM32F100RB

Le microcontrôleur STM32F100RB [164] est un microcontrôleur 32 bits fabriqué par la société ST Microelectronics dans la technologie CMOS 80 nm. Dans nos expériences, le microcontrôleur est monté sur la plateforme d'évaluation matérielle ChipWhisperer [125], [131] dont les plans de conception sont disponibles en accès libre sur le dépôt GitHub de l'entreprise NewAE [126]. Le microcontrôleur embarque :

— un cœur de calcul Arm Cortex-M3 [11] ;

— une SRAM de 8 ko ;

— une mémoire NOR Flash de 128 ko.

En revanche, il n'embarque pas de mémoire cache.

Le cœur de calcul est un RISC qui implémente un *pipeline* à trois étages (*fetch-decode-execute*). Il permet d'exécuter des programmes binaires écrits en assembleur dans le jeu d'instructions Thumb supporté par l'architecture ARMv7-M [14]. Les instructions de ces programmes sont codées sur 16 bits ou 32 bits.

Le microcontrôleur dispose en outre de seize GPR, de plusieurs GPIO et d'une interface de communication UART qui sont utilisés par nos programmes de tests. Dans nos expériences, le microcontrôleur est cadencé à une fréquence de 7.37 MHz.

La Figure 3.2 présente une image infrarouge de la face arrière du microcontrôleur STM32F100RB. Cette image est construite grâce au dispositif d'imagerie IR du banc d'injection laser. Les positions des mémoires Flash, de la SRAM, du cœur de calcul et de la circuiterie analogique du circuit y sont encadrées en rouge.

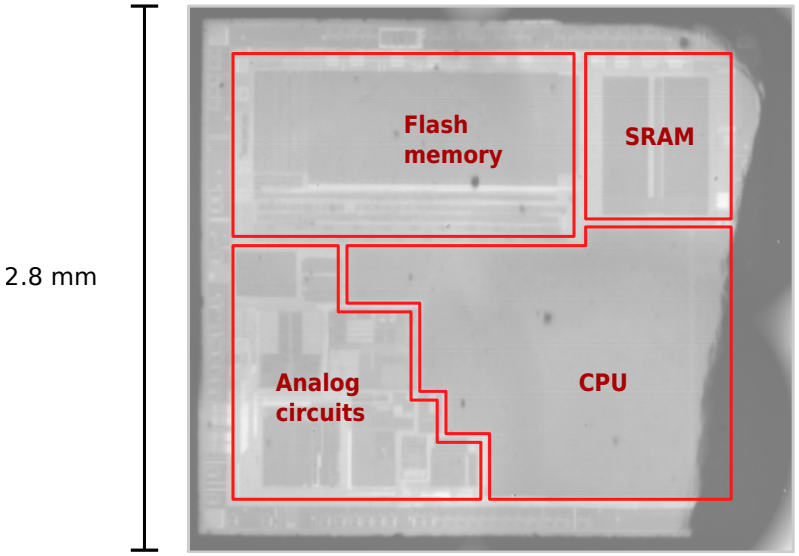


FIGURE 3.2 – Image par infrarouge de la face arrière du microcontrôleur STM32F100RB.

3.1.2.C Microcontrôleur SAMD21G18A

Le MCU SAMD21G18A [117] est un MCU 32 bits fabriqué par la société Microchip Technology Inc. La technologie de fabrication du circuit est estimée à 65 nm en mesurant la surface de la mémoire Flash sur les images infrarouges de la face arrière d'un circuit décapsulé. La méthode utilisée pour cette estimation est détaillée dans l'Annexe A. Dans nos expériences, le MCU est monté sur la carte de prototypage Arduino Zero [10].

Le MCU embarque :

— un cœur de calcul Arm Cortex-M0+ [12] ;

— une SRAM de 32 ko ;

— une mémoire NOR Flash de 256 ko divisée en deux bancs de 128 ko.

Bien que le cœur de calcul n'embarque pas de mémoire cache pour les instructions et les données, une mémoire cache directe de 64 octets dans la mémoire Flash permet d'optimiser les accès en lecture à cette dernière. Nous avons cependant choisi de désactiver cette mémoire lors de nos expérimentations pour simplifier l'analyse des modèles de fautes.

Le cœur de calcul est un RISC qui implémente un *pipeline* à deux étages (*fetch-execute*). Il permet d'exécuter des programmes binaires écrits en assembleur dans le jeu d'instructions Thumb supporté par l'architecture ARMv6-M [13]. Les instructions de ces programmes sont codées sur 16 bits ou 32 bits.

Le microcontrôleur dispose en outre de 16 GPR, de plusieurs GPIO et d'une interface de communication UART qui sont utilisés par nos programmes de tests. Dans nos expériences, le microcontrôleur est cadencé à une fréquence de 48 MHz.

La Figure 3.3 présente une image par microscopie de la face avant du microcontrôleur SAMD21G18A, tandis que la Figure 3.4 présente une image par infrarouge de la face arrière du microcontrôleur. Les positions des mémoires Flash et de la SRAM du circuit y sont encadrées en rouge.

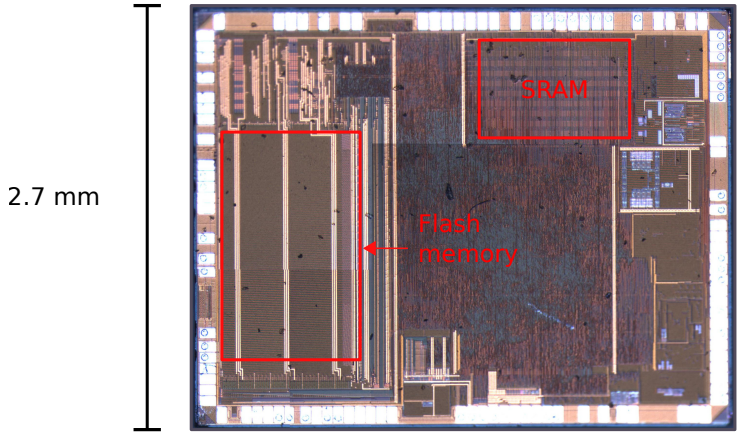


FIGURE 3.3 – Image par microscopie de la face avant du microcontrôleur SAMD21G18A.

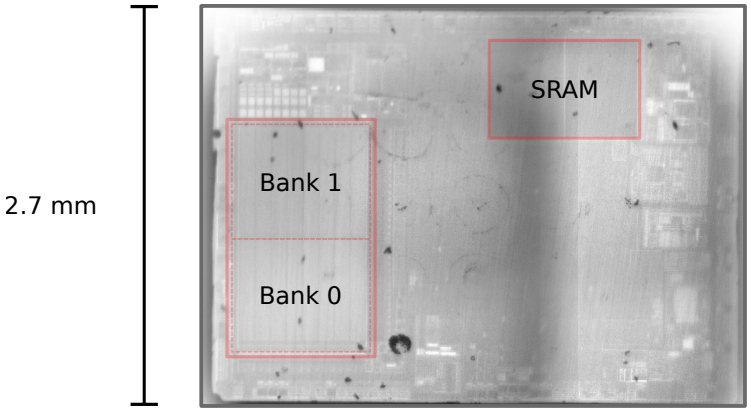


FIGURE 3.4 – Image par infrarouge de la face arrière du microcontrôleur SAMD21G18A.

3.1.3 Programme de tests

Nous avons choisi d'étudier les fautes injectées sur les transferts d'instructions et de données grâce à des programmes de tests (PT) écrits principalement en assembleur. Dans un premier temps, les fautes injectées sur les mots lus en mémoire Flash sont analysées au niveau matériel sur les transferts de données entre la mémoire Flash et les GPR d'un circuit de tests. Pour ce faire, un programme similaire au programme utilisé dans nos expérimentations sur les EMFI est utilisé (le lecteur est renvoyé au Code source 2.1 de la section 2.2.3).

Ce programme utilise l'instruction de chargement de registre `"ldr r1, [r0]"` pour provoquer le chargement dans le registre `"r1"` d'un mot de 32 bits enregistré dans la mémoire Flash des circuits de tests à l'adresse contenue par le registre `"r0"`. L'injection d'une faute sur les bits des données lues en mémoire Flash se traduit ainsi par la modification de la valeur chargée dans le registre `"r1"`. Un version résumée de ce programme est présentée sur le Code source 3.1.

```
1  .rept 16
2      nop
3  .endr
4      ldr      r1, [r0]
5  .rept 16
6      nop
7  .endr
```

CODE SOURCE 3.1 – Programme en langage assembleur pour la détection des fautes de type *bit set* sur le chargement d'un mot de 32 bits.

Notons que, pour des raisons de portabilité, les programmes de tests sont exécutés à partir de la mémoire Flash des circuits de tests dans nos expérimentations sur les LFI. Deux séries d'instructions `"nop"` permettent alors d'isoler le chargement d'une donnée du chargement des instructions du programme de tests. Cependant, l'instruction de chargement de registre utilisée pour provoquer un accès en lecture peut être corrompue en même temps que le chargement de données. Nous démontrons dans la section 3.3.3 qu'une durée d'impulsion laser de l'ordre de la période d'horloge du circuit de tests permet d'obtenir une résolution temporelle suffisante pour injecter une faute unique sur les données transférées, sans corrompre l'instruction de chargement. Un effet similaire est aussi obtenu en ajustant l'instant d'injection après la lecture de l'instruction de chargement de registre.

Au début de chaque exécution, le programme de tests génère un signal de *trigger* sur une GPIO qui permet de synchroniser la génération d'une impulsion laser sur l'exécution du programme de tests. Du fait des similitudes entre les bancs d'injection laser et EM, le déroulement d'une campagne d'injection de fautes laser est alors identique à celui des campagnes d'injection de fautes EM décrit dans la section 2.2.4.

3.2 Analyse spatiale des modèles de fautes

La précision d'une LFI permet de modifier l'état d'un bit à l'intérieur d'un circuit d'intégré. Les effets d'une LFI sur un circuit de tests dépendent donc principalement de la position du spot laser sur la surface du circuit. Cette section présente une analyse spatiale des modèles de fautes matériels obtenus par LFI sur la face arrière des MCU STM32F100RB et SAMD21G18A au niveau de la mémoire Flash embarquée des deux MCU pendant la lecture d'un mot de 32 bits. Les résultats obtenus illustrent le rôle des colonnes d'une mémoire NOR Flash, à l'intérieur des tableaux de cellules mémoire organisés en lignes et en colonnes, dans le mécanisme d'injection de fautes observé.

3.2.1 Influence de la position de l'objectif dans le plan du circuit

Lors de nos premières expérimentations avec différents programmes de tests, nous avons constaté que des corruptions d'instructions et de données pouvaient être obtenues pendant l'exécution des programmes de tests en positionnant l'objectif au niveau des mémoires Flash des circuits de tests. Cependant, aucune corruption des programmes de tests n'a été constatée lors de la relecture des mémoires Flash après une

LFI. Par conséquent, les fautes observées lors de nos expérimentations ont été injectées dynamiquement sous l'effet d'une perturbation transitoire des circuits de tests.

Une analyse des fautes matérielles injectées pendant le transfert d'un mot de 32 bits entre la mémoire Flash et le registre `"r1"` des circuits de tests a été réalisée avec le programme de tests décrit dans la section 3.1.3. Pour ce faire, l'objectif a tout d'abord été positionné à l'intérieur de la mémoire Flash, facilement identifiable sur les images IR de la face arrière du MCU STM32F100RB (Figure 3.2) et du MCU SAMD21G18A (Figure 3.4). Un balayage du délai entre le signal de *trigger* et la génération d'une impulsion laser a ensuite été réalisé. Les valeurs du délai d'injection associées à une corruption du mot de 32 bits chargé dans le registre `"r1"` ont finalement été retenues pour analyser l'influence des coordonnées x et y de l'objectif sur les modèles de fautes observés, à savoir :

— les fautes de type *bit set* ;

— les fautes de type *bit reset* ;

— les fautes de type *bit flip* ;

— la perte de communication avec les circuits de tests, ou *mute*.

L'objectif ×5 à spot large (20 µm de diamètre) a été privilégié pour quadriller rapidement la surface de la mémoire Flash. Une mise au point de l'image IR de la face arrière des circuits, obtenue grâce au dispositif décrit dans la section 3.1.1.D, nous a permis d'ajuster la coordonnée z de l'objectif de manière à focaliser le faisceau laser au niveau des zones actives des transistors, sous les niveaux de métallisation. Les cartographies des fautes obtenues sur les deux circuits de tests sont détaillées dans les paragraphes suivants.

3.2.1.A Cartographie des fautes sur le MCU STM32F100RB

Un balayage des coordonnées x et y de l'objectif dans la surface de la mémoire Flash du MCU STM32F100RB a été réalisé avec un délai d'injection fixé à 1 340 ns, une puissance de la source laser fixée à 1 W et une durée de l'impulsion laser fixée à 50 ns. La cartographie des modèles de fautes matériels obtenus dans cette expérience est présentée sur la Figure 3.5. Cette cartographie est superposée à l'image IR de la face arrière du MCU STM32F100RB sur la Figure 3.6.

La quasi-totalité des fautes identifiées sur le mot chargé dans le registre `"r1"` correspondent à des fautes de type *bit set*, à l'exception d'un *cluster* de fautes de type *mute* au voisinage des coordonnées (0 µm, 540 µm). Les fautes de type *bit set* sont distribuées à l'intérieur de trente-deux colonnes distinctes, indicées de c_0 à c_{31} sur la Figure 3.5. Ces colonnes recouvrent toute la surface de la mémoire Flash. Elles se répètent le long de l'axe x avec une période de 45 µm environ. La forme des zones de sensibilité suggère d'ores et déjà que les colonnes du tableau d'une mémoire NOR Flash jouent un rôle central dans les phénomènes observés.

Chaque colonne est associée à l'injection d'une faute de type *bit set* sur un bit du mot chargé dans le registre `"r1"`. L'indice de ce bit est déterminé par la colonne sur laquelle l'objectif est positionné. Lorsque la coordonnée x de l'objectif est fixée à 100 µm par

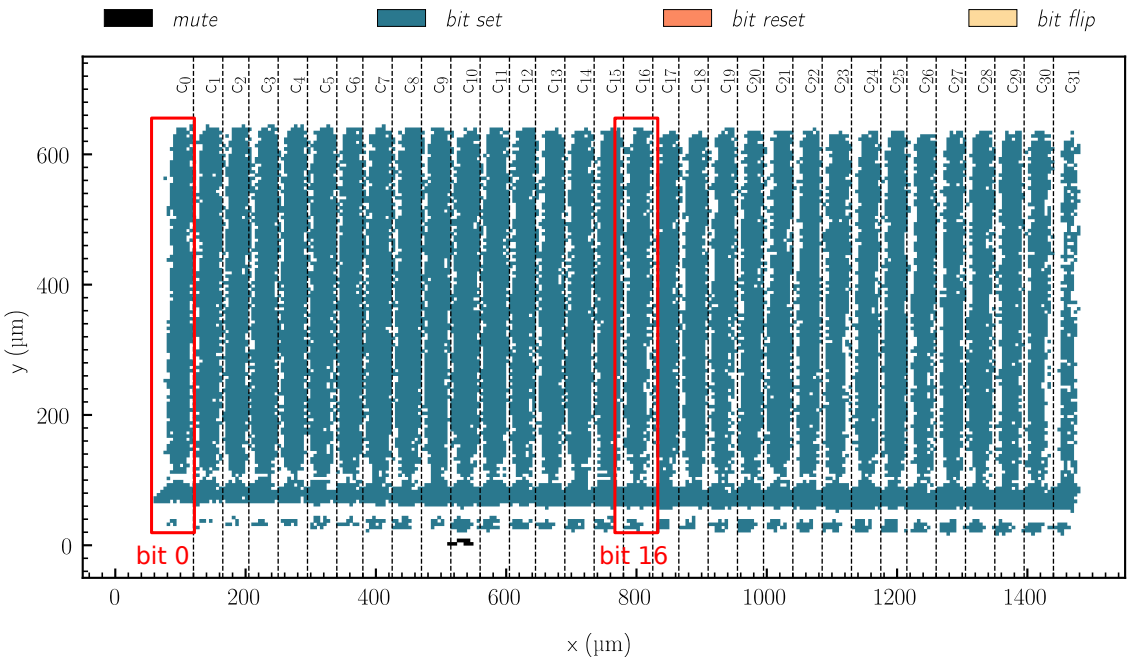


FIGURE 3.5 – Cartographie spatiale des fautes obtenues sur la mémoire NOR Flash du microcontrôleur STM32F100RB (objectif : ×5; puissance = 1 W; durée d'impulsion : 50 ns; pas d'exploration : 5 µm).

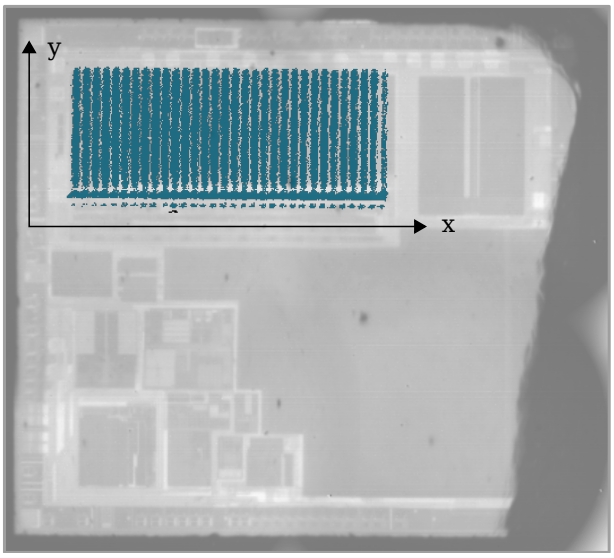


FIGURE 3.6 – Superposition de la cartographie spatiale sur l'image IR de la face arrière du MCU STM32F100RB.

exemple (colonne c_0), le bit d'indice 0 est mis à `'1'` dans le registre de 32 bits. Lorsqu'elle est fixée à 800 µm (colonne c_{16}), le bit d'indice 16 est mis à `'1'` dans le registre de 32 bits. L'indice des bits mis à `'1'` est reporté sur la Figure 3.7 en fonction de la coordonnée x de l'objectif lorsque la coordonnée y est fixée à 300 µm. Les positions des colonnes c_0 et c_{16} sont indiquées en rouge par une barre verticale.

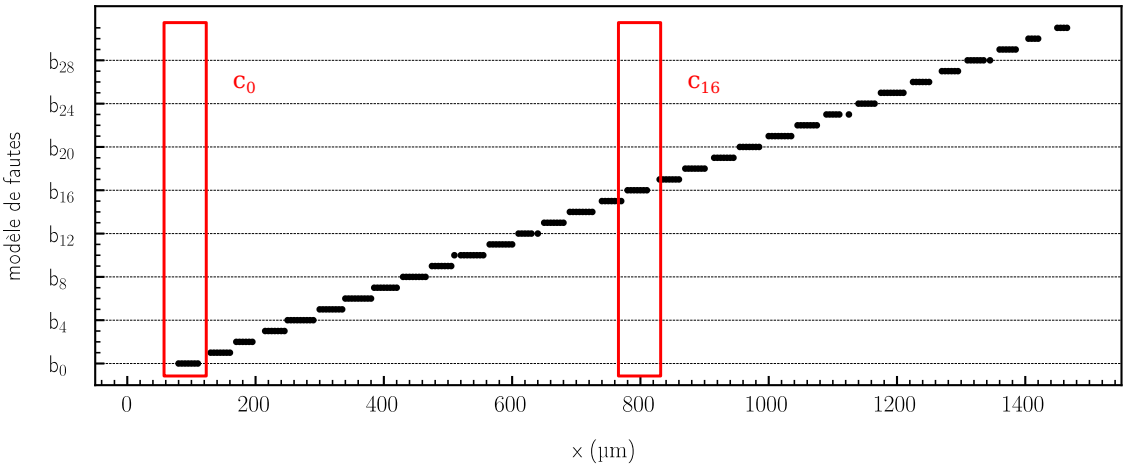


FIGURE 3.7 – Analyse de l'indice des bits mis à `'1'` sur l'ensemble des trente-deux colonnes en fonction de la coordonnée x de l'objectif (objectif : ×5 ; pas d'exploration : 5 µm ; coordonnée y 300 µm ; puissance : 1 000 mW ; durée d'impulsion : 50 ns).

Cette figure met en évidence une corrélation linéaire entre la position des colonnes suivant l'axe x et l'indice du bit sur lequel une faute de type *bit set* est injectée. Ainsi, un attaquant peut forcer un bit de son choix avec une grande précision lors de la lecture d'un mot de 32 bits lu dans la mémoire Flash du MCU STM32F100RB en ajustant la position de l'objectif sur une des colonnes de la mémoire Flash. Des figures identiques ont été obtenues pour différentes valeurs de la coordonnée y de l'objectif. L'influence de la coordonnée y de l'objectif sur les fautes de type *bit set* est donc négligeable, exception faite du défaut d'alignement du circuit par rapport au dispositif d'injection visible sur la Figure 3.5.

Cette analyse questionne donc le rapport entre les fautes observées et l'architecture de la mémoire NOR Flash du MCU. Les paragraphes suivants comparent à cet égard les résultats obtenus sur le MCU STM32F100RB et ceux obtenus sur le MCU SAMD21G18A.

3.2.1.B Cartographie des fautes sur le MCU SAMD21G18A

Un balayage des coordonnées x et y de l'objectif dans la surface de la mémoire Flash du MCU SAMD21G18A a été réalisé avec un délai d'injection fixé à 750 ns, une puissance de la source laser fixée à 0.5 W et une durée de l'impulsion laser fixée à 100 ns. La cartographie des modèles de fautes matériels obtenus dans cette expérience est présentée sur la Figure 3.8. Cette cartographie est superposée à l'image IR de la face arrière du MCU SAMD21G18A sur la Figure 3.9.

Comme pour le MCU STM32F100RB, la quasi-totalité des fautes identifiées sur le mot chargé dans le registre `"r1"` correspondent à des fautes de type *bit set*. Ces fautes sont distribuées à l'intérieur de seize colonnes distinctes, indicées de c_0 à c_{15} sur la Figure 3.8.

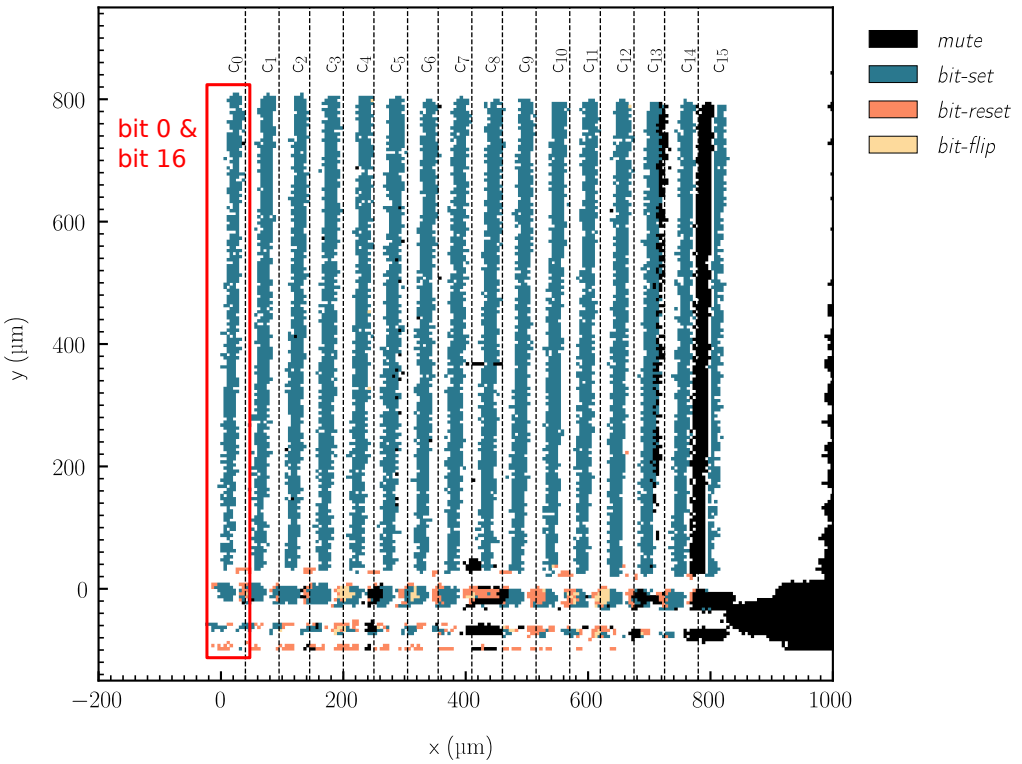


FIGURE 3.8 – Cartographie spatiale des fautes obtenues sur la mémoire NOR Flash du microcontrôleur SAMD21G18A (objectif : ×5 ; pas d'exploration : 5 µm ; puissance : 500 mW ; durée d'impulsion : 100 ns).

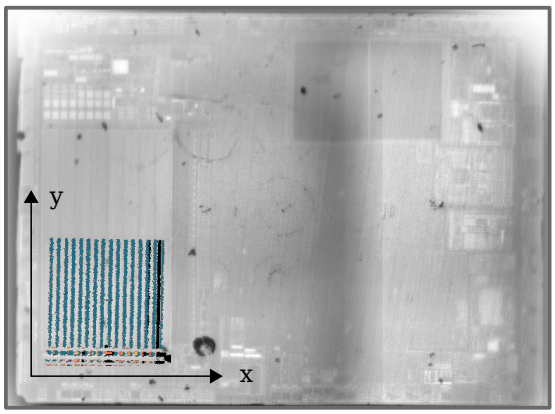


FIGURE 3.9 – Superposition de la cartographie spatiale sur l'image IR de la face arrière du MCU SAMD21G18A.

Ces colonnes recouvrent une partie de la surface du banc 0 de la mémoire Flash, où sont initialement mémorisés le programme de tests et le mot chargé dans le registre `"r1"`. Elles se répètent le long de l'axe x avec une période de 60 µm environ.

Chaque colonne est associée à l'injection d'une faute de type *bit set* sur deux bits du mot chargé dans le registre `"r1"`. Par exemple, lorsque la coordonnée x de l'objectif est fixée à 0 µm (colonne c_0), le bit d'indice 0 et le bit d'indice 16 sont mis à `'1'` dans le registre de 32 bits. Remarquons que les indices des deux bits sont congrus modulo 16. L'indice des bits mis à `'1'` est reporté sur la Figure 3.10 en fonction de la coordonnée x de l'objectif lorsque la coordonnée y est fixée à 400 µm. La position de la colonne c_0 est indiquée en rouge par une barre verticale.

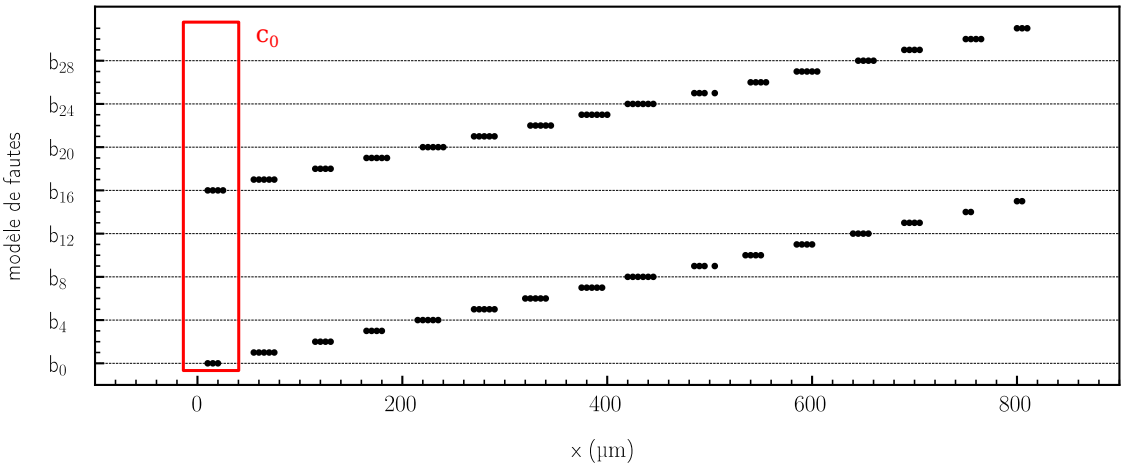


FIGURE 3.10 – Analyse de l'indice des bits mis à `'1'` sur l'ensemble des seize colonnes en fonction de la coordonnée x de l'objectif (objectif : ×5 ; pas d'exploration : 5 µm ; coordonnée y : 400 µm ; puissance : 500 mW ; durée d'impulsion : 100 ns).

Cette figure met en évidence une corrélation linéaire entre la position des colonnes suivant l'axe x et l'indice des paires de bits sur lequel une faute de type *bit set* est injectée. Si i désigne un entier strictement inférieur à 16, les bits d'indices i et $i+16$ d'un mot de 32 bits lu en mémoire Flash peuvent ainsi être forcés à `'1'` en positionnant l'objectif sur la colonne c_i. Ce modèle de faute permet à un attaquant d'injecter une faute identique sur deux mots de 16 bits consécutifs avec une seule impulsion laser.

Les trente-deux bits d'un mot enregistré dans la mémoire Flash d'un MCU 32 bits sont mémorisés dans trente-deux colonnes différentes. Nous supposons donc que les colonnes où sont mémorisés les couples de bits dont les indices sont congrus modulo 16 sont placées à proximité l'une de l'autre dans la mémoire Flash du MCU SAMD21G18A. Une LFI avec un spot large affecterait alors deux colonnes simultanément.

Des fautes de type *bit reset* et de type *bit flip* sont également identifiées sur la Figure 3.8 à l'extrémité inférieure de chacune des colonnes c_0 à c_{15}, à l'interface entre la mémoire Flash et le circuit. L'indice des deux bits modifiés à l'intérieur d'une colonne est identique pour les fautes de type *bit set* et pour les fautes de type *bit reset* et de type *bit flip*. Un attaquant peut donc manipuler précisément les bits d'un mot lu en mémoire Flash grâce à des fautes de type *bit set* et de type *bit reset* en positionnant l'objectif à l'interface entre la mémoire Flash et le circuit. Ces résultats peuvent ainsi être comparés

aux fautes obtenues par SAKAMOTO, FUJIMOTO et MATSUMOTO [146] en injectant un photocourant parasite par illumination laser sur les amplificateurs de détection de la mémoire NOR Flash d'un MCU 32 bits. Ils confortent donc l'hypothèse selon laquelle les colonnes des mémoires NOR Flash des MCU STM32F100RB et SAMD21G18A sont impliquées dans le mécanisme physique à l'origine des fautes observées.

3.2.1.C Synthèse des résultats

Les résultats obtenus sur les MCU STM32F100RB et SAMD21G18A confirment l'existence d'un mécanisme d'injection de fautes singulier dont la zone associée se confond avec la surface de la mémoire NOR Flash des circuits. Les fautes injectées dans cette zone possèdent en outre des propriétés similaires :

1. Les fautes sont injectées sur les mots lus en mémoire Flash au moment de leur lecture. Le contenu de la mémoire Flash n'est donc pas corrompu par une LFI.
2. Les fautes observées correspondent à un modèle de faute *bit set* sur un ou deux bits avec un spot laser de 20 µm de diamètre. Le mécanisme d'injection de fautes étudié permet donc d'injecter des fautes avec une grande précision sur des technologies 80 nm et 65 nm.
3. L'indice des bits peut être choisi en positionnant l'objectif sur les colonnes du tableau d'une mémoire NOR Flash.
4. Un déplacement de l'objectif perpendiculairement aux lignes du tableau, dans la direction des colonnes, n'a pas d'influence sur les propriétés du modèle de faute.

Les résultats obtenus sur le MCU STM32F100RB nous amènent enfin à questionner l'existence de trente-deux colonnes distinctes dans la mémoire NOR Flash du MCU SAMD21G18A, chacune associée à l'injection d'une faute de type *bit set* sur un bit d'indice choisi. Plus généralement, nous souhaitons évaluer la capacité d'un attaquant à obtenir des fautes avec une forte répétabilité sur chacun des bits d'un mot de 32 bits.

3.2.2 Étude de la répétabilité des fautes

Dans les paragraphes suivants, nous analysons l'influence de la position de l'objectif sur la répétabilité des fautes obtenues sur un ou deux bits avec différents objectifs. Les paramètres expérimentaux sont identiques à ceux des expériences décrites dans la section 3.2.1.

3.2.2.A Métriques

Pour un ensemble de paramètres d'injection fixés, la répétabilité d'un modèle de faute est définie par le ratio entre le nombre de fautes associées au modèle de faute et le nombre total de tentative réalisées afin d'injecter une faute. Nous introduisons donc deux métriques pour évaluer la capacité d'un attaquant à injecter une faute sur un bit d'indice choisi.

Métrique inclusive. Nous souhaitons analyser les effets d'une LFI bit par bit. Nous introduisons à cet effet la répétabilité b_i définie par le ratio entre le nombre n_i de fautes

de type *bit set* obtenues sur le bit d'indice i et le nombre total N de tentatives d'injection de fautes réalisées :

$$b_i = \frac{n_i}{\mathrm{N}} \tag{3.4}$$

Dans ce cas de figure, les fautes obtenues sur les bits d'indices autres que l'indice i n'ont pas d'influence sur la métrique b_{i}.

Métrique exclusive. Nous souhaitons également analyser la possibilité d'injecter une faute exclusivement sur un groupe de bits grâce à une LFI. Nous introduisons à cet effet la répétabilité $b_{i_1/i_2/\ldots/i_n}$ définie par le ratio :

$$b_{i_1/i_2/\ldots/i_n} = \frac{n_{i_1/i_2/\ldots/i_n}}{\mathrm{N}} \tag{3.5}$$

où le nombre $n_{i_1/i_2/\ldots/i_n}$ désigne le nombre de fautes obtenues simultanément sur les bits d'indices $i_1, i_2, \ldots, i_n$ et sur aucun autre bit, et le nombre N désigne le nombre total de tentatives d'injection de fautes réalisées. Par abus de notation, la quantité $b_{i/i/\ldots/i}$ désigne la répétabilité des fautes sur le seul bit d'indice i.

3.2.2.B Résultats pour le MCU STM32F100RB

Un balayage de la coordonnée x de l'objectif ×5 a été réalisé par pas de 1 µm sur la surface de la mémoire Flash du MCU STM32F100RB avec une coordonnée y fixée à 300 µm, un délai d'injection fixé à 1 340 ns, une puissance de la source laser fixée à 1 W et une durée de l'impulsion laser fixée à 50 ns. Pour chaque position, le programme de tests décrit dans la section 3.1.3 a été exécuté cent fois. Une tentative d'injection de fautes a été réalisée à chaque exécution. Ces conditions expérimentales sont identiques aux conditions fixées pour obtenir la Figure 3.7 qui décrit l'influence de la position de l'objectif sur l'indice des bits mis à `'1'`.

La Figure 3.11 décrit la répétabilité des fautes obtenues sur les bits d'indices 16 à 19 avec la métrique inclusive définie par l'équation 3.4. Chacun des bits d'indices 16 à 19 peut être mis à `'1'` de manière systématique dans une fenêtre de positions de l'objectif :

— la fenêtre définie par l'intervalle 781 µm – 804 µm pour le bit d'indice 16 ;

— la fenêtre définie par l'intervalle 824 µm – 847 µm pour le bit d'indice 17 ;

— la fenêtre définie par l'intervalle 870 µm – 891 µm pour le bit d'indice 18 ;

— la fenêtre définie par l'intervalle 914 µm – 935 µm pour le bit d'indice 19.

La largeur de ces fenêtres est estimée à 20 µm, soit environ le diamètre du spot laser produit dans le plan focal de l'objectif ×5. Le décalage entre le début d'une fenêtre et le début de la suivante est estimé à 45 µm. Il est donc aisé pour un attaquant de localiser les zones où un bit d'indice choisi peut être mis à `'1'` avec une répétabilité de 100 %.

3.2.2.C Résultats pour le MCU SAMD21G18A

L'expérience précédente a été réalisée avec l'objectif ×20 sur le MCU SAMD21G18A avec une coordonnée y fixée à 400 µm, un délai d'injection fixé à 750 ns, une puissance de la source laser fixée à 0.75 W et une durée de l'impulsion laser fixée à 100 ns. Ces

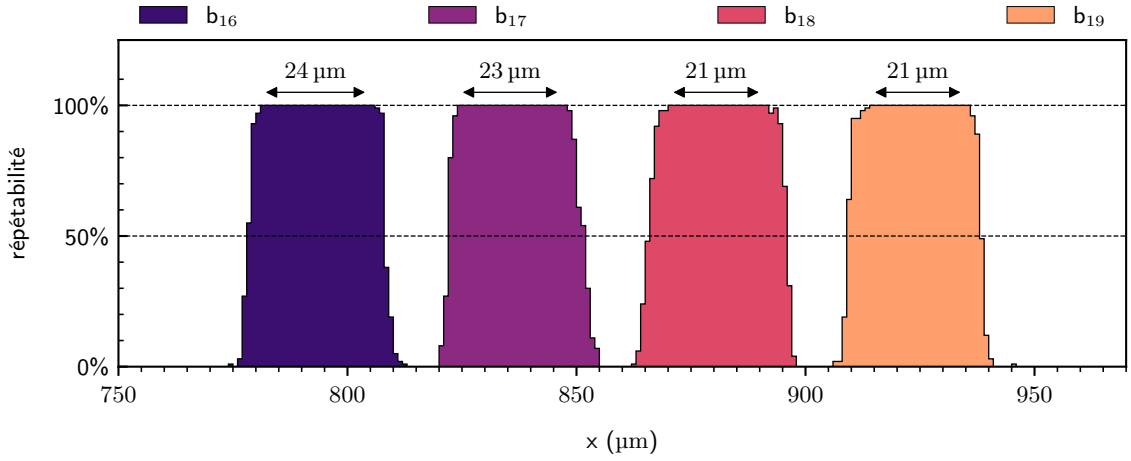


FIGURE 3.11 – Analyse de la répétabilité de l'injection de fautes de type *bit set* sur les bits d'indices 16 à 19 en fonction de la coordonnée x de l'objectif (objectif : ×5 ; pas d'exploration : 1 µm ; puissance : 1 000 mW ; durée d'impulsion : 50 ns).

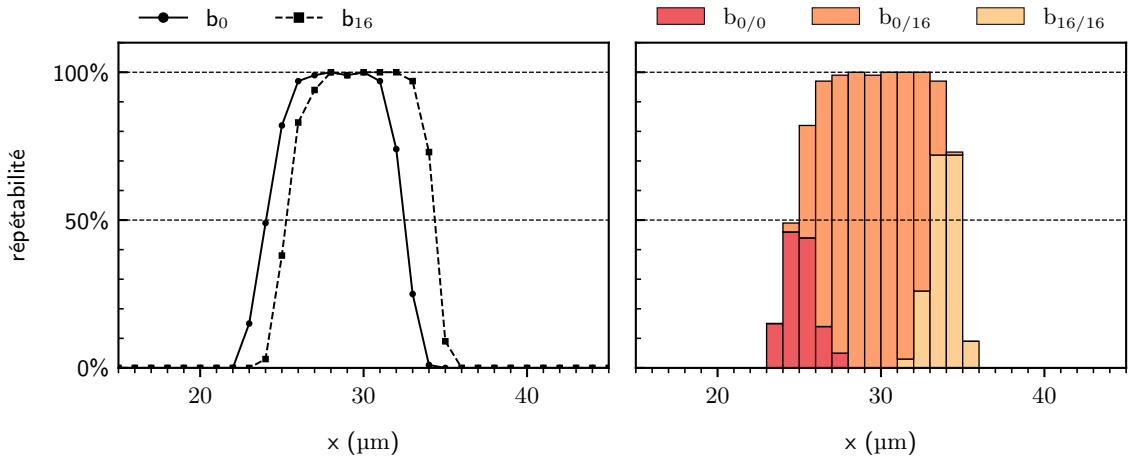


FIGURE 3.12 – Analyse de la répétabilité de l'injection de fautes de type *bit set* sur les bits d'indices 0 et 16 en fonction de la coordonnée x de l'objectif (objectif : ×20 ; pas d'exploration : 1 µm ; puissance : 750 mW ; durée d'impulsion : 100 ns).

conditions expérimentales sont identiques aux conditions fixées pour obtenir la Figure 3.10, qui décrit l'influence de la position de l'objectif sur l'indice des bits mis à `'1'`, à l'exception de la puissance de la source laser qui a été augmentée de 0.5 W à 0.75 W pour compenser la diminution du facteur de transmission entre l'objectif ×5 et l'objectif ×20. La Figure 3.11 décrit la répétabilité des fautes obtenues sur les bits d'indices 0 et 16 avec, à gauche, la métrique inclusive définie par l'équation 3.4 et, à droite, la métrique exclusive définie par l'équation 3.5.

Nous avons constaté dans la section 3.2.1.B que les bits d'indices 0 et 16 sont affectés simultanément par une LFI sur la colonne c_0. Ces résultats se retrouvent sur la figure de droite de la Figure 3.11 qui illustre notamment la répétabilité $b_{0/16}$ des fautes de type *bit set* injectées simultanément sur les bits d'indice 0 et d'indice 16. Ainsi, par exemple, pour une coordonnée x de l'objectif fixée à 30 µm les deux bits sont systématiquement mis à `'1'` par une LFI.

L'hypothèse selon laquelle ces fautes correspondent en fait à une perturbation simultanée de deux colonnes peut aussi être vérifiée. En effet, les fautes injectées sur le seul bit d'indice 0 (répétabilité $b_{0/0}$) sont majoritairement obtenues lorsque la coordonnée x de l'objectif est fixée au voisinage de 25 µm. Ainsi, sur les cent tentatives réalisées pour une coordonnée x de l'objectif fixée à 25 µm :

— 48 % des tentatives correspondent aux fautes obtenues sur le seul bit d'indice 0 ;

— 34 % des tentatives correspondent aux fautes obtenues sur les bits d'indices 0 et 16 simultanément ;

— 18 % des tentatives ont été réalisées sans observer de faute.

En revanche, les fautes injectées sur le seul bit d'indice 16 (répétabilité $b_{16/16}$) sont majoritairement obtenues lorsque la coordonnée x de l'objectif est fixée au voisinage de 34 µm. Sur les cent tentatives réalisées pour une coordonnée x de l'objectif fixée à 34 µm :

— 73 % des tentatives correspondent aux fautes obtenues sur le seul bit d'indice 16 ;

— 1 % des tentatives correspond aux fautes obtenue sur les bits d'indices 0 et 16 simultanément ;

— 26 % des tentatives ont été réalisées sans observer de faute.

Pour évaluer le décalage entre les deux colonnes à l'origine des fautes sur les bits d'indice 0 et d'indice 16, une analyse de la distribution des fautes est réalisée avec la métrique inclusive sur la figure de gauche de la Figure 3.11. Les distributions des bits d'indice 0 et d'indice 16 sont des distributions en cloche de largeur équivalente qui présentent un plateau en leur centre. Cette caractéristique se justifie notamment par la distribution gaussienne de l'intensité du faisceau laser dans le plan du circuit intégré [149]. La largeur des distributions est estimée à 8 µm, soit environ le diamètre du spot laser produit dans le plan focal de l'objectif ×20. Les deux distributions sont décalées d'environ 1.5 µm. Ce décalage est attribué au décalage spatial entre les deux colonnes à l'origine de l'injection d'une faute sur les bits d'indice 0 et d'indice 16.

Dès lors, s'il est possible d'injecter une faute de type *bit set* sur un unique bit d'un mot lu dans la mémoire Flash du MCU SAMD21G18A, ces résultats s'obtiennent toutefois au détriment de la répétabilité des fautes (50 % pour les fautes sur le seul bit d'indice 0 et 75 % pour celles sur le seul bit d'indice 16). Remarquons néanmoins qu'une amélioration de la précision de l'injection avec un spot laser de l'ordre du micromètre peut être envisagée pour améliorer la répétabilité du modèle de faute.

3.2.2.D Synthèse des résultats

Le modèle de faute *bit set* décrit dans les paragraphes précédents permet à un attaquant d'injecter une faute sur un bit d'indice choisi avec une grande précision et une forte répétabilité en positionnant l'objectif laser sur une colonne de la mémoire Flash d'un MCU.

Lorsque les colonnes sont espacées régulièrement sur tout la surface de la mémoire, comme c'est le cas pour le MCU STM32F100RB, une LFI permet de perturber individuellement chacune des trente-deux colonnes pendant la lecture d'un mot de 32 bits

avec une résolution faible comparée à la taille de gravure des transistors. Nous estimons qu'une résolution maximale de l'ordre de 45 µm est nécessaire pour injecter une faute sur un seul bit pour le MCU STM32F100RB fabriqué dans le technologie CMOS 80 nm.

Lorsque plusieurs colonnes sont placées à proximité l'une de l'autre, comme c'est le cas du MCU SAMD21G18A, une LFI permet de perturber ces colonnes simultanément. Une résolution élevée permet néanmoins de perturber individuellement chacune des trente-deux colonnes. Des fautes sur un seul bit ont ainsi été obtenues avec une répétabilité de plus de 50 % et un spot de 10 µm sur des colonnes décalées de 1.5 µm dans la mémoire Flash du MCU SAMD21G18A fabriqué dans le technologie CMOS 65 nm.

3.2.3 Influence de l'adresse de lecture

Les fautes analysées dans les sections précédentes ont été injectées pendant la lecture d'un mot de 32 bits à partir d'une adresse fixe dans la mémoire NOR Flash des deux circuits de tests. Les différences observées entre les deux mémoires sont attribuées aux différentes organisations physiques des mémoires. L'organisation d'une mémoire relie notamment le placement physique des colonnes et l'adresse physique des mots enregistrés. Or, cette dernière n'est pas documentée publiquement. Les paragraphes suivants analysent donc, dans un premier temps, l'influence de l'adresse de chargement sur les propriétés du modèle de faute *bit set* observé sur les colonnes de la mémoire afin de réaliser, dans un second temps, une rétro-ingénierie de l'organisation des deux mémoires NOR Flash étudiées. Ces résultats illustrent le rôle des colonnes mémoire sélectionnées dans le mécanisme d'injection de fautes observé.

3.2.3.A Résultats pour le MCU STM32F100RB

Un balayage par pas de 1 µm de la coordonnée x de l'objectif ×5 a été réalisé au-dessus de la surface de la mémoire Flash du MCU STM32F100RB avec une coordonnée y de l'objectif fixée à 300 µm, un délai d'injection fixé à 1 340 ns, une puissance de la source laser fixée à 0.75 W et une durée de l'impulsion laser fixée à 50 ns. Pour chaque position, une version modifiée du programme de tests décrit dans la section 3.1.3 a été exécutée. Cette version est détaillée succinctement dans le Code source 3.2.

```
1  .rept 16
2      nop
3  .endr
4      ldr     r1, [r0, r2]
5  .rept 16
6      nop
7  .endr
```

CODE SOURCE 3.2 – Chargement avec décalage d'un mot de 32 bits dans un tableau de mots.

Ce programme réalise un chargement du registre `"r1"` avec un mot enregistré dans un tableau dont l'adresse est mémorisée par le registre `"r0"`. La valeur du registre `"r2"`, passée en argument du programme de tests, détermine le décalage – exprimé en octet – entre l'adresse du tableau et l'adresse du mot chargé dans le registre `"r1"`. La valeur

de ce décalage est fixée par l'ordinateur de contrôle qui met à jour les arguments du programme de tests via la liaison UART du circuit de tests.

Pour chaque valeur explorée lors du balayage de la coordonnée x de l'objectif, le programme de tests décrit ci-dessus a été exécuté plusieurs fois pour charger chacun des mots d'un tableau de 1 ko. Ce tableau est initialisé à zéro dans la mémoire Flash du MCU STM32F100RB. Les fautes de type *bit set* obtenues sur les bits d'indices 18 (en bleu), 19 (en orange) et 20 (en gris) des mots lus sont reportées sur la Figure 3.13 en fonction de l'adresse des mots et de la coordonnée x de l'objectif. Dans un souci de lisibilité, l'adresse des mots est donnée relativement à l'origine du tableau.

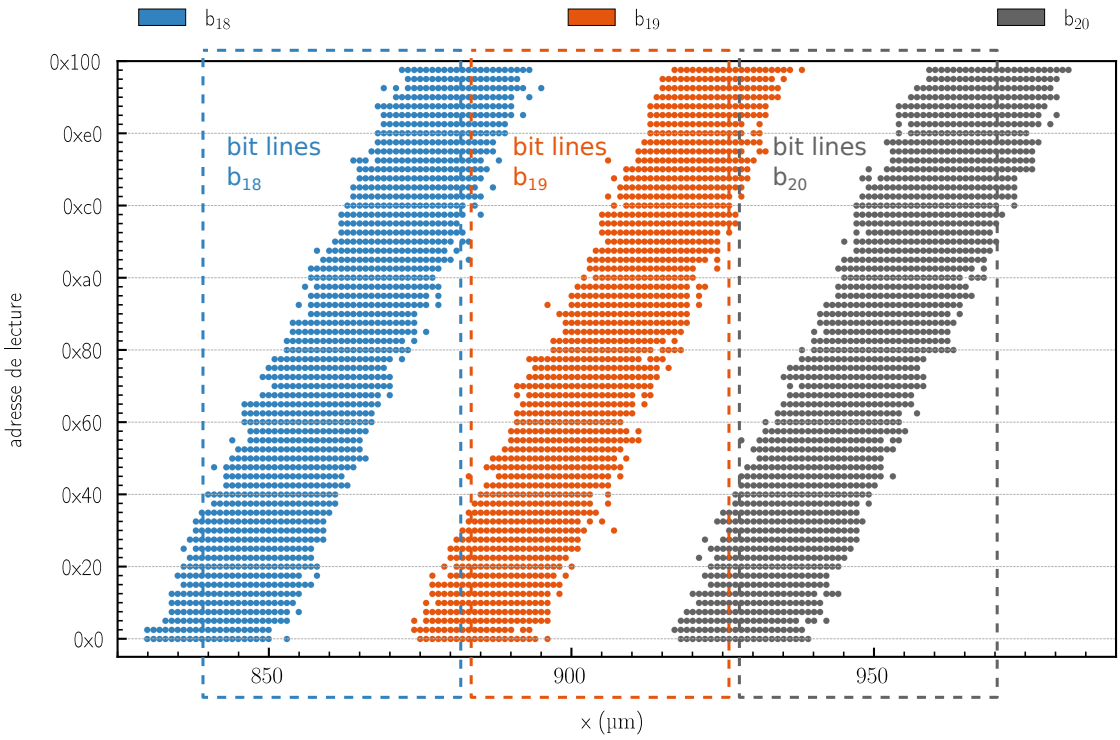


FIGURE 3.13 – Localisation; suivant l'axe x; des zones vulnérables associées aux bits d'indices 18 à 20 en fonction de l'adresse physique du mot lu dans la mémoire NOR Flash du microcontrôleur STM32F100RB (objectif : ×5; pas d'exploration : 1 µm; coordonnée y : 300 µm; puissance : 750 mW; durée d'impulsion : 50 ns).

Pour chaque adresse exlorée, les fautes injectées sur le bit d'indice i sont regroupées à l'intérieur d'un même intervalle de coordonnées x de l'objectif. Puisque la largeur d'une colonne est faible devant le diamètre du spot laser, nous pouvons supposer en première approximation que la colonne où est mémorisée le bit d'indice i est situées au centre de cet intervalle.

Pour l'adresse `"0x0"` par exemple, trois intervalles peuvent être oservés :

— l'intervalle 830 µm – 850 µm pour lequel une faute de type *bit set* est injectée sur le bit d'indice 18;

— l'intervalle 875 µm – 895 µm pour lequel une faute de type *bit set* est injectée sur le bit d'indice 19;

— l'intervalle 917 µm – 940 µm pour lequel une faute de type *bit set* est injectée sur le bit d'indice 20.

Nous pouvons donc supposé que les bits d'indices 18, 19 et 20 du mot lu à l'adresse `"0x0"` sont mémorisés dans des colonnes mémoire situées aux coordonnées x 840 µm, 885 µm et 928.5 µm respectivement.

Nous observons que la position des intervalles se décale dans le sens des valeurs croissantes de la coordonnée x de l'objectif à chaque incrément de l'adresse du mot lu en mémoire Flash. En notant (a_i, x_i) l'observation d'une faute de type *bit set* sur le bit d'indice i du mot de 32 bits lu à l'adresse a_i lorsque la coordonnée x de l'objectif est fixée à la valeur x_i, une régression linéaire sur l'ensemble des observations $(a_i^{(0)}, x_i^{(0)}), (a_i^{(1)}, x_i^{(1)}), \ldots, (a_i^{(n)}, x_i^{(n)})$ permet d'estimer la valeur de ce décalage à 0.7 µm. Nous supposons donc que les bits de même indice de deux mots consécutifs sont mémorisés dans deux colonnes adjacentes espacées de 0.7 µm.

Nous observons en outre que les bits de même indice sont mémorisés dans la même région. Les régions associées aux bits d'indices 18, 19 et 20 sont encadrées en bleu, en orange et en gris sur la Figure 3.13. La mémoire NOR Flash du MCU STM32F100RB est ainsi divisée en trente-deux régions.

Nous rapportons enfin que les résultats obtenus sur un bloc de 64 mots consécutifs sont identiques à ceux obtenus sur les 64 mots du bloc suivant. Nous supposons donc que les bits de même indice des mots dont les adresses sont congrus modulo `0x100` sont mémorisés dans la même colonne. Cette observation suggère que chaque ligne de la mémoire est composée de 64 mots de 32 bits, ou 2048 transistors à grille flottante. Puisque la mémoire NOR Flash du MCU STM32F100RB possède une capacité de 128 ko, nous en déduisons que cette dernière est divisée en 2048 colonnes et 512 lignes.

3.2.3.B Résultats pour le MCU SAMD21G18A

Un balayage par pas de 1 µm de la coordonnée x de l'objectif ×20 a été réalisé au-dessus de la surface du banc 0 de la mémoire NOR Flash du MCU SAMD21G18A avec une coordonnée y de l'objectif fixée à 400 µm, un délai d'injection fixé à 750 ns, une puissance de la source laser fixée à 0.4 W et une durée de l'impulsion laser fixée à 50 ns. Pour chaque position, le programme de tests décrit dans la section précédente a été exécuté plusieurs fois pour charger chacun des mots d'un tableau de 1 ko initialisé à zéro dans la mémoire Flash du MCU SAMD21G18A. Les fautes de type *bit set* obtenues sur les bits d'indices 1 (en bleu), 17 (en bleu clair), 2 (en orange), 18 (en orange clair), 3 (en gris) et 19 (en gris clair) des mots lus sont reportées sur la Figure 3.14 en fonction de l'adresse des mots et de la coordonnée x de l'objectif. Afin de mettre en évidence l'architecture 16 bits de la mémoire NOR Flash du microcontrôleur SAMD21, les bits d'indices 1, 2, 3, 17, 18 et 19 d'un mot de 32 bits mémorisé à l'adresse i sont représentés comme les bits d'indices 1, 2 et 3 d'un mot de 16 bits mémorisé à l'adresse i et les bits d'indices 1, 2 et 3 d'un mot de 16 bits mémorisé à l'adresse $i + 2$. Dans un souci de lisibilité, l'adresse des mots est donnée relativement à l'origine du tableau.

Comme pour le MCU STM32F100RB, nous constatons tout d'abord que les positions

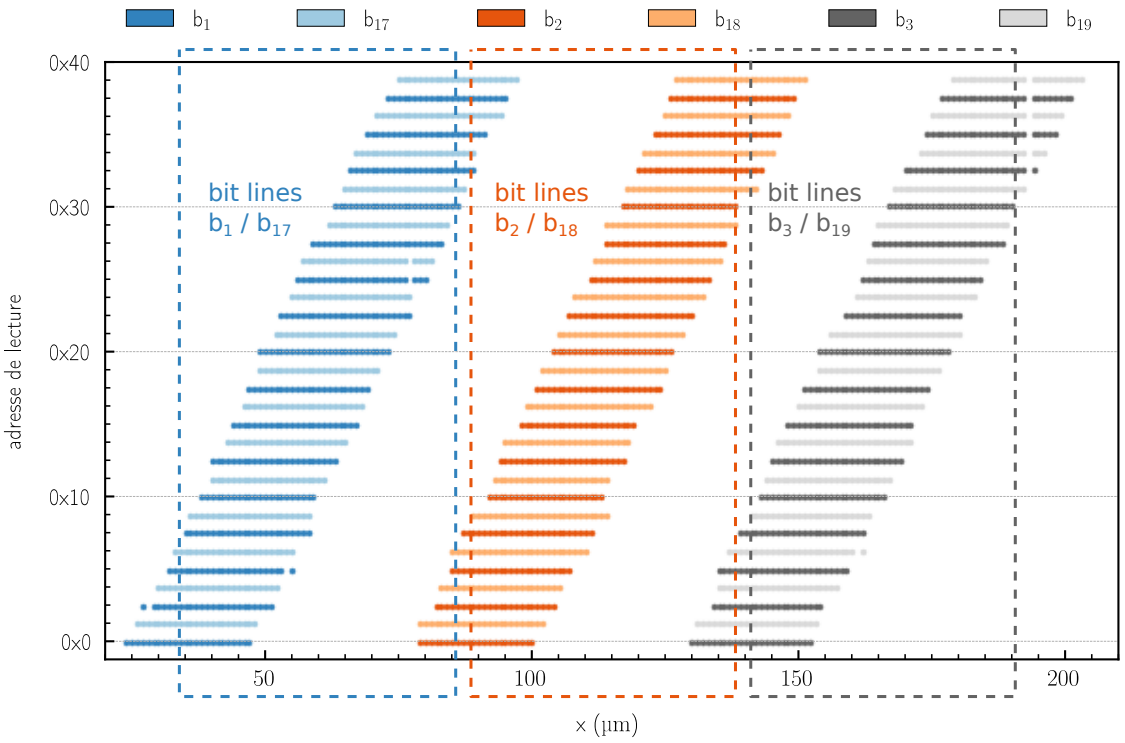


FIGURE 3.14 – Analyse de l'indice des bits mis à '1' en fonction de l'adresse physique du mot lu dans la mémoire NOR Flash du microcontrôleur SAMD21 et de la coordonnée x de l'objectif (objectif : ×20 ; pas d'exploration : 1 µm ; coordonnée y : 400 µm ; puissance : 400 mW : durée d'impulsion : 100 ns).

des colonnes dépendent de l'adresse du mot lu. En effet, ces positions se décalent dans le sens des valeurs croissantes de la coordonnée x de l'objectif à chaque incrément de l'adresse de lecture. En notant (a_i, x_i) l'observation d'une faute de type *bit set* sur le bit d'indice i du mot de 16 bits lu à l'adresse a_i lorsque la coordonnée x de l'objectif est fixée à la valeur x_i, une régression linéaire sur l'ensemble des observations $(a_i^{(0)}, x_i^{(0)}), (a_i^{(1)}, x_i^{(1)}), \ldots, (a_i^{(n)}, x_i^{(n)})$ permet d'estimer à 1.6 µm la valeur du décalage de la position entre deux colonnes adjacentes. Notons que cette valeur correspond à la valeur déjà calculée dans la section 3.2.2.C.

Nous constatons par ailleurs que les colonnes associées à l'injection d'une faute sur un bit d'indice donné dans un mot de 16 bits sont regroupées dans la même région. Les régions associées aux bits d'indices 1, 2 et 3 dans un mot de 16 bits sont respectivement encadrées en bleu, en orange et en gris sur la Figure 3.14. Le banc 0 de la mémoire NOR Flash du MCU SAMD21G18A est donc divisée en 16 régions.

Nous rapportons enfin que la position des colonnes est identique pour les mots lus à l'adresse i et à l'adresse $i + 0\text{x}40$. Chaque ligne de la mémoire est donc composée de 16 mots de 32 bits, soit 512 transistors à grille flottante. Puisque le banc 0 de la mémoire NOR Flash du MCU SAMD21G18A possède une capacité de 128 ko, nous en déduisons que ce dernier est divisé en 512 colonnes et 2048 lignes.

3.2.3.C Rétro-ingéniérie de l'organisation des mémoires Flash

Dans une architecture NOR Flash, les transistors à grille flottante sont organisés en lignes et en colonnes à l'intérieur d'un tableau. Les grilles des transistors à grille flottante d'une ligne sont reliées à une même ligne de mots, aussi appelée *word line*, qui permet de sélectionner simultanément tous les mots mémorisés dans la ligne. Les drains des transistors à grille flottante d'une colonne sont reliés à une ligne de bits commune, aussi appelée *bit line*, qui permet de lire le transistor sélectionné dans une colonne en connectant son drain à un amplificateur de détection (*Sense Amplifier* – SA). Les sources des transistors à grille flottante sont connectées ensemble à une ligne de sources qui est connectée à la masse pendant une opération de lecture. La lecture d'un mot de 32 bits nécessite ainsi de sélectionner une *word line* et trente-deux *bit lines*.

Microcontrôleur STM32F100RB. Les observations réalisées dans la section 3.2.3.A sont utilisées pour reconstituer l'organisation physique de la mémoire NOR Flash du MCU STM32F100RB. Cette reconstitution est présentée sur la Figure 3.15. Le repère utilisé dans nos expérimentations y figure en noir. Les lignes de sources ne sont pas représentées pour simplifier la réprésentation.

Les bits d'indice 18 b_{18} et d'indice 19 b_{19} d'un mot de 32 bits enregistré à l'adresse 0 dans la mémoire Flash sont mémorisés par deux transistors à grille flottante situés à l'intersection entre la *word line* WL_0 (en rouge) et les *bit lines* BL_{1152} et BL_{1216} (en bleu). La lecture du courant de ces deux transistors permet aux amplificateurs SA_{18} et SA_{19} de détecter la valeur des deux bits. Le bit d'indice 18 et d'indice 19 d'un mot de 32 bits enregistré à l'adresse suivante, c'est-à-dire l'adresse 4, sont mémorisés par deux transistors à grille flottante situés à l'intersection entre la *word line* WL_0 et les

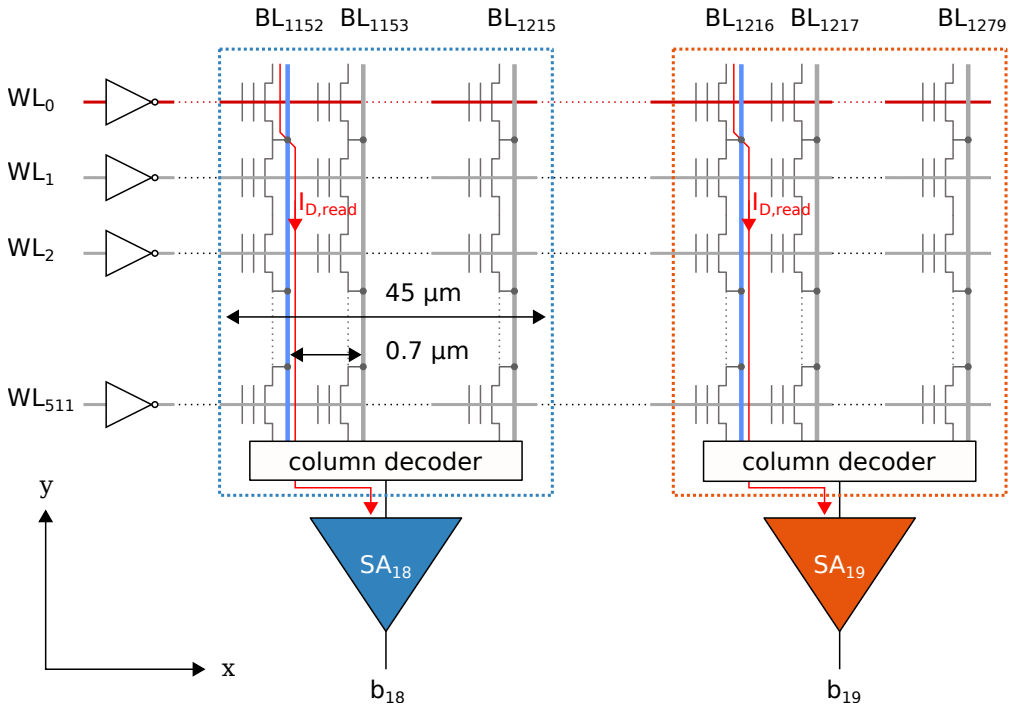


FIGURE 3.15 – Organisation de la mémoire NOR Flash du MCU STM32F100RB.

bit lines BL_{1153} et BL_{1217}. Un incrément de l'adresse de lecture décale ainsi les *bit lines* sélectionnées dans le sens des valeurs croissantes de la coordonnée x. Les mots dont les adresses sont congrues modulo `0x100` sont mémorisés par ailleurs dans les mêmes colonnes à des lignes différentes. Ces caractéristiques expliquent, d'une part, le décalage des zones de sensibilité avec l'adresse de lecture pour le modèle de faute *bit set* étudié et, d'autre part, la périodicité des résultats obtenus sur une plage de 64 mots. Les régions de la mémoire où sont mémorisées les bits de même indice sont encadrées en pointillé bleu pour les bits d'indice 18 et en pointillé orange pour les bits d'indice 19.

Microcontrôleur SAMD21G18A. Les observations réalisées dans la section 3.2.3.B sont utilisées pour reconstituer l'organisation physique du banc 0 de la mémoire NOR Flash du MCU SAMD21G18A. Cette reconstitution est présentée sur la Figure 3.16. Le repère utilisé dans nos expérimentations y figure en noir. Les lignes de sources ne sont pas représentées pour simplifier la réprésentation.

Les bits d'indice 1 b_1, d'indice 17 b_{17}, d'indice 2 b_2 et d'indice 18 b_{18} d'un mot de 32 bits enregistré à l'adresse 0 dans la mémoire Flash sont mémorisés par quatre transistors à grille flottante situés à l'intersection entre la *word line* WL_0 (en rouge) et les *bit lines* BL_{32}, BL_{33}, BL_{64} et BL_{65} (en bleu). La lecture du courant de ces quatre transistors permet aux amplificateurs de détecter la valeur des quatre bits. Comme pour le MCU STM32-F100RB, un incrément de l'adresse de lecture décale les *bit lines* sélectionnées dans le sens des valeurs croissantes de la coordonnée x. Les mots dont les adresses sont congrues modulo `0x40` sont mémorisés dans les mêmes colonnes, mais à des lignes différentes. Les régions de la mémoire où sont mémorisés les bits dont les indices sont

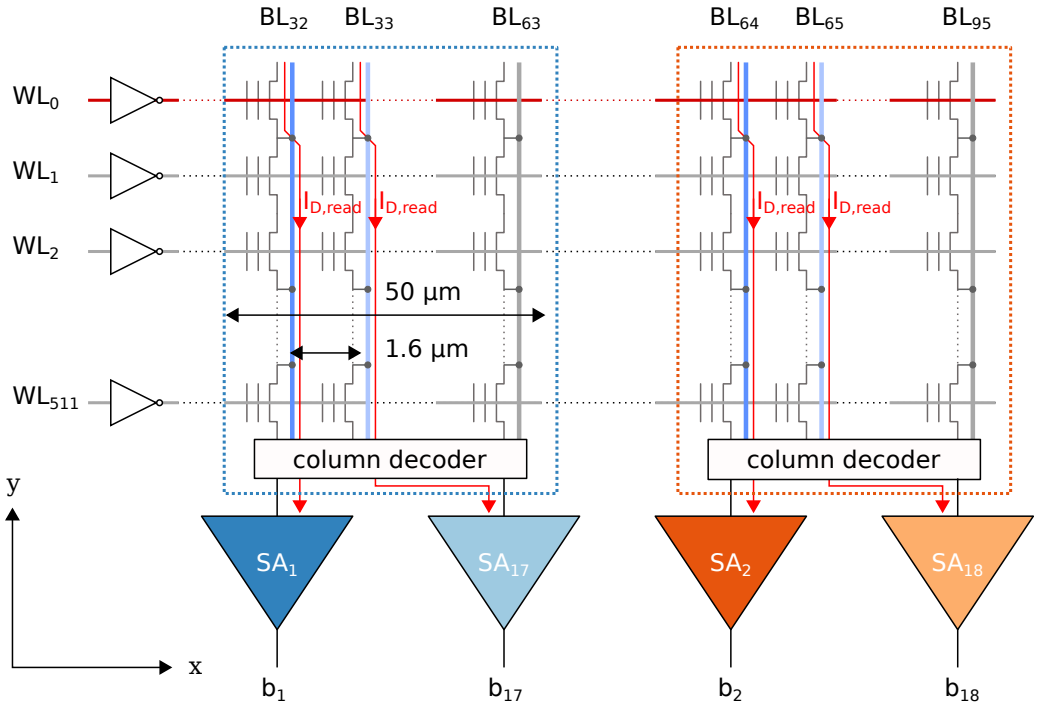


FIGURE 3.16 – Organisation de la mémoire NOR Flash du MCU SAMD21G18A.

congrus modulo 16 sont encadrées en pointillé bleu pour les bits d'indice 1 et 17 et en pointillé orange pour les bits d'indice 2 et 18. L'alternance entre les colonnes dédiées à la mémorisation des bits des demi-mots de poids faible et celles dédiées à la mémorisation des bits des demi-mots de poids fort explique les différences entre les résultats obtenus pour le MCU STM32F100RB et le MCU SAMD21G18A.

3.2.4 Influence de la position du plan focal de l'objectif

La largeur des fenêtres spatiales de sensibilité observées dans les sections précédentes correspond au diamètre du spot laser utilisé pour injecter une faute sur une colonne dans la mémoire Flash des MCU. L'espacement des colonnes sélectionnées pendant une opération de lecture permet ainsi de perturber une ou deux colonnes simultanément avec un spot laser de 10 µm ou 20 µm, en fonction de l'organisation de la mémoire testée. Dans les paragraphes suivants, nous étudions la possibilité de perturber plusieurs colonnes mémoire simultanément avec un spot très large en ajustant la coordonnée z de l'objectif.

3.2.4.A Évolution du diamètre du spot laser hors du plan focal

L'intensité I du faisceau dans un plan orthogonal à la direction de propagation du faisceau suit une distribution de la forme [38] :

$$\forall r \in \mathbb{R}_+, \quad \mathrm{I}(r) = \mathrm{I}_0 \left(\frac{\omega_0}{\omega(z)} \right) e^{-\frac{2r^2}{\omega(z)^2}} \tag{3.6}$$

où le paramètre $\omega(z)$ détermine la dispersion de la distribution gaussienne en dehors du plan focal et z la distance entre le plan focal et le plan de mesures considéré. L'évolution du paramètre ω avec la distance au plan focal satisfait l'équation [38] :

$$\omega(z) = \omega_0 \sqrt{1 + \left(\frac{z}{z_R}\right)^2} \tag{3.7}$$

La quantité z_R représente la distance de Rayleigh du faisceau dans le silicium, définie par :

$$z_R = \frac{\pi \eta \omega_0^2}{\lambda} \tag{3.8}$$

où η est l'indice de réfraction de la lumière dans le silicium.

L'équation 3.2 et l'équation 3.8 peuvent être combinées pour exprimer la distance de Rayleigh en fonction de l'ouverture numérique NA de l'objectif :

$$z_R = \frac{4\eta\lambda}{\pi \cdot \mathrm{NA}^2} \tag{3.9}$$

Pour un indice de réfraction de 3.5 dans le silicium et une longueur d'onde de 1 064 nm, la distance de Rayleigh associée à l'objectif ×5 peut ainsi être estimée à 3 mm.

Le diamètre d du spot FWHM à une distance z du plan focal de l'objectif peut finalement être relié au diamètre d_0 du spot FWHM dans le plan focal de l'objectif en combinant l'équation 3.3 et l'équation 3.7 suivant l'équation :

$$d = d_0 \sqrt{1 + \left(\frac{z}{z_R}\right)^2} \tag{3.10}$$

À une distance de 5 mm du plan focal, le diamètre du spot FWHM produit par l'objectif ×5 peut ainsi être estimé théoriquement à 39 µm.

3.2.4.B Analyse expérimentale sur le MCU STM32F100RB

Un balayage par pas de 1 µm de la coordonnée x de l'objectif ×5 a été réalisé au-dessus de la région de la mémoire Flash du MCU STM32F100RB associée à la mémorisation des bits d'indice 25 pour plusieurs valeurs de la coordonnée z de l'objectif et de la puissance de la source laser avec une coordonnée y fixée à 300 µm, un délai d'injection fixé à 1 340 ns et une durée de l'impulsion laser fixée à 135 ns. Une tentative d'injection d'une faute de type *bit set* sur le bit d'indice 25 d'un mot de 32 bits lu en mémoire Flash a été réitérée vingt fois. Le programme de tests utilisé à cet effet est décrit dans la section 3.1.3.

Les valeurs minimales de la puissance de la source laser pour lesquelles une faute de type *bit set* a été observée sur le bit d'indice 25 avec une répétabilité supérieure à 50 % sont reportées sur la Figure 3.17. Nous appelons « seuil d'injection » la courbe obtenue en reliant les points où ces minimums sont obtenus. La coordonnée z de l'objectif est calculée relativement au point où le contraste de l'image IR de la face arrière du circuit

est maximal. Le spot laser est alors focalisé sous les niveaux de métallisation du circuit de tests. Le plan focal de l'objectif s'éloigne des niveaux de métallisation vers l'intérieur du substrat dans le sens des valeurs croissantes de la coordonnée z.

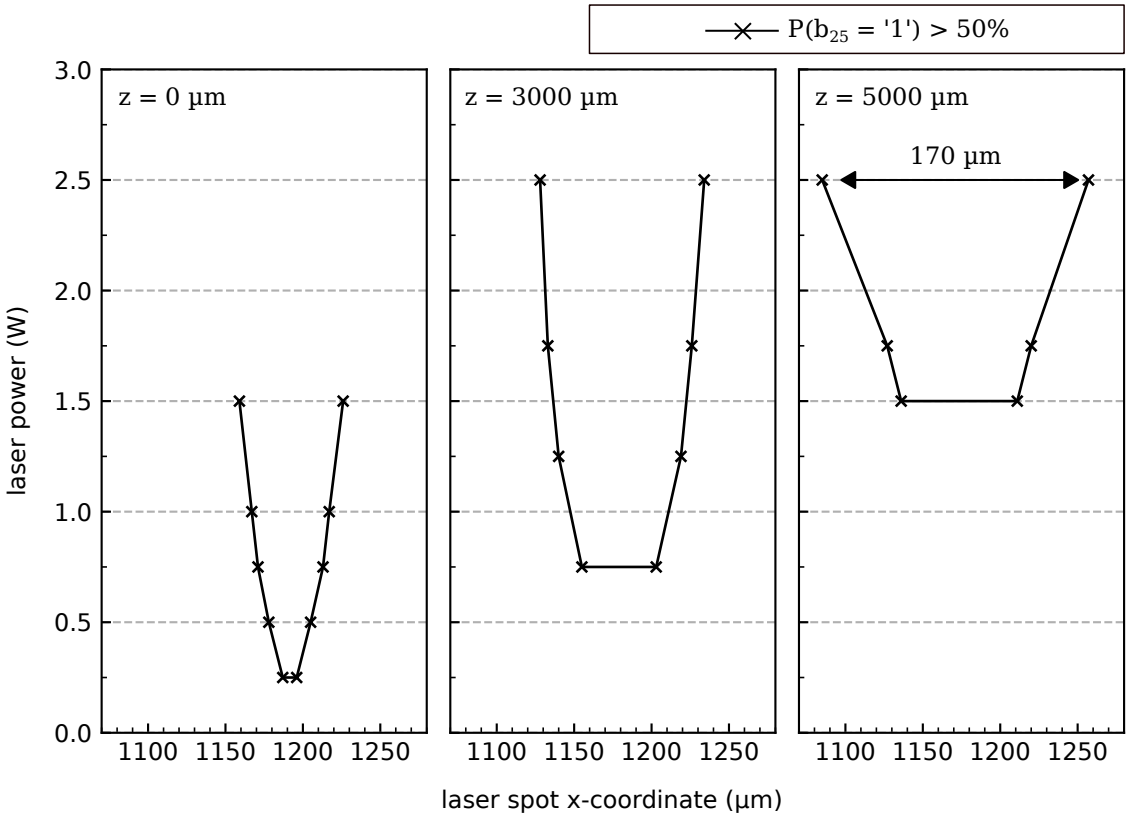


FIGURE 3.17 – Seuil d'injection des fautes de type *bit set* sur le bit d'indice 25 pour trois valeurs de la coordonnée z de l'objectif.

La fenêtre de coordonnée x de l'objectif à l'intérieur de laquelle une faute de type *bit set* peut être injectée sur le bit d'indice 25 du mot lu en mémoire Flash dépend, en premier lieu, de la puissance de la source laser. Par exemple, pour une coordonnée z de l'objectif fixée à 0 µm, la largeur de cette fenêtre est de 25 µm pour une puissance de 0.5 W et de 50 µm pour une puissance de 1 W. Or, ces valeurs sont supérieures aux valeurs obtenues dans les sections précédentes avec une durée de l'impulsion laser de 50 ns. La taille de la zone d'effet d'une impulsion laser sur la colonne qui mémorise le bit d'indice 25 du mot lu en mémoire Flash augmente ainsi avec la puissance de la source laser et la durée de l'impulsion.

Cependant, cette augmentation n'est pas linéaire et une augmentation conséquente de la puissance de la source laser est nécessaire pour obtenir un spot laser large, au risque d'endommager les fonctionnalités du circuit. Une alternative consiste alors à moduler la taille du spot laser en éloignant le plan focal de l'objectif des zones de diffusion du circuit de tests. Les seuils d'injection obtenus pour trois valeurs de la coordonnée z de l'objectif, à savoir 0 µm, 3 000 mm et 5 000 mm, permettent d'évaluer l'augmentation de la taille de la zone d'effet d'une impulsion laser sur une colonne mémoire en fonction de la position du plan focal de l'objectif. Ainsi, pour une coordonnée z de l'objectif fixée à 5 000 µm et une puissance de la source laser de 2.5 W la largeur de cette fenêtre est de

170 µm.

Remarquons dans un premier temps que ce résultat dépasse largement les estimations théoriques réalisées dans les paragraphes précédents. Ces observations peuvent être expliquées par un phénomène de migration des porteurs de charge générés par une impulsion laser dans le substrat, à distance des zones actives du circuit de tests. Remarquons dans un second temps que la puissance minimale requise pour injecter une faute augmente avec la coordonnée z de l'objectif. Cette caractéristique limite ainsi la taille de spot maximale qu'il est possible d'obtenir avec notre dispositif d'injection sur le circuit testé.

Puisque les colonnes associées à la mémorisation des bits de même indice sont regroupées dans une région d'environ 45 µm de large dans la mémoire Flash du MCU STM32F100RB, une zone d'effet de 170 µm de large doit permettre d'injecter une faute de type *bit set* sur trois à cinq bits simultanément. Cette caractéristique est illustrée sur la Figure 3.18 qui indique la répétabilité des fautes obtenues sur les bits d'indices 20 à 29 d'un mot de 32 bits lu en mémoire Flash en fonction de la coordonnée x de l'objectif avec une puissance de la source laser fixée à 2.5 W et une coordonnée z de l'objectif fixée à 5 000 µm. Pour une coordonnée x de l'objectif fixée à 1 085 µm, une faute de type *bit set* est injectée simultanément sur les bits d'indices 21 à 25 dans la moitié des tentatives réalisées. Pour une coordonnée x de l'objectif fixée à 1 200 µm, une faute de type *bit set* est injectée simultanément sur les bits d'indices 24 à 27 avec une répétabilité de 100 %.

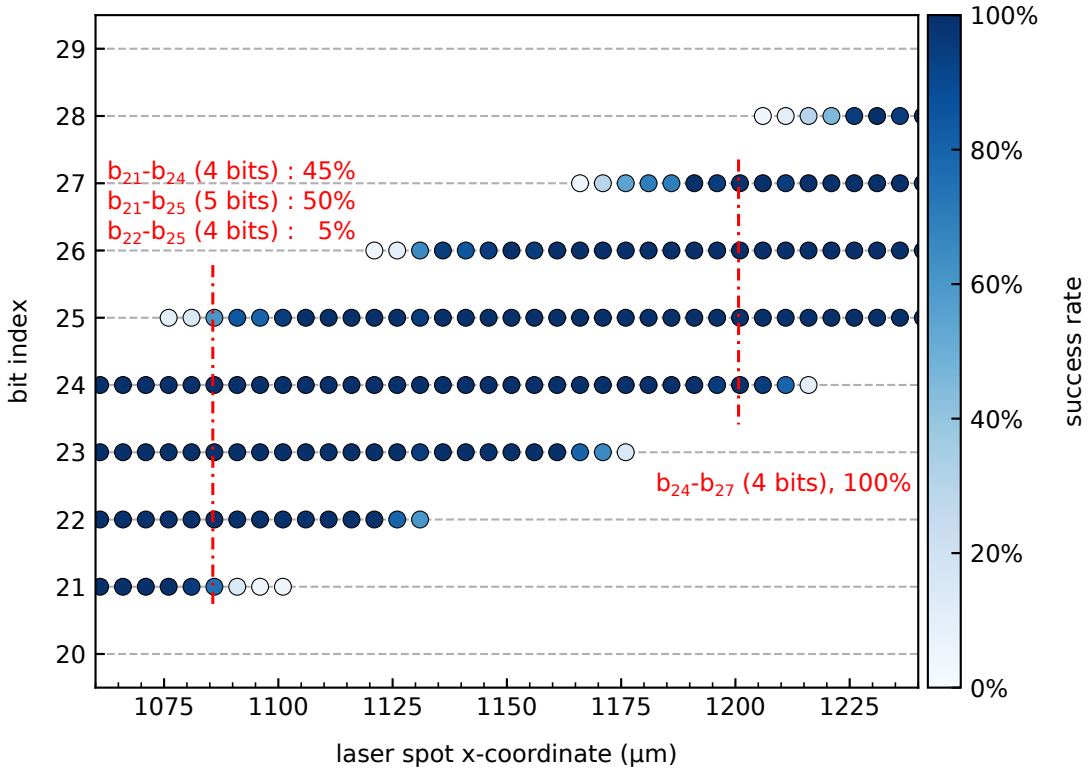


FIGURE 3.18 – Injection d'une faute de type *bit set* sur plusieurs bits adjacents en ajustant la coordonnée x de l'objectif.

Ces résultats démontrent ainsi que le modèle de faute *bit set* étudié dans ces travaux n'est

pas limité à la corruption d'un ou deux bits. En ajustant précisément la taille de la zone d'effet d'une perturbation laser, jusqu'à cinq bits ont été corrompus simultanément avec notre dispositif d'injection. Ce modèle de faute questionne l'utilisation des ECC, dont le dimensionnement suppose une connaissance du nombre maximal de fautes pouvant être corrigées, afin de contrer l'injection de fautes par impulsion laser sur les colonnes d'une mémoire NOR Flash.

3.3 Analyse temporelle des modèles de fautes

Le modèle de faute *bit set* décrit dans la section précédente traduit l'injection d'un photocourant dans les colonnes sélectionnées d'une mémoire NOR Flash lors de la lecture d'un mot dans la mémoire. Puisque la perturbation produite par une impulsion laser est de nature transitoire, cette dernière doit être précisément synchronisée sur une opération de lecture en mémoire Flash. Cette section analyse l'influence des paramètres d'une LFI sur la synchronisation temporelle d'une perturbation pendant une série de chargements de données à partir de la mémoire NOR Flash des MCU STM32F100RB et SAMD21G18A.

3.3.1 Influence de l'instant d'injection

Le programme de tests présenté dans la section 3.1.3 a été adapté pour charger consécutivement plusieurs mots de la mémoire Flash aux GPR des circuits de tests. La version modifiée du programme est détaillée succinctement dans le Code source 3.3.

```
1  .rept 16
2      nop
3  .endr
4      ldm     r0, {r1-r7}
5  .rept 16
6      nop
7  .endr
```

CODE SOURCE 3.3 – Chargement consécutifs des registres `"r1"` à `"r7"` avec sept mots de 32 bits consécutifs.

Ce programme réalise un chargement des registres `"r1"` à `"r7"` avec les sept mots de 32 bits d'un tableau dont l'adresse est mémorisée par le registre `"r0"`. Un programme similaire est utilisé dans la section 2.3 pour analyser les propriétés temporelles des modèles de fautes obtenus grâce à une impulsion EM sur un MCU 32 bits.

Le programme génère un signal de déclenchement, ou signal de *trigger*, à destination du banc d'injection laser au début de son exécution. Un balayage du délai d'injection entre le signal de déclenchement et la génération d'une impulsion EM a été réalisé sur les MCU STM32F100RB et SAMD21G18A. Les paragraphes suivants détaillent les résultats obtenus.

3.3.1.A Résultats pour le MCU STM32F100RB

Dans cette expérience, un balayage du délai d'injection a été réalisé par pas de 5 ns en répétant cent fois le chargement de sept mots consécutifs dans les registres `"r1"` à `"r7"` sur le MCU STM32F100RB grâce au programme décrit par le Code source 3.3. Les coordonnées x et y de l'objectif ×5 sont fixées à 320 µm et 300 µm [1], la puissance de la source laser à 1 W et la durée de l'impulsion laser à 50 ns. La répétabilité des fautes de type *bit set* obtenues sur le bit d'indice 7 de chacun des sept mots de 32 bits transférés dans les registres `"r1"` à `"r7"` est reportée sur la Figure 3.19 en fonction du délai d'injection.

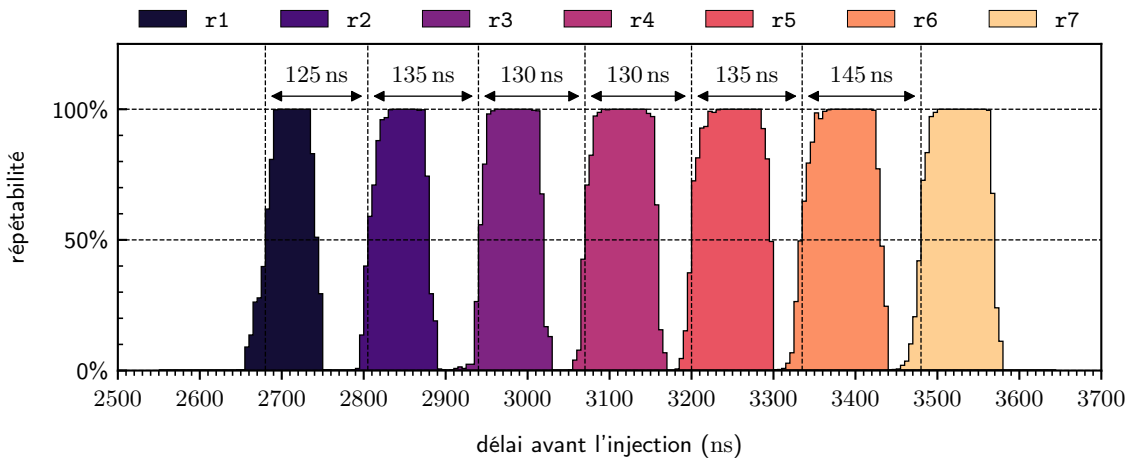


FIGURE 3.19 – Répétabilité des fautes de type *bit set* injectées sur le bit d'indice 7 pendant le chargement des registres `"r1"` à `"r7"` en fonction de l'instant d'injection pour le microcontrôleur STM32F100RB (répétition : 100 ; objectif : ×5 ; puissance : 1 000 mW ; durée d'impulsion : 50 ns ; pas d'exploration : 5 ns.

Pour chacun des sept registres, il existe une fenêtre temporelle où une faute de type *bit set* peut être injectée sur le bit d'indice 7 avec une répétabilité de 100 %. La largeur de ces fenêtres est légèrement inférieure à la durée d'une impulsion laser. Elle est estimée à 40 ns. L'intervalle entre deux fenêtres consécutives est estimé à 136 ns en moyenne, soit environ une période d'horloge (135 ns) du MCU STM32F100RB cadencé à 7.37 MHz. Cette période correspond également à la période des opérations de lecture en Flash lors de l'exécution de l'instruction `"ldm r0, {r1-r7}"` qui provoque la lecture d'un mot de 32 bits et son transfert entre la mémoire Flash et les GPR du circuit de tests à chaque cycle d'horloge. Un attaquant peut ainsi choisir d'injecter une faute de type *bit set* sur un seul mot de 32 bits en synchronisant une impulsion laser de durée inférieure à la période de fonctionnement de la mémoire Flash sur la lecture de ce mot en mémoire.

1. Le circuit de tests STM32F100RB a été démonté et remonté sur le dispositif d'injection laser entre les expériences décrites dans la section 3.2 et les expériences décrites dans les sections qui suivent. Nous estimons le décalage entre les repères utilisés à 80 µm dans la direction de la coordonnée x de l'objectif.

3.3.1.B Résultats pour le MCU SAMD21G18A

L'expérience précédente a été réitérée sur le MCU SAMD21G18A en fixant les coordonnée x et y de l'objectif ×5 à 140 µm et 300 µm [2], la puissance de la source laser à 1 W et la durée de l'impulsion laser à 50 ns. L'indice des mots pour lesquels une faute de type *bit set* a été injectée sur les bits d'indices 2 et 18 avec une répétabilité de 100 % est reporté sur la Figure 3.19 en fonction du délai d'injection.

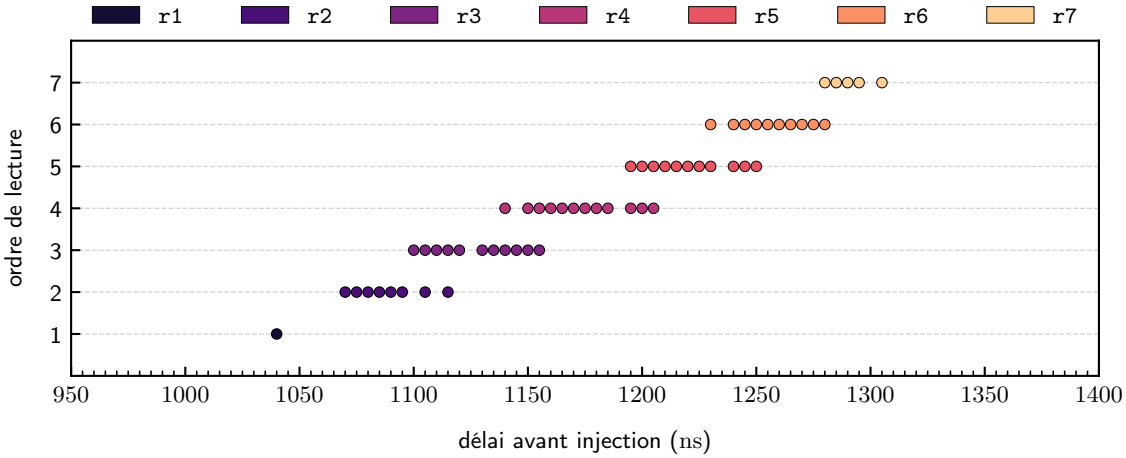


FIGURE 3.20 – Localisation temporelle des fautes de type *bit set* injectées simultanément sur les bits d'indice 2 et 18 avec une répétabilité de 100 % pendant le chargement des registres `"r1"` à `"r7"` pour le microcontrôleur SAMD21G18A. (répétition : 25 ; objectif : ×5 ; puissance : 1 000 mW ; durée d'impulsion : 50 ns ; pas d'exploration : 5 ns.

Comme pour le MCU STM32F100RB, l'injection d'une faute de type *bit set* pendant la lecture d'un mot en mémoire n'est possible qu'à l'intérieur d'une fenêtre temporelle dont la largeur est équivalente à la durée d'une impulsion laser. En revanche, les fenêtres consécutives se recouvrent partiellement. Par exemple, pour un délai d'injection fixé à 1 200 ns, les bits d'indice 2 et 18 des mots chargés dans les registres `"r4"` et `"r5"` sont systématiquement mis à `'1'`.

La période entre deux fenêtres consécutives est estimée à 40 ns, soit deux périodes d'horloge du circuit cadencé à 48 MHz (T_{clk} = 21 ns). Cette période correspond à la période théorique des opérations de lecture provoquées par l'instruction de chargement multiple lors du chargement des registres `"r1"` à `"r7"`. En effet, le constructeur Microchip Technology Inc. préconise de configurer le contrôleur de la mémoire Flash du MCU SAMD21G18A pour insérer un cycle de latence sur le bus de données entre chaque opération de lecture lorsque ce dernier fonctionne à une fréquence de 48 MHz [117]. Le chargement de sept mots consécutifs provoque donc une opération de lecture tous les deux cycles d'horloge. Un attaquant peut ainsi choisir d'injecter une faute de type *bit set* sur deux mots de 32 bits lus séquentiellement en mémoire en synchronisant une impulsion laser d'une durée supérieure à la période de fonctionnement de la mémoire Flash sur la lecture de ces deux mots. Dans la section suivante, nous étudions donc la

2. Le circuit de tests STM32F100RB a été démonté et remonté sur le dispositif d'injection laser entre les expériences décrites dans la section 3.2 et les expériences décrites dans les sections qui suivent. Nous estimons le décalage entre les repères utilisés à 20 µm dans la direction de la coordonnée x de l'objectif.

possibilité d'injecter une faute de type *bit set* sur plus de deux chargements consécutifs en augmentant la durée de l'impulsion laser.

3.3.2 Influence de la durée de l'impulsion laser

L'expérience décrite dans la section précédente a été réitérée sur le MCU SAMD21-G18A en fixant le délai d'injection à 1 000 ns avant la fenêtre temporelle de sensibilité associée au chargement du registre `"r1"` et en augmentant progressivement la durée de l'impulsion laser. Les fautes de type *bit set* obtenues sur les bits d'indice 2 et 18 avec une répétabilité de 100 % pendant le chargement des registres `"r1"` à `"r7"` sont reportées sur la Figure 3.21 en fonction de la durée de l'impulsion laser.

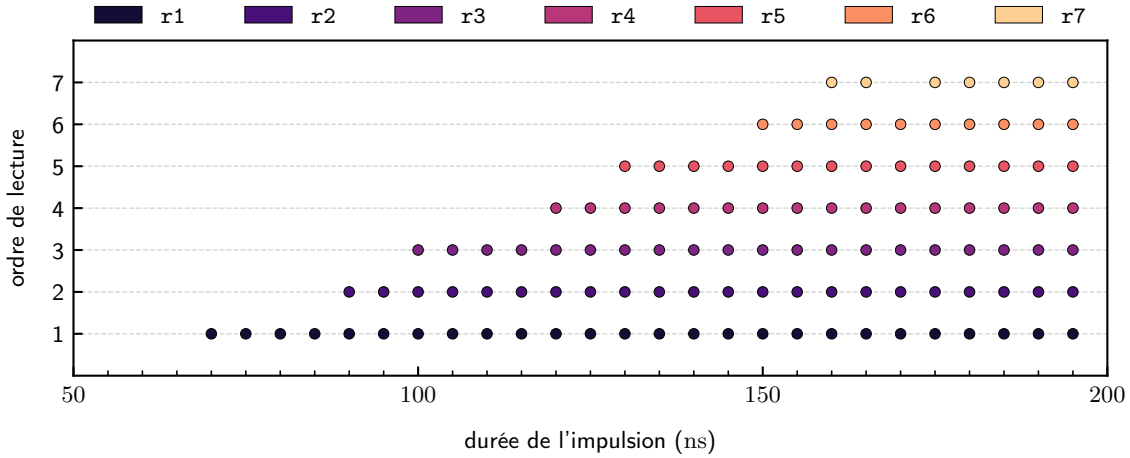


FIGURE 3.21 – Effets d'une augmentation progressive de la durée d'une impulsion laser sur le nombre de chargements corrompus par l'injection systématique d'une faute de type *bit set* sur les bits d'indices 2 et 18 des mots transférés entre la mémoire Flash et les registres `"r1"` à `"r7"` du microcontrôleur SAMD21G18A (répétition : 25 ; objectif : ×5 ; puissance : 1 000 mW ; pas d'exploration : 5 ns).

Cette figure illustre l'augmentation du nombre de transferts consécutifs corrompus par l'injection d'une faute de type *bit set* avec l'augmentation de la durée de l'impulsion laser. Pour une impulsion d'une durée de 170 ns par exemple, les bits d'indice 2 et 18 des sept mots lus en mémoire Flash ont systématiquement été mis à `'1'`. Ainsi, un attaquant peut injecter une faute sur un nombre choisi d'opérations de lecture consécutives en ajustant précisément la durée d'une impulsion laser. Nous rapportons que des résultats similaires ont été obtenus pour le MCU STM32F100RB avec un délai d'injection fixé à 2 500 ns. Une faute de type *bit set* a été injectée avec une répétabilité de 100 % sur les sept chargements de registres consécutifs avec une seule impulsion laser d'une durée de 370 ns.

Remarquons cependant que la durée de l'exécution de sept chargements consécutifs est de 280 ns sur le MCU SAMD21G18A et de 945 ns sur le MCU STM32F100RB. Les effets d'une perturbation laser persistent donc au-delà de la stimulation laser et contribuent, de fait, à l'injection d'une faute. Ce phénomène pourrait être attribué au filtrage du photocourant induit dans une colonne mémoire sous l'effet de la charge capacitive de la *bit line* de la colonne. L'étalement dans le temps des effets d'une perturbation laser

sur une colonne mémoire participerait ainsi à une réduction de la précision temporelle du modèle de faute étudié. Ces résultats questionnent les limites de notre dispositif d'injection pour les circuits de tests étudiés.

3.3.3 Optimisation de la précision temporelle

La forme temporelle d'une perturbation dépend de nombreux paramètres, dont la puissance de la source laser et la durée d'une impulsion.

3.3.3.A Optimisation du couple délai-puissance

Dans cette expérience, un balayage du délai d'injection a été réalisé pour plusieurs valeurs de la puissance de la source laser comprises entre 500 mW et 2 000 mW en répétant cinquante fois le chargement d'un mot de 32 bits dans le registre `"r1"` du MCU STM32F100RB. La répétabilité des fautes de type *bit set* obtenues sur le bit d'indice 7 du mot de 32 bits chargé dans le registre `"r1"` est reportée sur la Figure 3.22.

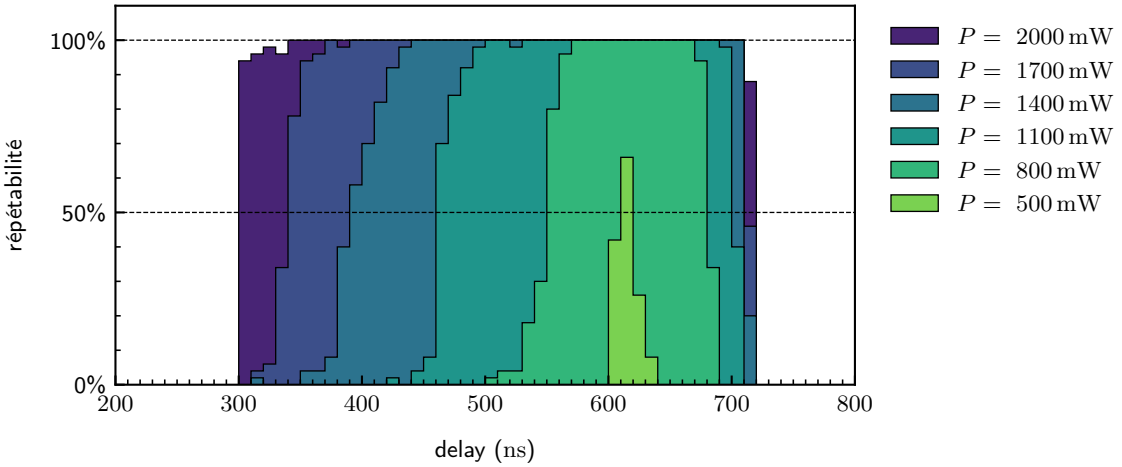


FIGURE 3.22 – Répétabilité des fautes injectées pendant le chargement du registre `"r1"` en fonction de l'instant d'injection et de la puissance de la source laser sur le microcontrôleur STM32F100RB (répétition : 50 ; objectif : ×5 ; durée d'impulsion : 135 ns ; pas d'exploration : 10 ns).

La génération d'une impulsion laser avec un délai d'injection compris dans l'intervalle 610 ns – 620 ns et une puissance de 500 mW nous a permis de perturber très précisément l'échantillonnage d'un bit lu en mémoire Flash avec une répétabilité de 70 % à l'intérieur d'une fenêtre temporelle étroite, d'environ 10 ns de largeur. Si cette précision temporelle permet de sélectionner très précisément l'opération de lecture pendant laquelle une faute est injectée, elle impose néanmoins des contraintes fortes en matière de synchronisation dans un scénario d'attaque réelle.

Une augmentation de la puissance de la source laser nous a permis d'injecter une faute de type *bit set* avec une répétabilité de 100 % en générant une impulsion laser en amont de la fenêtre temporelle 610 ns – 620 ns. Des fautes ont ainsi été obtenues, par exemple, pour un délai d'injection compris entre 440 ns et 690 ns à l'intérieur d'une fenêtre temporelle de 250 ns de largeur pour une puissance de la source laser fixée à 1 400 mW .

Remarquons que les fenêtres temporelles obtenues pour différentes valeurs de la puissance de la source laser ne sont pas centrées autour de l'intervalle 610 ns – 620 ns, que nous supposons encadrer l'opération d'échantillonnage du mot lu en mémoire Flash et transféré dans le registre `"r1"`. Nous supposons donc qu'une impulsion laser générée avec un délai d'injection supérieur à 620 ns perturbe majoritairement le circuit de tests après l'opération de lecture du mot transféré dans le registre `"r1"`.

Pour rappels, nous avions déjà démontré dans la section 3.2.4 que la puissance de la source laser est l'un des paramètres clés pour ajuster la précision spatiale de l'injection d'une faute. Les résultats présentés dans cette section démontrent que cette dernière joue également un rôle important dans l'ajustement de la précision temporelle d'une injection de faute. Puisque le profil temporel de la perturbation laser dépend à la fois de la durée de l'impulsion et de la puissance de la source, il convient d'étudier conjointement l'influence de l'une et de l'autre pour déterminer les paramètres d'injection optimaux.

3.3.3.B Optimisation du couple durée-puissance

Dans cette expérience, un balayage du délai d'injection a été réalisé pour plusieurs valeurs de la puissance de la source laser (comprises entre 500 mW et 2 000 mW) et plusieurs valeurs de la durée de l'impulsion laser. Le chargement d'un mot de 32 bits dans le registre `"r1"` du MCU STM32F100RB a été répété cinquante fois pour chaque ensemble de valeurs explorées. La puissance minimale pour laquelle une faute de type *bit set* est obtenue sur le bit d'indice 7 du mot de 32 bits chargé dans le registre `"r1"` avec une répétabilité de 100 % est reportée en fonction de l'instant d'injection et de la durée de l'impulsion laser sur la Figure 3.23.

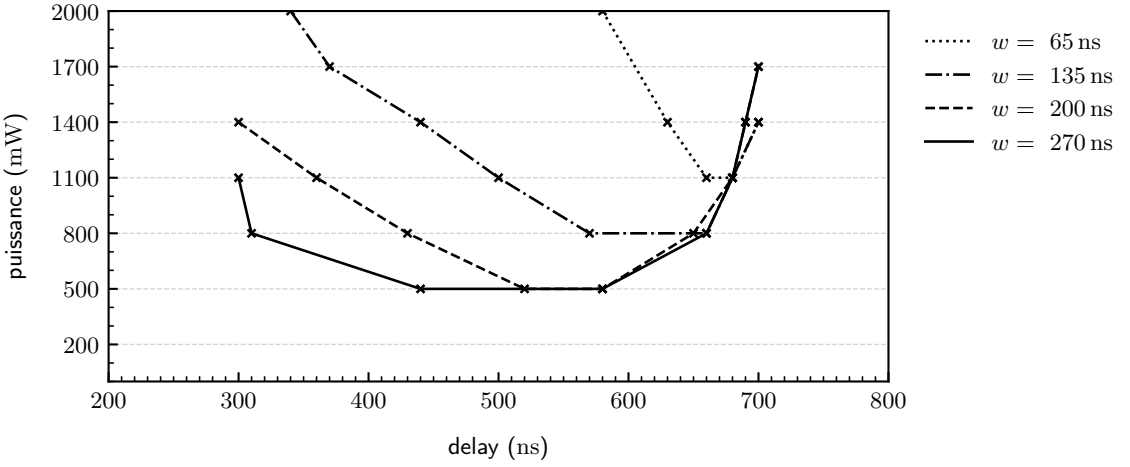


FIGURE 3.23 – Seuils de puissance de la source laser pour l'injection d'une faute de type *bit set* avec 100 % de répétabilité sur le bit d'indice 7 d'un mot de 32 bits chargé dans le registre `"r1"` en fonction de l'instant d'injection (répétition : 50 ; objectif : ×5 ; pas d'exploration : 10 ns ; circuit de tests : microcontrôleur STM32F100RB).

La représentation adoptée permet de retrouver les résultats de l'influence de la puissance sur la résolution temporelle pour une durée d'impulsion fixée. Par exemple, pour une durée d'impulsion de 135 ns et une puissance de 1 400 mW, une faute de type *bit set* peut être injectée avec 100 % de répétabilité lorsque le délai d'injection est réglé entre

440 ns et 690 ns. Plusieurs observations peuvent alors être faites.

Tout d'abord, la puissance minimale pour laquelle une faute peut être injectée dépend de la durée de l'impulsion. Un attaquant a donc intérêt à augmenter la durée de l'impulsion laser pour injecter des fautes avec une puissance faible. Puisque la durée de l'impulsion limite la résolution temporelle de l'attaquant, une valeur proche d'une période d'horloge semble être le meilleur compromis.

Par ailleurs, l'instant optimal d'injection dépend de la durée de l'impulsion et la puissance de la source. Ce dernier tend en effet à se rapprocher de l'opération d'échantillonnage lorsque la durée de l'impulsion est faible et tend à s'en éloigner lorsque cette dernière augmente. Les couples de valeurs optimales pour la durée d'impulsion et la puissance de la source laser sont donc à déterminer conjointement avec l'instant d'injection et non à délai constant.

Enfin, la résolution temporelle de la procédure d'injection est intrinsèquement limitée par le dispositif expérimental et par le circuit de tests. La meilleure résolution possible obtenue avec la source 50 ns sur le microcontrôleur STM32F1 est ainsi de l'ordre de 20 ns pour une durée d'impulsion de 65 ns et une puissance de 1 100 mW.

3.4 Mécanisme d'injection de fautes laser sur une mémoire NOR Flash

Dans cette section, nous appliquons tout d'abord le modèle électrique d'une perturbation laser aux structures de transistor à grille flottante. Nous étudions ensuite les conditions et les effets de l'injection d'un photocourant parasite dans les cellules d'une mémoire NOR Flash. L'analyse théorique de ce mécanisme sur une architecture mémoire de référence permet alors d'expliquer les points de convergence et de divergence du modèle de faute *bit set* analysé sur les MCU STM32F100RB et SAMD21G18A dans les sections précédentes.

3.4.1 Implémentation physique d'une mémoire NOR Flash

Dans une mémoire Flash, l'information est mémorisée sous forme de charges électriques dans un tableau de cellules mémoire. Chaque cellule mémoire est composée d'un transistor () qui conserve la charge emprisonnée dans sa grille indépendamment de la présence ou non d'une alimentation électrique. Des amplificateurs de détections permettent de traduire cette charge en une valeur binaire qui peut être exploitée par le circuit.

Les tableaux d'une mémoire NOR Flash sont organisés en lignes et en colonnes. Les grilles des transistors à grille flottante d'une ligne sont reliées à une même ligne de mots, aussi appelée *word line*, qui permet de sélectionner simultanément tous les mots mémorisés dans la ligne. Les drains des transistors à grille flottante d'une colonne sont reliés à une ligne de bits commune, aussi appelée *bit line*, qui permet de lire le transistor sélectionné dans une colonne en connectant son drain à un amplificateur de détection.

La lecture d'un mot de 32 bits nécessite ainsi de sélectionner une *word line* et 32 *bit lines*. Cette architecture est illustrée sur la Figure 3.15 dans le cas du MCU STM32F100RB et sur la Figure 3.16 dans le cas du MCU SAMD21G18A.

Le *layout* d'un tableau de cellules mémoire de 512 lignes et 2048 colonnes dans l'architecture NOR Flash est présenté sur la Figure 3.24. Il est obtenu en répétant périodiquement le *layout* d'un transistor à grille flottante encadré en noir en bas à gauche de la figure.

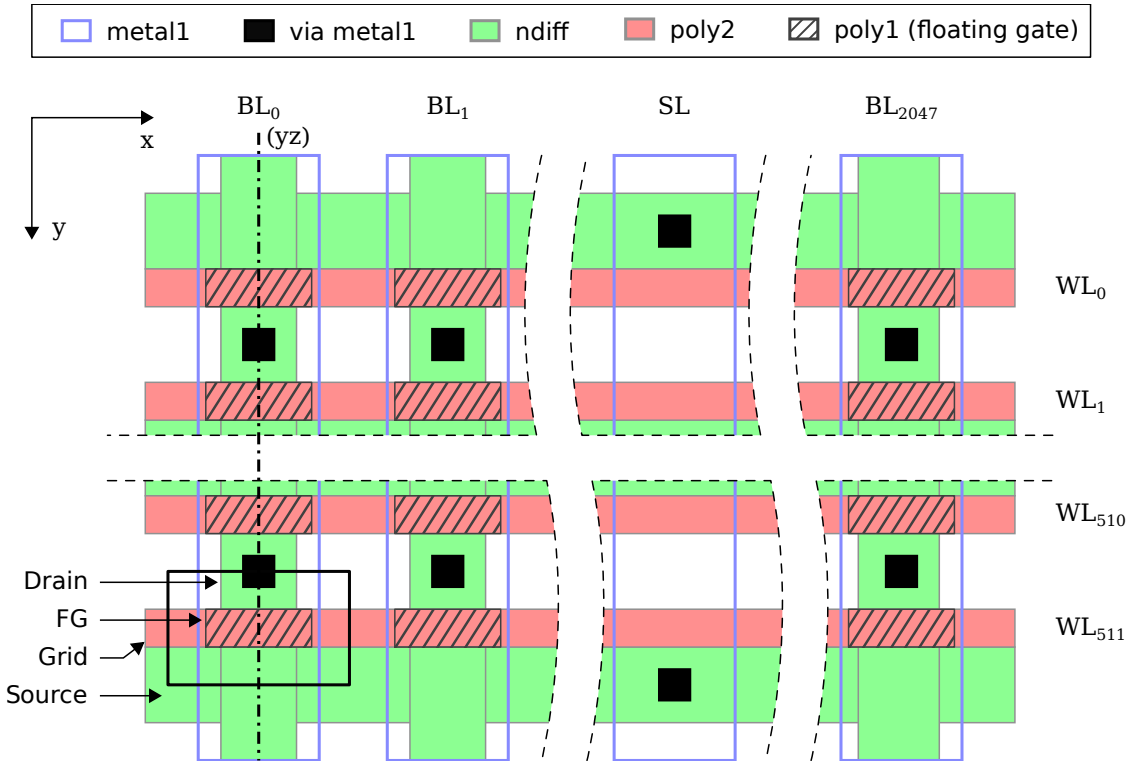


FIGURE 3.24 – Layout d'un tableau de cellules mémoire dans l'architecture NOR Flash. Le *layout* d'une cellule mémoire est encadré en noir.

Ce transistor est composé d'une grille flottante et d'une grille de contrôle en polysilicium, notées « poly1 » et « poly2 » respectivement, ainsi que d'une source et d'un drain délimités de part et d'autre de la grille de contrôle par les zones de diffusion $n+$, notées « ndiff ». Afin d'optimiser la densité de la mémoire, chaque transistor partage son drain et sa source avec les transistors adjacents dans la même colonne mémoire. Le drain des transistors d'une même colonne sont connectés à une *bit line* déposé au premier niveau de métal, notée « metal1 », grâce à des vias métalliques, notés « via metal1 ». La *bit line* associée à la colonne d'indice i est notée BL_i. Les grilles de contrôle en polysilicium des transistors d'une même ligne, notées « poly2 », sont connectées ensemble pour former une *word line*. La *word line* associée à la ligne d'indice i est notée WL_i. Les sources des transistors à grille flottante d'un même secteur sont connectées par l'intermédiaire d'une zone de diffusion à une ligne de source, notée SL. Cette ligne permet ainsi de polariser l'ensemble des sources des transistors à la masse [37].

3.4.2 Mécanisme d'injection de fautes laser sur une mémoire NOR Flash

Une vue en coupe d'une colonne mémoire NOR Flash est présentée sur la Figure 3.25. Sur cette vue sont représentés : **(1)** la grille de contrôle, **(2)** l'isolant interpoly, **(3)** la grille flottante, **(4)** l'oxyde tunnel, **(5)** la source, **(6)** le drain et **(7)** le substrat d'un transistor, ainsi que **(8)** les vias métalliques et **(9)** la *bitline* de la colonne. L'axe de coupe, noté (yz), est indiqué en pointillé noir sur la Figure 3.24.

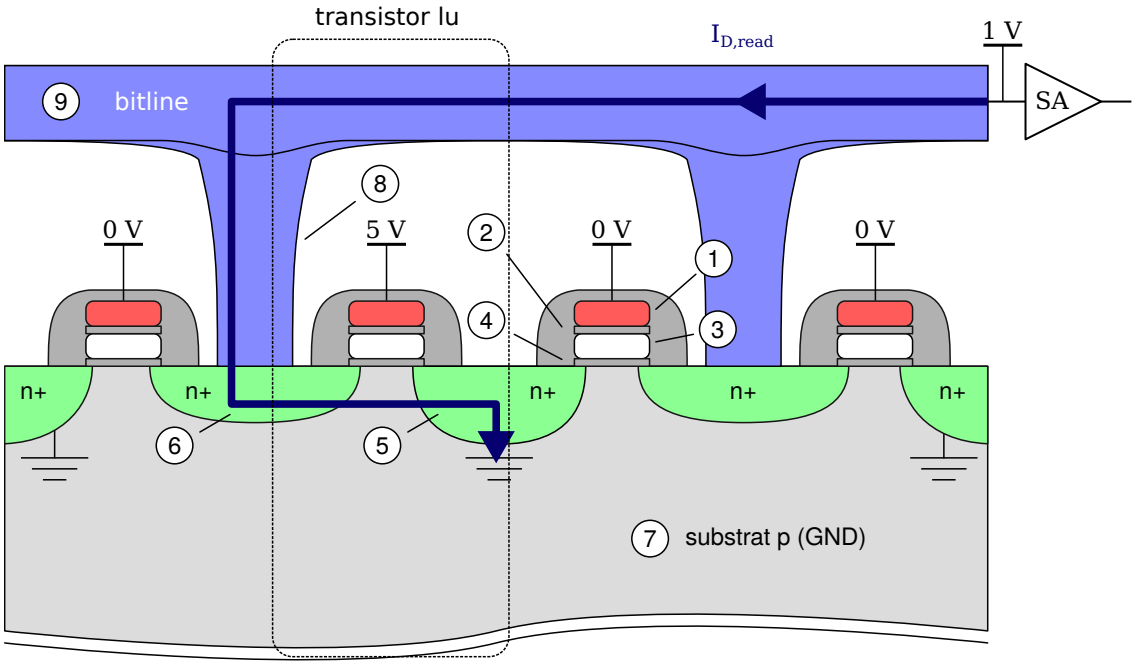


FIGURE 3.25 – Vue en coupe d'une colonne mémoire NOR Flash pendant une opération de lecture.

Les tensions reportées sur la Figure 3.25 correspondent à une polarisation de la colonne pendant une lecture du transistor à grille flottante encadré en pointillé. La *bit line* de la colonne est polarisée à 1 V. La *word line* du transistor lu est polarisée à 5 V. La *word line* des transistors non-sélectionnés est quant à elle polarisée à la masse.

Dans ces conditions de polarisations, le courant de lecture $I_{D,read}$ qui traverse la *bit line* et le transistor sélectionné dépend de l'état du transistor. Lorsque celui-ci est effacé, un canal de conduction se forme entre le drain et la source du transistor et un courant de lecture non-négligeable traverse la *bit line*. En revanche, lorsque il est programmé, le courant de lecture reste quasi-nul. Un amplificateur de détection (*Sense Amplifier* – SA) connecté à la *bit line* traduit le courant qui traverse la *bit line* en un état logique : l'état `'1'` si le transistor lu est effacé, l'état `'0'` s'il est programmé. La charge stockée sur la grille flottante du transistor lu est ainsi converti en un état logique. Le lecteur est redirigé vers la section 1.4.4.C pour de plus amples informations sur l'opération de lecture d'un transistor à grille flottante.

Remarquons que les drains de tous les transistors d'une colonne sélectionnée sont polarisés à 1 V à travers la *bit line* lors d'une opération de lecture. La jonction PN entre le drain et le substrat de ces transistors est alors polarisée en inverse. Or, le mécanisme

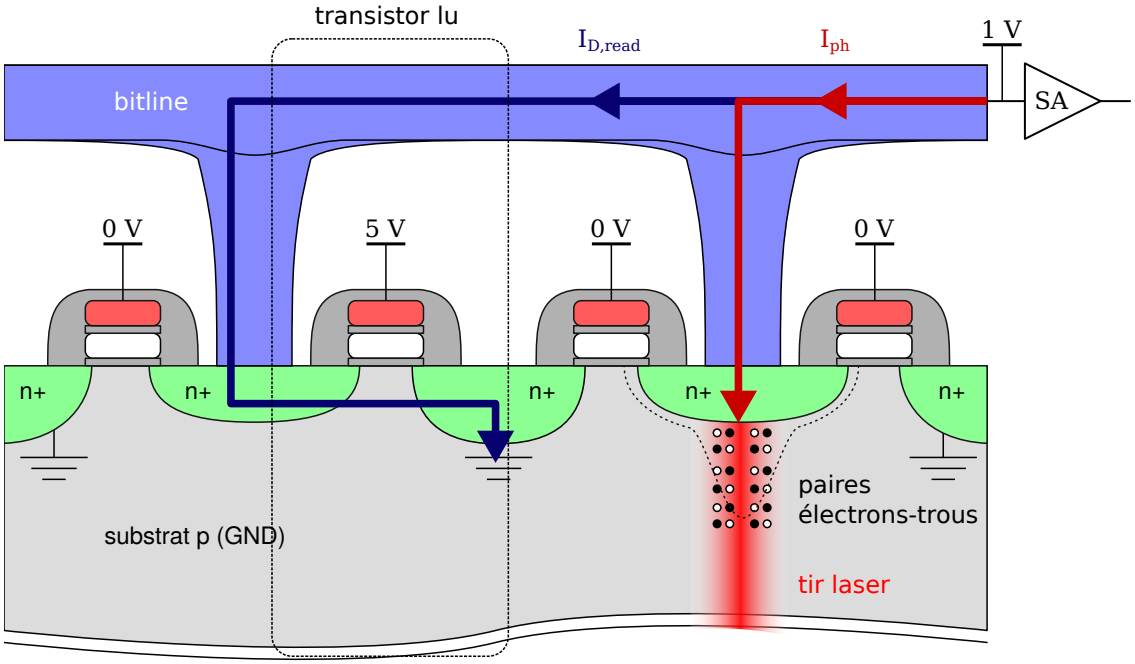


FIGURE 3.26 – Vue en coupe d'une colonne mémoire NOR Flash sous illumination laser.

d'injection d'un photocourant par illumination laser décrit dans la section 1.4.1 s'applique aux jonctions PN polarisées en inverse d'un circuit intégré. Les porteurs de charges générés par un tir laser à proximité de ces jonctions sont donc collectés par les transistors à grille flottante illuminés et injectés sous la forme d'un photocourant I_{ph} à l'entrée de l'amplificateur de détection. Ce mécanisme d'injection de fautes est illustré sur la Figure 3.26.

Une LFI sur une colonne mémoire sélectionnée permet ainsi d'augmenter le courant de lecture d'un transistor, de sorte à forcer la comparaison réalisée par l'amplificateur de détection à l'état `'1'`, indépendamment de l'état du transistor lu. Puisque ce mécanisme ne permet pas de réduire le courant de lecture, les fautes observées correspondent uniquement à des fautes de type *bit set*. Cette caractéristique est illustrée sur la Figure 3.27.

Remarquons par ailleurs que tous les transistors illuminés d'une colonne sélectionnée contribuent à l'injection d'un photocourant. La position de l'objectif au-dessus de la colonne n'a donc pas d'influence sur le modèle de faute observé. Cette caractéristique explique l'indépendance par rapport à la coordonnée y de l'objectif des résultats obtenus sur les MCU STM32F100RB et SAMD21G18A. Une perturbation à l'origine d'une faute peut ainsi être générée à distance des transistors sélectionnés.

L'injection d'une faute sur le bit d'indice 18 d'un mot de 32 bits est finalement illustrée sur la Figure 3.28. La région de la mémoire dédiée à la mémorisation des bits d'indice 18 est encadrée en bleu et celle dédiée aux bits d'indice 19 en orange. La position de la zone d'effet du spot laser, délimitée en rouge, détermine les transistors sur lesquels un photocourant est injecté, parmi les transistors sélectionnés de la région de la mémoire où les bits d'indice 18 sont mémorisés. La somme des contributions des transistors illuminés force la sortie de l'amplificateur de détection SA_{18} à `'1'`. Une faute de type

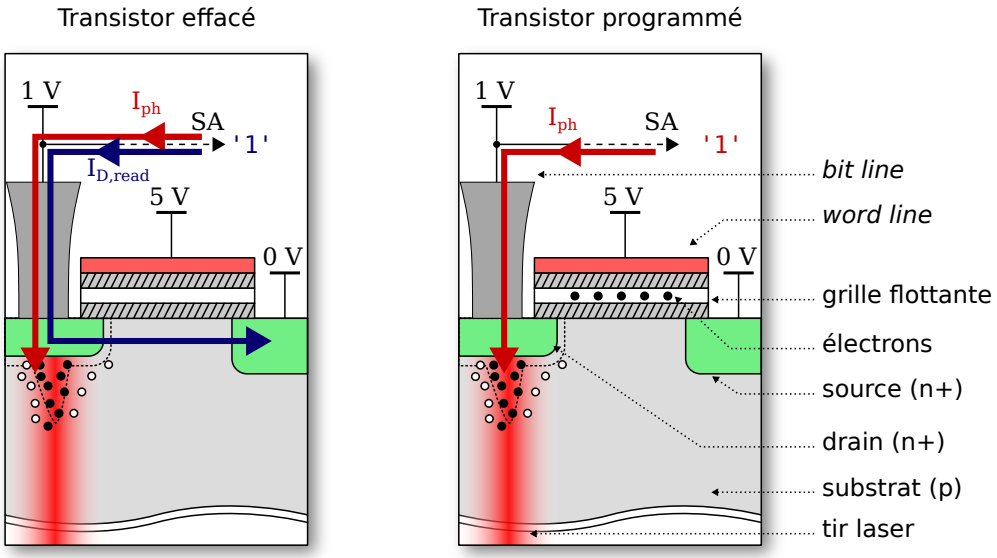


FIGURE 3.27 – Injection d'une faute *bit set* par illumination laser d'un transistor à grille flottante effacé (à gauche) et programmé (à droite).

bit set est ainsi injectée sur le bit d'indice 18.

3.4.3 Synthèse du mécanisme d'injection de fautes

Le mécanisme d'injection de fautes décrit dans cette section est une application directe du mécanisme d'injection d'un photocourant par illumination laser sur les transistors à grille flottante d'une mémoire NOR Flash. L'injection d'un photocourant est rendue possible par la polarisation des colonnes mémoire sélectionnées pendant une opération de lecture. Puisque l'injection d'un photocourant permet d'augmenter, mais pas de diminuer, le courant de lecture d'un transistor, une illumination laser force systématiquement à `'1'` le résultat d'une lecture de l'état d'un transistor. Ce mécanisme d'injection de fautes explique ainsi l'observation de fautes de type *bit set* uniquement lorsque l'objectif du laser est dirigé à l'intérieur du tableau de la mémoire NOR Flash des MCU STM32F100RB et SAMD21G18A.

Tous les transistors illuminés d'une colonne sélectionnée peuvent par ailleurs contribuer à la collection des porteurs de charge générés par une LFI. Cette caractéristique du mécanisme d'injection de fautes explique que les résultats observés ne dépendent pas de la coordonnée y de l'objectif au-dessus du tableau de la mémoire NOR Flash des MCU STM32F100RB et SAMD21G18A.

L'organisation de la mémoire permet, enfin, de sélectionner précisément les colonnes sur lesquelles une faute est injectée. Le découpage de la mémoire par groupes de colonnes adjacentes où les bits de même indice sont mémorisés facilite en effet la sélection de l'indice des bits mis à `'1'` grâce à la coordonnée x de l'objectif. En réglant la taille du spot laser pour recouvrir toutes les colonnes d'un de ces groupes, une faute peut être injectée sur un bit d'indice choisi indépendamment de l'adresse du mot lu en mémoire Flash.

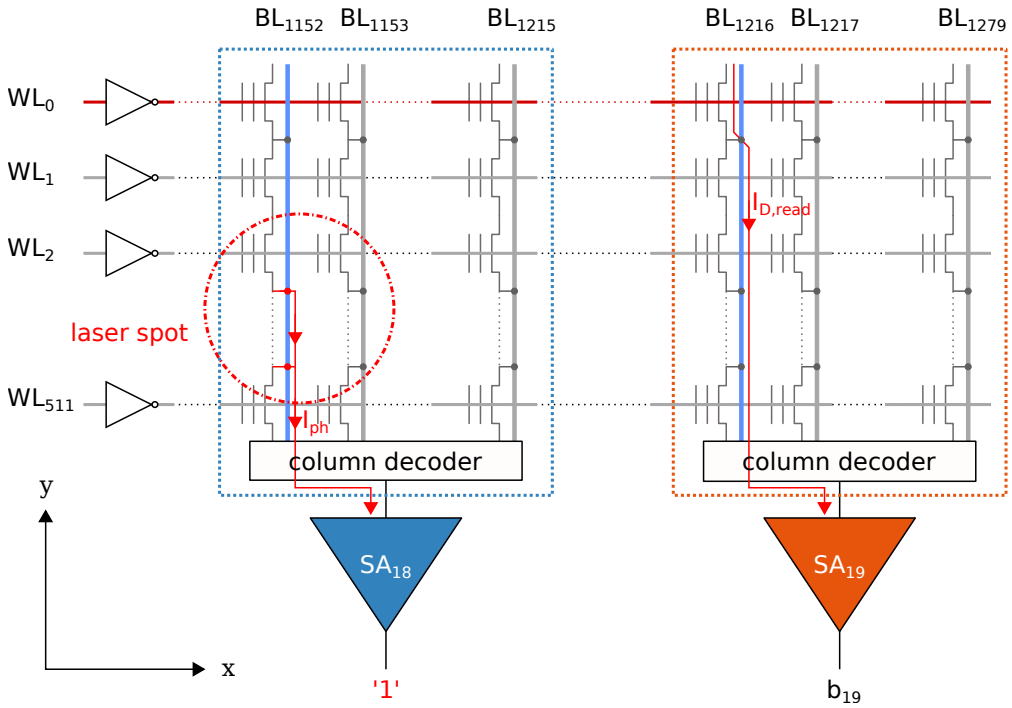


FIGURE 3.28 – Injection d'un photocourant dans la colonne mémoire du bit d'indice 18 d'un mot de 32 bits.

Le mécanisme d'injection de fautes étudié permet ainsi d'injecter des fautes de type *bit set* de manière très précise sur un bit d'indice choisi pendant la lecture d'une instruction ou d'une donnée dans la mémoire NOR Flash d'un MCU avec très peu de contraintes en termes de positionnement. Différents modèles de fautes logiciels obtenus grâce à ce dernier sont détaillés dans la section suivante.

3.5 Étude des modèles de fautes logiciels

Cette section explore les modèles de fautes logiciels obtenus en pratique en appliquant le mécanisme d'injection de fautes décrit dans les sections précédentes aux transferts d'instructions entre la mémoire NOR Flash et le cœur de calcul d'un MCU.

3.5.1 Corruption d'une instruction de chargement

Les paragraphes suivants décrivent les tests réalisés sur l'instruction `"movw r0, #0"` qui charge les 16 bits de poids faible du registre `"r0"` avec la valeur immédiate 0 codée sur 16 bits. L'encodage sur 32 bits de cette instruction est composé :

— d'un opcode codé sur 12 bits qui définit l'opération réalisée par le cœur de calcul ;

— d'un registre de destination `"Rd"` codé sur 3 bits ;

— d'une valeur immédiate codée sur 16 bits obtenue en concatenant les valeurs `"imm4"`, `"i"`, `"imm3"` et `"imm8"`.

Afin d'évaluer les différentes corruptions obtenues sur le transfert de l'instruction `"movw r0, #0"`, les registres `"r0"` à `"r11"` sont tout d'abord initialisés à `0xFFFFFFFF`. En l'absence de faute, les 16 bits de poids faible du registre `"r0"` sont chargés avec la valeur `0x0000`. La valeur du registre `"r0"` est alors `0xFFFF0000`.

Plusieurs expériences ont été réalisées sur le MCU STM32F100RB pour injecter une faute de type *bit set* sur un bit de l'instruction pendant le transfert de cette dernière pour différentes valeurs de la coordonnée x de l'objectif. Les différentes corruptions obtenues sont illustrées sur la Figure 3.29. Les fautes de type *bit set* sont indiquées par un carré rouge au niveau du bit forcé à `'1'` dans l'instruction de chargement.

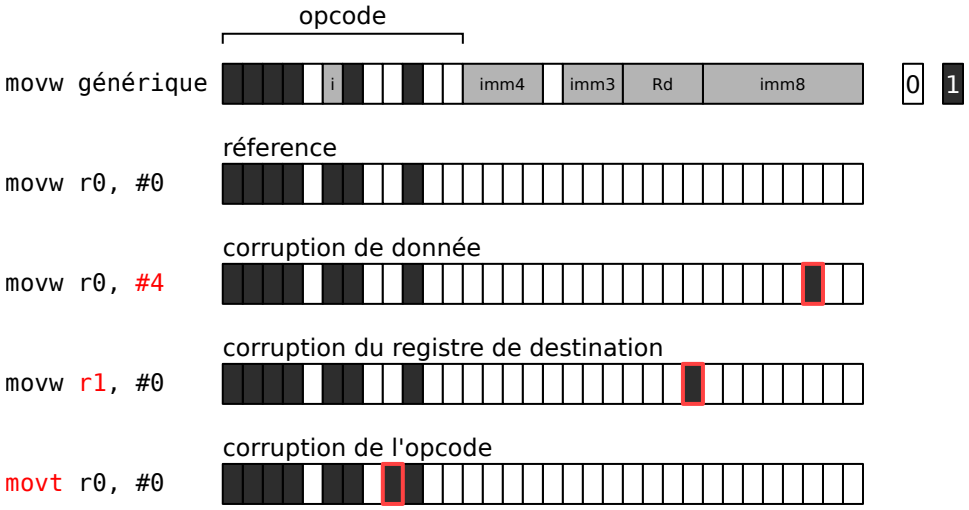


FIGURE 3.29 – Exemples de corruptions possibles sur l'instruction `"movw"`. Les bits ciblés par une faute de type *bit set* sont encadrés en rouge.

Corruption de la valeur immédiate. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 190 µm, le bit d'indice 2 de l'instruction est mis à `'1'`, conformément au mécanisme de faute étudié. Le cœur de calcul exécute alors l'instruction `movw r0, #4` qui charge les 16 bits de poids faible du registre `"r0"` avec la valeur `0x0004`. En pratique, la valeur `0xFFFF0004` est alors observée dans le registre `"r0"` après l'injection d'une faute.

Corruption du registre de destination. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 450 µm, le bit d'indice 8 de l'instruction est mis à `'1'`. Le cœur de calcul exécute alors l'instruction `movw r1, #0` qui charge les 16 bits de poids faible du registre `"r1"` avec la valeur `0x0000`. En pratique, les valeurs `0xFFFFFFFF` et `0xFFFF0000` sont alors observées dans le registre `"r0"` et `"r1"` après l'injection d'une faute.

Corruption de l'opcode. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 1 100 µm, le bit d'indice 23 de l'instruction est mis à `'1'`. Le cœur de calcul exécute alors l'instruction `movt r0, #0` qui charge les 16 bits de poids fort du registre `"r0"` avec la valeur `0x0000`. En pratique, la valeur `0x0000FFFF` est alors observée dans le registre `"r0"` après l'injection d'une faute.

3.5.2 Corruption d'une instruction arithmétique

Les paragraphes suivants décrivent les tests réalisés sur l'instruction `"add r0, #1"` qui incrémente la valeur du registre `"r0"`. Cette instruction est utilisée dans la plupart des boucles avec incrément d'une variable. L'encodage sur 16 bits de cette instruction est composé

— d'un opcode codé sur 5 bits ;

— d'un registre `"Rd"` – à la fois source et destination – codé sur 3 bits ;

— d'une valeur immédiate `"imm8"` codée sur 8 bits.

Afin d'évaluer les différentes corruptions obtenues sur le transfert de l'instruction `"add r0, #1"`, les registres `"r0"` à `"r11"` sont tout d'abord initialisés à `0x00000000`. En l'absence de faute, la valeur du registre `"r0"` est incrémentée de 1. Elle correspond alors à la valeur `0x00000001`.

Plusieurs expériences ont été réalisées sur le MCU STM32F100RB pour injecter une faute de type *bit set* sur un bit de l'instruction pendant le transfert de cette dernière pour différentes valeurs de la coordonnée x de l'objectif. Les différentes corruptions obtenues sont illustrées sur la Figure 3.30. Les fautes de type *bit set* sont indiquées par un carré rouge au niveau du bit forcé à `'1'` dans l'instruction arithmétique.

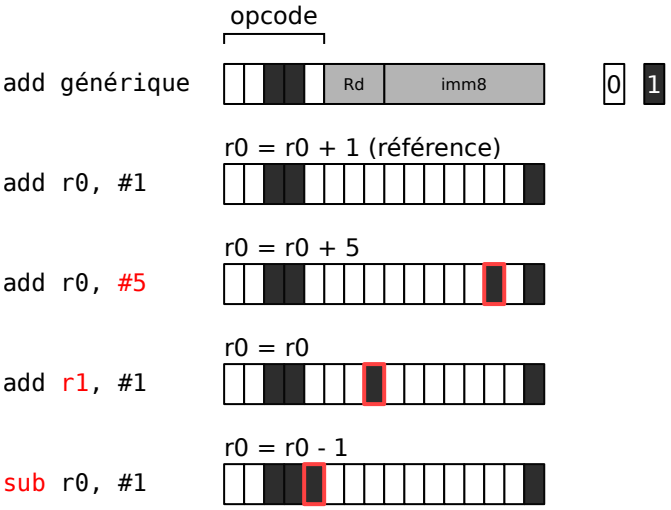


FIGURE 3.30 – Manipulations de la valeur d'incrément d'une instruction `add`. Les bits ciblés par une faute de type *bit set* sont encadrés en rouge.

Corruption de la valeur immédiate. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 190 µm ou à 890 µm, en fonction de l'alignement de l'instruction, le bit d'indice 2 de l'instruction est mis à `'1'`, conformément au mécanisme de faute étudié. Le cœur de calcul exécute alors l'instruction `add r0, #5` qui incrémente de 5 le contenu du registre `"r0"`. En pratique, la valeur `0x00000005` est alors observée dans le registre `"r0"` après l'injection d'une faute.

Corruption du registre de destination. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 450 µm ou à 1 160 µm, en fonction de l'alignement de l'instruction, le bit d'indice 8 de l'instruction est mis à `'1'`. Le cœur de calcul exécute alors

l'instruction `add r1, #1` qui incrémente de 1 le contenu du registre `"r1"`. En pratique, les valeurs `0x00000000` et `0x00000001` sont alors observées dans le registre `"r0"` et `"r1"` après l'injection d'une faute.

Corruption de l'opcode. Lorsqu'une faute est injectée en fixant la coordonnée x de l'objectif à 580 µm ou à 1 280 µm, en fonction de l'alignement de l'instruction, le bit d'indice 11 de l'instruction est mis à `'1'`. Le cœur de calcul exécute alors l'instruction `sub r0, #1` qui décrémente de 1 le contenu du registre `"r0"`. En pratique, la valeur `0xFFFFFFFF` est alors observée dans le registre `"r0"` après l'injection d'une faute.

Les corruptions obtenues sur la valeur immédiate, le registre de destination et l'opcode de l'instruction `"add r0, #1"` permettent ainsi d'augmenter, d'empêcher ou d'inverser le signe d'un incrément. Cependant, puisque l'instruction est codée sur 16 bits et qu'un attaquant ignore *a priori* l'alignement des instructions en mémoire, deux positions de l'objectif doivent être explorées pour chaque indice ciblé. Une application directe de la manipulation des valeurs d'incrément consiste à provoquer une sortie anticipée d'une boucle de comparaison, par exemple dans le cas de la vérification d'un mot de passe ou d'un code de vérification [76].

3.5.3 Corruption d'une instruction de branchement

Nous étudions enfin la possibilité de modifier l'issue du test conditionnel présenté dans le Code source 3.4.

```
1 ldr r0, =0xFFFFFFFF
2 nop
3 ...
4 nop
5 cmp r0, #0
6 bhi .+1
7 movw r0, #0
```

CODE SOURCE 3.4 – Test conditionnel sur la valeur du registre `"r0"`.

L'instruction de chargement de registre `"ldr"` charge tout d'abord le registre `"r0"` avec le mot de 32 bits `0xFFFFFFFF`. Une série d'instructions `"nop"` permet d'isoler temporellement la lecture des instructions du test conditionnel de celle de l'instruction de chargement de registres. L'instruction de comparaison `"cmp"` met ensuite à jour les drapeaux du MCU en fonction du résultat de la comparaison entre le registre `"r0"` et la valeur immédiate 0. L'instruction de branchement `"bhi"` redirige finalement l'exécution du programme après l'instruction de chargement `"movw r0, #0"` si la valeur non-signée du registre `"r0"` est strictement supérieure à 0. Puisque cette condition est remplie, le registre `"r0"` conserve la valeur `0xFFFFFFFF` en l'absence de faute.

L'encodage sur 16 bits de l'instruction `"bhi"` est composé :

— d'un opcode codé sur 4 bits ;

— d'une condition `"cond"` codée sur 4 bits, qui définie la condition testée par le cœur de calcul ;

— d'une valeur immédiate `"imm8"` codée sur 8 bits, qui détermine l'adresse du branchement relativement au compteur du programme.

Cet encodage est illustré sur la Figure 3.31. L'injection d'une faute de type *bit set* sur le bit d'indice 8 de l'instruction `"bhi"` permet de modifier la condition de branchement `"hi"` (strictement supérieure) en la condition de branchement `"ls"` (inférieure ou égale). Cette faute réalise ainsi une inversion du test conditionnel de l'instruction de branchement. Elle est illustrée sur la Figure 3.31 par un carré rouge au niveau du bit forcé à `'1'` dans l'instruction de branchement.

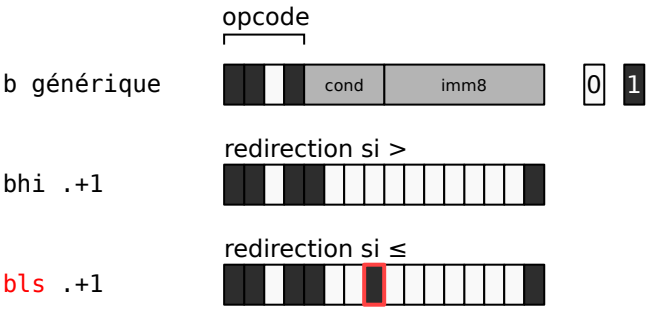


FIGURE 3.31 – Modification d'un test conditionnel. Les bits ciblés par une faute de type *bit set* sont encadrés en rouge.

Afin d'obtenir cette corruption, la coordonnée x de l'objectif a été fixée à 450 µm dans une première série de tests et à 1 160 µm dans une deuxième série de tests. Avec ces réglages, la valeur `0xFFFF0000` a été observée dans le registre `"r0"` après l'injection d'une faute. Remarquons qu'une faute sur le registre source de l'instruction de comparaison permettrait également d'expliquer les caractéristiques observées. La valeur du registre `"r0"` dépendrait alors de la valeur du registre utilisée par l'instruction de comparaison corrompue. Remarquons que la corruption d'une instruction de branchement est plus efficace qu'un saut d'instruction, dans la mesure où elle permet à un attaquant d'atteindre une portion de code sécurisé, située en amont de l'instruction corrompue.

3.6 Conclusion

Dans ce chapitre, nous avons étudié en pratique l'injection de fautes par illumination laser sur la mémoire NOR Flash de deux microcontrôleurs 32 bits de technologies 65 nm et 80 nm. Le modèle de faute étudié sur le MCU STM32F100RB permet à un attaquant d'injecter une faute de type *bit set* pendant la relecture d'un nombre arbitraire de mots en mémoire Flash sur 1 à 4 bits adjacents avec une répétabilité de 100 %. Des fautes de type *bit set* ont également été obtenues sur 5 bits adjacents avec une répétabilité de 50 %. Le modèle de faute étudié sur le MCU SAMD21G18A, quant à lui, permet à un attaquant d'injecter une faute de type *bit set* pendant la relecture d'un nombre arbitraire de mots en mémoire Flash sur 2 bits dont les indices sont congrus modulo 16. Des fautes de type *bit set* ont également été obtenues sur un seul bit avec une répétabilité supérieure à 50 % en ajustant le diamètre du spot laser et la position de l'objectif. Les paramètres d'injection utilisé pour ajuster ces caractéristiques sont résumés sur la Table 3.2.

PROPRIÉTÉS	PARAMÈTRES
Indice des bits mis à '1'	- Coordonnées **X** de l'objectif
Précision spatiale	- **Grossissement** de l'objectif - Coordonnées **Z** de l'objectif - **Puissance** de la source laser - **Durée** de l'impulsion laser
Précision temporelle	- **Délai** d'injection - **Durée** de l'impulsion laser - **Puissance** de la source laser
Répétabilité	- **Délai** d'injection - **Durée** de l'impulsion laser - **Puissance** de la source laser

TABLE 3.2 – Influence des paramètres expérimentaux sur les modèles de fautes par injection laser étudiés dans ces travaux.

La comparaison des résultats obtenus sur les deux microcontrôleurs, de technologies et d'architectures différentes, confirme l'existence d'un mécanisme d'injection de fautes commun aux deux circuits. Ce mécanisme est décrit en appliquant le mécanisme d'injection de fautes laser par effet photoélectrique sur les transistors à grille flottante des tableaux de cellules mémoire NOR Flash. En particulier, nous avons mis en évidence que les jonctions drains-substrats des transistors dont les drains sont connectés à une *bitline* sélectionnée pendant une opération de lecture sont sensibles au mécanisme d'injection de fautes laser par effet photoélectrique. L'injection d'un photocourant parasite dans une colonne mémoire permet ainsi de forcer à '1' le résultat d'une opération de lecture sur les bits mémorisés dans cette colonne mémoire.

Ce mécanisme explique les spécificités des résultats obtenus sur les deux circuits, à savoir :

— un modèle de faute unipolaire (*bit set*) pour les deux circuits étudiés ;

— la possibilité de choisir l'indice des bits mis à '1' en ajustant la position de l'objectif sur une colonne ;

— l'injection de fautes identiques pour un déplacement de l'objectif dans l'axe colinéaire à la direction des colonnes ;

— un décalage des zones de sensibilité associées aux colonnes sélectionnées lors d'un incrément de l'adresse de lecture.

Les architectures de mémoire NOR Flash facilitent l'injection de fautes laser avec une forte précision spatiale. En effet, la surface de ces mémoires représente une part importante de la surface des circuits intégrés étudiés (environ 10 %). Elles sont donc facilement identifiables sur une image infrarouge de ces circuits. La répartition des

colonnes sélectionnées pendant une opération de lecture facilite par ailleurs l'injection sur une ou deux colonnes choisies, en fonction de l'architecture de la mémoire. Nous avons notamment démontré sur un circuit fabriqué dans la technologie 80 nm qu'une faute peut être injectée sur une seule colonne mémoire avec 100 % de répétabilité grâce à un spot laser de 40 µm de diamètre. La géométrie des colonnes permet enfin d'ajuster les paramètres d'injection suivant un seul axe, à savoir l'axe perpendiculaire à la direction des colonnes, contrairement à la plupart des scénarios d'injection de fautes laser qui nécessitent d'ajuster la position de l'objectif suivant deux axes.

Nous avons également vérifié en pratique la possibilité de remplacer une instruction par une autre avec une grande précision grâce à une illumination laser. La surface importante et la régularité du tableau de cellules d'une mémoire NOR Flash facilitent ainsi l'identification et l'exploitation de fautes injectées par illumination laser pendant l'exécution d'un programme sur des microcontrôleurs non-sécurisés. Plusieurs vulnérabilités résultant de ce modèle de faute sont étudiées dans le chapitre 4.

Ces contributions ont fait l'objet de deux publications :

— B. COLOMBIER, A. MENU, J. DUTERTRE et al., « Laser-induced Single-bit Faults in Flash Memory : Instructions Corruption on a 32-bit Microcontroller, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2019, p. 1-10 ;

— A. MENU, J. DUTERTRE, J. RIGAUD et al., « Single-bit Laser Fault Model in NOR Flash Memories : Analysis and Exploitation, » in *Proceedings of the International Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, Milan, Italy, 2020, p. 41-48.

Chapitre 4

Exploitation des modèles de fautes EM et laser

Dans ce chapitre, nous exploitons les modèles de fautes décrits dans ces travaux pour compromettre l'implémentation logicielle de plusieurs fonctions de sécurité. Nous démontrons, dans un premier temps, que ces modèles permettent de construire et de tester facilement des scénarios d'attaques par injection de fautes à travers plusieurs cas d'étude réalisés en pratique. Nous exploitons, dans un second temps, la connaissance de ces modèles pour analyser les vulnérabilités d'un circuit sécurisé destiné à l'internet des objets.

4.1 Synthèse des modèles de fautes

Dans cette section, nous résumons les modèles de fautes par injection EM décrits sur les microcontrôleurs non-sécurisés ATmega328P et SAM3X8E dans le chapitre 2, ainsi que les modèles de fautes par injection laser décrits sur les microcontrôleurs non-sécurisés STM32F100RB et SAMD21G18A dans le chapitre 3.

4.1.1 Résumé des caractéristiques principales

L'injection d'une faute au niveau matériel sur l'interface de lecture d'une mémoire Flash permet de manipuler les instructions et les données au niveau logiciel pendant une opération de lecture. Les fautes injectées sur les instructions et les données lues en mémoire Flash sont caractérisées par :

- **un type**, à savoir *bit set, bit reset,* ou *bit flip*;
- **une précision spatiale**, définie par la granularité et le nombre de bits qu'un attaquant peut corrompre simultanément pendant la lecture d'un mot en mémoire;
- **une précision temporelle**, définie par le nombre de cycles d'horloge qu'un attaquant peut perturber;
- **une répétabilité**, qui quantifie le taux de succès d'une injection de faute.

Les propriétés des modèles de fautes étudiés sur chacun des dispositifs sont résumées dans la Table 4.1.

Publication	DTIS 2020 [112]	FDTC 2019 [111]	HOST 2019 [51]	FDTC 2020 [113]
Microcontrôleur	ATmega328P	SAM3X8E	STM32F100RB	SAMD21G18A
ISA	16 bits	16-32 bits	16-32 bits	16-32 bits
Technologie	0.35 µm	0.25 µm	80 nm	65 nm
Fréquence d'horloge	16 MHz	84 MHz	7.37 MHz	48 MHz
Technique d'injection	EMFI	EMFI	LFI 1 064 nm	LFI 1 064 nm
Type de fautes	*bit set* ou *bit reset*	*bit set* ou *bit reset*	*bit set*	*bit set*
Précision spatiale	16 bits	de 1 à 128 bits	de 1 à 5 bits	1 ou 2 bits
Précision temporelle	jusqu'à 6 cycles	1 cycle	au moins 7 cycles	au moins 7 cycles
Répétabilité	très forte	très forte	très forte	très forte

TABLE 4.1 – Synthèse des modèles de fautes sur les instructions et les données en transfert entre la mémoire Flash et le cœur de calcul des microcontrôleurs étudiés.

4.1.2 Modèles de fautes EM

Au niveau matériel, une EMP se traduit par l'injection de fautes de type *bit set* ou de type *bit reset* sur les instructions ou les données en transfert entre la mémoire Flash et une mémoire tampon, aussi appelée *buffer*, dédiée aux résultats des opérations de lecture en mémoire. Le nombre et l'indice des bits corrompus dépendent par ailleurs de la position de la sonde d'injection par rapport à la mémoire.

Dans le cas du microcontrôleur SAM3X8E, de un à cent vingt-huit bits du *buffer* de préchargement des données peuvent être mis soit à `'0'`, soit à `'1'` avec une granularité d'un octet et une répétabilité de 100 %. En ajustant la position de la sonde d'injection, les fautes peuvent être injectées soit sur les bits de poids fort, soit sur les bits de poids faible des quatre mots consécutifs de 32 bits chargés dans le *buffer* de données.

Dans le cas du microcontrôleur ATmega328P, un saut d'instructions peut affecter de une à six instructions consécutives avec une répétabilité de 100 %.

4.1.3 Modèles de fautes laser

Nous avons démontré que le mécanisme d'injection d'un photocourant par illumination laser s'applique aux jonctions drains-substrats des transistors à grille flottante polarisées en inverse à l'intérieur d'une mémoire NOR Flash. Lorsqu'une colonne mémoire NOR Flash est illuminée pendant une opération de lecture, ce photocourant augmente le courant de lecture qui traverse la colonne. Au niveau matériel, une impulsion laser se traduit ainsi par l'injection de fautes de type *bit set* sur les instructions et les données en transfert entre la mémoire Flash et le *buffer* de lecture de la mémoire.

Dans le cas du microcontrôleur STM32F100RB, de un à quatre bits adjacents peuvent être mis à `'1'` avec une répétabilité de 100 %. Cinq bits adjacents ont été mis à `'1'` avec une répétabilité de 50 %. Le nombre de bits modifiés par une injection de fautes peut être choisi en ajustant précisément la taille de la zone d'effet d'une perturbation laser. Les colonnes mémoire affectées par une perturbation laser et, par suite, l'indice

des bits modifiés, peuvent également être choisis en ajustant la position de l'objectif du laser. À l'instar des résultats obtenus par DUTERTRE, RIOM, POTIN et al. [69], nous avons mis en évidence une corrélation entre la durée d'une impulsion laser et le nombre d'instructions ou de données consécutives dans lesquelles une faute est injectée. Nous avons illustré cette propriété pour une à sept instructions consécutives.

Un mécanisme similaire au mécanisme décrit sur le microcontrôleur STM32F100RB a été observé sur le microcontrôleur SAMD21G18A. Dans le cas du microcontrôleur SAMD21G18A, une faute de type *bit set* peut être injectée sur un ou deux bits en ajustant la position de l'objectif et la taille de la zone d'effet du faisceau laser. Deux bits dont les indices sont congrus modulo 16 ont ainsi été mis simultanément à `'1'` avec une répétabilité de 100 %. Un seul bit parmi ces deux bits a été mis à `'1'` avec une répétabilité de 50 % à 75 %.

Dans les sections suivantes, nous détaillons plusieurs cas d'étude pratiques et un cas d'étude théorique dans lesquels ces modèles de fautes sont utilisés pour compromettre l'implémentation de plusieurs algorithmes de sécurité.

4.2 Attaques par injection de fautes EM

Dans cette section, nous présentons trois attaques par injection de fautes EM, dont une sur un algorithme de vérification d'un code PIN et deux sur un algorithme de chiffrement AES-128. Ces attaques ont été implémentées en pratique sur les microcontrôleurs ATmega328P et SAM3X8E grâce à l'analyse détaillée des modèles de fautes par injection EM réalisée dans le chapitre 2. Elles ont fait l'objet de deux publications : [112] pour l'analyse et l'exploitation d'un saut d'instruction et [111] pour l'analyse et l'exploitation des corruptions d'un *buffer* de préchargement des données.

4.2.1 Attaque sur un algorithme de vérification d'un code PIN

Un numéro d'identification personnel (*Personal Identification Number* – PIN) est un code secret composé généralement de quatre digits. Ce code permet, par exemple, de déverrouiller un circuit électronique, comme une carte SIM, ou d'authentifier l'utilisateur d'une carte banquaire lors d'une transaction. L'utilisation d'un circuit protégé par un code PIN nécessite ainsi la connaissance de ce code.

Un utilisateur dispose initialement d'un nombre d'essais limité pour déverrouiller le circuit. À chaque comparaison entre le code PIN de référence – mémorisé dans le circuit – et le code PIN entré par l'utilisateur, un essai est soustrait au nombre d'essais disponibles. La valeur du code PIN de référence est ainsi protégée contre une attaque exhaustive, aussi appelée *brute-force* dans la littérature anglo-saxonne.

Dans les paragraphes suivants, nous décrivons l'exploitation d'un saut d'instruction causé par une EMP sur le microcontrôleur ATmega328P pour contourner le nombre d'essais maximal d'un algorithme de vérification d'un code PIN.

4.2.1.A Présentation de l'algorithme verifyPIN

Un algorithme de vérification exécuté par le circuit compare, à chaque tentative d'authentification, le code PIN entré par l'utilisateur avec un code PIN de référence mémorisé dans une mémoire non-volatile du circuit. Si les codes correspondent, le circuit est déverrouillé et le nombre d'essais est réinitialisé. En revanche, si le nombre d'essais est épuisé, le circuit reste verrouillé et aucune autre tentative ne peut être réalisée.

La collection de logiciels FISSC (*Fault Injection and Simulation Secure Collection*) est dédiée à l'analyse et à la simulation des attaques par injection de fautes sur des codes sécurisés [64]. Nous avons choisi d'implémenter en pratique une attaque de vérification d'un code PIN sur l'algorithme `"verifyPIN"` issu de la collection FISSC. Une implémentation en langage C de l'algorithme est reportée ci-dessous dans le Code source 4.1 :

```
1  #define BOOL_TRUE 0xAA
2  #define BOOL_FALSE 0x55
3
4  uint8_t g_userPIN[4];
5  uint8_t g_cardPIN[4];
6
7  uint8_t g_authenticated = BOOL_FALSE;
8  int8_t g_ptc = 3;
9
10 void verifyPIN()
11 {
12     g_authenticated = BOOL_FALSE;
13     if(g_ptc > 0)
14     {
15         g_ptc--;
16         if(byteArrayCompare(g_userPIN, g_cardPIN, 4) == BOOL_TRUE)
17         {
18             g_ptc = 3;
19             g_authenticated = BOOL_TRUE;
20         }
21     }
22 }
```

CODE SOURCE 4.1 – Implémentation de la fonction `"verifyPIN"` en langage C.

Les valeurs booléennes « vrai » et « faux » sont représentées sous la forme de booléens durcis contre l'injection de fautes `"BOOL_TRUE"` (ligne 1) et `"BOOL_FALSE"` (ligne 2).

La variable `"g_ptc"` mémorise le nombre d'essais dont dispose l'utilisateur. La variable `"g_authenticated"` indique le statut de l'authentication. Le tableau `"g_userPIN"` est utilisé pour mémoriser le code PIN entré par l'utilisateur lors d'une tentative d'authentification. Le tableau `"g_cardPIN"`, enfin, contient une copie du code PIN de référence mémorisé dans le circuit.

L'utilisateur dispose initialement de trois essais (ligne 8). À chaque appel de la fonction `"verifyPIN"`, l'algorithme réinitialise la variable d'authentification à « faux » (ligne 12) et vérifie le nombre d'essais restants (ligne 13).

Si le nombre d'essais restants est supérieur à zéro, un essai lui est soustrait (ligne 15) et la fonction `"byteArrayCompare"` est appelée (ligne 16). Cette fonction prend en arguments deux tableaux de digits – le code PIN entré par l'utilisateur et le code PIN de référence – et le nombre de digits à comparer. Si les codes PIN sont identiques, le nombre d'essais est réinitialisé (ligne 18) et l'utilisateur est authentifié (ligne 19).

En revanche, si le nombre d'essais est inférieur ou égal à zéro, la fonction `"byteArrayCompare"` n'est pas appelée et la variable d'authentification garde sa valeur réinitialisée. Par conséquent, l'utilisateur n'est pas authentifié.

La fonction `"byteArrayCompare"` implémente par ailleurs un premier niveau de protection contre les attaques matérielles par observation et par perturbation. Une implémentation en langage C de la fonction est reportée ci-dessous dans le Code source 4.2 :

```
1  uint8_t byteArrayCompare(uint8_t *a1, uint8_t *a2, uint8_t size)
2  {
3      int i;
4      uint8_t status = BOOL_FALSE;
5      uint8_t diff = BOOL_FALSE;
6      for(i=0; i < size; i++)
7          if(a1[i] != a2[i])
8              diff = BOOL_TRUE;
9      if(i != size)
10         countermeasure();
11     if(diff == BOOL_FALSE)
12         status = BOOL_TRUE;
13     else
14         status = BOOL_FALSE;
15     return status;
16 }
```

CODE SOURCE 4.2 – Implémentation de la fonction `"byteArrayCompare"` en langage C.

La comparaison des deux tableaux est implémentée en temps constant dans une boucle `"for"` (lignes 6–8). L'observation de la durée de l'exécution de l'algorithme ne permet donc pas de déduire la valeur du code PIN de référence. L'algorithme est ainsi protégé contre les attaques par SPA introduites par KOCHER [97].

À l'issue de la comparaison, l'algorithme vérifie que tous les digits du code PIN de l'utilisateur ont été comparés à ceux du code PIN de référence en testant la valeur du compteur d'itérations `"i"` (ligne 9) . Si le nombre de digits vérifiés diffère du nombre de digits du code PIN, une alerte est envoyée au système (ligne 10) et la vérification échoue. L'algorithme est ainsi protégé contre une attaque par injection de fautes qui empêcherait la comparaison de deux digits dans l'objectif de réaliser une attaque exhaustive sur le code PIN.

Cependant, ces contre-mesures ne protègent pas intégralement l'algorithme `"verifyPIN"` contre les attaques matérielles. DUREUIL, PETIOT, POTET et al. [64] identifient, en effet, plusieurs chemins d'attaques en faute en simulant un saut d'instruction ou une corruption d'instruction. En particulier, l'injection d'une faute pendant la vérification du compteur d'essais permet à un attaquant de disposer d'un nombre d'essais illimité, à condition de pouvoir reproduire la faute avec une forte répétabilité.

4.2.1.B Description du chemin d'attaque

Le programme "verifyPIN" est compilé pour le microcontrôleur 8 bits ATmega328P. La traduction du programme en langage assembleur AVR est présentée sur le Code source 4.3 suivant.

Signalement du début du programme (ligne 3). La deuxième GPIO du port B du microcontrôleur est tout d'abord basculée à l'état haut. Ce basculement constitue le premier front du signal *synchronization* qui sert de commande pour le générateur d'impulsions de tension.

Initialisation du statut de l'utilisateur (ligne 5–8). La variable d'authentification "g_authenticated" est ensuite initialisée à la valeur « faux ». Le registre "r17" indique qu'une étape de l'authentification a échoué. Il est donc initialisé à la valeur « faux » avant l'exécution de la routine de comparaison "byteArrayCompare".

Vérification du nombre d'essais restants (ligne 10–12). La valeur du compteur d'essais est chargée dans le registre "r20". L'instruction "cp r1, r20" met à jour les drapeaux du registre d'état du microcontrôleur en fonction du résultat de la comparaison entre le registre "r20" et le registre "r1". Par convention, la valeur du registre "r1" est fixée à zéro. Deux cas de figure se présentent alors :

— si le compteur d'essais est strictement supérieur à zéro, l'instruction de branchement "brge" n'a pas d'effet et l'exécution se poursuit normalement ;

— en revanche, si le compteur d'essais est inférieur ou égal à zéro, l'instruction de branchement "brge" redirige l'exécution du programme à l'adresse du label ".end_verifyPIN". La routine de comparaison "byteArrayCompare" n'est donc pas exécutée et la variable d'authentification garde sa valeur d'initialisation, à savoir la valeur « faux ».

L'injection d'un saut d'instruction sur l'instruction de branchement permet ainsi de contourner le nombre limite d'essais en forçant l'exécution de la routine "byteArrayCompare".

Signalement du début de la comparaison des digits (ligne 13). La troisème GPIO du port B du microcontrôleur est basculée à l'état haut. Ce basculement constitue le premier front du signal *core* qui indique le début d'une vérification.

Mise à jour du nombre d'essais restants (ligne 15–16). Le compteur d'essais mémorisé dans le registre "r20" est décrémenté et mémorisé en SRAM dans la variable "g_ptc" avant l'exécution de la routine "byteArrayCompare".

Signalement de la fin de la comparaison des digits (ligne 41). La troisième GPIO du port B du microcontrôleur est basculée à l'état bas. Ce basculement constitue le second front du signal *core* qui indique la fin d'une vérification.

```
1  verifyPIN:
2      push    r2-r17, r28, r29 ; prolog
3      sbi     0x05, 2 ; set sync trigger
4  @   g_authenticated = BOOL_FALSE;
5      ldi     r16, 0x55
6      sts     g_authenticated, r16
7  @   diff = BOOL_FALSE
8      ldi     r17, 0x55
9  @   if(g_ptc > 0){
10     lds     r20, g_ptc
11     cp      r1, r20
12     brge    .end_verifyPIN
13     sbi     0x05, 3  ; set core trigger
14 @   g_ptc--;
15     subi    r20, 0x01
16     sts     g_ptc, r20
17 ; -- fixed time comparison loop --
18 @   if(byteArrayCompare(g_userPIN, g_cardPIN) == BOOL_TRUE){
19     movw    YL, ZL
20     adiw    YL, 0x04
21     movw    YH, ZH
22 .comparison_loop:
23     ld      r18, Z+
24     ld      r19, X+
25 @   if(a1[i] != a2[i]){
26     cpse    r18, r19
27 @       diff = BOOL_TRUE;
28     ldi     r17, 0xAA
29 @   }
30     cp      ZL, YL
31     cpc     ZH, YH
32     brne    .comparison_loop
33 ; -- check for fault in loop --
34 @   if(i != size){
35     cp      ZL, YL
36     cpc     ZH, YH
37     breq    .+2
38 @       diff = BOOL_TRUE;
39     ldi     r17, 0xAA
40 @   }
41     cbi     0x05, 3 ; clear core trigger
42 ; -- update status based on diff value --
43 @   if(byteArrayCompare(g_userPIN, g_cardPIN) == BOOL_TRUE){
44     cpi     r17, 0x55
45     brne    .+6
46 @       g_authenticated = BOOL_TRUE;
47     ldi     r16, 0xAA
48     sts     g_authenticated, r16
49 @       g_ptc = 3;
50     ldi     r20, 0x03
51     sts     g_ptc, r20
52     cbi     0x05, 2 ; clear sync trigger
53 @   }
54 .end_verifyPIN:
55     pop     r2-r17, r28, r29 ; epilog
56     ret
```

CODE SOURCE 4.3 – Implémentation du programme "verifyPIN" en code assembleur AVR.

Mise à jour du statut de l'utilisateur (ligne 44–51). L'instruction `"cp r17, 0x55"` met à jour les drapeaux du registre d'état en fonction du résultat de la comparaison entre le registre `"r17"` et la valeur « faux ». Deux cas de figure se présentent alors :

— si une étape de l'authentification a échoué, le registre `"r17"` contient la valeur « vrai ». L'instruction de branchement `"brne"` redirige alors l'exécution à la fin du programme. La variable d'authentification garde donc sa valeur d'initialisation, à savoir la valeur « faux » ;

— dans le cas contraire, le registre `"r17"` contient la valeur « faux » et l'exécution du programme se poursuit après l'instruction de branchement. La variable d'authentification prend alors la valeur « vrai » et le compteur d'essais est réinitialisé à trois essais. L'utilisateur est authentifié.

Remarquons que l'injection d'un saut d'instruction pendant la mise à jour du statut de l'utilisateur permet de forcer l'authentification sans connaître le code PIN de référence.

Signalement de la fin du programme (ligne 52). La deuxième GPIO du port B du microcontrôleur est enfin basculée à l'état bas. Ce basculement constitue le deuxième front du signal *synchronization* qui indique la fin de l'exécution du programme.

4.2.1.C Résultats expérimentaux

L'attaque réalisée consiste à transformer l'instruction de branchement `"brge"` (ligne 12) en instruction `"nop"` afin de contourner la vérification du nombre d'essais. Le dispositif d'injection utilisé est identique au dispositif utilisé dans le chapitre 2 pour injecter des sauts d'instructions grâce à une EMP. La sonde d'injection utilisée est la sonde 6N/∅500 composée d'une bobine de six tours de fil de cuivre autour d'un noyau de ferrite 78 aminci par abrasion mécanique. Le diamètre du fil de cuivre de la bobine est d'environ 200 µm, celui du noyau de ferrite d'environ 500 µm.

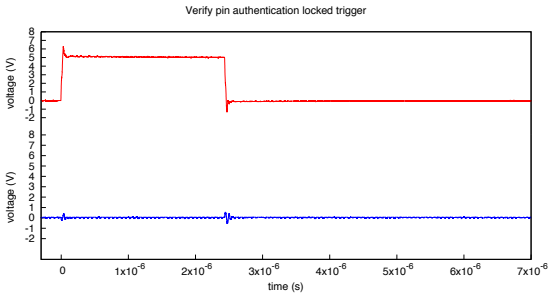
L'injection est réalisée par la face avant du microcontrôleur 8 bits ATmega328P, au niveau de la mémoire Flash, conformément aux résultats obtenus dans le chapitre 2. Les microcontrôleurs utilisés sont décapsulés par la face avant, de sorte à rapprocher au maximum la sonde d'injection du circuit intégré.

Les sauts d'instructions causés par une EMP sont obtenus au niveau de la zone indiquée par une étoile bleue sur l'image par microscopie de la face avant du microcontrôleur ATmega328P (voir Figure 2.5 du chapitre 2).

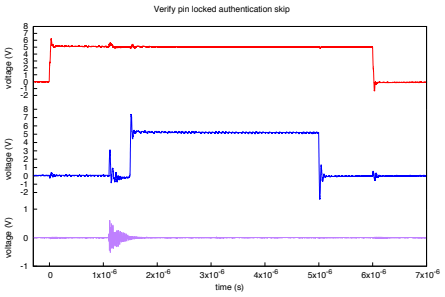
Les signaux *synchronization* et *core* générés par le circuit de test sont reportés sur la Figure 4.1. Le signal *synchronization* (en rouge) délimite l'exécution du programme `"verifyPIN"` et le signal *core* (en bleu) délimite l'exécution de la routine `"byteArrayCompare"`. La perturbation électromagnétique est synchronisée sur le signal *synchronization*.

Lorsque le compteur d'essais `"g_ptc"` est à zéro (Figure 4.1a), la routine `"byteArrayCompare"` n'est pas exécutée et le signal *core* (en bleu) reste à l'état bas.

La Figure 4.1b illustre une exécution forcée de la routine `"byteArrayCompare"` lors de l'injection d'un saut d'instruction sur l'instruction `"brge"` (ligne 12 du Code source 4.3).



(a) Entrée refusée dans la routine `"byteArrayCompare"`



(b) Entrée forcée dans la routine `"byteArrayCompare"`

FIGURE 4.1 – Illustration d'une attaque exhaustive sur un code PIN grâce à l'injection d'un saut d'instruction.

L'image de la perturbation électromagnétique par une antenne d'observation est reportée en violet sur la figure. Dans ces conditions, le code PIN rentré par un attaquant est comparé au code PIN de référence malgré l'épuisement du nombre d'essais.

Les conclusions formulées dans le chapitre 2 nous ont permis d'ajuster les paramètres d'injection de sorte à forcer l'exécution de la routine `"byteArrayCompare"` avec 100 % de répétabilité. Une exploration exhaustive des 10^4 codes PIN possibles permet alors d'extraire la valeur du code PIN de référence. Ce cas d'étude démontre ainsi l'efficacité des techniques d'injection de fautes par impulsion EM pour empêcher l'exécution d'un test conditionnel dans le flot de contrôle d'un algorithme de sécurité.

4.2.2 Implémentation d'une PFA

Les paragraphes suivants détaillent l'implémentation en pratique d'une attaque par PFA sur le microcontrôleur SAM3X8E grâce à une EMFI.

4.2.2.A Attaque par analyse de fautes persistantes

En 2018, ZHANG, LOU, ZHAO et al. [180] introduisent une nouvelle famille d'attaques contre les algorithmes de chiffrement par blocs appelée PFA. Les auteurs supposent qu'un attaquant peut injecter une faute sur un élément d'une table de correspondances (une SBOX par exemple dans le cas d'un AES) et que l'effet de la faute persiste jusqu'à la remise à zéro du circuit corrompu. Les auteurs supposent, en outre, que l'indice de l'élément corrompu et sa valeur après l'injection d'une faute sont connus. Une publication ultérieure démontre que cette dernière condition n'est pas nécessaire si l'indice et la valeur de la faute sont extraits par une analyse statistique des textes chiffrés [181].

L'attaque est alors réalisée en deux temps. Dans un premier temps, l'attaquant répète la procédure d'injection d'une faute persistante dans la table de correspondances jusqu'à satisfaire les conditions du schéma d'attaque. Dans un second temps, environ 2000 chiffrements sont réalisés avec la SBOX corrompue et les textes chiffrés sont collectés. Il n'est pas nécessaire pour l'attaquant de collecter les textes chiffrés corrects associés à chacun des textes chiffrés corrompus. L'attaque diffère, en ce point, des attaques par DFA.

Parmi les chiffrements réalisés, ceux qui utilisent l'élément corrompu dans la SBOX introduisent un biais statistique dans la distribution des octets du dernier état de l'algorithme de chiffrement. Une analyse statistique des textes chiffrés collectés dans la deuxième phase de l'attaque permet alors d'extraire la valeur de la dernière clé de ronde.

Les auteurs ont implémenté une injection de fautes persistantes en pratique sur une mémoire DRAM DDR3 cadencée à 1.3 GHz en utilisant le *bug* Rowhammer [96] qui exploite le couplage capacitif entre les lignes de cellules adjacentes dans une DRAM pour modifier matériellement l'état logique d'une ou plusieurs cellules mémoire par des accès en lecture répétés aux lignes adjacentes à ces cellules. Si toutes les technologies mémoire sont potentiellement affectées par le bug Rowhammer, et en particulier les technologies DRAM, aucune exploitation n'a été démontrée jusqu'alors sur une mémoire NOR Flash ou sur une SRAM embarquée [123]. Cette technique d'exploitation se transpose donc difficilement sur les microcontrôleurs, car ces derniers ne possèdent pas de mémoire DRAM. Dans ces travaux, nous démontrons que la corruption d'un transfert de données par une injection de fautes EM permet d'implémenter en pratique une PFA.

4.2.2.B Description de l'attaque

Nous supposons qu'un microcontrôleur est programmé avec une implémentation logicielle d'un algorithme de chiffrement AES-128. Une implémentation classique de l'algorithme de chiffrement applique la transformation SUBBYTES grâce à une table de correspondances de 256 octets appelée SBOX. Cette implémentation peut être améliorée pour les architectures 32 bits grâce à quatre tables de correspondances, appelées TTABLES, qui permettent d'appliquer simultanément les transformations SUBBYTES, SHIFTROWS et MIXCOLUMNS [55].

Chaque TTABLE est composée de 256 colonnes de 32 bits. Chaque colonne s'exprime en fonction de l'indice a de la colonne et de l'élément $\mathrm{S}[a]$ d'indice a de la SBOX par l'équation 4.1, où l'opérateur "$\cdot$" désigne la multiplication polynomiale modulo `0x11B` dans le corps fini $\mathbb{F}_{2^8}$.

$$\mathrm{T}_0[a] = \begin{bmatrix} 02\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \\ 03\cdot\mathrm{S}[a] \end{bmatrix} \quad \mathrm{T}_1[a] = \begin{bmatrix} 03\cdot\mathrm{S}[a] \\ 02\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \end{bmatrix} \quad \mathrm{T}_2[a] = \begin{bmatrix} 01\cdot\mathrm{S}[a] \\ 03\cdot\mathrm{S}[a] \\ 02\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \end{bmatrix} \quad \mathrm{T}_3[a] = \begin{bmatrix} 01\cdot\mathrm{S}[a] \\ 01\cdot\mathrm{S}[a] \\ 03\cdot\mathrm{S}[a] \\ 02\cdot\mathrm{S}[a] \end{bmatrix} \tag{4.1}$$

Une implémentation alternative de l'algorithme n'utilise que la TTABLE T_0 pour réduire l'empreinte mémoire de l'algorithme. Les colonnes des TTABLES $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3$ sont alors calculées dynamiquement lors du chiffrement en appliquant une, deux, ou trois rotations sur les colonnes de la TTABLE T_0 [55].

Notons $\widetilde{\mathrm{T}_0}[v]$ la valeur d'une faute sur la colonne d'indice v de la TTABLE T_0. Cette faute est équivalente à l'injection d'une faute persistante $\widetilde{\mathrm{S}}[v]$ sur l'élément d'indice v d'une

SBOX si elle satisfait l'équation 4.2.

$$\widetilde{T_0}[v] = \begin{bmatrix} 02 \cdot \widetilde{S}[v] \\ 01 \cdot \widetilde{S}[v] \\ 01 \cdot \widetilde{S}[v] \\ 03 \cdot \widetilde{S}[v] \end{bmatrix} \tag{4.2}$$

C'est le cas en particulier si les 32 bits d'une colonne sont mis à `'0'`. Une mise à `'0'` de tous les bits de la colonne d'indice v de la TTABLE T_0 équivaut ainsi à une mise à `'0'` de tous les bits de l'élément d'indice v de la SBOX.

4.2.2.C Résultats expérimentaux

Dans cette section, nous évaluons la possibilité d'injecter une faute persistante sur la TTABLE de la fonction de chiffrement grâce à une EMP. Cette attaque a été implémentée en pratique sur le microcontrôleur non-sécurisé SAM3X8E. Ce dernier a été programmé avec une implémentation en langage assembleur de l'algorithme de chiffrement symétrique AES-128 proposée par SCHWABE et STOFFELEN. Cet algorithme est implémenté de sorte à n'utiliser qu'une seule TTABLE de 1 ko lors d'un chiffrement [157].

Dans notre attaque, nous supposons que les tables de correspondances sont chargées en SRAM après le démarrage du microcontrôleur. Cette technique permet d'améliorer le temps d'exécution de l'algorithme en réduisant la latence des accès en mémoire lors de l'utilisation des tables de correspondances. Un espace de 1 ko doit néanmoins être réservé dans la SRAM pour la TTABLE dédiée aux opérations de chiffrement.

Pour ce faire, le code source original de l'implémentation proposée par SCHWABE et STOFFELEN est modifié afin de placer les tables de correspondances dans une section identifiable que nous nommons `".rodata.tables"`. Cette modification est illustrée sur le Code source 4.4.

```
1  .section    .rodata.tables
2  .align      2
3  .type       AES_Te0,%object
4  AES_Te0:
5  @ AES_Te0[0], AES_Te0[1], AES_Te0[2], AES_Te0[3]
6  .word 0x63c6a563, 0x7cf8847c, 0x77ee9977, 0x7bf68d7b
7  @ AES_Te0[4], AES_Te0[5], AES_Te0[6], AES_Te0[7]
8  .word 0xf2ff0df2, 0x6bd6bd6b, 0x6fdeb16f, 0xc59154c5

   ⋮

73 @ AES_Te0[252], AES_Te0[253], AES_Te0[254], AES_Te0[255]
74 .word 0xb07bcbb0, 0x54a8fc54, 0xbb6dd6bb, 0x162c3a16
75
76 .section    .text
```

CODE SOURCE 4.4 – Attribution d'une section identifiable aux TTABLES.

La section `".rodata.tables"` est tout d'abord programmée dans la mémoire Flash du

microcontrôleur qui mémorise le micrologiciel en l'absence d'alimentation électrique. À chaque remise à zéro du circuit, elle est transférée dans la SRAM embarquée par le code d'initialisation du microcontrôleur. Suite à cette opération, chacune des tables de correspondance peut être adressée en SRAM.

La section `".rodata.tables"` possède donc deux adresses : l'adresse de chargement en mémoire Flash où elle est initialement mémorisée, ou LMA (*Load Memory Address*), et l'adresse virtuelle en SRAM utilisée par le micrologiciel, ou VMA (*Virtual Memory Address*). Ces adresses sont déterminées à partir du script d'assemblage du micrologiciel dont un extrait est présenté sur le Code source 4.5.

```
1  MEMORY
2  {
3    flash (rx)  : ORIGIN = 0x00080000, LENGTH = 0x00080000
4    sram  (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00018000
5  }
6
7
8  SECTIONS
9  {
10     /* ... */
11     _etext = .;
12     .data : ALIGN(4)
13     {
14         _srelocate = .;
15         *(.rodata.tables)
16         . = ALIGN(4);
17         /* ... */
18         _erelocate = .;
19     } > sram AT > flash
20     /* ... */
21 }
```

CODE SOURCE 4.5 – Script d'assemblage d'un programme en vue d'une relocalisation des tables de correspondance en SRAM.

La section `".rodata.tables"` est placée dans la section `".data"` définie dans le script d'assemblage (ligne 15). La commande `"> sram AT > flash"` (ligne 19) attribue à la section `".data"` une VMA dans la plage d'adresses de la SRAM, définie ligne 4, et une LMA dans la plage d'adresses de la mémoire Flash, définie ligne 3. L'adresse du début de la section en mémoire Flash est assignée par ailleurs au symbole `"_etext"` (ligne 11), celle du début de la section en SRAM au symbole `"_srelocate"` (ligne 14) et celle de la fin de la section en SRAM au symbole `"_erelocate"` (ligne 18).

Ces symboles sont utilisés par le code d'initialisation exécuté à chaque remise à zéro du microcontrôleur pour transférer les tables de correspondances de la mémoire Flash à la SRAM. La routine en charge de ce transfert est présentée en langage C sur le Code source 4.6. L'instruction `"*pDest++ = *pSrc++;"` charge un mot de 32 bits à l'adresse contenue par le pointeur `"pSrc"` et mémorise ce mot à l'adresse contenue par le pointeur `"pDest"` avant d'incrémenter de quatre octets l'adresse des deux pointeurs (ligne 5). Ce transfert est la cible de notre attaque.

```
uint32_t *pSrc, *pDst;
pSrc = &_etext;
pDest = &_srelocate;
for(; pDest < &_erelocate ;)
    *pDest++ = *pSrc++;
```

CODE SOURCE 4.6 – Implémentation en langage C de la copie des tables de correspondances de la mémoire Flash à la SRAM dans le code d'initialisation du microcontrôleur.

Dans une implémentation en langage assembleur obtenue en compilant le Code source 4.6 avec le niveau d'optimisation `"-O3"`, le premier accès en mémoire Flash à une adresse alignée sur un bloc de 128 bits provoque un préchargement de 4 mots de 32 bits dans le *buffer* de données de la mémoire Flash. Une EMP pendant le préchargement de quatre colonnes consécutives dans le *buffer* de données de la mémoire Flash permet ainsi de mettre à zéro 4 colonnes de la TABLE. Puisque la TABLE corrompue est mémorisée en SRAM, les fautes persistent jusqu'à la remise à zéro du circuit.

Nous supposons que l'attaquant est capable de synchroniser précisément la perturbation électromagnétique sur l'exécution du code d'initialisation du microcontrôleur. Dans le cadre de ces travaux, nous utilisons à cet effet un signal logique généré sur une GPIO du microcontrôleur par le code d'initialisation. L'amplitude de l'impulsion de tension est fixée à 200 V et la largeur de l'impulsion à 6 ns. La position de la sonde est fixée conformément aux résultats du chapitre 2, de sorte à mettre à `'1'` ou à `'0'` tous les bits du *buffer* de données de la mémoire Flash. Un balayage du délai est réalisé par pas de 1 ns. Après chaque injection, une relecture de la SRAM est réalisée pour identifier les paramètres d'injection qui satisfont aux prérequis d'une injection de fautes persistantes dans la TABLE dédiée aux opérations de chiffrement.

Nous avons vérifié en pratique que tous les bits de quatre colonnes consécutives de 32 bits dans la TABLE peuvent être mis à `'0'` avec 100 % de répétabilité sur cent répétitions pour un délai d'injection constant. Ce modèle de faute permet d'implémenter les conditions d'une PFA dans le cas où 4 octets de la SBOX sont corrompus. Dans ce cas de figure, l'analyse réduit théoriquement l'entropie de la clé secrète à 32 bits sur lesquels une recherche exhaustive peut être réalisée [180].

4.2.3 Attaque de Biham et Shamir

Un algorithme de chiffrement symétrique permet de protéger la confidentialité, l'intégrité et l'authenticité des échanges entre un microcontrôleur et une tierce partie. Le secret des échanges repose intégralement sur le secret d'une clé de chiffrement connue seulement du microcontrôleur et de la tierce partie, conformément au principe de Kerckhoffs [92]. L'algorithme de chiffrement, quant à lui, est publique.

Dans les microcontrôleurs qui ne disposent pas d'une EEPROM, la clé secrète est mémorisée dans la mémoire Flash embarquée. Or, cette clé doit être transférée hors de la mémoire Flash après le démarrage du circuit si une série de chiffrements est réalisée. L'opération de lecture de la clé secrète en mémoire Flash est alors vulnérable à l'injection d'une faute EM.

Dans les paragraphes qui suivent, nous présentons tout d'abord l'attaque de BIHAM et SHAMIR sur la clé secrète d'un cryptosystème inconnu. Nous adaptons ensuite cette attaque au modèle de faute EM décrit dans ces travaux. Nous décrivons enfin les résultats expérimentaux obtenus par injection de fautes EM sur le microcontrôleur SAM3X8E pour extraire une clé de chiffrement de 128 bits enregistrée dans la mémoire Flash du microcontrôleur.

4.2.3.A Présentation de l'attaque de Biham et Shamir

Dans leur article pionnier de 1997, BIHAM et SHAMIR [30] proposent plusieurs schémas d'attaques permettant d'exploiter les effets d'une faute sur un algorithme de chiffrement afin d'en extraire la clé secrète. Parmi ces attaques, les auteurs proposent une variante des attaques par DFA qui exploite des fautes injectées sur la clé de chiffrement.

Dans cette attaque, les auteurs supposent, tout d'abord, qu'un attaquant dispose d'un accès physique à un circuit intégré à l'intérieur duquel une clé secrète est mémorisée. Ils supposent, en outre, que l'attaquant peut provoquer le chiffrement d'un même texte clair avec plusieurs clés de chiffrement et observer le résultat du chiffrement. Chaque clé de chiffrement est ainsi associée à un texte chiffré. Ils supposent enfin que l'attaquant est capable d'injecter des fautes mono-bit de type *bit set*, ou *bit reset*, dans la mémoire non-volatile du circuit intégré où est mémorisée la clé secrète. Chaque faute injectée dans la clé persiste donc jusqu'à la reprogrammation du circuit.

Nous choisissons d'illustrer l'attaque dans le cas d'un modèle de faute de type *bit reset*. Nous nous plaçons, par ailleurs, dans le cas où la fonction de chiffrement et le texte clair sont connus de l'attaquant.

Première phase de l'attaque. L'attaquant chiffre le même texte clair p_0, d'abord avec la clé originale, puis avec chacune des clés obtenues par l'injection d'une faute dans la mémoire non-volatile, jusqu'à chiffrer le texte clair avec une clé dont tous les bits ont été mis à `'0'`. Remarquons à ce stade que deux clés de chiffrement différentes – obtenues en injectant un nombre différent de fautes de type *bit reset* pour chaque clé – produisent des textes chiffrés différents. L'attaquant constitue ainsi une suite de textes chiffrés fautés $(c_n, c_{n-1}, \ldots, c_0)$ associés à une suite de clés inconnues $(k_n, k_{n-1}, \ldots, k_0)$, où la clé k_n correspond à la clé originale et la clé k_i possède exactement i bits à `'1'`. Avec cette notation, l'indice d'une clé désigne à la fois sa position dans la suite et son poids de Hamming. La quantité n désigne ainsi le poids de hamming de la clé secrète, inférieur à 128 pour une clé de 128 bits. La clé k_0 désigne quant à elle la clé dont tous les bits sont à `'0'`.

Deuxième phase de l'attaque. L'objectif de l'attaquant est ensuite de reconstruire la suite de clés $k_0, k_1, \ldots, k_n$ à partir de la suite de textes chiffrés $c_0, c_1, \ldots, c_n$, obtenue dans la première phase de l'attaque, afin d'extraire la clé secrète k_n. Pour ce faire, il extrait, à chaque étape de l'algorithme, la position d'un bit à `'1'` dans la clé secrète. Supposons, en effet, que l'attaquant connaît la clé k_i pour un entier i inférieur à $n-1$. Remarquons alors que la clé k_{i+1} est obtenue en positionnant un bit à `'1'` parmi les bits à `'0'` dans

la clé k_i. Puisque ce bit n'est pas connu *a priori*, une hypothèse est formulée sur son indice. L'attaquant constitue ainsi une clé candidate pour la clé k_{i+1}. Si le résultat du chiffrement du texte clair p_0 avec la clé candidate est le texte chiffré c_{i+1}, cette clé est, en toute vraisemblance, la clé k_{i+1}. Sinon, une autre hypothèse est formulée, jusqu'à identifier l'indice du bit corrompu. L'attaquant construit ainsi la clé k_{i+1} à partir de la clé k_i. Puisque la clé k_0 est connue, l'attaquant peut remonter, par récurrence, à la clé secrète k_n. Les premières étapes de cette procédure, en partant de la clé k_0, sont illustrées sur la Table 4.2.

i	k_i	$\rightarrow c_i$
0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow c_0$
1	00 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow c_1$
2	00 00 00 00 05 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow c_2$
3	00 80 00 00 05 00 00 00 00 00 00 00 00 00 00 00	$\rightarrow c_3$
4	00 80 00 00 05 00 20 00 00 00 00 00 00 00 00 00	$\rightarrow c_4$
⋮		
n (clé originale)	XX 80 XX XX 05 XX 20 XX XX XX XX XX XX XX XX XX	$\rightarrow c_n$

TABLE 4.2 – Reconstruction d'une clé secrète dans l'attaque de BIHAM et SHAMIR.

La principale difficulté de l'attaque originale de BIHAM et SHAMIR réside dans l'implémentation en pratique du modèle de faute proposé par les auteurs [22]. Ces derniers supposent, en effet, qu'un attaquant peut fixer la clé secrète à une valeur connue avec une précision au bit près en injectant des fautes qui persistent jusqu'à la reprogrammation de la mémoire non-volatile. Or, un tel modèle de faute nécessite *a priori* une forte résolution spatiale. Son implémentation en pratique n'est donc pas attestée par l'état de l'art.

4.2.3.B Variante de l'attaque dans le cas d'une injection de fautes EM

Nous proposons une variante de l'attaque de BIHAM et SHAMIR adaptée au modèle de faute EM sur le microcontrôleur SAM3X8E détaillé dans le chapitre 2. Cette variante se fonde sur le modèle de menace suivant :

1. L'attaquant dispose d'un accès physique au microcontrôleur. Une clé secrète de 128 bits est enregistrée dans la mémoire Flash du circuit.
2. L'attaquant peut provoquer le transfert de la clé secrète de la mémoire Flash vers le *buffer* de données de 128 bits et synchroniser la génération d'une perturbation EM sur le chargement de la clé.
3. L'attaquant peut chiffrer un même texte clair avec la clé secrète et observer le résultat du chiffrement.

Sous ces conditions :

— des fautes de type *bit reset* ou *bit set* peuvent être injectées grâce à une EMP pendant le transfert de la clé secrète entre la mémoire Flash et le *buffer* de données sans modifier le contenu de la mémoire Flash ;

— de 1 à 128 bits de la clé secrète peuvent être progressivement mis à `'0'` (ou à un) en ajustant la position de la sonde d'injection, de sorte qu'entre deux positions différentes, au plus d bits supplémentaires soient corrompus.

La quantité d dépend notamment du pas de déplacement de la sonde d'injection. Elle est estimée à 16 bits à partir des résultats expérimentaux présentés dans la section 2.4.

Un cas particulier de corruption d'une clé de 128 bits avec ce modèle de faute est illustré sur la Table 4.3 pour des fautes de type *bit reset*. L'octet d'indice i dans la clé est représenté en notation hexadécimale dans la colonne B_i du tableau. En reprenant les notations précédentes, nous notons $k_{i_n}, k_{i_{n-1}}, \ldots, k_{i_0}$ la suite de clés générées lors de la mise à `'0'` progressive des bits de la clé originale k_{i_n} grâce à une série d'injections de fautes EM. La suite d'indices $i_n, i_{n-1}, \ldots, i_0$ est une suite entière strictement décroissante dont l'élément i_0 a la valeur 0. L'indice i_j désigne donc le poids de Hamming de la clé k_{i_j} et la notation k_{i_0} la clé dont tous les bits sont à `'0'`.

			Clés fautées (modèle de faute *bit reset*)															
m	y_m	k_{i_m}	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}
8	y_8	k_{i_8}	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX
7	y_7	k_{i_7}	X0	XX	XX	XX	X0	XX	XX	XX	X0	XX	XX	XX	X0	XX	XX	XX
6	y_6	k_{i_6}	00	XX	XX	XX	00	XX	XX	XX	00	XX	XX	XX	00	XX	XX	XX
5	y_5	k_{i_5}	00	X0	XX	XX	00	X0	XX	XX	00	X0	XX	XX	00	X0	XX	XX
4	y_4	k_{i_4}	00	00	XX	XX	00	00	XX	XX	00	00	XX	XX	00	00	XX	XX
3	y_3	k_{i_3}	00	00	X0	XX	00	00	X0	XX	00	00	X0	XX	00	00	X0	XX
2	y_2	k_{i_2}	00	00	00	XX	00	00	00	XX	00	00	00	XX	00	00	00	XX
1	y_1	k_{i_1}	00	00	00	X0	00	00	00	X0	00	00	00	X0	00	00	00	X0
0	y_0	k_{i_0}	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

TABLE 4.3 – Mise à `'0'` progressive des bits d'une clé de 128 bits en commençant par les bits de poids faible de chaque mot de 32 bits. La notation `X` désigne les semi-octets qui correspondent à ceux de la clé originale. La coordonnées y_m de la sonde d'injection EM est ajustée à chaque itération.

À l'instar de l'attaque originale de BIHAM et SHAMIR, l'attaquant peut reconstruire la suite de clés inconnues $k_{i_0}, k_{i_1}, \ldots, k_{i_n}$ en reconstruisant progressivement la suite des différences entre deux clés successives. Chaque différence $k_{i_{j+1}} \oplus k_{i_j}$ correspond au masque des fautes de type *bit reset* injectées sur la clé $k_{i_{j+1}}$ afin d'obtenir la clé k_{i_j} pour une position donnée de la sonde d'injection. Les deux clés sont ainsi liées par l'équation suivante :

$$k_{i_j} = k_{i_{j+1}} \,\&\, \sim (k_{i_{j+1}} \oplus k_{i_j}) \tag{4.3}$$

où l'opérateur « & » désigne l'opération booléenne « ET » appliquée bit par bit sur un entier en base 2 et l'opérateur « ~ » réalise le complément à deux d'un entier en base 2. La clé originale k_{i_n} s'obtient alors en recombinant l'ensemble des masques de fautes selon l'équation suivante :

$$k_{i_n} = (k_{i_n} \oplus k_{i_{n-1}}) \mid (k_{i_{n-1}} \oplus k_{i_{n-2}}) \mid \ldots \mid (k_{i_1} \oplus k_{i_0}) \tag{4.4}$$

où l'opérateur « | » désigne l'opération booléenne « OU » appliquée bit par bit sur un

entier en base 2.

Par hypothèse, pour tout entier j inférieur ou égal à $n-1$, le masque de fautes de 128 bits $k_{i_{j+1}} \oplus k_{i_j}$ contient au plus 16 bits à '1' dont la position dépend de la position y_j de la sonde d'injection lors de l'injection de fautes. Nous modélisons cette propriété par l'existence d'une fenêtre d'hypothèses, notée m_{i_j}, que nous définissons pour chaque position y_j de la sonde d'injection par l'équation suivante :

$$m_{i_j} = \mathbf{ROT}_{r_j}\,(\underbrace{x\ldots x}_{4}\underbrace{00\ldots00}_{28}\underbrace{x\ldots x}_{4}\underbrace{00\ldots00}_{28}\underbrace{x\ldots x}_{4}\underbrace{00\ldots00}_{28}\underbrace{x\ldots x}_{4}\underbrace{00\ldots00}_{28})_2 \tag{4.5}$$

La notation x désigne l'emplacement possible d'un bit à '1' et l'opérateur $\mathbf{ROT}_{r_j}$ applique une rotation à droite de r_j bits sur un mot binaire de 128 bits.

Les fenêtres d'hypothèses utilisées pour reconstruire la clé secrète dans l'exemple de la Table 4.3 sont illustrées sur la Table 4.4. L'octet d'indice i de la différence entre deux clés successives est reporté dans la colonne B_i du tableau en notation hexadécimale.

			Clés partielles (modèle de faute *bit reset*)															
m	j_m	Notation	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}
0	0	$k_{i_1} \oplus k_{i_0}$	00	00	00	X0	00	00	00	X0	00	00	00	X0	00	00	00	X0
1	4	$k_{i_2} \oplus k_{i_1}$	00	00	00	XX	00	00	00	XX	00	00	00	XX	00	00	00	XX
2	8	$k_{i_3} \oplus k_{i_2}$	00	00	X0	XX	00	00	X0	XX	00	00	X0	XX	00	00	X0	XX
3	12	$k_{i_4} \oplus k_{i_3}$	00	00	XX	XX	00	00	XX	XX	00	00	XX	XX	00	00	XX	XX
4	16	$k_{i_5} \oplus k_{i_4}$	00	X0	XX	XX	00	X0	XX	XX	00	X0	XX	XX	00	X0	XX	XX
5	20	$k_{i_6} \oplus k_{i_5}$	00	XX	XX	XX	00	XX	XX	XX	00	XX	XX	XX	00	XX	XX	XX
6	24	$k_{i_7} \oplus k_{i_6}$	X0	XX	XX	XX	X0	XX	XX	XX	X0	XX	XX	XX	X0	XX	XX	XX
7	28	$k_{i_8} \oplus k_{i_7}$	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX

TABLE 4.4 – Reconstruction progressive d'une clé de 128 bits dans une fenêtre glissante de 16 bits en commençant par les bits de poids forts de chaque mot de 32 bits. Les semi-octets dans la fenêtre glissante sont marqués en rouge. Les semi-octets de la clé originale sont désignés par la notation X.

Afin d'extraire la clé secrète, il convient donc de choisir un nombre n d'étapes et un ensemble de fenêtres d'hypothèses $m_{i_0}, m_{i_1}, m_{i_2}, \ldots, m_{i_{n-1}}$ pour reconstruire les différences entre deux clés successives. Nous définissons, en outre, les opérations suivantes qui traduisent la mise à jour d'un bit inconnu x :

$$\begin{aligned}
\mathrm{Update}(x,0) &= x \\
\mathrm{Update}(x,1) &= 1 \\
\mathrm{Update}(0,x) &= 0 \\
\mathrm{Update}(1,x) &= 1
\end{aligned}$$

La procédure de reconstruction de la clé est détaillée sur l'Algorithme 1.

La clé secrète est initialisée à zéro (ligne 2). Pour chacun des indices $i_0, i_1, i_2, \ldots, i_{n-1}$, une fenêtre d'hypothèses est définie (lignes 4–5). L'algorithme explore alors au plus

Algorithme 1 Reconstruction d'une clé secrète pour des fautes de type *bit reset*.

Input: p_0 : reference plaintext, c_0 : reference ciphertext, $\mathscr{C}$: collection of faulty ciphertexts
Output: k_{i_n} : 128-bit secret key

1: **procedure** RECOVERKEY128($p_0, c_0, \mathscr{C}$)
2: $k_{i_n} \leftarrow (00\ldots0)_2$
3: **for** $i_j = i_0, i_1, i_2, \ldots, i_{n-1}$ **do**
4: $m_{i_j} \leftarrow$ GetWindow(i_j) ▷ Define hypothesis window for difference $k_{i_{j+1}} \oplus k_{i_j}$
5: $m_{i_j} \leftarrow$ Update(m_{i_j}, k_{i_n}) ▷ Update fault mask with known bits in key k_{i_n}
6: **while** a hypothesis $k_{i_{j+1}} \oplus k_{i_j}$ can be made for m_{i_j} **do**
7: $q \leftarrow k_{i_n} \mid (k_{i_{j+1}} \oplus k_{i_j})$
8: $c \leftarrow$ **Enc**(p_0, q)
9: **if** $c = c_0$ **then**
10: $k_{i_n} \leftarrow q$
11: **break**
12: **else if** $c \in \mathscr{C}$ **then**
13: $m_{i_j} \leftarrow$ Update($m_{i_j}, k_{i_{j+1}} \oplus k_{i_j}$)
14: $k_{i_n} \leftarrow$ Update(k_{i_n}, m_{i_j})

2^{16} hypothèses sur la position des bits à `'1'` dans cette fenêtre (ligne 6). Pour chaque hypothèse, une clé candidate est construite (ligne 7) et le texte chiffré associé est calculé (ligne 8). Si ce texte chiffré correspond à l'un des textes chiffrés collectés lors de la première phase de l'attaque, le masque de fautes est mis à jour (lignes 12–13) avec au plus 16 bits à la valeur `'1'`et une nouvelle hypothèse est explorée. À la fin de chaque étape, la clé en construction est mise à jour avec l'état reconstruit du masque de fautes (ligne 14). L'algorithme s'arrête si l'un des textes chiffrés correspond au texte chiffré de référence (lignes 9–11) ou si les hypothèses pour les indices $i_0, i_1, i_2, \ldots, i_{n-1}$ ont été épuisées. Dans ce dernier cas, l'algorithme renvoie une clé partielle en cours de construction.

Dans le meilleur des cas, la clé à reconstruire est constituée uniquement de bits à la valeur `'1'` et les textes chiffrés collectés pendant la campagne d'injection de fautes de type *bit reset* sont associés à une suite de clés dont deux clés consécutives ne diffèrent que d'un bit. Dans ce cas, l'algorithme possède une complexité linéaire en la taille de la clé, soit, dans le cas d'une clé de 128 bits, une complexité en $O(2^7)$. Dans le pire des cas, la clé à reconstruire est constituée uniquement de bits à la valeur `'0'`. Dans ce cas, l'algorithme possède une complexité qui dépend principalement de la taille de la fenêtre glissante. Pour une clé de 128 bits, l'algorithme possède alors une complexité en $O(15 \times 2^{16})$. Or, par analogie, le pire des cas pour l'algorithme de reconstruction d'une clé secrète fondé sur le modèle de faute *bit reset* correspond au meilleur des cas pour un algorithme similaire fondé sur le modèle de faute *bit set*. La complexité de l'algorithme pourraient ainsi être améliorée en combinant les informations obtenues sur la clé secrète lors des injections de fautes de type *bit reset* et lors des injections de fautes de type *bit set*.

4.2.3.C Résultats expérimentaux

Nous avons validé notre attaque en pratique sur le microcontrôleur non-sécurisé SAM-3X8E avec un dispositif d'injection de fautes par impulsions EM. Le microcontrôleur a été programmé avec l'implémentation en langage assembleur de l'algorithme de chiffrement symétrique AES-128 proposé par SCHWABE et STOFFELEN [157].

Puisque le microcontrôleur ne dispose pas d'une EEPROM, la clé secrète a été programmée, avec le logiciel, dans la mémoire Flash embarquée du circuit. Notre attaque cible la lecture de cette clé dans la mémoire Flash au début de l'exécution de la fonction de dérivation des clés de rondes de l'AES.

Dans le cadre de cette expérience, le circuit est programmé avec la clé de chiffrement `"3168E5522368815A6F8A6910766EC3F2"` et le texte clair `"E84F94DD10F3E5EE-718DA7E975D3FBA4"` tous deux choisis aléatoirement.

Le chiffrement d'un message avec un AES-128 se décompose en rondes. Chaque ronde utilise une clé de ronde qui est calculée à partir de la clé secrète grâce à la fonction de dérivation des clés de rondes. Le début d'une implémentation en langage assembleur de cette fonction est présentée sur le Code source 4.7 suivant :

```
1  @ void AES_128_keyschedule(const uint8_t *key,
2  @       uint8_t *rk) {
3  .global     AES_128_keyschedule
4  .type       AES_128_keyschedule,%function
5  AES_128_keyschedule:
6
7      //function prologue, preserve registers
8      push    {r4-r11}
9
10     //load key
11     ldm     r0, {r4-r7}
```

CODE SOURCE 4.7 – Début de la fonction de dérivation des clés de rondes en assembleur ARM dans l'implémentation de SCHWABE et STOFFELEN [157].

La fonction prend, en premier argument, l'adresse de la clé secrète et, en deuxième argument, l'adresse d'un tableau en SRAM où seront mémorisées les clés de rondes. Notons que, par convention, le premier argument est passé dans le registre `"r0"` et le deuxième dans le registre `"r1"`. L'instruction `"ldm r0, {r4-r7}"` (ligne 11) charge donc les registres `"r4"` à `"r7"` avec les 128 bits de la clé secrète. Pour ce faire, la mémoire Flash précharge simultanément les 128 bits de la clé secrète dans un *buffer* de données. Les données contenues dans le *buffer* sont ensuite transférées par blocs de 32 bits dans les registres du microcontrôleur.

Afin de simplifier l'implémentation de notre attaque, nous avons choisi de déclencher le générateur d'impulsions EM avec un signal logique généré par le microcontrôleur avant l'appel à la fonction de dérivation des clés de rondes. Le délai d'injection entre la réception du signal par le générateur et la production d'une EMP est ajusté par la méthode suivante afin d'injecter des fautes dans la clé secrète pendant son préchargement dans le *buffer* de données.

La sonde d'injection EM est d'abord positionnée sur la face avant du microcontrôleur, dans le voisinage du circuit, aux coordonnées $x = 9\,\text{mm}$ et $y = 8.5\,\text{mm}$, afin de fixer à `'0'` ou à `'1'` les 128 bits de la clé secrète grâce à une EMP. Un balayage du délai d'injection est ensuite réalisé par pas de 1 ns. Pour chaque valeur du délai, une faute est injectée pendant le calcul des clés de rondes et le texte clair p_0 est chiffré. Les textes chiffrés fautés et les valeurs des paramètres d'injection sont mémorisés dans un fichier de résultats au format CSV.

Ces textes chiffrés sont ensuite déchiffrés une fois avec la clé k_0 dont tous les bits sont à `'0'`, et une fois avec la clé k_{128} dont tous les bits sont à `'1'`. Les valeurs du délai d'injection pour lesquels le texte clair p_0 est retrouvé correspondent alors aux délais pour lesquels des fautes sont injectées pendant le transfert de la clé secrète. Si le texte clair p_0 n'est pas connu, il est néanmoins possible d'en extraire la valeur. En effet, cette dernière apparaît deux fois : une fois dans les textes clairs obtenus avec la clé k_0 et une fois dans ceux obtenus avec la clé k_{128}. Par ailleurs, les délais d'injection associés à ces deux occurrences sont situés dans un intervalle de quelques nanosecondes. Ces critères permettent donc d'identifier le texte clair p_0, sans connaissances *a priori* sur ce dernier et, par suite, d'identifier les valeurs du délai d'injection pour lesquelles des fautes de type *bit reset* ou de type *bit set* sont injectées dans la clé secrète.

Nous avons choisi de fixer le délai programmable de sorte à injecter des fautes de type *bit reset* sur 1 à 128 bits de la clé secrète. Pour ce délai, un balayage de la coordonnée y de la sonde d'injection EM a donc été réalisé par pas de 25 µm. Pour chaque position de la sonde d'injection EM, une faute a été injectée pendant le calcul des clés de rondes et le texte clair p_0 a été chiffré. Les textes chiffrés et les valeurs des paramètres d'injection ont été mémorisés dans un fichier de résultats au format CSV. Lors de ce balayage, nous avons collecté 80 textes chiffrés fautés.

L'algorithme 1 a ensuite été utilisé pour extraire les 128 bits de la clé secrète en commençant par les bits de poids fort des quatre mots de 32 bits. Nous avons choisi d'extraire la clé secrète en utilisant une fenêtre glissante de 16 bits décalée de 8 bits à chaque itération. Deux masques consécutifs se superposent donc sur un octet. La clé secrète a été intégralement reconstruite avec environ 2^{14} hypothèses. Les étapes de la reconstruction sont détaillées dans la Table 4.5. L'octet d'indice i d'une clé de 128 bits est représenté en notation hexadécimale dans la colonne B_i du tableau. Notre attaque nous a ainsi permis de réduire la complexité de la recherche de la clé de 2^{128} à 2^{14} hypothèses.

	i	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}
	0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	1	00	00	00	50	00	00	00	50	00	00	00	10	00	00	00	F0
	2	00	00	00	50	00	00	00	58	00	00	00	10	00	00	00	F0
	3	00	00	00	52	00	00	00	5A	00	00	00	10	00	00	00	F2
	4	00	00	C0	52	00	00	80	5A	00	00	40	10	00	00	C0	F2
	5	00	00	E0	52	00	00	80	5A	00	00	60	10	00	00	C0	F2
	6	00	00	E4	52	00	00	80	5A	00	00	68	10	00	00	C0	F2
	7	00	00	E5	52	00	00	81	5A	00	00	69	10	00	00	C3	F2
	8	00	40	E5	52	00	40	81	5A	00	80	69	10	00	40	C3	F2
	9	00	60	E5	52	00	60	81	5A	00	80	69	10	00	60	C3	F2
	10	00	68	E5	52	00	68	81	5A	00	88	69	10	00	6C	C3	F2
	11	00	68	E5	52	00	68	81	5A	00	8A	69	10	00	6E	C3	F2
	12	00	68	E5	52	00	68	81	5A	40	8A	69	10	40	6E	C3	F2
	13	30	68	E5	52	20	68	81	5A	60	8A	69	10	70	6E	C3	F2
	14	30	68	E5	52	20	68	81	5A	6C	8A	69	10	74	6E	C3	F2
(secret key)	15	31	68	E5	52	23	68	81	5A	6F	8A	69	10	76	6E	C3	F2

TABLE 4.5 – Reconstruction d'une clé secrète de 128 bits en commençant par les bits de poids fort des quatre mots de 32 bits. Les semi-octets dans la fenêtre glissante sont marqués en rouge.

4.3 Attaques par injection de fautes laser

Dans cette section, nous présentons deux attaques pour extraire la clé secrète d'un algorithme de chiffrement AES-128 grâce à une injection de fautes laser. Ces attaques exploitent le mécanisme d'injection de fautes laser sur les transistors à grille flottante d'une mémoire NOR Flash décrit dans le chapitre 3. L'analyse de ce mécanisme et son exploitation ont fait l'objet de deux publications : [51] et [113].

4.3.1 Attaque sur la dernière transformation AddRoundKey d'un AES

4.3.1.A Présentation de la transformation AddRoundKey

Une implémentation standard de l'algorithme de chiffrement AES-128 se décompose en dix rondes. Chaque ronde effectue les transformations ADDROUNDKEY, SUBBYTES, SHIFTROWS et MIXCOLUMNS, à l'exception de la dernière ronde qui n'inclut pas la transformation MIXCOLUMNS. Une dernière transformation ADDROUNDKEY est effectuée à l'issue des dix rondes [55].

Notre attaque cible cette dernière transformation dans le but d'extraire la dernière clé de ronde. Dans le cas de l'AES-128, la connaissance de la dernière clé de ronde est suffisante pour inverser l'algorithme de dérivation des clés de rondes et ainsi extraire l'intégralité de la clé secrète de chiffrement [23].

Lors de la dernière transformation ADDROUNDKEY, l'état de l'algorithme de chiffrement, noté S, est combiné octet par octet avec la dernière clé de ronde, notée K^{10}, par une opération ou-exclusif, notée « $\oplus$ ». Cette transformation est décrite en pseudo-code par l'Algorithme 2. L'état de l'algorithme de chiffrement à l'issue de la dernière opération

ADDROUNDKEY constitue le message chiffré, noté C.

Algorithme 2 Addition de la dernière clé de ronde.

1: **for** $i \leftarrow 0$ **to** 3 **do**
2: **for** $j \leftarrow 0$ **to** 3 **do**
3: $S_{i,j} \leftarrow S_{i,j} \oplus K^{10}_{i,j}$
4: $C \leftarrow S$

Suivant la notation du standard, l'état de l'algorithme de chiffrement, la dernière clé de ronde et le message chiffré sont représentés par des matrices carrées de 16 octets. L'octet situé à l'intersection de la colonne i et de la ligne j de l'état de l'algorithme de chiffrement est noté $S_{i,j}$, celui de la dernière clé de ronde est noté $K^{10}_{i,j}$ et celui du message chiffré est noté $C_{i,j}$.

La dernière clé de ronde s'obtient en combinant le message chiffré avec l'état de l'algorithme de chiffrement avant la dernière transformation ADDROUNDKEY grâce à l'équation suivante :

$$\forall (i,j) \in [0..3]^2, \quad S_{i,j} \oplus C_{i,j} = K^{10}_{i,j} \tag{4.6}$$

La combinaison d'un message chiffré avec et sans la dernière transformation ADDROUNDKEY permet ainsi d'extraire la dernière clé de ronde.

4.3.1.B Implémentation en langage assembleur

L'implémentation logicielle de l'Algorithme 2 se décompose en deux boucles imbriquées. Une version simplifiée de cet algorithme compilée pour le microcontrôleur STM32F100RB en langage assembleur ARM est donnée dans le Code source 4.8.

Les deux boucles imbriquées peuvent être décomposées en une boucle extérieure, délimitée par les labels `".loop_i"` et `".end_loop_i"`, et une boucle intérieure, délimitée par les labels `".loop_j"` et `".end_loop_j"`.

Initialement, les compteurs d'itérations des boucles extérieure et intérieure sont mis à `'0'` dans les registres `"r0"` et `"r1"` respectivement (ligne 2 et ligne 5).

À chaque itération, un octet de l'état de l'algorithme de chiffrement est combiné avec un octet de la dernière clé de ronde. Les instructions associées à cette opération sont résumées par un commentaire à la ligne 7.

Après chaque itération, le compteur d'itérations de la boucle intérieure est incrémenté (ligne 9). L'exécution du programme est redirigée au début de la boucle intérieure tant que la valeur de ce compteur est inférieure au nombre d'itérations à réaliser (lignes 10–11). Dans le cas contraire, le programme sort de la boucle intérieure.

Après chaque sortie de la boucle intérieure, le compteur d'itérations de la boucle extérieure est incrémenté (ligne 15). L'exécution du programme est redirigée au début de la boucle extérieure tant que la valeur de ce compteur est inférieure au nombre

```
1  @    for (i=0;i<3;i++) {
2       mov     r0, #0
3  .loop_i:
4  @         for (j=0;j<3;j++) {
5       mov     r1, #0
6  .loop_j:
7  @             S(i,j) = S(i, j) ^ K(Nr, i, j)
8  @         for (j=0;j<3;j++) {
9       add     r1, #1
10      cmp     r1, #3
11      ble     .loop_j
12 .end_loop_j:
13 @         }
14 @    for (i=0;i<3;i++) {
15      add     r0, #1
16      cmp     r0, #3
17      ble     .loop_i
18 .end_loop_i:
19 @    }
```

CODE SOURCE 4.8 – Implémentation en langage assembleur ARM de la transformation ADDROUNDKEY.

d'itérations à réaliser (lignes 16–17). Dans le cas contraire, le programme sort de la boucle extérieure et l'addition de la dernière clé de ronde s'achève.

4.3.1.C Description de l'attaque

Nous avons démontré dans la section 3.5 que l'injection d'une faute de type *bit set* par illumination laser de la mémoire Flash pendant la lecture d'une instruction permet notamment de modifier les opérandes de l'instruction. Plusieurs chemins d'attaques permettent alors de modifier le compteur d'itérations :

— le premier consiste à augmenter la valeur immédiate des instructions qui initialisent les compteurs d'itérations à 0 (ligne 2 et ligne 5) ;

— le second consiste à augmenter la valeur immédiate des instructions qui incrémentent les compteurs d'itérations (ligne 15 et ligne 9).

L'injection d'une faute provoque alors une sortie anticipée, soit de la boucle intérieure, soit de la boucle extérieure, si le compteur d'itérations correspondant est supérieur au nombre d'itérations de la boucle.

Cependant, la manipulation des compteurs d'itérations ne nous permet pas d'empêcher la combinaison du premier octet de l'état de l'algorithme avec le premier octet de la dernière clé de ronde. Une recherche exhaustive devra donc être réalisée sur cet octet.

L'attaque proposée suppose qu'un attaquant a connaissance des textes chiffrés. Nous désignons par $\tilde{C}^{\text{outer}}_{i,j}$ l'octet i de la ligne j du message chiffré lorsqu'une faute est injectée à la fin de la première itération de la boucle extérieure. Dans ces conditions, les trois dernières lignes ne sont pas combinées avec la clé de ronde. Le message chiffré fauté satisfait alors les relations 4.7. En observant le message chiffré fauté, il est donc possible d'extraire tous les octets du dernier état de l'algorithme de chiffrement après la 10^{e}

ronde, à l'exception des états $S_{0,0}$, $S_{0,1}$, $S_{0,2}$ et $S_{0,3}$. Tous les octets de la dernière clé de ronde peuvent ainsi être extraits, à l'exception des octets $K^{10}_{0,0}$, $K^{10}_{0,1}$, $K^{10}_{0,2}$ et $K^{10}_{0,3}$, grâce à l'équation suivante :

$$\tilde{C}^{\text{outer}}_{i,j} = \begin{cases} S_{i,j} & \text{si } i \in [1..3] \wedge j \in [0..3] \\ C_{i,j} & \text{sinon} \end{cases} \tag{4.7}$$

Nous désignons par $\tilde{C}^{\text{inner}}_{i,j}$ l'octet i de la ligne j du message chiffré lorsqu'une faute est injectée à la fin de la première itération de la boucle intérieure. Dans ces conditions, les trois derniers octets de la première ligne ne sont pas combinés avec la clé de ronde. Le message chiffré fauté satisfait alors les relations 4.8. En observant le message chiffré fauté, il est donc possible d'extraire les octets $S_{0,1}$, $S_{0,2}$ et $S_{0,3}$ du dernier état de l'algorithme de chiffrement. Les octets $K^{10}_{0,1}$, $K^{10}_{0,2}$ et $K^{10}_{0,3}$ de la dernière clé de ronde peuvent ainsi être extraits grâce à l'équation suivante :

$$\tilde{C}^{\text{inner}}_{i,j} = \begin{cases} S_{i,j} & \text{si } (i = 0) \wedge j \in [1..3] \\ C_{i,j} & \text{sinon} \end{cases} \tag{4.8}$$

Dans le cas de l'AES-128, la clé secrète de chiffrement s'obtient en inversant l'opération de dérivation des clés de rondes à partir de la dernière clé de ronde. En réalisant une recherche exhaustive sur les 2^8 valeurs potentielles de l'octet $K^{10}_{0,0}$, il est alors possible de retrouver la clé secrète de chiffrement.

4.3.1.D Résultats expérimentaux

Le dispositif utilisé pour injecter des fautes de type *bit set* par impulsion laser dans les instructions transférées entre la mémoire NOR Flash et le cœur de calcul du MCU STM32F100RB est décrit dans la section 3.1.1. L'injection laser est réalisée par la face arrière du microcontrôleur avec l'objectif ×5. Le spot laser est positionné à l'intérieur de la mémoire NOR Flash grâce au système d'imagerie IR du banc d'injection laser.

Nous avons choisi d'implémenter en pratique le remplacement d'une instruction de la forme `"add Rd, #1"` (ligne 9 ou ligne 15 du Code source 4.8) en une instruction de la forme `"add Rd, #5"` en injectant une faute de type *bit set* sur le bit d'indice 2 de l'instruction. Ce cas de figure est illustré sur la Figure 4.2 qui détaille l'encodage sur 16 bits de l'instruction `"add Rd, #1"` et l'injection d'une faute de type *bit set* en rouge. Notons que cette attaque s'étend aisément au cas où le compteur d'itérations est décrémenté avec une instruction `"sub"`.

La coordonnée Z du système optique est ajustée de sorte à injecter des fautes sur un seul bit. Pour ce faire, la coordonnée Z est ajustée de sorte à maximiser le contraste de l'image de la face arrière du circuit de test obtenue grâce au dispositif d'imagerie infrarouge du banc laser. L'objectif est positionné à l'intérieur de la matrice de cellules NOR Flash en ajustant sa coordonnée Y.

En fonction de l'alignement en mémoire des instructions ciblées, deux configurations

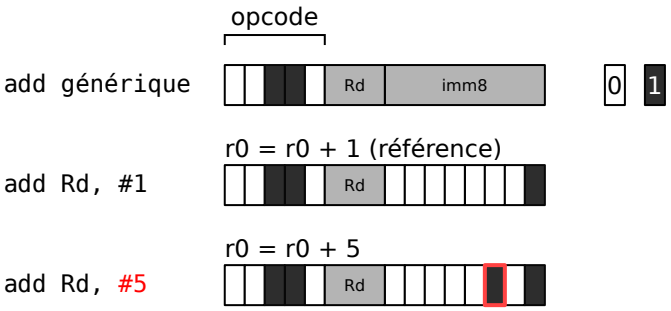


FIGURE 4.2 – Modification de l'incrément d'un compteur d'itérations par injection d'une faute de type *bit set*.

conduisent au modèle de faute sus-mentionné conformément aux données expérimentales décrites dans la section 3.2.1.A. La première nécessite d'ajuster la coordonnée X de l'objectif au-dessus des colonnes mémoire qui mémorisent les bits d'indice 2 et la seconde d'ajuster la coordonnée X de l'objectif au-dessus de celles qui mémorisent les bits d'indice 18. Ces deux configurations sont donc explorées l'une après l'autre.

La durée de l'impulsion est choisie inférieure à une période d'horloge du circuit (127 ns) et la puissance de la source laser est fixée à 1 W. Un signal logique généré par le circuit de test est utilisé pour synchroniser la perturbation laser sur l'exécution de l'algorithme de chiffrement.

L'intervalle d'exécution de la dernière ronde de l'algorithme de chiffrement est identifié sur les traces de consommation du circuit. Un balayage des valeurs du délai d'injection par pas d'une demi-période d'horloge est réalisé dans cet intervalle pour les paramètres d'injection sus-mentionnés.

L'injection d'une faute sur l'instruction `"add"` à la fin de la première itération de la boucle extérieure se traduit par la modification de tous les octets du message chiffré, à l'exception des octets $C_{0,0}$, $C_{0,1}$, $C_{0,2}$ et $C_{0,3}$.

L'injection d'une faute sur l'instruction `"add"` à la fin de la première itération de la boucle intérieure se traduit par la modification des octets $C_{0,1}$, $C_{0,2}$ et $C_{0,3}$ du message chiffré.

Un message chiffré correct et deux messages chiffrés fautés nous ont ainsi permis d'extraire quinze des seize octets d'une clé de chiffrement avec une excellente reproductibilité.

4.3.2 Extraction d'une clé secrète par SEA

4.3.2.A Attaque par SEA

Le principe de redondance permet de protéger un circuit microélectronique d'une défaillance singulière. La fonction protégée peut être exécutée plusieurs fois par le même bloc matériel ou simultanément par plusieurs blocs matériels. Une faute injectée dans une des exécutions peut ainsi être détectée ou corrigée en comparant le résultat des différentes exécutions. Ce principe est appliqué dans le domaine de la sécurité

matérielle pour protéger les primitives de sécurité contre les attaques par injection de fautes [67].

YEN et JOYE [178] remarquent cependant qu'une simple vérification de l'intégrité des résultats ne suffit pas à protéger un algorithme cryptographique. Les auteurs soulignent, en effet, que la propagation d'une faute dans les calculs d'un algorithme dépend du modèle de faute et des données manipulées par l'algorithme. Une application de ce principe, proposée par CLAVIER [47], consiste à utiliser la dépendance aux données des fautes de type *bit set* et *bit reset* pour extraire directement la valeur d'un bit. En effet, l'injection d'une faute de type *bit reset* sur un bit à la valeur `'0'`, par exemple, ne modifie pas la valeur du bit. Ces fautes, nommées *safe errors* ou *ineffective errors*, ne modifient donc pas le résultat d'un algorithme. En comparant les résultats obtenus avec et sans l'injection d'une faute, un attaquant peut ainsi connaître la valeur du bit sur lequel une faute a été injectée.

Ce principe est utilisé par BLÖMER et SEIFERT [31] dans une attaque à texte clair connu par IFA sur l'algorithme de chiffrement AES-128. Dans leur attaque, les auteurs supposent qu'un attaquant peut injecter une faute de type *bit reset* sur un bit choisi à l'intérieur de l'état de l'algorithme de chiffrement après la première transformation ADDROUNDKEY. Une opération ou-exclusif entre la valeur extraite pour cet état et le texte clair permet alors d'extraire la clé de chiffrement. Afin de réaliser cette attaque en pratique, les auteurs ont proposé de modifier l'état de l'AES contenu dans la mémoire SRAM grâce à une injection de fautes laser.

Une attaque par IFA sur l'algorithme de chiffrement DES a aussi été proposée par LOUBET-MOUNDI, VIGILANT et OLIVIER [105]. Dans leur attaque, les auteurs supposent qu'un attaquant peut forcer un bit à `'1'`, ou à `'0'`, dans les registres d'un système cryptographique où est mémorisée la clé secrète. La validité du résultat du chiffrement dépend alors de la valeur initiale du bit de la clé secrète sur lequel une faute a été injectée. En répétant ce procédé pour chacun des bits d'une clé secrète, l'intégralité de la clé peut être extraite. Les auteurs ont réalisé avec succès le modèle de faute proposé grâce à une injection de fautes laser par la face arrière de deux circuits fabriqués dans les technologies CMOS 0.18 µm et 0.13 µm. Ils remarquent cependant que la complexité du processus d'identification des fautes et la précision du dispositif d'injection limitent leur approche sur des technologies récentes.

Le modèle de faute de type *bit set*, décrit dans la section 3.5 sur un circuit fabriqué dans la technologie CMOS 80 nm, permet de mettre à `'1'` un bit d'indice choisi pendant la lecture d'un mot de 32 bits en mémoire NOR Flash avec une forte répétabilité. Une résolution au bit près est obtenue par ailleurs avec un spot laser large, d'un diamètre de 20 µm. Nous proposons donc d'implémenter l'attaque de LOUBET-MOUNDI, VIGILANT et OLIVIER grâce à une injection de fautes laser pendant le transfert d'une clé secrète entre la mémoire NOR Flash embarquée et les registres généraux d'un microcontrôleur non-sécurisé.

4.3.2.B Résultats expérimentaux

L'attaque présentée dans ce paragraphe a été réalisée en pratique sur le microcontrôleur non-sécurisé STM32F100RB dans des conditions similaires à l'extraction d'une clé par injection de fautes EM sur le microcontrôleur SAM3X8E (section 4.2.3).

Le MCU est programmé avec une implémentation en langage assembleur de l'algorithme de chiffrement symétrique AES-128 proposée par SCHWABE et STOFFELEN [157]. Puisque le microcontrôleur ne dispose pas d'une EEPROM, la clé secrète de 128 bits est mémorisée, avec le logiciel, dans la mémoire NOR Flash embarquée du circuit.

Les registres généraux du microcontrôleur sont chargés avec la clé secrète par une instruction de chargement multiple `"ldm"` au début de la fonction de dérivation des clés de rondes (voir Code source 4.7 de la section 4.2.3). Puisque la mémoire Flash du MCU STM32F100RB possède un *buffer* de lecture de 32 bits, quatre opérations de lecture en mémoire Flash sont nécessaires pour lire une clé de 128 bits. Ces opérations de lecture sont la cible de notre attaque.

Afin de simplifier la mise en place de l'expérience, nous avons choisi de déclencher les impulsions laser avec un signal logique généré par le microcontrôleur avant l'appel à la fonction de dérivation des clés de rondes. Un premier balayage du délai d'injection Δt nous a permis d'identifier les instants de chargement de la clé secrète :

— $\Delta t = 5\,670$ ns pour les bits 0 à 31 ;

— $\Delta t = 5\,805$ ns pour les bits 32 à 63 ;

— $\Delta t = 5\,940$ ns pour les bits 64 à 95 ;

— $\Delta t = 6\,075$ ns pour les bits 96 à 127.

Dans un scénario d'attaque réelle, les impulsions laser peuvent être déclenchées, par exemple, sur l'envoi d'une commande de chiffrement. Par ailleurs, le calcul des clés de rondes peut être forcé en réalisant une remise à zéro du circuit. Enfin, une analyse des traces de consommation du circuit permet d'identifier l'intervalle temporel où la fonction est exécutée [172].

L'objectif du laser est ensuite positionné au niveau de la mémoire NOR Flash embarquée du microcontrôleur. La coordonnée Y est fixée à 400 µm de sorte à placer l'objectif du laser à l'intérieur de la mémoire Flash. La coordonnée Z de l'objectif est ajustée pour injecter des fautes de type *bit set* sur un seul bit dans un mot de 32 bits. L'indice de ce bit est choisi en ajustant la coordonnée X de l'objectif. La puissance de la source laser est réglée sur 1 W et la durée de l'impulsion laser sur 50 ns.

Pour chacun des quatre délais d'injection retenus – à savoir 5 670 ns, 5 805 ns, 5 940 ns et 6 075 ns – un balayage de la coordonnée X de l'objectif du laser est réalisé par pas de 10 µm. L'injection d'une faute de type *bit set* sur un bit de valeur `'0'` ou `'1'` peut être détectée en chiffrant deux fois le même texte clair : une fois en conditions normales et une fois lorsqu'une impulsion laser est générée pendant le transfert de la clé. Afin d'accélérer les expériences, seule la fonction de dérivation des clés de rondes est exécutée pour chacune des clés testées. Après chaque injection de fautes, le programme vérifie l'intégrité des clés de rondes mémorisées dans la SRAM du microcontrôleur et renvoie

un booléen qui traduit l'injection d'une faute. Les paramètres d'injection et le résultat de la vérification sont mémorisés à l'intérieur d'un fichier de résultats au format CSV.

Une cartographie similaire à celle présentée dans le chapitre 3 permet d'identifier la position des *bit lines* où une faute de type *bit set* est injectée sur un bit d'indice donné dans un mot de 32 bits. Les valeurs retenues à l'issue de la rétro-ingénierie des positions des *bit lines* pour l'adresse de lecture choisie dans cette expérience sont reportées dans la Table 4.6. Les valeurs pour une adresse de lecture différente sont obtenues en décalant chacun des intervalles d'une quantité constante qui dépend de la position relative de cette adresse par rapport à l'adresse choisie dans cette expérience, conformément aux résultats présentés dans la section 3.2.3.

indice du bit	y_{min} (µm)	y_{max} (µm)	indice du bit	y_{min} (µm)	y_{max} (µm)
0	80	110	16	780	810
1	130	155	17	820	860
2	165	200	18	870	900
3	205	240	19	915	945
4	250	290	20	965	985
5	300	335	21	1000	1035
6	345	380	22	1050	1075
7	385	420	23	1090	1115
8	430	460	24	1135	1165
9	470	505	25	1180	1210
10	515	550	26	1230	1250
11	560	595	27	1270	1295
12	605	635	28	1310	1340
13	655	685	29	1355	1385
14	695	725	30	1400	1420
15	735	770	31	1450	1465

TABLE 4.6 – Ingéniérie inverse des positions des *bit lines* pour chacun des bits d'un mot de 32 bits.

Les résultats expérimentaux obtenus avec la clé `"6FDA6B0DAD03FEF29290FC3E0C5BF9-24"` sont présentés sur la Figure 4.3. Les fautes observées expérimentalement sont repérées par un point noir. Elles correspondent à l'injection d'une faute de type *bit set* sur un bit dont la valeur initiale est `'0'`. Pour chaque *bit line* identifiée dans la Table 4.6, notre hypothèse sur la valeur initiale du bit lu en mémoire est indiquée par un point de couleur grise (valeur initiale `'1'`) ou de couleur rouge (valeur initiale `'0'`). Les 128 bits de la clé secrète peuvent ainsi être lus directement sur la Figure 4.3.

Cette procédure nous a permis d'extraire la quasi-totalité des bits de cent clés de chiffrement choisies aléatoirement. Les bits non-extraits sont soit le bit de poids le plus fort, soit le bit de poids le plus faible dans un des mots de 32 bits qui composent la clé de chiffrement. Ces résultats peuvent donc probablement être améliorés en ajustant l'intervalle d'exploration suivant l'axe Y.

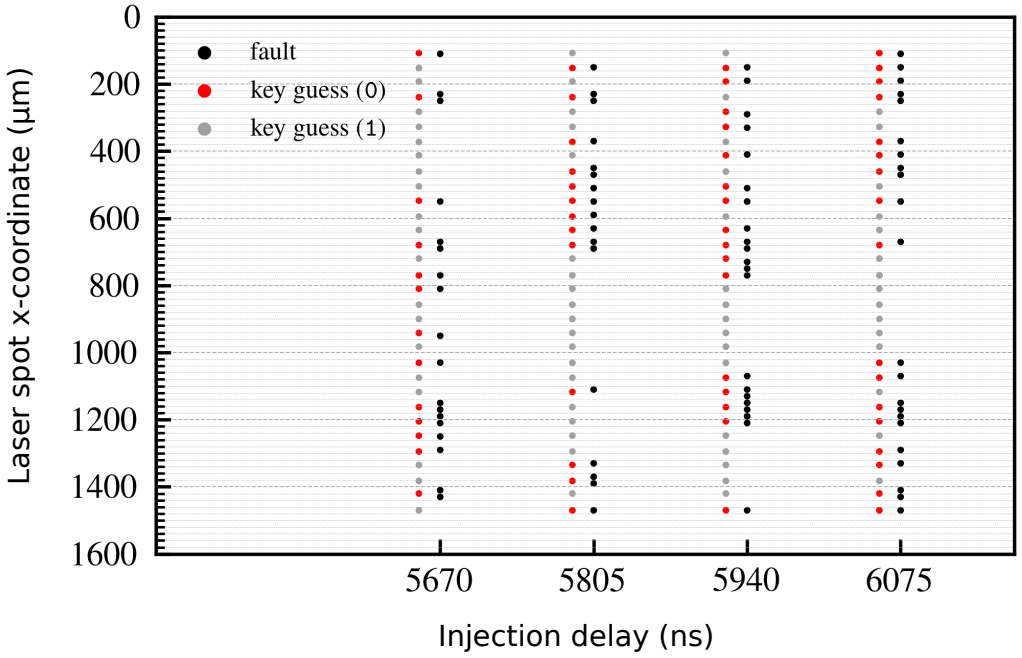


FIGURE 4.3 – Attaque *safe-error* sur la clé secrète `"6FDA6B0D AD03FEF2 9290FC3E 0C5BF924"` pour 256 injections de fautes. Pour déduire la valeur de la clé secrète, les fautes (en noir) sont associées aux positions des *bit lines* extraites par ingénierie inverse de l'organisation de la mémoire Flash (en rouge et gris).

4.4 Analyse du *secure boot* d'un microcontrôleur

Dans le cadre de ces travaux, nous avons porté notre intérêt sur la sécurité matérielle des objets connectés face aux attaques par injection de fautes, et plus particulièrement sur la sécurité maérielle des microcontrôleurs non-sécurisés. Afin d'identifier les menaces qui pèsent sur ces circuits, nous avons analysé plusieurs modèles de fautes originaux sur des microcontrôleurs non-sécurisés. Les vulnérabilités induites par ces modèles de fautes ont été validées sur ces microcontrôleurs en réalisant plusieurs attaques contre des algorithmes de sécurité.

La fonctionnalité d'amorçage sécurisé, ou *secure boot*, est l'une des pierres angulaires de l'internet des objets. Dans les paragraphes suivants, nous proposons plusieurs applications théoriques des modèles de fautes étudiés dans ces travaux pour contourner l'implémentation d'un *secure boot* dans le cas d'un circuit sécurisé destiné à l'IoT.

4.4.1 Motivations

Nous proposons d'étendre l'application des modèles de fautes étudiés dans ces travaux à la phase de démarrage d'un microcontrôleur conçu pour l'internet des objets. Lors de cette phase, deux logiciels d'amorçage, ou *bootloaders*, sont exécutés l'un après l'autre. Le premier logiciel initialise le circuit, met en place une politique de sécurité et vérifie l'intégrité et l'authenticité du deuxième logiciel avant de l'exécuter. Cette fonctionnalité, appelée *secure boot*, implémente une étape critique dans la sécurisation du circuit. Il

convient alors d'estimer sa résistance face aux attaques par injection de fautes dans l'objectif de concevoir des protections adaptées à cette menace.

Pour ce faire, nous proposons une analyse des vulnérabilités du *secure boot* d'un microcontrôleur commercialisé pour des applications de l'IoT. Cette analyse se fonde sur les modèles de fautes étudiés dans le chapitre 2 et le chapitre 3 pour proposer plusieurs vecteurs d'attaques en faute pendant la phase de démarrage du circuit. Elle articule ainsi la connaissance des modèles de fautes produite dans ces travaux avec l'exploration des vulnérabilités matérielles dans le cadre des objets connectés.

4.4.2 Principe d'un amorçage sécurisé

Le premier programme exécuté par un microcontrôleur est le programme d'amorçage, ou *bootloader*. Son rôle consiste à initialisier le circuit avant d'exécuter le micrologiciel, ou *firmware*, qui implémente les fonctionnalités principales du circuit. Dans le cas d'un démarrage sécurisé, la charge revient donc au *bootloader* de valider le *firmware* avant d'exécuter ce dernier. Nous présentons, en premier lieu, un mécanisme de validation classique et décrivons, en second lieu, comment ce mécanisme peut être étendu à une chaîne de programmes.

4.4.2.A Signature numérique

Un schéma de signature numérique permet d'automatiser la validation d'un code binaire. Il est constitué d'une fonction de hachage et d'un algorithme de chiffrement asymétrique. Ces fonctions sont utilisées, d'un côté, par le propriétaire du *firmware* pour générer la signature du *firmware* avant la programmation du microcontrôleur et, de l'autre, par le microcontrôleur pour valider le *firmware* avec sa signature à chaque démarrage.

Avant la programmation du microcontrôleur, un couple composé d'une clé privée et d'une clé publique est d'abord généré par le propriétaire du *firmware* pour la méthode de chiffrement sélectionnée. Ce couple fonctionne alors de manière asymétrique : la clé privée permet de chiffrer les messages, tandis que la clé publique permet de les déchiffrer. La clé privée est gardée secrète par le propriétaire du *firmware* et la clé publique est généralement programmée dans une mémoire à programmation unique (*One Time Programming* – OTP) du circuit [150]. L'empreinte du *firmware* est ensuite calculée grâce à la fonction de hachage. Cette empreinte est chiffrée avec la clé privée pour constituer la signature du *firmware*. Le *firmware* et sa signature sont finalement programmés dans une mémoire non-volatile du microcontrôleur. Puisque le *bootloader* constitue, avec la clé publique, l'essentiel du mécanisme de validation, il doit être impossible de le modifier. Pour ce faire, il est généralement programmé par le constructeur du microcontrôleur dans une ROM du circuit [150].

Un mécanisme d'amorçage sécurisé est illustré sur la Figure 4.4. Le code d'amorçage sécurisé mémorisé dans la ROM est exécuté par le cœur de calcul à chaque remise à zéro du circuit **(1)**. Il calcule tout d'abord l'empreinte du *firmware* grâce à la fonction de hachage de l'accélérateur cryptographique (*cryptographic co-processor*) et mémorise

cette dernière en SRAM **(2)**. Il calcule ensuite l'empreinte de référence du *firmware* en déchiffrant la signature du *firmware* avec la clé publique mémorisée dans une mémoire OTP du circuit grâce à la fonction de déchiffrement de l'accélérateur cryptographique **(3)**. Il vérifie enfin que les deux empreintes obtenues sont identiques **(4)**. Le cas échéant, il exécute le *firmware*. En revanche, si le *firmware* a été modifié, son empreinte par la fonction de hachage ne correspond plus à l'empreinte de la signature. Or un attaquant doit connaître la clé privée pour forger une signature avec l'empreinte de son choix. Ainsi, le logiciel embarqué n'est exécuté que s'il est intègre et authentique.

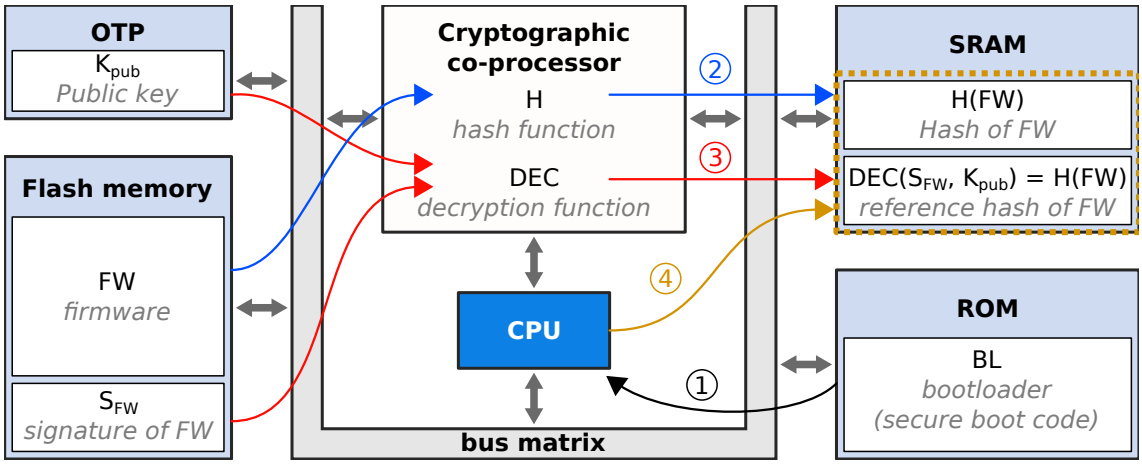


FIGURE 4.4 – Implémentation d'un mécanisme de *secure boot* grâce à un schéma de signature numérique.

4.4.2.B Chaîne de confiance

La clé publique et le *bootloader* programmés dans le microcontrôleur constituent le premier maillon d'une chaîne de validations. Ce maillon est souvent désigné comme la racine de confiance (*Root of Trust* – RoT) du circuit. Cependant, certains scénarios applicatifs nécessitent de pouvoir modifier le *bootloader*. C'est le cas, en particulier, si le *bootloader* doit être corrigé suite à la découverte d'une vulnérabilité. Une solution classique consiste alors à diviser le *bootloader* en plusieurs programmes, ou étages, qui s'exécutent à la chaîne l'un après l'autre. Le premier étage est programmé par le constructeur dans une ROM du microcontrôleur, tandis que les étages suivants sont programmés dans une mémoire non-volatile reprogrammable, généralement une mémoire Flash. Chaque étage met en œuvre le mécanisme de signature décrit plus haut pour valider l'étage suivant avant de l'exécuter. Une chaîne de confiance s'établit ainsi entre la RoT et le *firmware*.

Les fonctionnalités de nombreux objets connectés sont implémentées grâce à des microcontrôleurs embarqués. Dans le cas de capteurs connectés, ces microcontrôleurs sont des circuits à basse consommation et à faible coût de production. Or, les algorithmes de chiffrement asymétrique possèdent un coût d'implémentation, une consommation énergétique et un temps d'exécution élevés. Nous décrivons ainsi dans la section suivante comment les propriétés de sécurité garanties par un algorithme de signature numérique sont obtenues grâce à l'algorithme de chiffrement symétrique AES-128 sur un MCU commercialisé destiné aux applications de l'IoT.

4.4.3 Analyse d'une solution commercialisée

Dans les paragraphes suivants, nous détaillons l'implémentation d'un mécanisme de *secure boot* dans un MCU à basse consommation commercialisé pour des applications de l'IoT. Cette étude sert de fondement à notre analyse de la sécurité du circuit face aux attaques par injection de fautes pendant sa phase de démarrage.

4.4.3.A Présentation du microcontrôleur SAML11

Le microcontrôleur SAML11E16 est un MCU 32 bits à basse consommation fabriqué par la société Microchip Technology Inc. Il implémente plusieurs fonctionnalités de sécurité adaptées à différents cas d'applications de l'IoT :

1. Il intègre la technologie TrustZone qui permet d'isoler le contexte d'exécution d'une solution sécurisée de celui d'une solution non-sécurisée. Par conséquent, l'exploitation d'une vulnérabilité logicielle dans la solution non-sécurisée ne compromet pas, en principe, la solution sécurisée.
2. Il met en œuvre une protection anti-débogage qui filtre les accès au circuit selon le niveau d'autorisation programmé, à savoir DAL2, DAL1 ou DAL0. Le niveau DAL2 ne présente aucune restriction, le niveau DAL1 interdit les accès à la solution sécurisée et le niveau DAL0, n'autorise que certaines fonctions de tests préprogrammées.
3. Il implémente un mécanisme d'amorçage sécurisé, ou *secure boot*, programmé dans la ROM du circuit. Ce mécanisme vérifie l'intégrité et l'authenticité d'un logiciel d'amorçage, ou *bootloader*, programmé dans la mémoire Flash du circuit à chaque démarrage du microcontrôleur.

Un tableau auxiliaire de la mémoire Flash, nommé *NVM rows*, permet de configurer ces fonctionnalités lors de la programmation du circuit.

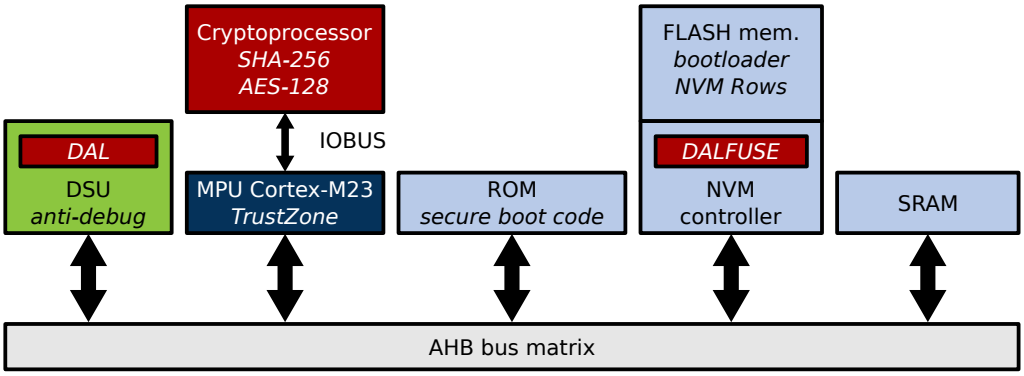


FIGURE 4.5 – Schéma bloc des principaux composants du microcontrôleur SAML11.

Les principaux composants du microcontrôleur et leurs interconnexions sont détaillés sur le schéma bloc de la Figure 4.5. Le microcontrôleur implémente un cœur de calcul ARM Cortex-M23 qui peut être cadencé à une fréquence maximale de 32 MHz. Il embarque une ROM de 8 ko, une SRAM de 16 ko et une mémoire NOR Flash de 64 ko. Ces mémoires sont intégrées au microcontrôleur dans une architecture von Neumann. Un

accélérateur cryptographique implémente la fonction de hachage SHA-256 et l'algorithme de chiffrement symétrique AES-128. Le périphérique DSU (*Device Service Unit*) implémente, enfin, une protection anti-débogage qui filtre les requêtes faites au circuit suivant le niveau de débogage (*Debug Access Level* – DAL) enregistré dans un registre du périphérique.

Nous remarquons que le registre `"DAL"` du DSU est synchronisé avec la valeur du registre `"DALFUSE"` du contrôleur de la mémoire NOR Flash. Or, la valeur du registre `"DAL"` n'est pas conservée lorsque l'alimentation du microcontrôleur est déconnectée. Par ailleurs, une augmentation du niveau d'accès en débogage nécessite un redémarrage du circuit pour prendre effet. Il est donc probable que le registre `"DAL"` du DSU soit mis à jour avec la valeur du registre `"DALFUSE"` du contrôleur de la mémoire NOR Flash pendant le démarrage du circuit, bien que cette information ne soit pas fournie explicitement par le constructeur.

4.4.3.B Scénarios de déploiement

La Figure 4.6 illustre les deux scénarios de déploiement possibles du microcontrôleur SAML11. Ces scénarios sont implémentés grâce aux trois niveaux d'accès en débogage du microcontrôleur : DAL2, DAL1 et DAL0. Dans ces scénarios, une entreprise, que nous nommons l'entreprise A, commercialise une solution qui utilise le microcontrôleur SAML11.

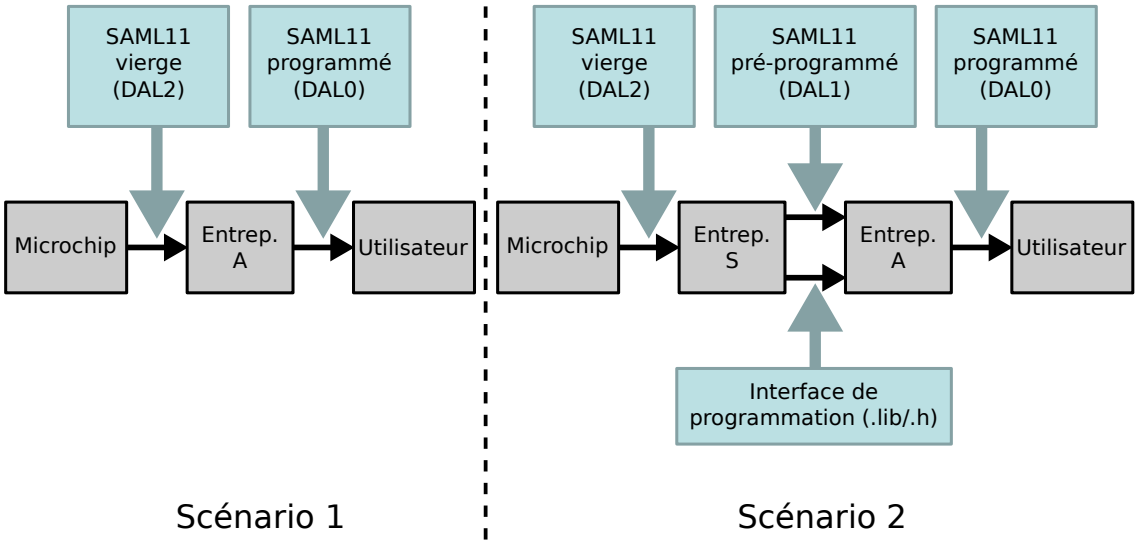


FIGURE 4.6 – Déploiement d'un circuit SAML11 dans un scénario à une ou deux entreprises. Chaque entreprise modifie le niveau d'accès autorisé par la protection anti-débogage, à savoir DAL2, DAL1 ou DAL0, après la programmation de sa solution [115].

Premier scénario. L'entreprise A programme la mémoire Flash du microcontrôleur, initialement vierge, avec sa solution. Elle configure ensuite la protection anti-débogage au niveau DAL0. L'utilisateur final ne peut alors ni relire, ni modifier la solution de l'entreprise A au travers des fonctionnalités de débogage.

Deuxième scénario. L'entreprise A sous-traite l'implémentation des algorithmes de sécurité du microcontrôleur à une deuxième entreprise, appelée l'entreprise S. Cette dernière programme tout d'abord la zone sécurisée de la mémoire Flash du microcontrôleur avec sa solution et configure la protection anti-débogage au niveau DAL1. L'entreprise A ne peut alors ni relire, ni modifier la solution de l'entreprise S. L'entreprise A programme ensuite la zone non-sécurisée de la mémoire Flash du microcontrôleur avec sa solution et configure la protection anti-débogage au niveau DAL0. L'utilisateur final ne peut alors ni relire, ni modifier les solutions des entreprises S et A.

Le deuxième scénario, plus complexe, fait appel à la technologie TrustZone pour isoler la solution sécurisée, conçue par l'entreprise S, de la solution non-sécurisée, conçue par l'entreprise A. Dans ce scénario, l'entreprise S configure en outre les plages mémoire associées à la solution sécurisée, les permissions en lecture et en écriture de l'environnement non-sécurisé et, si nécessaire, les espaces d'échanges entre les environnements sécurisé et non-sécurisé dans le tableau *NVM rows* de la mémoire Flash. En particulier, l'entreprise S définit les fonctions de la solution sécurisée qui peuvent être appelées par la solution non-sécurisée de l'entreprise A, suivant une convention d'appel sécurisée. Une interface de programmation est alors mise à disposition des développeurs de l'entreprise A sous la forme de librairies (`".lib"`) et de leurs fichiers d'en-tête (`".h"`).

Afin de garantir la confidentialité des informations enregistrées dans le microcontrôleur SAML11, la protection anti-débogage et le contrôleur de la mémoire Flash fonctionnent en tandem. Le passage du niveau DAL0 au niveau DAL1 provoque l'effacement de la solution non-sécurisée de l'entreprise A, tandis que celui d'un niveau quelconque au niveau DAL2 provoque l'effacement de l'intégralité du circuit. Ainsi, pour rétablir les accès en débogage à une zone protégée, il est préalablement nécessaire de l'effacer.

4.4.3.C Mécanisme de *secure boot*

Programme du *secure boot*. Le premier logiciel exécuté au démarrage du microcontrôleur SAML11 est le logiciel d'amorçage du constructeur programmé dans la ROM du circuit. Ce logicel implémente toutes les étapes du *secure boot* : il initialise le circuit, configure l'environnement sécurisé et vérifie l'intégrité et l'authenticité du *bootloader* programmé dans la mémoire Flash du circuit avant de l'exécuter. Afin d'empêcher la relecture et la modification du logiciel, la ROM est configurée par le constructeur en exécution seule. Nous n'avons trouvé aucun détail technologique sur l'implémentation de cette mémoire.

Paramètres du *secure boot*. Un tableau auxiliaire de la mémoire Flash du microcontrôleur, nommé *NVM rows*, enregistre les paramètres du *secure boot*, la configuration de l'environment sécurisé et les données de calibration du circuit. Les paramètres du *secure boot*, en particulier, sont mémorisés dans une ligne du tableau désignée par l'acronyme BOCOR (*Boot Configuration Row*). Ils sont listés ci-dessous :

— `"BS"` (8 bits) : taille du *bootloader* sécurisé ;
— `"BNSC"` (6 bits) : taille de la zone du *bootloader* sécurisé réservée aux appels de fonctions sécurisées depuis l'environnement non-sécurisé ;

— "BOOTOPT" (8 bits) : sélection du mode d'amorçage ;

— "BOOTPROT" (8 bits) : plage mémoire protégée par le mécanisme de *secure boot* ;

— "BCWEN" (1 bit) : autorisation des accès en écriture sur la ligne BOCOR ;

— "BCREN" (1 bit) : autorisation des accès en lecture sur la ligne BOCOR ;

— "BOCORCRC" (32 bits) : CRC-32 de "BS", "BNSC", "BOOTOPT", "BOOTPROT", "BCWEN" et "BCREN" ;

— "BOOTKEY" (256 bits) : clé d'authentification ;

— "BOCORHASH" (256 bits) : empreinte de la ligne BOCOR par la fonction SHA-256, à l'exception de "BOCORHASH".

Mode d'amoçage. Il est défini par la valeur du paramètre "BOOTOPT" qui est testée par le programme du *secure boot* à chaque démarrage du microcontrôleur.

— Si cette valeur est 0x01, le programme du *secure boot* calcule l'empreinte du *bootloader* et compare le résultat à une empreinte de référence mémorisée, avec le *bootloader*, dans la mémoire Flash. Si les empreintes sont identiques, le *bootloader* est exécuté. Sinon, le circuit est remis à zéro.

— Si cette valeur est 0x02 ou 0x03, la valeur du paramètre "BOOTKEY" est utilisée par le programme du *secure boot* lors du calcul de l'empreinte du *bootloader*. Par conséquent, il est nécessaire de connaître la valeur du paramètre "BOOTKEY" pour forger une nouvelle empreinte.

— Pour toutes les autres valeurs, le microcontrôleur exécute le *bootloader* sans le vérifier.

L'utilisation du *secure boot* est détaillée dans le cas d'une vérification de l'intégrité et de l'authenticité d'un *bootloader* programmé dans la zone sécurisée de la mémoire Flash du circuit.

Programmation du microcontrôleur. Avant la programmation du microcontrôleur, une clé d'authentification de 256 bits ("BOOTKEY") est d'abord choisie par le propriétaire du *bootloader* qui est programmé dans la mémoire Flash du circuit. L'empreinte de cette clé par la fonction de hachage SHA-256 est alors utilisée comme vecteur d'initialisation pour calculer l'empreinte du *bootloader*. Cette empreinte, ou *hash*, est insérée dans le *bootloader*, puis le code binaire ainsi constitué est programmé au début de la mémoire Flash du microcontrôleur. La ligne BOCOR est ensuite programmée avec notamment la taille du *bootloader* "BOOTPROT" et la clé d'authentification "BOOTKEY". La valeur '0' est assignée aux paramètres "BCWEN" et "BCREN" pour empêcher, d'un côté, la relecture de la clé d'authentification et, de l'autre, la modification des paramètres du *secure boot*. Les valeurs des paramètres "BOCORCRC" et "BOCORHASH" sont finalement calculées et programmées avec la ligne BOCOR. Les spécifications de l'algorithme CRC-32 utilisé par le circuit sont détaillées dans la documentation technique du MCU [116]. L'agencement de la mémoire Flash à l'issue de ce processus est illustré sur la Figure 4.7

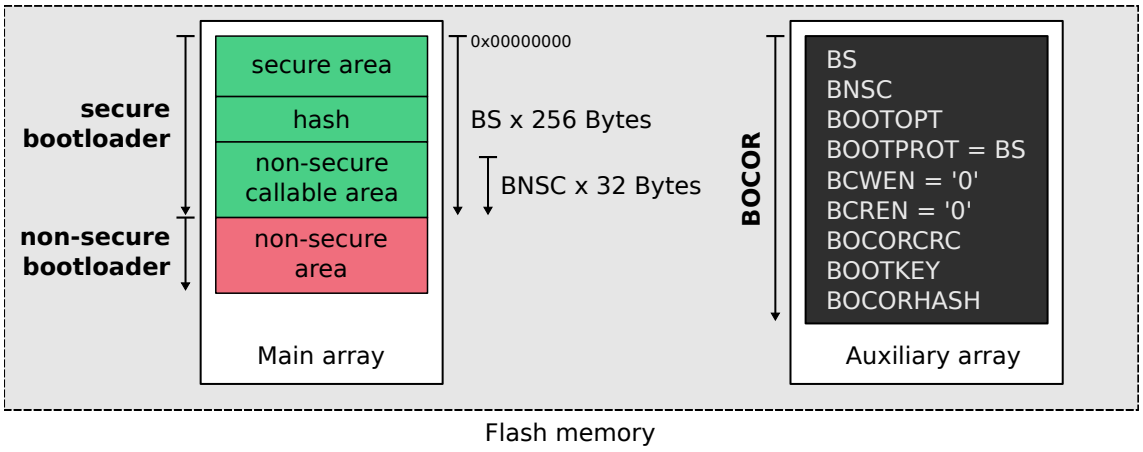


FIGURE 4.7 – Mémoire Flash du microcontrôleur SAML11 après la programmation du *bootloader*.

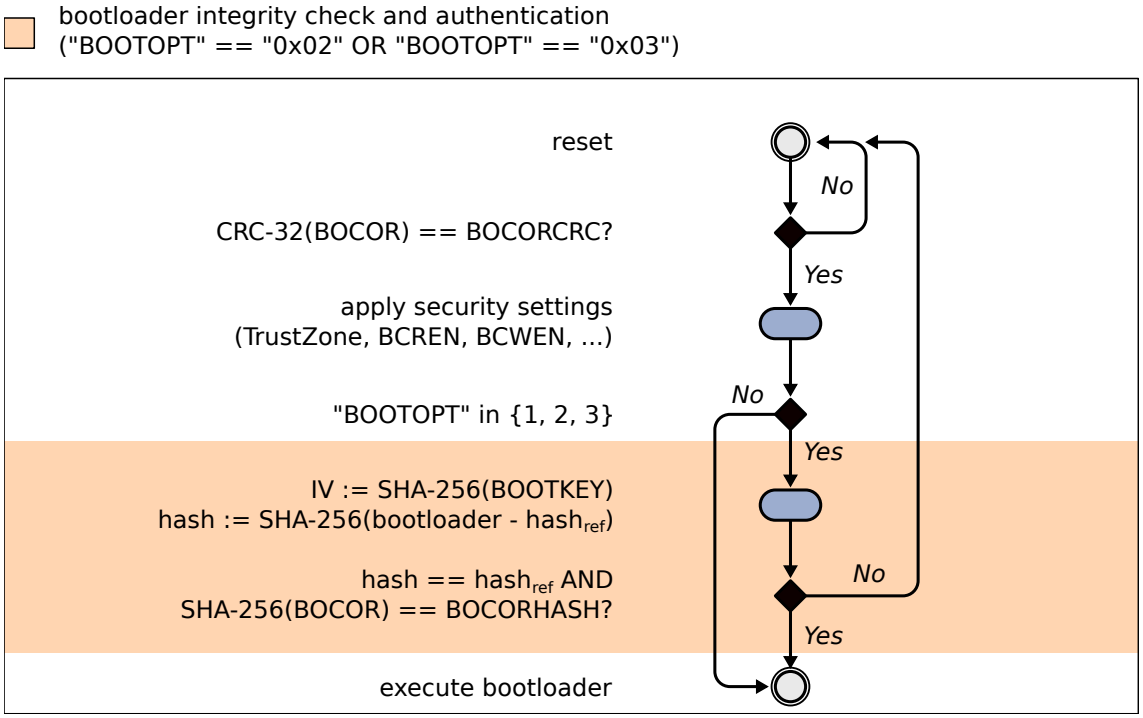


FIGURE 4.8 – Séquence d'amorçage du microcontrôleur SAML11.

Séquence d'amorçage. Au démarrage du microcontrôleur, le programme d'amorçage mémorisé dans la ROM vérifie l'intégrité de la ligne BOCOR et d'une partie du tableau *NVM rows* grâce à un CRC-32. Si la vérification réussit, le programme utilise la configuration de sécurité mémorisée dans le tableau *NVM rows* pour mettre à jour les attributs de sécurité des mémoires et des périphériques du microcontrôleur. Il copie notamment les paramètres `"BCREN"` et `"BWREN"` de la ligne BOCOR pour mettre à jour le contrôleur de la mémoire Flash. Le programme détermine ensuite le mode d'amorçage à partir de la valeur du paramètre `"BOOTOPT"`. Dans le cas où le microcontrôleur a été configuré pour utiliser la clé d'authentification `"BOOTKEY"`, les étapes suivantes sont réalisées :

1. La fonction SHA-256 de l'accélérateur cryptographique est tout d'abord utilisée pour calculer l'empreinte de la clé `"BOOTKEY"` et le résultat est mémorisé en SRAM.
2. La même fonction est ensuite utilisée pour calculer l'empreinte du *bootloader* avec, pour vecteur d'initialisation, le résultat du calcul précédent. L'empreinte de référence du *bootloader*, programmée dans la mémoire Flash, est exclue du calcul.
3. L'empreinte calculée par l'accélérateur cryptographique est finalement comparée à l'empreinte de référence. Si les empreintes sont identiques, le *bootloader* est intègre et authentique. Une procédure similaire est réalisée pour vérifier l'intégrité de la ligne BOCOR. Si les deux vérifications réussissent, le *bootloader* est exécuté. En revanche, si l'une des vérifications échoue, le circuit est remis à zéro.

Cette séquence d'amorçage est schématisée sur la Figure 4.8

4.4.4 Identification des vecteurs d'attaques en faute

4.4.4.A Modèle de menace

Nous limitons notre analyse des vulnérabilités du microcontrôleur SAML11 face aux attaques par injection de fautes au mécanisme de *secure boot*. Nous supposons, dans ce cadre, qu'une seule entreprise, l'entreprise A, développe la solution qui utilise le microcontrôleur SAML11. Cette solution s'exécute, par défaut, dans l'environnement sécurisé du microcontrôleur.

L'entreprise A programme tout d'abord le microcontrôleur avec son *bootloader*, une application et une clé d'authentification `"BOOTKEY"` de 256 bits. Elle configure ensuite le microcontrôleur pour valider le *bootloader* à chaque démarrage du circuit grâce à la fonctionnalité *secure boot*. Le *bootloader* valide à son tour l'application avant de l'exécuter. Une chaîne de confiance s'établit ainsi entre la RoT du microcontrôleur – constituée de la ROM et de la clé d'authentification – et l'application de l'entreprise A. Cette dernière bascule enfin le niveau d'accès en débogage du microcontrôleur sur DAL0. Les fonctionnalités de débogage sont alors limitées à quelques fonctionnalités de tests et la solution de l'entreprise ne peut être ni relue, ni modifiée par l'utilisateur final.

Nous supposons, en premier lieu, qu'un attaquant dispose d'un accès physique à un circuit programmé et configuré par l'entreprise A. L'attaquant ne peut donc ni relire, ni modifier la solution de l'entreprise A grâce aux fonctionnalités de débogage. Nous

supposons, en second lieu, que l'attaquant peut se procurer des circuits SAML11E16 vierges. Ces circuits peuvent alors être utilisés pour calibrer une attaque par injection de fautes avant de reproduire l'attaque sur le microcontrôleur ciblé. En partant de ces hypothèses, nous présentons plusieurs vecteurs d'attaques par injection de fautes qui permettent à une attaquant d'extraire la clé d'authentification `"BOOTKEY"` ou de contourner le mécanisme de *secure boot* afin d'exécuter un *bootloader* modifié à partir de la mémoire Flash.

4.4.4.B Extraction de la clé d'authentification

Dans cette attaque, nous supposons que l'attaquant peut synchroniser précisément une injection de faute avec l'exécution du code d'amorçage mémorisé dans la ROM. Pour ce faire, le déclenchement d'un générateur d'impulsions EM ou d'un banc d'injection laser peut être synchronisé sur le signal de remise à zéro du circuit. En effet, l'exécution du code d'amorçage après la remise à zéro du circuit est particulièrement prédictible, car ce dernier implémente un flot de contrôle très simple qui n'est pas perturbé par des interruptions matérielles. Un attaquant peut ainsi espérer corrompre les mêmes instructions avec un fort taux de succès en ajustant le délai entre le basculement du signal de remise à zéro du circuit et la génération d'une impulsion EM ou laser.

Le modèle de faute *bit set* étudié dans le chapitre 3 permet à un attaquant de forcer un bit d'indice choisi à `'1'` lors de la lecture d'un mot dans la mémoire NOR Flash d'un MCU grâce à une impulsion laser. Or, la clé `"BOOTKEY"` est mémorisée dans la mémoire NOR Flash du MCU SAML11E16. Une faute de type *bit set* peut donc être injectée sur un bit d'indice choisi dans la clé d'authentification pendant la lecture de la clé. Puisque l'intégrité de cette dernière est vérifiée avec celle de la colonne `"BOCOR"` grâce à un algorithme CRC-32 pendant le démarrage du circuit, l'injection d'une faute de type *bit set* sur un bit de la clé dont la valeur initiale est `'0'` fait systématiquement échouer le processus d'amorçage. L'attaque par SEA sur le transfert d'une clé secrète décrite dans la section 4.3.2 s'applique ainsi à la relecture de la clé `"BOOTKEY"` lors de la vérification de son intégrité.

Pour réaliser cette attaque, un circuit de la série SAML11E16 doit d'abord être cartographié afin de reconstruire l'organisation de la mémoire et d'identifier les positions des colonnes mémoire. Un balayage temporel et spatial des fautes obtenues sur le même circuit pendant la vérification de l'intégrité d'une clé d'authentification connue de l'attaquant permet ensuite d'identifier les instants pour lesquels les différentes parties de la clé de 256 bits sont relues dans la mémoire Flash. L'attaque peut finalement être reproduite sur le circuit cible avec les instants et les positions identifiés afin d'extraire les 256 bits de la clé d'authentification.

Remarquons que le MCU SAML11E16 permet de réaliser un *scrambling* d'une partie de la mémoire Flash réservée à la mémorisation des données sensibles. Ce procédé permet de dissimuler la valeur des données sensibles en combinant ces dernières avec un masque pseudo-aléatoire généré à partir d'une clé configurée dans les registres du contrôleur de la mémoire Flash. Cette fonctionnalité constitue ainsi une bonne protection contre l'attaque par SEA décrite dans la section 4.3.2, car un attaquant

doit alors reconstruire les masques utilisés après avoir extrait la valeur masquée de la clé [105]. Cependant, ce mécanisme ne concerne pas la colonne "BOCOR" de la mémoire Flash qui est la cible de cette attaque. Une extension de la fonctionnalité de *scrambling* au tableau auxiliaire de la mémoire pourrait ainsi être envisagée pour protéger le circuit contre les attaques en faute par SEA.

4.4.4.C Corruption de la configuration de sécurité

La configuration "BOCOR" mémorisée dans la mémoire Flash contient notamment la clé "BOOTKEY" utilisée pour authentifier le *bootloader* lors d'un amorçage sécurisé, le paramètre "BOOTOPT" qui détermine le mode d'amorçage du circuit, les paramètres "BREN" et "BWEN" qui définissent les droits d'accès en lecture et en écriture à la configuration "BOCOR" et la valeur d'un CRC-32 qui permet de vérifier l'intégrité de la configuration lors du démarrage du MCU SAML11E16. Une corruption du paramètre "BOOTOPT" permettrait ainsi de désactiver le mode d'amorçage sécurisé du circuit. Une corruption des paramètres "BREN" et "BWEN" permettrait de rétablir les accès en lecture et en écriture à la configuration "BOCOR" dans l'objectif de relire la clé "BOOTKEY". Les paragraphes suivants détaillent nos hypothèses de travail et les chemins d'attaques possibles avec les modèles de fautes décrits dans ces travaux.

Si la configuration "BOCOR" est transférée dans une mémoire volatile lors du démarrage du circuit, l'injection d'une faute sur l'interface de lecture de la mémoire Flash du circuit permet de modifier de manière persistante les valeurs de la configuration "BOCOR". Cependant, ces modifications sont détectées ultérieurement par la vérification de l'intégrité de la configuration. Une solution alternative consiste alors à perturber les mémoires volatiles où la configuration "BOCOR" est mémorisée grâce à une impulsion laser après la vérification de l'intégrité de la configuration. Une modification de la configuration permet alors de modifier les droits d'accès en lecture et en écriture à la configuration, ou de forcer le mode d'amorçage du circuit en mode non-sécurisé de manière analogue aux résultats obtenus par VASSELLE, THIEBEAULD, MAOUHOUB et al. [173].

Cependant, la mémorisation de la configuration "BOCOR" dans une mémoire volatile nécessite soit de réserver une partie du circuit intégré pour implémenter un registre fantôme – ou *shadow register* – qui n'est utilisé que dans la phase d'amorçage du circuit, soit d'exposer la configuration dans la mémoire SRAM au risque qu'une vulnérabilité permette à un attaquant de relire ou de modifier la configuration dans cette mémoire [127]. Il est donc probable que la configuration "BOCOR" ne soit pas transférée dans une mémoire volatile lors du démarrage du circuit. Nous pouvons ainsi supposer que les valeurs des paramètres mémorisés dans la configuration "BOCOR" sont relues dans la mémoire Flash à chaque utilisation.

Nous supposons qu'un attaquant peut synchroniser précisément une perturbation électromagnétique ou laser de sorte à injecter une faute de type *bit set* pendant la lecture des paramètres "BREN", "BWEN" ou "BOOTOPT" en mémoire Flash, conformément aux modèles de fautes présentés dans ces travaux. Si une faute de type *bit set* est injectée sur les paramètres "BREN" et "BWEN" juste avant la mise à jour des registres du contrôleur

de la mémoire Flash (voir Figure 4.8), alors les accès en lecture et en écriture à la colonne "BOCOR" sont rétablis. Dès lors, la clé "BOOTKEY" peut être relue en exploitant par exemple la commande de relecture "CMD_RAUX" mise à disposition par le programme d'amorçage sécurisé pour relire la configuration du circuit dans le mode DAL0 [116]. Si une faute de type *bit set* est injectée sur un ou plusieurs bits du paramètre "BOOTOPT" juste avant que ce dernier soit testé par le programme d'amorçage, le circuit démarre en mode non-sécurisé. Puisque les modèles de fautes décrits dans ces travaux ne modifient pas le contenu de la mémoire Flash, les fautes injectées pendant la lecture des paramètres avant leur utilisation ne sont pas détectées par la vérification de l'empreinte de la colonne "BOCOR" réalisée lors d'un amorçage sécurisé. L'injection d'une faute transitoire pendant la lecture des paramètres permet ainsi de modifier précisément la configuration de sécurité des périphériques et le flot de contrôle du mécanisme d'amorçage sécurisé du MCU SAML11E16.

4.5 Conclusion

Dans ce chapitre, nous avons évalué en pratique les modèles de fautes étudiés dans nos expérimentations sur l'injection de fautes EM (chapitre 2) et laser (chapitre 3).

Nous avons montré que ces modèles de fautes peuvent être exploités pour corrompre le flot de contrôle d'un logiciel avec une forte répétabilité, en empêchant l'exécution d'une instruction grâce à une impulsion EM ou en modifiant la valeur de l'incrément d'un compteur d'itérations grâce à une impulsion laser. La précision spatiale des techniques d'injection de fautes laser permet dans ce cas d'obtenir des effets plus complexes que les techniques d'injection de fautes EM dont la précision spatiale limite les chemins d'attaques possibles.

Nous avons également montré que les fautes injectées localement sur les transferts de données grâce à une impulsion EM peuvent être exploitées pour extraire la clé secrète d'un algorithme de chiffrement grâce à une attaque par PFA ou grâce à une attaque sur le transfert d'une clé secrète. Ces attaques exploitent la possibilité d'injecter des fautes de type *bit set* ou de type *bit reset* sur un grand nombre de bits avec une forte répétabilité pendant les transferts de données entre la mémoire Flash et le *buffer* de préchargement des données d'un MCU. La possibilité de corrompre simultanément un grand nombre de bits grâce à une seule impulsion EM est, dans ce cas, utilisée à l'avantage de l'attaquant.

Nous avons réalisé en pratique une attaque par SEA en illuminant des zones ciblées de la mémoire NOR Flash d'un MCU lors du transfert d'une clé secrète entre la mémoire et les registres du circuit. Les valeurs des positions de l'objectif pour lesquelles une faute a été injectée pendant le transfert de la clé permettent d'extraire directement la valeur de cette dernière.

Nous avons enfin proposé une analyse des chemins d'attaques par injection de fautes sur une implémentation commerciale d'un mécanisme de *secure boot*. Parmi les trois chemins d'attaques proposés, le premier utilise l'attaque par SEA décrite précédemment pour extraire la valeur d'une clé d'authentification, tandis que les deux autres exploitent

les propriétés des modèles de fautes étudiés pour corrompre localement la configuration de sécurité d'un MCU lors de sa relecture en mémoire Flash.

Conclusion et perspectives

Les objets connectés embarquent généralement un MCU non-sécurisé. À la différence des MCU sécurisés, ces microcontrôleurs ne sont pas protégés contre les attaques matérielles. Un premier objectif consistait donc à analyser les modèles de fautes obtenus sur différents MCU non-sécurisés grâce à des techniques d'injection de fautes EM et laser. Un deuxième objectif consistait ensuite à évaluer les vulnérabilités causées par ces modèles de fautes. Un troisième objectif consistait enfin à évaluer la possibilité d'utiliser les attaques en faute comme point d'entrée pour compromettre la sécurité d'un objet connecté.

Dans le premier chapitre, nous avons présenté les différentes attaques matérielles et plus particulièrement les techniques d'injection de fautes, dans le contexte de l'internet des objets. Nous avons également étudié les mécanismes d'injection de fautes associé à deux techniques d'injection de fautes locales : l'injection de fautes EM et l'injection de fautes laser. Nous avons enfin identifié des lacunes dans la compréhension et l'évaluation des modèles de fautes obtenus grâce à ces deux techniques sur un MCU non-sécurisé.

Dans le deuxième chapitre, nous avons analysé les modèles de fautes obtenus grâce à une impulsion EM sur les *buffers* de préchargement des instructions et des données de deux MCU fabriqués dans les technologies CMOS 0.25 µm et 0.35 µm. Nous avons d'abord analysé les modèles de fautes matériels obtenus sur les transferts de données entre la mémoire Flash et le *buffer* de préchargement des données et nous avons mis en évidence la possibilité d'injecter arbitrairement des fautes de type *bit set* ou de type *bit reset* sur une partie seulement des données transférées en ajustant les coordonnées de la sonde d'injection, l'instant d'injection et la polarité d'une impulsion de tension à l'origine d'une EMP. Nous avons également montré que plusieurs mécanismes d'injection de fautes peuvent être observés simultanément en fonction de l'amplitude de l'impulsion de tension. Nous avons enfin étudié les propriétés des sauts d'instructions injectés sous l'effet d'une EMP et nous avons mis en évidence la possibilité d'injecter un saut d'une à six instructions de manière répétable en ajustant la largeur de l'impulsion de tension à l'origine d'une EMP.

Dans le troisième chapitre, nous avons analysé les modèles de fautes obtenus grâce à une impulsion laser sur les mémoires NOR Flash de deux MCU fabriqués dans les technologies CMOS 80 nm et 65 nm. Nous avons d'abord analysé les propriétés d'un modèle de faute de type *bit set* unipolaire sur la mémoire NOR Flash des deux MCU. Nous avons mis en évidence la possibilité d'injecter facilement des fautes sur un seul bit ou sur plusieurs bits avec une grande précision, une répétabilité de 100 % et peu de

contraintes de positionnement. Une faute a également été injectée sur 5 bits adjacents grâce à une seule impulsion laser avec 50 % de répétabilité en augmentant le diamètre du spot. Nous avons ensuite proposé une analyse du mécanisme physique d'injection de fautes au niveau des colonnes d'une mémoire NOR Flash et nous avons montré que ce mécanisme permet d'expliquer les spécificités des résultats obtenus sur les deux MCU. Nous avons enfin détaillé plusieurs applications de ce modèle de faute réalisées en pratique pour corrompre les instructions en transfert entre une mémoire Flash et un cœur de calcul.

Dans le quatrième chapitre, nous avons étudié différents scénarios d'applications des modèles de fautes de corruptions d'instructions et de données étudiés dans ces travaux. Nous avons montré en pratique que ces modèles de fautes permettent de corrompre le flot de contrôle d'un algorithme de sécurité ou d'extraire une clé secrète. En particulier, nous avons montré que les techniques d'injection de fautes EM et laser permettent d'explorer des chemins d'attaques différents en fonction de la localité des fautes injectées. Finalement, nous avons proposé une analyse des chemins d'attaques sur un mécanisme de *secure boot* implémenté dans un MCU conçu pour des applications sécurisées de l'IoT. Si ces chemins d'attaques n'ont pas été implémentés en pratique, ils soulignent toutefois la vulnérabilité de l'interface de la mémoire Flash où les configurations de sécurité et les clés secrètes sont mémorisées dans un microcontrôleur qui n'est pas sécurisé contre les attaques en faute.

Ces travaux constituent ainsi une première analyse expérimentale de plusieurs modèles de fautes originaux obtenus sur des MCU non-sécurisés grâce à des techniques d'injection de fautes locales. La compréhension de ces modèles de fautes accélère l'identification des paramètres d'injection sur un nouveau circuit, facilite la réalisation d'attaques et informe la conception de nouvelles contre-mesures.

Plusieurs directions de recherches ont été identifiées pour étendre ces résultats. Une première direction de recherche consisterait à analyser l'influence des paramètres d'une injection de fautes EM sur les mécanismes de fautes observés afin d'explorer d'autre modèles de fautes. Cette étude permettrait également de comprendre plus finement les spécificités du mécanisme d'injection de fautes EM. Une deuxième direction de recherche consisterait à explorer l'injection de fautes multiples, par exemple grâce à plusieurs impulsions successives, ou encore grâce à un banc d'injection laser multi-spots. Une troisième direction de recherche consisterait à étudier les effets du mécanisme d'injection de fautes laser décrit dans ces travaux sur les opérations d'écriture en mémoire Flash afin d'empêcher la mise à jour d'un paramètre de sécurité ou de corrompre la mise à jour d'un logiciel. Enfin, une dernière direction de recherche consisterait à étendre l'analyse de ce mécanisme, d'une part, à des nœuds technologiques plus avancés que les mémoires NOR Flash 80 nm et 65 nm étudiées dans ces travaux et, d'autre part, à des architectures de mémoire différentes, comme les mémoires EEPROM.

Annexe A

Synthèse des circuits étudiés

Les caractéristiques des circuits étudiés sont résumées dans la Table A.1. Les technologies de fabrication des microcontrôleurs ATmega328P et STM32F100RB nous sont connues. Bien que les technologies de fabrication des microcontrôleurs SAM3X8E et SAMD21G18A ne nous soient pas connues, elles peuvent être estimées en comparant les densités des mémoires Flash des deux microcontrôleurs à celles des microcontrôleurs ATmega328P et STM32F100RB respectivement.

Nous supposons, d'une part, que la densité d d'une mémoire Flash, définie comme le rapport entre le nombre de ses cellules mémoire et sa surface, est inversement proportionnelle au carré de la taille de gravure minimale des transistors λ^2 et, d'autre part, que le rapport de proportionnalité est relativement constant pour deux nœuds technologiques proches. Par exemple, la surface des cellules mémoire NOR Flash de technologie antérieures à la technologie 90 nm peut être estimée à $10\lambda^2$, tandis que ce rapport de proportionnalité augmente légèrement pour les technologies plus récentes [29]. Le produit $d \times \lambda^2$ peut ainsi être considéré constant pour les mémoires NOR Flash fabriquées à des nœuds technologiques proches.

Les densités des mémoires NOR Flash sont estimées à partir de mesures réalisées sur les images par microscopie de la face avant des circuits et sur les images infrarouges de la face arrière des circuits présenté dans la section 2.2.2 et dans la section 3.1.2. Nous vérifions alors l'approximation A.1 pour les microcontrôleurs SAM3X8E et ATmega328P, et l'approximation A.2 pour les microcontrôleurs SAMD21G18A et STM32F100RB. Ces approximations nous permettent d'estimer la taille de gravure de la mémoire Flash du microcontrôleur SAM3X8E à 250 nm, et celle du microcontrôleur SAMD21G18A à 65 nm.

$$d_{\text{ATmega328P}} \times 350\,\text{nm}^2 \approx d_{\text{SAM3X8E}} \times 250\,\text{nm}^2 \tag{A.1}$$

$$d_{\text{STM32F100RB}} \times 80\,\text{nm}^2 \approx d_{\text{SAMD21G18A}} \times 65\,\text{nm}^2 \tag{A.2}$$

Circuit test	SAM3X8E [16]	ATmega328P [15]	STM32F100RB [164]	SAMD21G18A [117]
Architecture				
CPU	32 bits Cortex-M3	8 bits AVR	32 bits Cortex-M3	32 bits Cortex-M0+
Architecture	mod. Harvard	Harvard	mod. Harvard	von Neumann
ISA	ARMv7M	AVR	ARMv7M	ARMv6M
Instructions	16 / 32 bits	16 bits	16 / 32 bits	16 / 32 bits
Fréquence	84 MHz	16 MHz	7.37 MHz	48 MHz
Technologie	250 nm	350 nm	80 nm	65 nm
Mémoires				
SRAM	96 ko	2 ko	8 ko	32 ko
NVM (Flash)	512 ko	32 ko	128 ko	256 ko
Densité (Flash)	0.3 bit/μm^2	0.16 bit/μm^2	1.12 bit/μm^2	1.63 bit/μm^2
Conditionnement				
Surface du circuit	6.1 mm × 6.1 mm	2.8 mm × 2.8 mm	2.8 mm × 3.2 mm	2.7 mm × 3.5 mm
Boîtier	LQFP144	DIP-28	LQFP64	TQFP48

TABLE A.1 – Synthèse des caractéristiques matérielles des circuits de tests.

Liste des sigles

AES	*Advanced Encryption Standard* p. 33
ASIC	circuit intégré spécifique p. 40
CC	Critères Communs p. 25
CFI	*Control Flow Integrity* p. 34
CHEI	*Channel Hot-Electron Injection* p. 54
CMOS	MOS complémentaire p. 11
CSV	*Comma-Separated Values* p. 72
DDoS	déni de service distribué p. 17
DES	*Data Encryption Standard* p. 33
DFA	*Differential Fault Analysis* p. 32
DFF	bascule à verrouillage D p. 36
DPA	analyse différentielle de la consommation p. 23
ECC	code correcteur d'erreurs p. 34
EM	électromagnétique p. 18
EMFI	injection de fautes électromagnétique p. 27
EMI	interférence électromagnétique p. 27
EMP	impulsion électromagnétique p. 27
FA	attaque en faute p. 23
FN	Fowler-Nordheim p. 54
FPGA	matrice prédiffusée programmable par l'utilisateur p. 40
FWHM	*full-width-at-half-maximum* p. 94
GPIO	broche générale d'entrée-sortie p. 22
GPR	registre général p. 12

IFA	*Ineffective Fault Analysis* p. 33
IoT	internet des objets p. 17
IR	infrarouge p. 95
ISA	architecture de jeu d'instructions p. 29
LFI	injection de fautes laser p. 44
MCU	microcontrôleur p. 12
MOSFET	transistor MOS à effet de champ p. 53
NIR	infrarouges proches p. 27
PFA	*Persistent Fault Analysis* p. 33
PGN	réseau d'alimentation électrique p. 26, 38
PIN	numéro d'identification personnel p. 139
RISC	processeur à jeu d'instructions réduit p. 43
RoT	racine de confiance p. 167
RSA	*Rivest-Shamir-Adleman* p. 32
RTL	*Register Transfer Level* p. 35
SA	amplificateur de détection p. 55
SCA	analyse des canaux cachés p. 18
SEA	*Safe Error Analysis* p. 33
SEM	microscopie électronique à balayage p. 25
SoC	système sur puce p. 22
SPA	analyse simple de la consommation p. 23
UART	émetteur-récepteur asynchrone universel p. 65
UV	ultraviolet p. 28
ZCE	zone de charge d'espace p. 47

Bibliographie

[1] K. M. ABDELLATIF et O. HÉRIVEAUX, « SiliconToaster : A Cheap and Programmable EM Injector for Extracting Secrets, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Milan, Italy, 2020, p. 35-40.

[2] M. AGOYAN, J. DUTERTRE, D. NACCACHE, B. ROBISSON et A. TRIA, « When Clocks Fail : On Critical Paths and Clock Faults, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, D. GOLLMANN, J. LANET et J. IGUCHI-CARTIGNY, éd., sér. LNCS, t. 6035, Springer, 2010, p. 182-193.

[3] M. AGOYAN, J.-M. M. DUTERTRE, A.-P. P. MIRBAHA, D. NACCACHE, A.-L. RIBOTTA et A. TRIA, « How to flip a bit ? » In *Proceedings of the IEEE International On-line Testing Symposium (IOLTS)*, Corfu, Greece, 2010, p. 235-239.

[4] M. M. ALAM, S. TAJIK, F. GANJI, M. TEHRANIPOOR et D. FORTE, « RAM-Jam : Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, 2019, p. 48-55.

[5] S. ANCEAU, P. BLEUET, J. CLÉDIÈRE, L. MAINGAULT, J. RAINARD et R. TUCOULOU, « Nanofocused X-Ray Beam to Reprogram Secure Circuits, » in *Proceedings of the International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, W. FISCHER et N. HOMMA, éd., sér. LNCS, t. 10529, Springer, 2017, p. 175-188.

[6] R. ANDERSON et M. KUHN, « Tamper Resistance : A Cautionary Note, » in *Proceedings of USENIX Workshop on Electronic Commerce*, t. 2, Oakland, California, 1996, p. 1.

[7] M. ANTONAKAKIS, M. BAILEY, M. BERNHARD, E. BURSZTEIN, J. COCHRAN, Z. DURUMERIC, J. A. HALDERMAN, L. INVERNIZZI, M. KALLITSIS, D. KUMAR, C. LEVER, Z. MA, J. MASON, D. MENSCHER, C. SEAMAN, N. SULLIVAN, K. THOMAS et Y. ZHOU, « Understanding the Mirai Botnet, » in *Proceedings of the USENIX Security Symposium*, Vancouver, BC, Canada, avr. 2017, p. 1093-1110.

[8] ARDUINO. (2019). Arduino Due, adresse : https://store.arduino.cc/arduino-due.

[9] ——, (2019). Arduino UNO Rev3, adresse : https://store.arduino.cc/arduino-uno-rev3.

[10] ——, (2019). Arduino Zero, adresse : https://store.arduino.cc/arduino-zero.

[11] ARM LTD. (2010). Cortex-M3 Revision r2p0 : Technical Reference Manual, adresse : https://developer.arm.com/documentation/ddi0337/h.

[12] ——, (2012). Cortex-M0+ Revision r0p1 : Technical Reference Manual, adresse : https://developer.arm.com/documentation/ddi0484/c.

[13] ——, (2018). ARMv6-M Architecture Reference Manual, adresse : https://developer.arm.com/documentation/ddi0419/latest/.

[14] ——, (2021). ARMv7-M Architecture Reference Manual, adresse : https://developer.arm.com/documentation/ddi0403/latest/.

[15] ATMEL CORPORATION. (2015). ATmega328P, adresse : https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

[16] ——, (2015). SAM3X / SAM3A Series, adresse : https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.

[17] ——, (2016). AVR Instruction Set Manual, adresse : http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf.

[18] AVTECH ELECTROSYSTEMS LTD. (2020). AVIR Series : 200 VOLT, 20 AND 50 kHz PULSE GENERATORS PULSE WIDTHS 2 ns TO 1 us, adresse : https://www.avtechpulse.com/catalog/avir_rev17.pdf.

[19] ——, (2020). AVL Series : 100, 160, 240, 450 VOLT 1 – 5 NS RISE TIME PULSE GENERATORS, adresse : https://www.avtechpulse.com/catalog/avl_rev29.pdf.

[20] J. BALASCH, D. ARUMÍ et S. MANICH, « Design and validation of a platform for electromagnetic fault injection, » in *Proceedings of the Conference on Design of Circuits and Integrated Systems*, Barcelona, Spain, 2017, p. 1-6.

[21] 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 *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Nara, Japan, 2011, p. 105-114.

[22] H. BAR-EL, H. CHOUKRI, D. NACCACHE, M. TUNSTALL et C. WHELAN, « The sorcerer's apprentice guide to fault attacks, » *Proceedings of the IEEE*, t. 94, n° 2, p. 370-382, fév. 2006.

[23] A. BARENGHI, G. BERTONI, L. BREVEGLIERI, M. PELLICIOLI et G. PELOSI, « Low Voltage Fault Attacks to AES and RSA on General Purpose Processors., » version 20130321 :101357, 2010. Cryptology ePrint Archive : https://eprint.iacr.org/2010/130.

[24] A. BARENGHI, L. BREVEGLIERI, I. KOREN, G. PELOSI et F. REGAZZONI, « Countermeasures against Fault Attacks on Software Implemented AES : Effectiveness and Cost, » in *Proceedings of the Workshop on Embedded Systems Security (WESS)*, Scottsdale, Arizona, 2010.

[25] R. BAUMANN, « Soft errors in advanced computer systems, » *IEEE Design and Test of Computers*, t. 22, n° 3, p. 258-266, 2005.

[26] P. BAYON, L. BOSSUET, A. AUBERT, V. FISCHER, F. POUCHERET, B. ROBISSON et P. MAURINE, « Contactless Electromagnetic Active Attack on Ring Oscillator Based True Random Number Generator, » in *Proceedings of the International Workshop on Constructive Side-channel Analysis and Secure Design (COSADE)*, W. SCHINDLER et S. A. HUSS, éd., sér. LNCS, t. 7275, Springer, 2012, p. 151-166.

[27] A. BECKERS, J. BALASCH, B. GIERLICHS, I. VERBAUWHEDE, S. OSUKA, M. KINUGAWA, D. FUJIMOTO et Y. HAYASHI, « Characterization of EM faults on ATmega328P, » in *Proceedings of the Joint International Symposium on Electromagnetic Compatibility*, Sapporo, Japan, 2019, p. 1-4.

[28] N. BERINGUIER-BOHER, M. LACRUCHE, D. EL-BAZE, J.-M. DUTERTRE, J.-B. RIGAUD et P. MAURINE, « Body Biasing Injection Attacks in Practice, » in *Proceedings of the Workshop on Cryptography and Security in Computing Systems*, Prague, Czech Republic, 2016, p. 49-54.

[29] R. BEZ, E. CAMERLENGHI, A. MODELLI et A. VISCONTI, « Introduction to flash memory, » *Proceedings of the IEEE*, t. 91, n° 4, p. 489-502, avr. 2003.

[30] E. BIHAM et A. SHAMIR, « Differential Fault Analysis of Secret Key Cryptosystems, » in *Advances in Cryptology – CRYPTO '97*, B. S. K. JR., éd., sér. LNCS, t. 1294, Springer, 1997, p. 513-525.

[31] J. BLÖMER et J. P. SEIFERT, « Fault based cryptanalysis of the Advanced Encryption Standard (AES), » in *Proceedings of the International Conferance on Financial Cryptography (FC)*, R. N. WRIGHT, éd., sér. LNCS, t. 2742, Guadeloupe, France : Springer, 2003, p. 162-181.

[32] D. BONEH, R. A. DEMILLO et R. J. LIPTON, « On the Importance of Eliminating Errors in Cryptographic Computations, » *J. Cryptology*, t. 14, n° 2, p. 101-119, mar. 2001.

[33] G. BOUFFARD, J. IGUCHI-CARTIGNY et J.-L. L. LANET, « Combined software and hardware attacks on the Java Card control flow, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, E. PROUFF, éd., sér. LNCS, t. 7079, Springer, 2011, p. 283-296.

[34] C. BOZZATO, R. FOCARDI et F. PALMARINI, « Shaping the Glitch : Optimizing Voltage Fault Injection Attacks, » *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, t. 2019, n° 2, p. 199-224, fév. 2019.

[35] J. BREIER et D. JAP, « Testing feasibility of back-side laser fault injection on a microcontroller, » in *Proceedings of the Workshop on Embedded Systems Security (WESS)*, New York, NY, USA, 2015, p. 1-6.

[36] J. BREIER, D. JAP et C.-N. N. CHEN, « Laser profiling for the back-side fault attacks : With a practical laser skip instruction attack on AES, » in *Proceedings of ACM Cyber-physical System Security Workshop (CPSS)*, Singapore, 2015, p. 99-103.

[37] J. BREWER et M. GILL, éd., *Nonvolatile Memory Technologies with Emphasis on Flash, A Comprehensive Guide to Understanding and Using Flash Memory Devices*. Hoboken, New Jersey : John Wiley & Sons, Inc., 2008.

[38] S. P. BUCHNER, F. MILLER, V. POUGET et D. P. MCMORROW, « Pulsed-Laser Testing for Single-Event Effects Investigations, » *IEEE Trans. Nucl. Sci.*, t. 60, n° 3, p. 1852-1875, 2013.

[39] T. CALIN, M. NICOLAIDIS et R. VELAZCO, « Upset hardened memory design for submicron CMOS technology, » *IEEE Trans. Nucl. Sci.*, t. 43, n° 6, p. 2874-2878, 1996.

[40] P. CAPPELLETTI, C. GOLLA, P. OLIVO et E. ZANONI, éd., *Flash Memories*. New York, NY, USA : Springer US, 1999.

[43] M. CHAMBONNEAU, S. SOUIKI-FIGUIGUI, P. CHIQUET, V. D. MARCA, J. POSTEL-PELLERIN, P. CANET, J.-M. PORTAL et D. GROJO, « Suppressing the memory state of floating gate transistors with repeated femtosecond laser backside irradiations, » *Appl. Phys. Lett.*, t. 110, n° 16, p. 161 112, avr. 2017.

[44] C. CHAMPEIX, N. BORREL, J.-M. M. DUTERTRE, B. ROBISSON, M. LISART et A. SARAFIANOS, « Experimental validation of a Bulk Built-In Current Sensor for detecting laser-induced currents, » in *Proceedings of the IEEE International On-line Testing Symposium (IOLTS)*, Halkidiki, Greece, 2015, p. 150-155.

[45] S. CHIH-TANG, « Evolution of the MOS transistor-from conception to VLSI, » *Proceedings of the IEEE*, t. 76, n° 10, p. 1280-1326, 1988.

[46] L. CLAUDEPIERRE et P. BESNIER, « Microcontroller Sensitivity to Fault-Injection Induced by Near-Field Electromagnetic Interference, » in *Joint International Symposium on Electromagnetic Compatibility, Proceedings of Sapporo Asia-pacific International Symposium Electromagnetic Compatibility (EMC Sapporo/APEMC)*, Sapporo, Japan, 2019, p. 673-676.

[47] C. CLAVIER, « Secret External Encodings Do Not Prevent Transient Fault Analysis, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, P. PAILLIER et I. VERBAUWHEDE, éd., sér. LNCS, t. 4727, Springer, 2007, p. 181-194.

[48] R. de CLERCQ, J. GÖTZFRIED, D. ÜBLER, P. MAENE et I. VERBAUWHEDE, « SOFIA : Software and control flow integrity architecture, » *Comput. Secur.*, t. 68, p. 16-35, 2017.

[49] L. COJOCAR, K. PAPAGIANNOPOULOS et N. TIMMERS, « Instruction Duplication : Leaky and Not Too Fault-Tolerant ! » In *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, T. EISENBARTH et Y. TEGLIA, éd., sér. LNCS, t. 10728, Springer, 2017, p. 160-179.

[52] (2021). Common Methodology for Information Technology Security Evaluation, adresse : https://www.commoncriteriaportal.org.

[53] F. COURBON, S. SKOROBOGATOV et C. WOODS, « Reverse Engineering Flash EEPROM Memories Using Scanning Electron Microscopy, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, K. LEMKE-RUST et M. TUNSTALL, éd., sér. LNCS, t. 10146, Springer, 2016, p. 57-72.

[54] A. CUI et R. HOUSLEY, « BADFET : Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection, » in *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, Vancouver, BC, Canada, 2017.

[55] J. DAEMEN et V. RIJMEN, *The design of Rijndael : AES — the Advanced Encryption Standard*, 1re éd. Berlin, Germany : Springer, 2002.

[56] J.-L. DANGER, A. FACON, S. GUILLEY, K. HEYDEMANN, U. KÜHNE, A. SI MERABET et M. TIMBERT, « CCFI-Cache : A Transparent and Flexible Hardware Protection for Code and Control-Flow Integrity, » in *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, Prague, Czech Republic, 2018, p. 529-536.

[57] A. DEHBAOUI, J.-M. M. DUTERTRE, B. ROBISSON et A. TRIA, « Electromagnetic transient faults injection on a hardware and a software implementations of AES, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Leuven, Belgium, 2012, p. 7-15.

[58] C. DESHPANDE, B. YUCE, N. F. GHALATY, D. GANTA, P. SCHAUMONT et L. NAZHANDALI, « A Configurable and Lightweight Timing Monitor for Fault Attack Detection, » in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, USA, 2016, p. 461-466.

[59] C. DOBRAUNIG, M. EICHLSEDER, T. KORAK, S. MANGARD, F. MENDEL et R. PRIMAS, « SIFA : Exploiting Ineffective Fault Inductions on Symmetric Cryptography, » *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, t. 2018, no 3, p. 547-572, 2018.

[60] A. DOUIN, V. POUGET, D. LEWIS, P. FOUILLAT et P. PERDU, « Electrical modeling for laser testing with different pulse durations, » in *Proceedings of the IEEE International On-line Testing Symposium (IOLTS)*, French Riviera, France, 2005, p. 9-13.

[61] M. DUMONT, « Modélisation de l'injection de faute électromagnétique sur circuits intégrés sécurisés et contre-mesures, » Ph.D. dissertation, Université de Montpellier, Montpellier, France, 2020.

[62] M. DUMONT, M. LISART et P. MAURINE, « Electromagnetic Fault Injection : How Faults Occur, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, 2019, p. 9-16.

[63] L. DUREUIL, « Analyse de code et processus d'évaluation des composants sécurisés contre l'injection de fautes, » Ph.D. dissertation, Université Grenoble Alpes, Grenoble, France, 2016.

[64] L. DUREUIL, G. PETIOT, M. POTET, T. LE, A. CROHEN et P. de CHOUDENS, « FISSC : A fault injection and simulation secure collection, » in *Proceedings of the International Conference on Computer Safety, Reliability, and Security*, A. SKAVHAUG, J. GUIOCHET et F. BITSCH, éd., sér. LNCS, t. 9922, Springer, 2016, p. 3-11.

[65] L. DUREUIL, M.-L. L. POTET, P. de CHOUDENS, C. DUMAS et J. CLÉDIÈRE, « From code review to fault injection attacks : Filling the gap using fault model inference, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, N. HOMMA et M. MEDWED, éd., sér. LNCS, t. 9514, Springer, 2015, p. 107-124.

[66] J. DUTERTRE, V. BEROULLE, P. CANDELIER, S. D. CASTRO, L. FABER, M. FLOTTES, P. GENDRIER, D. HÉLY, R. LEVEUGLE, P. MAISTRI, G. D. NATALE, A. PAPADIMITRIOU et B. ROUZEYRE, « Laser Fault Injection at the CMOS 28 nm Technology Node : an Analysis of the Fault Model, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Amsterdam, Netherlands, 2018, p. 1-6.

[67] J. M. DUTERTRE, J. J. FOURNIER, A. P. MIRBAHA, D. NACCACHE, J. B. RIGAUD, B. ROBISSON et A. TRIA, « Review of fault injection mechanisms and consequences on countermeasures design, » in *Proceedings of the IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Athens, Greece, 2011, p. 1-6.

[69] J. DUTERTRE, T. RIOM, O. POTIN et J. RIGAUD, « Experimental analysis of the laser-induced instruction skip fault model, » in *Proceedings of the Nordic Conference on Secure IT Systems (NordSec)*, A. ASKAROV, R. R. HANSEN et W. RAFNSSON, éd., sér. LNCS, t. 11875, Springer, 2019, p. 221-237.

[70] J.-M. M. DUTERTRE, A.-P. P. MIRBAHA, D. NACCACHE, A.-L. RIBOTTA, A. TRIA et T. VASCHALDE, « Fault Round Modification Analysis of the advanced encryption standard, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Francisco, CA, USA, 2012, p. 140-145.

[71] D. EL-BAZE, « Conception et évaluation d'un détecteur d'attaque par injection de fautes, » Ph.D. dissertation, École des mines de Saint-Étienne, Gardanne, France, 2017.

[72] D. EL-BAZE, J.-B. B. RIGAUD et P. MAURINE, « A fully-digital em pulse detector, » in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Dresden, Germany, 2016, p. 439-444.

[73] M. A. ELMOHR, H. LIAO et C. H. GEBOTYS, « EM Fault Injection on ARM and RISC-V, » in *Proceedings of the International Symposium on Quality Electronic Design*, Santa Clara, CA, USA, 2020, p. 206-212.

[74] ESA, *Single event effects test method and guidelines*, ESCC Basic Specification No. 25100. adresse : https://escies.org/download/specdraftapppub?id=3095.

[75] T. FUHR, É. JAULMES, V. LOMNÉ et A. THILLARD, « Fault Attacks on AES with Faulty Ciphertexts Only, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Los Alamitos, CA, USA, 2013, p. 108-118.

[76] C. GAINE, D. ABOULKASSIMI, S. PONTIÉ, J.-P. NIKOLOVSKI et J.-M. DUTERTRE, « Electromagnetic Fault Injection as a New Forensic Approach for SoCs, » in *Proceedings of the International Workshop on Information Forensics and Security*, New York, NY, USA, 2020, p. 1-6.

[77] GARTNER, INC. PRESS RELEASE. (2019). Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020, adresse : https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io.

[78] N. F. GHALATY, B. YUCE et P. SCHAUMONT, « A Systematic Approach to Fault Attack Resistant Design, » in *Fundamentals of IP and SoC Security : Design, Verification, and Debug*, S. BHUNIA, S. RAY et S. SUR-KOLAY, éd. Springer, 2017, p. 223-245.

[79] N. F. GHALATY, B. YUCE, M. TAHA et P. SCHAUMONT, « Differential Fault Intensity Analysis, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Busan, Korea, 2014, p. 49-58.

[80] M. GHODRATI, B. YUCE, S. GUJAR, C. DESHPANDE, L. NAZHANDALI et P. SCHAUMONT, « Inducing local timing fault through EM injection, » in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, San Francisco, CA, USA, 2018, 142 :1-142 :6.

[81] B. GIERLICHS, J.-M. SCHMIDT et M. TUNSTALL, « Infective Computation and Dummy Rounds : Fault Protection for Block Ciphers without Check-before-Output, » in *Progress in Cryptology, Proceedings of the International Conference on Cryptology and Information Security in Latin America*, A. HEVIA et G. NEVEN, éd., sér. LNCS, t. 7533, Springer, 2012, p. 305-321.

[82] C. GIRAUD, « DFA on AES, » in *Proceedings of the International Conference on the Advanced Encryption Standard (AES), Revised Selected and Invited Papers*, H. DOBBERTIN, V. RIJMEN et A. SOWA, éd., sér. LNCS, t. 3373, Bonn, Germany : Springer, 2004, p. 27-41.

[83] C. GIRAUD et H. THIEBEAULD, « A Survey on Fault Attacks, » in *Smart Card Research and Advanced Applications VI*, J. QUISQUATER, P. PARADINAS, Y. DESWARTE et A. A. E. KALAM, éd., sér. IFIP International Federation for Information Processing, t. 153, Springer, 2004, p. 159-176.

[84] O. M. GUILLEN, M. GRUBER et F. DE SANTIS, « Low-Cost Setup for Localized Semi-invasive Optical Fault Injection Attacks : How Low Can We Go ? » In *Proceedings of the International Workshop on Constructive Side-channel Analysis and Secure Design (COSADE)*, S. GUILLEY, éd., sér. LNCS, t. 10348, Springer, 2017, p. 207-222.

[85] D. H. HABING, « The use of lasers to simulate radiation-induced transients in semiconductor devices and circuits, » *IEEE Trans. Nucl. Sci.*, t. 12, n° 5, p. 91-100, oct. 1965.

[86] N. HOMMA, Y.-i. HAYASHI, N. MIURA, D. FUJIMOTO, D. TANAKA, M. NAGATA et T. AOKI, « EM Attack Is Non-invasive? - Design Methodology and Validity Verification of EM Attack Sensor, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, sér. LNCS, t. 8731, Springer, 2014, p. 1-16.

[87] J. U. HORSTMANN, H. W. EICHEL et R. L. COATES, « Metastability behavior of CMOS ASIC flip-flops in theory and test, » *IEEE J. Solid-State Circuits*, t. 24, n° 1, p. 146-157, fév. 1989.

[88] D. KAHNG et S. M. SZE, « A floating gate and its application to memory devices, » *Bell Syst. Tech. J.*, t. 46, n° 6, p. 1288-1295, 1967.

[89] M. S. KELLY et K. MAYES, « High Precision Laser Fault Injection using Low-cost Components., » in *Proceedings of the International Symposium on Hardware Oriented Security and Trust (HOST)*, San Jose, CA, USA, 2020, p. 219-228.

[90] M. S. KELLY, K. MAYES et J. F. WALKER, « Characterising a CPU fault attack model via run-time data analysis, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2017, p. 79-84.

[91] Z. KENJAR, T. FRASSETTO, D. GENS, M. FRANZ et A.-R. SADEGHI, « V0LTpwn : Attacking x86 Processor Integrity from Software, » in *Proceedings of the USENIX Security Symposium*, Vancouver, BC, Canada, 2020, p. 1445-1461.

[92] A. KERCKHOFFS, « La Cryptographie Militaire, » *Journal des Sciences Militaires*, t. 9, p. 5-38, jan. 1883.

[93] S. KERNS, B. SHAFER, L. ROCKETT, J. PRIDMORE, D. BERNDT, N. van VONNO et F. BARBER, « The design of radiation-hardened ICs for space : a compendium of approaches, » *Proceedings of the IEEE*, t. 76, n° 11, p. 1470-1509, 1988.

[94] M. KHARBOUCHE-HARRARI, J. POSTEL-PELLERIN, G. DI PENDINA, R. WACQUEZ, D. ABOULKASSIMI, M. BOCQUET, R. SOUSA, R. DELATTRE et J.-M. PORTAL, « Impact of a Laser Pulse on a STT-MRAM Bitcell : Security and Reliability Issues, » in *Proceedings of the International Symposium on On-Line Testing And Robust System Design*, Platja d'Aro, Spain, 2018, p. 243-244.

[95] C. H. KIM et J.-J. QUISQUATER, « Faults, Injection Methods, and Fault Attacks, » *IEEE Design and Test of Computers*, t. 24, n° 6, p. 544-545, 2007.

[96] Y. KIM, R. DALY, J. KIM, C. FALLIN, J. H. LEE, D. LEE, C. WILKERSON, K. LAI et O. MUTLU, « Flipping bits in memory without accessing them : An experimental study of DRAM disturbance errors, » in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, Minneapolis, MN, USA, 2014, p. 361-372.

[97] P. C. KOCHER, « Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, » in *Advances in Cryptology – CRYPTO '96*, N. KOBLITZ, éd., sér. LNCS, t. 1109, Springer, 1996, p. 104-113.

[98] P. C. KOCHER, J. JAFFE et B. JUN, « Differential Power Analysis, » in *Advances in Cryptology – CRYPTO '99*, M. J. WIENER, éd., sér. LNCS, t. 1666, Springer, 1999, p. 388-397.

[99] T. KORAK et M. HOEFLER, « On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Busan, South Korea, 2014, p. 8-17.

[100] D. S. V. KUMAR, A. BECKERS, J. BALASCH, B. GIERLICHS et I. VERBAUWHEDE, « An In-Depth and Black-Box Characterization of the Effects of Laser Pulses on ATmega328P, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, B. BILGIN et J. FISCHER, éd., sér. LNCS, t. 11389, Springer, 2019, p. 156-170.

[101] R. KUMAR, P. JOVANOVIC et I. POLIAN, « Precise fault-injections using voltage and temperature manipulation for differential cryptanalysis, » in *Proceedings of the IEEE International On-line Testing Symposium (IOLTS)*, Girona, Spain, 2014, p. 43-48.

[102] J.-F. LALANDE, K. HEYDEMANN et P. BERTHOMÉ, « Software Countermeasures for Control Flow Integrity of Smart Card C Codes, » in *Proceedings of the European Symposium on Research in Computer Security*, M. KUTYLOWSKI et J. VAIDYA, éd., sér. LNCS, t. 8713, Springer, 2014, p. 200-218.

[103] Y. LI, K. SAKIYAMA, S. GOMISAWA, T. FUKUNAGA, J. TAKAHASHI et K. OHTA, « Fault Sensitivity Analysis, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, S. MANGARD et F. STANDAERT, éd., sér. LNCS, t. 6225, Springer, 2010, p. 320-334.

[104] H. LIAO et C. GEBOTYS, « Methodology for EM Fault Injection : Charge-based Fault Model, » in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Florence, Italy, 2019, p. 256-259.

[105] P. LOUBET-MOUNDI, D. VIGILANT et F. OLIVIER. (2011). Static Fault Attacks on Hardware DES Registers. version 20111005 :140737. Cryptology ePrint Archive : https://eprint.iacr.org/2011/531.

[106] M. MADAU, M. AGOYAN, J. BALASCH, M. GRUJIĆ, P. HADDAD, P. MAURINE, V. ROŽIĆ, D. SINGELÉE, B. YANG et I. VERBAUWHEDE, « The Impact of Pulsed Electromagnetic Fault Injection on True Random Number Generators, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Amsterdam, Netherlands, 2018, p. 43-48.

[107] F. MAJÉRIC, « Etude d'attaques matérielles et combinées sur les « System-on-Chip », » Ph.D. dissertation, Université Jean Monnet, Saint-Étienne, France, 2016.

[108] T. G. MALKIN, F.-X. STANDAERT et M. YUNG, « A Comparative Cost/Security Analysis of Fault Attack Countermeasures, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, L. BREVEGLIERI, I. KOREN, D. NACCACHE et J. SEIFERT, éd., sér. LNCS, t. 4236, Springer, 2006, p. 159-172.

[109] F. MASUOKA, M. ASANO, H. IWAHASHI, T. KOMURO et S. TANAKA, « A new flash E2PROM cell using triple polysilicon technology, » in *Proceedings of the IEEE International Electron Devices Meeting (IEDM)*, San Francisco, CA, USA, 1984, p. 464-467.

[110] G. S. MAY et C. J. SPANOS, *Fundamentals of semiconductor manufacturing and process control*, 1re éd. Hoboken, New Jersey : John Wiley & Sons, Inc., 2006.

[114] D. MERLI, D. SCHUSTER, F. STUMPF et G. SIGL, « Semi-Invasive EM Attack on FPGA RO PUFs and Countermeasures, » in *Proceedings of the Workshop on Embedded Systems Security (WESS)*, Taipei, Taiwan, 2011, p. 1-9.

[115] MICROCHIP TECHNOLOGY INC. (2018). AN5365 : SAM L11 Security Reference Guide, adresse : http://ww1.microchip.com/downloads/en/AppNotes/SAM-L11-Security-ReferenceGuide-AN-DS70005365A.pdf.

[116] ——, (2020). SAM L10/L11 Family, adresse : https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-L10L11-Family-DataSheet-DS60001513F.pdf.

[117] ——, (2021). SAM D21/DA1 Family, adresse : https://ww1.microchip.com/downloads/en/DeviceDoc/SAM-D21DA1-Family-Data-Sheet-DS40001882G.pdf.

[118] S. W. MOORE, R. D. MULLINS, P. A. CUNNINGHAM, R. J. ANDERSON et G. S. TAYLOR, « Improving Smart Card Security Using Self-Timed Circuits, » in *International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, Manchester, UK, 2002, p. 211-218.

[119] N. MORO, K. HEYDEMANN, E. ENCRENAZ et B. ROBISSON, « Formal verification of a software countermeasure against instruction skip attacks, » *J. Cryptogr. Eng.*, t. 4, no 3, p. 145-156, sept. 2014.

[120] N. MORO, A. DEHBAOUI, K. HEYDEMANN, B. ROBISSON et E. ENCRENAZ, « Electromagnetic fault injection : Towards a fault model on a 32-bit microcontroller, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Los Alamitos, CA, USA, 2013, p. 77-88.

[121] N. MORO, K. HEYDEMANN, A. DEHBAOUI, B. ROBISSON et E. ENCRENAZ, « Experimental evaluation of two software countermeasures against fault attacks, » in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Arlington, VA, USA, 2014, p. 112-117.

[122] K. MURDOCK, D. OSWALD, F. D. GARCIA, J. VAN BULCK, D. GRUSS et F. PIESSENS, « Plundervolt : Software-based Fault Injection Attacks against Intel SGX, » in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2020, p. 1466-1482.

[123] O. MUTLU et J. S. KIM, « RowHammer : A Retrospective, » *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, t. 39, no 8, p. 1555-1571, 2020.

[124] D. M. H. NEIL H. E. WESTE, *CMOS VLSI Design : A Circuits and Systems Perspective*, 4e éd. Boston, MA, USA : Addison Wesley, 2010.

[125] NEWAE TECHNOLOGY INC. (2018). CW308 UFO target board, adresse : https://media.newae.com/datasheets/NAE-CW308-datasheet.pdf.

[126] ——, (2020). ChipWhisperer official GitHub repository, adresse : https://github.com/newaetech/chipwhisperer.

[127] J. OBERMAIER et S. TATSCHNER, « Shedding too much Light on a Microcontroller's Firmware Protection, » in *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, Vancouver, BC, Canada, 2017.

[128] C. O'FLYNN, « Fault Injection using Crowbars on Embedded Systems, » version 20160825 :050844, 2016. Cryptology ePrint Archive : https://eprint.iacr.org/2016/810.

[129] ——, « MIN()imum Failure : EMFI Attacks against USB Stacks, » in *Proceedings of the USENIX Workshop on Offensive Technologies (WOOT)*, Santa Clara, CA, USA, 2019.

[130] ——, « Low-Cost Body Biasing Injection (BBI) Attacks on WLCSP Devices, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, P. LIARDET et N. MENTENS, éd., sér. LNCS, t. 12609, Springer, 2020, p. 166-180.

[131] C. O'FLYNN et Z. CHEN, « ChipWhisperer : An open-source platform for hardware embedded security research, » in *Proceedings of the International Workshop on Constructive Side-channel Analysis and Secure Design (COSADE)*, sér. LNCS, t. 8622, Springer, 2014, p. 243-260.

[132] S. ORDAS, L. GUILLAUME-SAGE et P. MAURINE, « Electromagnetic fault injection : the curse of flip-flops, » *J. Cryptogr. Eng.*, t. 7, n° 3, p. 183-197, sept. 2017.

[133] S. ORDAS, L. GUILLAUME-SAGE, K. TOBICH, J. DUTERTRE et P. MAURINE, « Evidence of a Larger EM-Induced Fault Model, » in *Proceedings of the International Conference on Smart Card Research and Advanced Application (CARDIS)*, M. JOYE et A. MORADI, éd., sér. LNCS, t. 8968, Springer, 2014, p. 245-259.

[134] S. ORDAS, L. GUILLAUME-SAGE et P. MAURINE, « EM injection : Fault model and locality, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Saint Malo, France, 2015, p. 3-13.

[135] G. PIRET et J.-J. QUISQUATER, « A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, C. D. WALTER, Ç. K. KOÇ et C. PAAR, éd., sér. LNCS, t. 2779, Cologne, Germany : Springer, 2003, p. 77-88.

[136] F. POUCHERET, L. CHUSSEAU, B. ROBISSON et P. MAURINE, « Local electromagnetic coupling with CMOS integrated circuits, » in *Proceedings of the International Workshop on Electromagnetic Compatibility of Integrated Circuits (EMC COMPO)*, Dubrovnik, Croatia, 2011, p. 137-141.

[137] F. POUCHERET, K. TOBICH, M. LISART, L. CHUSSEAU, B. ROBISSON et P. MAURINE, « Local and direct EM injection of power into CMOS integrated circuits, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Nara, Japan, 2011, p. 100-104.

[138] J. PROY, K. HEYDEMANN, A. BERZATI et A. COHEN, « Compiler-assisted loop hardening against fault attacks, » *ACM Trans. Archit. Code Optim. (TACO)*, t. 14, n° 4, 36 :1-36 :25, 2017.

[139] J. PROY, K. HEYDEMANN, A. BERZATI, F. MAJÉRIC et A. COHEN, « A first ISA-level characterization of EM pulse effects on superscalar microarchitectures – A secure software perspective, » in *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*, Canterbury, CA, UK, 2019, p. 1-10.

[140] P. QIU, D. WANG, Y. LYU et G. QU, « VoltJockey : Breaching TrustZone by Software-Controlled Voltage Manipulation over Multi-Core Frequencies, » in *Proceedings of the Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2019, p. 195-209.

[141] B. RAZAVI, *Design of Analog CMOS Integrated Circuits*, 1re éd. New York, NY, USA : McGraw-Hill, 2001.

[142] J.-B. RIGAUD, P. MANET et M. JOYE, « Strengthening Hardware AES Implementations against Fault Attack, » *IET Inf. Secur.*, t. 1, n° 3, p. 106-110, sept. 2007.

[143] L. RIVIÈRE, Z. NAJM, P. RAUZY, J. DANGER, J. BRINGER et L. SAUVAGE, « High precision fault injections on the instruction cache of ARMv7-M architectures, » in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Washington, DC, USA, 2015, p. 62-67.

[144] E. RONEN, A. SHAMIR, A.-O. WEINGARTEN et C. O'FLYNN, « IoT Goes Nuclear : Creating a ZigBee Chain Reaction, » in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, mai 2017, p. 195-212.

[145] M. SACHDEV, *Defect Oriented Testing for CMOS Analog and Digital Circuits*, 1re éd., sér. Frontiers in Electronic Testing. New York, NY, USA : Springer US, 1999, t. 10.

[146] J. SAKAMOTO, D. FUJIMOTO et T. MATSUMOTO, « Laser-Induced Controllable Instruction Replacement Fault Attack, » *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, t. 103-A, n° 1, p. 11-20, 2020.

[147] D. SAMYDE, S. P. SKOROBOGATOV, R. J. ANDERSON et J. QUISQUATER, « On a New Way to Read Data from Memory, » in *Proceedings of the International IEEE Security in Storage Workshop*, Washington, DC, USA, 2002, p. 65-69.

[148] A. SARAFIANOS, O. GAGLIANO, V. SERRADEIL, M. LISART, J. DUTERTRE et A. TRIA, « Building the electrical model of the pulsed photoelectric laser stimulation of an NMOS transistor in 90nm technology, » in *Proceedings of the IEEE International Reliability Physics Symposium (IRPS)*, Monterey, CA, USA, 2013, 5B.5.1-5B.5.9.

[149] A. SARAFIANOS, C. ROSCIAN, J.-M. DUTERTRE, M. LISART et A. TRIA, « Electrical modeling of the photoelectric effect induced by a pulsed laser applied to an SRAM cell, » *Microelectron. Reliab.*, t. 53, n° 9-11, p. 1300-1305, nov. 2013.

[150] O. SAVRY, T. HISCOCK et M. E. MAJIHI, *Sécurité matérielle des systèmes : vulnérabilités des processeurs et techniques d'exploitation*. Malakoff, France : Dunod, 2020.

[151] P. SCHAUMONT, « Security in the Internet of Things : A challenge of scale, » in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Lausanne, Switzerland, 2017, p. 674-679.

[152] F. SCHELLENBERG, M. FINKELDEY, N. GERHARDT, M. HOFMANN, A. MORADI et C. PAAR, « Large laser spots and fault sensitivity analysis, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2016, p. 203-208.

[153] M. SCHINK, A. WAGNER, F. UNTERSTEIN et J. HEYSZL, « Security and Trust in Open Source Security Tokens, » version 20210517 :063843, 2021. Cryptology ePrint Archive : https://eprint.iacr.org/2021/640.

[154] J. SCHMIDT, M. HUTTER et T. PLOS, « Optical Fault Attacks on AES : A Threat in Violet, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Lausanne, Switzerland, 2009, p. 13-22.

[155] J.-M. SCHMIDT et M. HUTTER, « Optical and EM Fault-Attacks on CRT-based RSA : Concrete Results, » in *Proceedings of the Austrian Workhop on Microelectronics*, Graz, Austria : Verlag der Technischen Universität Graz, 2007, p. 61-67.

[156] J.-M. M. SCHMIDT et C. HERBST, « A practical fault attack on square and multiply, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Washington, DC, USA, 2008, p. 53-58.

[157] P. SCHWABE et K. STOFFELEN, « All the AES You Need on Cortex-M3 and M4, » in *Selected Areas in Cryptography – SAC*, R. AVANZI et H. M. HEYS, éd., sér. LNCS, t. 10532, Springer, 2016, p. 180-194.

[158] N. SELMANE, S. GUILLEY et J.-L. L. DANGER, « Practical setup time violation attacks on AES, » in *Proceedings of the European Dependable Computing Conference (EDCC)*, Kaunas, Lithuania, 2008, p. 91-98.

[159] S. P. SKOROBOGATOV, « Data Remanence in Flash Memory Devices, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, J. R. RAO et B. SUNAR, éd., sér. LNCS, t. 3659, Springer, 2005, p. 339-353.

[160] ——, « Local Heating Attacks on Flash Memory Devices, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, San Francisco, CA, USA, 2009, p. 1-6.

[161] ——, « Flash Memory Bumping Attacks, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, S. MANGARD et F. STANDAERT, éd., sér. LNCS, t. 6225, Springer, 2010, p. 158-172.

[162] ——, « Optical Fault Masking Attacks, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Santa Barbara, California, USA, 2010, p. 23-29.

[163] S. P. SKOROBOGATOV et R. J. ANDERSON, « Optical Fault Induction Attacks, » in *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, B. S. K. JR., Ç. K. KOÇ et C. PAAR, éd., sér. LNCS, t. 2523, Springer, 2002, p. 2-12.

[164] STMICROELECTRONICS. (2016). STM32F100x4 STM32F100x6 STM32F100x8 STM32F100xB, adresse : https://www.st.com/en/microcontrollers-microprocessors/stm32f100rb.html.

[165] A. TANG, S. SETHUMADHAVAN et S. STOLFO, « CLKSCREW : Exposing the Perils of Security-Oblivious Energy Management, » in *Proceedings of USENIX Security Symposium*, Vancouver, BC, Canada, 2017, p. 1057-1074.

[166] K. TOBICH, P. MAURINE, P.-Y. LIARDET, M. LISART et T. ORDAS, « Voltage Spikes on the Substrate to Obtain Timing Faults, » in *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, 2013, p. 483-486.

[167] K. TOBICH, P. MAURINE, T. ORDAS et P. Y. LIARDET, « Yet Another Fault Injection Technique : by Forward Body Biasing Injection, » in *YACC : Yet Another Conference on Cryptography*, Porquerolles Island, France, 2012.

[168] O. TRABELSI, L. SAUVAGE et J.-L. DANGER, « Characterization at Logical Level of Magnetic Injection Probes, » in *Joint International Symposium on Electromagnetic Compatibility, Proceedings of Sapporo Asia-pacific International Symposium Electromagnetic Compatibility (EMC Sapporo/APEMC)*, Sapporo, Japan, 2019, p. 625-628.

[169] E. TRICHINA et R. KORKIKYAN, « Multi fault laser attacks on protected CRT-RSA, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Santa Barbara, California, USA, 2010, p. 75-86.

[170] T. TROUCHKINE, G. BOUFFARD et J. CLÉDIÈRE, « Fault Injection Characterization on Modern CPUs, » in *Information Security Theory and Practice*, M. LAURENT et T. GIANNETSOS, éd., sér. LNCS, t. 12024, Springer, 2019, p. 123-138.

[171] B. VAKIL et T. LINTON. (2021). Why We're in the Midst of a Global Semiconductor Shortage, adresse : https://hbr.org/2021/02/why-were-in-the-midst-of-a-global-semiconductor-shortage.

[172] J. G. VAN WOUDENBERG, M. F. WITTEMAN et F. MENARINI, « Practical optical fault injection on secure microcontrollers, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Tokyo, Japan, 2011, p. 91-99.

[173] A. VASSELLE, H. THIEBEAULD, Q. MAOUHOUB, A. MORISSET et S. ERMENEUX, « Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Taipei, Taiwan, 2017, p. 41-48.

[174] I. VERBAUWHEDE, D. KARAKLAJIC, J.-M. M. SCHMIDT, D. KARAKLAJIĆ et J.-M. M. SCHMIDT, « The fault attack jungle - A classification model to guide you, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Nara, Japan, 2011, p. 3-8.

[175] A. VIJAYAKUMAR, V. C. PATIL, D. E. HOLCOMB, C. PAAR et S. KUNDU, « Physical Design Obfuscation of Hardware : A Comprehensive Investigation of Device and Logic-Level Techniques, » *IEEE Trans. Inf. Forensics Secur.*, t. 12, n° 1, p. 64-77, 2017.

[176] M. WAGNER, « 700+ Attacks Published on Smart Cards : The Need for a Systematic Counter Strategy, » in *Proceedings of the International Workshop on Constructive Side-channel Analysis and Secure Design (COSADE)*, W. SCHINDLER et S. A. HUSS, éd., sér. LNCS, t. 7275, Darmstadt, Germany : Springer, 2012, p. 33-38.

[177] V. WERNER, L. MAINGAULT et M.-L. POTET, « An End-to-End Approach for Multi-Fault Attack Vulnerability Assessment, » in *Proceedings of the International Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, Milan, Italy, 2020, p. 10-17.

[178] S. YEN et M. JOYE, « Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis, » *IEEE Trans. Computers*, t. 49, n° 9, p. 967-970, sept. 2000.

[179] B. YUCE, N. F. GHALATY, H. SANTAPURI, C. DESHPANDE, C. PATRICK et P. SCHAUMONT, « Software Fault Resistance is Futile : Effective Single-Glitch Attacks, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Santa Barbara, CA, USA, 2016, p. 47-58.

[180] F. ZHANG, X. LOU, X. ZHAO, S. BHASIN, W. HE, R. DING, S. QURESHI et K. REN, « Persistent Fault Analysis on Block Ciphers, » *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, t. 2018, n° 3, p. 150-172, août 2018.

[181] F. ZHANG, Y. ZHANG, H. JIANG, X. ZHU, S. BHASIN, X. ZHAO, Z. LIU, D. GU et K. REN, « Persistent Fault Attack in Practice, » *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, t. 2020, n° 2, p. 172-195, 2020.

[182] K. M. ZICK, M. SRIVASTAV, W. ZHANG et M. FRENCH, « Sensing Nanosecond-Scale Voltage Attacks and Natural Transients in FPGAs, » in *Proceedings of the ACM/-SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, Monterey, CA, USA, 2013, p. 101-104.

[183] L. ZUSSA, A. DEHBAOUI, K. TOBICH, J.-M. M. DUTERTRE, P. MAURINE, L. GUILLAUME-SAGE, J. CLÉDIÈRE, A. TRIA, J. CLEDIERE et A. TRIA, « Efficiency of a glitch detector against electromagnetic fault injection, » in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, Dresden, Germany, 2014, p. 1-6.

[184] L. ZUSSA, J. DUTERTRE, J. CLÉDIÈRE et B. ROBISSON, « Analysis of the fault injection mechanism related to negative and positive power supply glitches using an on-chip voltmeter, » in *Proceedings of the IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Arlington, VA, USA, 2014, p. 130-135.

Publications

[41] P. CAYREL, B. COLOMBIER, V. DRAGOI, A. MENU et L. BOSSUET, « Message-recovery Laser Fault Injection Attack on Code-based Cryptosystems, » 2020. Cryptology ePrint Archive : https://eprint.iacr.org/2020/900/20210226:145132.

[42] ——, « Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem, » in *Advances in Cryptology – EUROCRYPT '2021*, A. CANTEAUT et F. STANDAERT, éd., sér. LNCS, t. 12697, Springer, 2021, p. 438-467.

[50] B. COLOMBIER, A. MENU, J. DUTERTRE, P. MOËLLIC, J. RIGAUD et J. DANGER, « Laser-induced Single-bit Faults in Flash Memory : Instructions Corruption on a 32-bit Microcontroller, » version 20190226 :143620, 2018. Cryptology ePrint Archive : https://eprint.iacr.org/2018/1042.

[51] ——, « Laser-induced Single-bit Faults in Flash Memory : Instructions Corruption on a 32-bit Microcontroller, » in *Proceedings of the IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, McLean, VA, USA, 2019, p. 1-10.

[68] J.-M. DUTERTRE, A. MENU, O. POTIN, J.-B. RIGAUD et J.-L. DANGER, « Experimental analysis of the electromagnetic instruction skip fault model and consequences for software countermeasures, » *Microelectron. Reliab.*, t. 121, p. 114 133, 2021.

[111] A. MENU, S. BHASIN, J. DUTERTRE, J. RIGAUD et J. DANGER, « Precise Spatio-Temporal Electromagnetic Fault Injections on Data Transfers, » in *Proceedings of the International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, 2019, p. 1-8.

[112] A. MENU, J. DUTERTRE, O. POTIN, J. RIGAUD et J. DANGER, « Experimental Analysis of the Electromagnetic Instruction Skip Fault Model, » in *Proceedings of the IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, Marrakech, Morocco, 2020, p. 1-7.

[113] A. MENU, J. DUTERTRE, J. RIGAUD, B. COLOMBIER, P. MOËLLIC et J. DANGER, « Single-bit Laser Fault Model in NOR Flash Memories : Analysis and Exploitation, » in *Proceedings of the International Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, Milan, Italy, 2020, p. 41-48.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *2021LYSEM032*

Alexandre MENU

HARDWARE SECURITY OF IOT DEVICES

Speciality : Microelectronics

Keywords : hardware security, IoT, microcontroller, fault attack, LFI, EMFI, NOR Flash.

Abstract :

Securing connected devices is one of the biggest challenge of the internet of things. Among existing threats, an attacker with a physical access to a device can use electromagnetic fault injection (EMFI) and laser fault injection (LFI) to alter the behavior of an electronic device in order to extract confidential information or be granted unauthorized privileges. While an extensive litterature describes how these techniques can be used to attack cryptographic devices, the threat that they represent for non-secure microcontrollers has not been fully assessed yet. This work provides an in-depth analysis of faults injected in four non-secure microcontrollers manufactured in CMOS 0.35 µm, 0.25 µm, 80 nm and 65 nm technologies with EMFI and LFI. Faults are injected with an EM pulse in instructions and data loaded in the buffers of the embedded Flash memory with a high repeatability and the possibility to inject either bit set or bit reset faults. An in-depth analysis of each injection parameter and an analysis of vulnerable parts in the microarchitecture of the devices are provided. Furthermore, we demonstrate that the pulse paramters can be tuned to skip until 6 consecutive instructions fetched from the embedded Flash memory of a microcontroller. High-precision bit set faults are also injected with a large laser spot in words loaded from the embedded NOR Flash memory with a high repeatability. The fault mechanism described in this work allow an attacker to inject a fault on a single bit or multiple bits depending on the size of the laser spot, and on one or several words depending on the duration of the laser pulse. An analysis of the effects of a laser induced photocurrent in the columns of a NOR Flash memory provides an understanding of the charactistics of this fault model. Finally, the threats that these fault models represent are assessed in several practical case studies which involve security functions. A theoritical analysis of the fault attack paths in the secure-boot implementation of a secure microcontroller for IoT applications highlight the weaknesses of the read interface of a non-volatile memories during the secure-boot process.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *2021LYSEM032*

Alexandre MENU

SÉCURITÉ MATÉRIELLE DES OBJETS CONNECTÉS

Spécialité: Microélectronique

Mots clefs : sécurité matérielle, internet des objets, microcontrôleur, LFI, EMFI, mémoire NOR Flash.

Résumé :

La sécurité des objets connectés dans l'internet des objets repose sur l'identification et l'évaluation des menaces qui pèsent sur ces circuits. Ces travaux proposent une analyse des vulnérabilités matérielles de plusieurs microcontrôleurs non-sécurisés face aux techniques d'injection de fautes par impulsions électromagnétiques (EM) et laser. Ces techniques permettent à un attaquant de perturber le fonctionnement d'un circuit pour contourner une protection ou extraire des informations confidentielles. Nous proposons d'évaluer le degré de maîtrise d'un attaquant sur des circuits qui ne sont pas protégés contre les attaques en faute. Les résultats obtenus sur quatre microcontrôleurs non-sécurisés fabriqués dans les technologies 0,35 µm, 0,25 µm, 80 nm et 65 nm mettent en évidence plusieurs modèles de fautes originaux sur les interfaces des mémoires NOR Flash de ces circuits. Nous montrons notamment qu'une impulsion EM permet de forcer localement les bits des mots sur leur trajet entre la mémoire NOR Flash et l'unité de calcul d'un circuit, et que la localité des fautes peut être maîtrisée en ajustant les paramètres d'une injection de fautes EM de manière à obtenir des corruptions ciblées ou des sauts d'instructions. Nous montrons également que l'injection d'un photocourant par illumination laser dans les colonnes d'une mémoire NOR Flash permet de forcer la valeur d'un ou plusieurs bits choisis dans les mots lus en mémoire avec une grande précision spatiale et temporelle. Une analyse théorique du mécanisme d'injection de fautes sur les architectures de mémoires NOR Flash permet de retrouver toutes les caractéristiques du modèle de faute étudié. Enfin, les vulnérabilités introduites par ces modèles de fautes sont évaluées à travers plusieurs cas d'étude réalisés en pratique. Une analyse théorique des vulnérabilités introduites par ces modèles de fautes dans un mécanisme d'amorçage sécurisé met en lumière le rôle clé de l'interface de lecture des mémoires non-volatiles dans la mise en œuvre d'une chaîne de confiance.