وزارة التـعليـم العـالــي والبـحـث العلمــي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
جـامعــة فرحــات عبــاس – سطيـف -1-
Université Ferhat Abbas – Sétif -1-
Faculté de Technologie
Département d'Électronique

THESE

Pour l'obtention du diplôme de

Doctorat en sciences

Spécialité : **Electronique**

Présentée par

Ibrahim MEZZAH

Thème

Diagnostic dans les systèmes embarqués

Thèse soutenue le 08/03/2018 devant le jury composé de :

M. BOULOUFA Abdeslam	Pr	Université de Sétif	Président
M. CHEMALI Hamimi	Pr	Université de Sétif	Rapporteur
M. TALBI Mohamed Lamine	MCA	Université de BBA	Examinateur
M. BOUROUBA Nacerdine	Pr	Université de Sétif	Examinateur
M. KHOUAS Abdelhakim	Pr	Université de Boumerdes	Examinateur
M. AIDEL Salih	MCA	Université de BBA	Examinateur

Remerciements

Mes remerciements les plus sincères vont tout d'abord à M. Hamimi Chemali qui a encadré ce travail et qui a consacré beaucoup de temps et d'efforts pour m'apprendre le métier de conception des systèmes électroniques. Je tiens aussi à lui exprimer mes sentiments de gratitude pour le soutien qu'il m'a apporté et la bienveillance qu'il n'a jamais cessée de manifester à mon égard.

Les travaux de cette thèse ont été réalisés au sein de l'équipe CSE (Conception des Systèmes Embarqués) du CDTA (Centre de Développement des Technologies Avancées) à Alger. Je voudrais remercier M. Omar Kermia, Responsable de l'équipe CSE, de m'avoir intégré dans son équipe et pour sa disponibilité, son aide et ses précieux conseils.

Je tiens à remercier M. Abdeslam Bouloufa, Professeur au département d'électronique de Sétif, pour avoir accepté de présider le jury.

Mes remerciements sont aussi adressés à M. Nacerdine Bourouba, Professeur au département d'électronique de Sétif, et M. Abdelhakim Khouas, Professeur à l'Université de Boumerdès, pour avoir accepté d'examiner ce travail.

Je tiens également à remercier M. Mohamed Lamine Talbi et M. Salih Aidel, Maîtres de conférences à l'université de Bordj Bou Arreridj, pour avoir accepté d'examiner ce travail.

Je remercie beaucoup ma sœur Samia pour la considérable aide qu'elle m'a apportée dans ce travail et durant toutes mes études, je la remercie aussi de m'avoir encouragé et soutenu.

Je remercie particulièrement M. Omar Abdelmalek pour son aide et sa contribution dans une partie du travail réalisé.

Mes remerciements sont aussi adressés à M. Vincent Beroulle et M. David Hely pour m'avoir accueilli durant le stage que j'ai effectué au sein du laboratoire LCIS – INP Grenoble.

Merci à tous les membres de l'équipe CSE et les collègues du CDTA Ramdane, Zakarya, Riadh, Fayçal, Oussama et beaucoup d'autres collègues pour leur soutien et aide.

J'adresse mes remerciements à mes amis qui m'ont soutenu et encouragé régulièrement, Tarek, Bilel, Abdelhak et Riad.

Je remercie profondément mes parents, et toute ma famille pour son considérable soutien tout au long de ces interminables années d'étude.

Je remercie infiniment ma femme pour son soutien, encouragement et sa patience. Merci à mes deux petites filles simplement pour leur présence.

Je remercie finalement tous ceux qui m'ont soutenu dans ce travail de près ou de loin et ils sont très nombreux. Merci à tous.

Table des matières

Liste des figures

Liste des tableaux

Abréviations & Acronymes

ASIC	Application Specific Integrated Circuit
CI	Circuit Intégré
CPU	Central Processing Unit
DSP	Digital Signal Processor
EPC	Electronic Product Code
FDX	Full Duplex
FF	Flip-Flop
FM0	Bi-Phase Space Coding
FPGA	Field Programmable Gate Array
HDL	Hardware Design Language
IoT	Internet of Things
IP	Intellectual Property
ITF	Interrogator Talks First
JTAG	Joint Test Action Group
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look Up Table
MBU	Multiple Bit Upsets
MCIP	Micro-Controller IP ou Mezzah-Chemali IP
MCIP−C	MCIP Configurable
MCU	Microcontroller Unit
MSB	Most Significant Bit
OVL	Open Verification Library
PIE	Pulse Interval Encoding
PRNG	Pseudo-Random Number Generator
PSL	Property Specification Language
R→T	Reader-to-Tag
RAM	Random Access Memory
RF	Radio Frequency
RFID	Radio Frequency Identificaion
RISC	Reduced Instruction Set Computer
RN16	16-bit random ou pseudo-random number
RTcal	Reader-to-Tag calibration symbol
RTL	Register Transfer Level
RTOS	Real-Time Operating System
SE	Système Embarqué
SET	Single Event Transient
SEU	Single Event Upset
SoC	System on Chip
SVA	SystemVerilog Assertions
T→R	Tag-to-Reader
TRcal	Tag-to-Reader calibration symbol
UCE	Unité de Contrôle d'Emulation
UHF	Ultra High Frequency
VHDL	Very high speed integrated circuits Hardware Description Language
VLSI	Very-Large-Scale Integration
WSN	Wireless Sensor Network

Chapitre 1 Introduction et motivations

Les progrès technologiques dans le domaine de fabrication des circuits intégrés (CI) ont permis de réaliser des systèmes électroniques, désormais de plus en plus complexes, mais ayant plus de performances et de fonctionnalités. La quasi-totalité de ces systèmes se base systématiquement sur l'utilisation d'une ou plusieurs unités de traitement (*Processing Units*) avec du logiciel incorporé. Cette construction, qui commence à être considérée comme primitive aujourd'hui, avait partiellement réussi à satisfaire une demande grandissante tout en procurant confort et souplesse d'exploitation des différents outils et équipements modernes.

En effet, le déploiement des systèmes embarqués (*Embedded Systems*) connaît actuellement une croissance intense surtout avec l'apparition des objets connectés (*Internet of Things* − IoT). Ces nouveaux systèmes sont devenus les briques qui composent les différentes parties de notre environnement moderne. Ainsi, les systèmes embarqués sont partie prenante de la très grande majorité des équipements, des outils, produits et réseaux actuels (transport, télécommunication, santé, sécurité, contrôle de processus, produits grand public… etc.).

Un système embarqué (SE) est caractérisé par une étroite cohabitation entre le matériel et le logiciel : un ordinateur spécialisé (mono-application) devient une simple carte électronique ou un circuit intégré incorporé dans une entité, plus ou moins complexe, dont il devient un élément parmi d'autres. Les SE sont alors très utilisés dans les dispositifs et appareils. Un produit moderne peut contenir un ou plusieurs SE à la fois.

Les SE sont conçus autour d'une ou plusieurs unités de traitement sous forme de processeurs, microcontrôleurs, DSP (Digital Signal Processors), FPGA (Field-Programmable Gate Arrays) ou ASIC (Application-Specific Integrated Circuits). Les

logiciels des SE, appelés souvent '*firmwares*', sont des éléments fondamentaux et leurs complexités varient d'un système à un autre : certains *firmwares* sont composés d'un programme de petite taille, et d'autres *firmwares* plus complexes peuvent en contenir des dizaines de millier de lignes de code et suivent des règles de codage particulières. Une grande partie des *firmwares* utilise des systèmes d'exploitation temps réel (*Real-Time Operating System* − RTOS) pour répondre aux exigences des contraintes temporelles.

Dans la majorité des cas, un SE doit satisfaire plusieurs types de contraintes à savoir les contraintes du temps de réponse, sûreté de fonctionnement, fiabilité, sécurité, consommation d'énergie, coût… etc. Les SE peuvent être classés sous différentes catégories selon plusieurs critères tels que les performances, les domaines d'application et les compositions logicielles-matérielles.

D'un point de vue d'effet sur la vie et la sécurité des personnes, on distingue principalement deux catégories de SE : les **SE critiques** et les **SE non-critiques**. Les SE non-critiques sont généralement des SE grand public tels que les téléphones cellulaires, tablettes, lecteurs MP3, caméras et téléviseurs. Si ces types de SE plantent, aucun dommage sérieux ne survient. De nombreux modules du domaine de contrôle industriel sont aussi des SE non-critiques, de même pour la plupart des réseaux de télécommunication et pour certains SE utilisés dans le transport. Les SE critiques, par contre, sont des systèmes ayant une incidence critique sur la vie humaine ou sur la sécurité en cas de défaillance ou de dysfonctionnement : − mort ou blessure grave de personnes, − perte ou endommagement sévères d'un équipement coûteux, − dommage environnemental ou − pertes financières importantes et non recouvrables. Le développement de ces systèmes doit offrir donc des garanties de bon fonctionnement et de bon comportement en cas de défaillance d'une partie interne ou en situation non prévue.

Le développement d'un SE fiable représente souvent un défi majeur aux développeurs. Malgré les innombrables techniques et outils proposés dans le domaine du développement et du vérification de ces systèmes, la réalisation d'un SE dépourvu de bugs que ce soit hardware ou *firmware* reste un objectif difficile à atteindre, particulièrement pour les systèmes complexes. Afin de faire face à ces défis de développement, les fabricants proposent régulièrement des mises à jour de *firmware* pour leurs produits afin d'y remédier aux bugs *firmware* ou hardware, détectés après la commercialisation de leurs produits. La détection de bugs, après la phase de fabrication, reste une tâche essentielle afin de traiter les bugs rapidement et particulièrement prévenir toute conséquence redoutable. La dotation

des SE de mécanismes de détection de bugs en ligne constitue alors une solution louable pour la détection des bugs après la phase de fabrication.

Par ailleurs, les dispositifs électroniques sont sensibles également aux perturbations externes comme la variation de l'alimentation, la variation de la température, les interférences électromagnétiques et les rayonnements ionisants. Ces perturbations peuvent induire des fautes à l'intérieur de ces systèmes et provoquer par la suite des erreurs et des défaillances. La meilleure méthode, pour contourner l'effet de ces influences, est la conception de systèmes robustes qui peuvent tolérer les fautes générées par ces perturbations surtout quand il s'agit de système soumis à des conditions de travail draconiennes. Un autre phénomène générant des fautes est le vieillissement de composants électroniques. Le vieillissement des CI modernes qui ont des capacités d'intégration élevées est plus important que celui des CI de capacité d'intégration faible [1]. De ce fait, le développement de techniques de détection du vieillissement des CI est fortement recommandé pour améliorer la conception de nouveaux systèmes fiables.

Une autre source de perturbation susceptible de provoquer des erreurs au niveau des SE est représentée par les attaques malveillantes qui peuvent être de différents types. Les SE communicants sont les plus sensibles à ces attaques notamment ceux dotés de moyens de communication sans fil. La menace de ces attaques est devenue sérieuse plus que jamais avec l'accroissement spectaculaire du nombre des objets connectés. Malgré la disposition de nombreuses méthodes d'authentification, de cryptage et de protection pour sécuriser les SE contre les attaques, la nécessité de nouvelles solutions qui renforcent cette sécurité demeure inévitable.

La détection de fautes, d'erreurs ou même de pannes dans les SE constitue un défi que ce soit à la phase de conception, pendant le débogage ou durant leur fonctionnement normal. Le diagnostic constitue, lui aussi, une tâche largement plus délicate qui nécessite une connaissance approfondie de ces erreurs et une identification de leurs causes. Avec la multiplicité des causes sus-citées, la grande majorité des fautes et des erreurs qui se produisent dans les systèmes ne sont pas détectées ni diagnostiquées. Les erreurs qui font réellement l'objet d'un diagnostic sont en général seulement celles qui produisent des pannes apparentes et permanentes.

Les différents aspects traités dans les paragraphes précédents nous permettent de prédire que les futurs SE seront dotés de formes d'intelligence, basiques ou avancées, qui

leurs permettront d'interpréter l'évolution de l'environnement et diagnostiquer les occurrences de fautes. Les travaux et les efforts déployés dans cette thèse rentrent dans cette optique.

1.1 Objectifs : instrumentations sur puce pour la détection de fautes, diagnostic et contrôle de la fiabilité

L'instrumentation sur puce (*on chip instrumentation*) est de plus en plus utilisée dans les systèmes modernes pour apporter des solutions à la complexité toujours croissante et, particulièrement, favoriser une flexibilité de développement et de vérification de systèmes. Ce type d'instrumentation est extensivement utilisé dans les domaines de test et de débogage des CI, mais son application dans d'autres activités inhérentes aux processus des CI demeure limitée.

Les objectifs de cette thèse se focalisent essentiellement sur le développement, pour les systèmes sur puce (*System on Chip* − SoC), des instrumentations sur puce qui permettent d'effectuer des tâches de détection en ligne des fautes et des erreurs et de réaliser des tâches de diagnostic assurant ainsi un contrôle permanent de la fiabilité. Cette instrumentation a la forme d'une infrastructure-IP (Intellectual Property) qui peut être incorporée à un SoC quelconque pour surveiller son fonctionnement et particulièrement indiquer si le système préserve ses caractéristiques et maintient sa fiabilité [2].

La Figure 1-1 donne un exemple d'implémentation de cette approche où l'I-IP est ajoutée au SoC comme un périphérique chargé de surveiller en permanence le fonctionnement du système, sans autant l'altérer ni l'influencer. La surveillance consiste à récupérer les données qui circulent dans le SoC et faire des analyses qui permettent de détecter les fautes et les erreurs qui se produisent afin d'évaluer les comportements des différents blocs constitutifs et le comportement global du système.

Comme illustré par la Figure 1-1, l'I-IP peut exploiter les ressources des circuits de test et de débogage, déjà existantes dans le SoC et qui ne sont pas utilisées durant son fonctionnement normal, pour récupérer les données à analyser. Une mémoire non volatile est associée à l'I-IP et sert à stocker les données nécessaires aux fonctions de test et de débogage. Ces données contiennent un historique des anomalies détectées et des diagnostics réalisés ainsi qu'une évaluation de la fiabilité. Ces données accessibles au CPU (Central Processing Unit) à travers l'I-IP, moyennant un programme, permettent l'analyse des différents aspects de fonctionnement du système. L'interaction l'I-IP / CPU s'effectue

à travers un mécanisme basé sur les interruptions. Ces interruptions sont multiples et sont générées pour tester la réponse du système et le prévenir des erreurs et des perturbations externes détectées. L'I-IP peut utiliser des capteurs intégrés sur la même puce (Figure 1-1) pour détecter toute perturbation externe susceptible d'entraîner des fautes à l'intérieur du système.

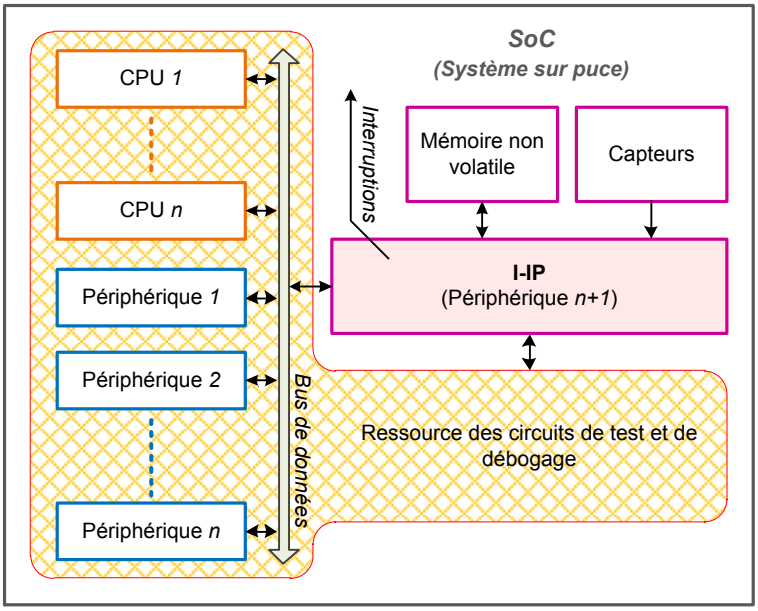


Figure 1-1 L'approche du contrôle de la fiabilité des SoC.

Pour répondre aux besoins des systèmes équipés d'un RTOS, l'I-IP peut intégrer des fonctionnalités de vérification du flot d'exécution du RTOS. Dans ce cas, le CPU procède au paramétrage de l'I-IP au cours du lancement du RTOS et à chaque création ou suppression d'une tâche avec son échéance d'exécution ainsi que les adresses des différentes variables qui pilotent le RTOS (*Semaphores*, *Message queue*… etc.). Une fois, paramétrée et activée, l'I-IP procède à la vérification du flot d'exécution du RTOS, contrôle l'exécution des différentes tâches et effectue la vérification des échéances d'exécution non dépassées. Muni de ces fonctionnalités, l'I-IP joue le rôle d'un '*Watchdog*' mais pour un RTOS.

L'autre fonctionnalité de l'I-IP est le diagnostic afin d'identifier les anomalies survenues et d'en déterminer leurs causes : bugs, perturbations externes, vieillissement, attaques… etc. Le système peut alors exécuter des actions qui permettent de corriger ou d'atténuer les effets de ces anomalies ou même de bloquer les attaques en cas d'occurrence. Le système peut aussi, au fur et à mesure, informer d'autres systèmes

externes sur son état de fonctionnement, ce qui permet de procéder et recourir aux tâches nécessaires en cas de mauvais fonctionnement tel qu'effectuer un diagnostic avancé, une correction de bugs, une amélioration de conditions de fonctionnement, un déclenchement de systèmes redondants et un remplacement rapide du système défaillant… etc. La généralisation de l'utilisation de l'approche proposée devrait non seulement augmenter la fiabilité et la sécurité des systèmes mais contribuerait qualitativement à l'amélioration des futurs circuits et designs.

1.2 Application à la RFID

La radio-identification (*Radio Frequency Identification* − RFID) constitue une technologie très utile vu son déploiement intensif dans de nombreux domaines d'application. La RFID est exploitée ainsi dans diverses applications telles que la gestion de stock, contrôle d'accès, passeport biométrique, cartes bancaires sans contact, localisation géométrique, réseaux de capteurs et aussi bien dans plusieurs d'autres applications. L'aspect de communication sans fil employé par la RFID ainsi que son déploiement étendu notamment dans des applications connues sensibles, rend l'utilisation de cette technologie confrontée à plusieurs défis concernant particulièrement la confidentialité et la sécurité de l'information.

Un système RFID est composé d'une étiquette électronique (appelé aussi tag, en anglais) dans laquelle est incorporé un code d'identification généralement unique. Le fait que les tags soient des dispositifs électroniques, cela implique qu'ils sont sujets à des fautes et à des erreurs causées par des défauts internes ou des perturbations externes. Un tag défectueux entraîne des défaillances de détection et d'identification, ce qui altère d'une manière significative la disponibilité, la fiabilité et la sécurité de tout le système RFID [3].

D'autre part, il existe des menaces sérieuses pour les systèmes RFID sujets à des attaques malveillantes de plusieurs types qui ne peuvent pas être maitrisées de coutume. En conséquence, ces attaques peuvent entraîner des suites dangereuses pour la vie des gens. Les attaques pour lesquelles les tags RFID sont très vulnérables sont les attaques par injection de fautes à travers plusieurs moyens tels que la variation de l'énergie, les interférences électromagnétiques et l'induction optique. Ce type d'attaques impose au tag des comportements fautifs comme le saut d'une tâche de cryptage ou le basculement dans un état sécurisé afin de délivrer des données sécurisées [4] [5]. Bien que des efforts importants aient été déployés pour améliorer la fiabilité et la sécurité des systèmes RFID notamment contre les attaques, d'autres recherches sont encore nécessaires pour répondre à la diversité croissante des fautes et effets indésirables.

Étant donné que les tags RFID sont des systèmes embarqués communicants (pour ceux qui contiennent un logiciel embarqué), ils sont considérés dans les travaux de cette thèse comme un support d'application pour les approches développées. Les problématiques multiples liées à la RFID ainsi que le peu de travaux de recherche recensés ont motivé ce choix. Le but à travers les recherches que nous avons menées est donc d'apporter de nouvelles solutions destinées pour la RFID et pour les objets communicants d'une façon générale.

1.3 Méthodologie

Le développement et l'application d'une approche quelconque nécessitent généralement plusieurs moyens et outils. Pour élaborer les approches proposées, nous avons utilisé plusieurs outils disponibles et nous avons développé plusieurs d'autres outils afin d'atteindre les finalités ciblées. La sélection de la RFID a exigé le développement d'un circuit de tag RFID en RTL (Register Transfer Level) pour pouvoir le traiter à ce niveau et en conséquence appliquer les solutions à développer. La non-disponibilité d'un circuit pareil, du moins dans la communauté '*open source*', et le besoin d'une maîtrise totale de son architecture nous a poussé à développer notre propre circuit ainsi que toute une plateforme RFID.

La méthodologie suivie dans la réalisation des travaux de cette thèse commence donc par le développement d'une plateforme RFID qui contient deux architectures différentes de tag RFID et un lecteur RFID. L'une des deux architectures de tag et celle du lecteur sont développés autour d'un microcontrôleur sous forme RTL déjà existant [6], mais son architecture a été restructurée pour le rendre configurable et plus flexible à supporter ce genre d'implémentation.

Le développement des solutions de diagnostic a nécessité au préalable la connaissance (l'étude) des effets des fautes potentielles sur le système ciblé. À cet effet, une plateforme d'injection de fautes a été développée et utilisée pour réaliser des analyses sur les effets des fautes sur les tags RFID en employant des circuits FPGA. Enfin, des solutions de détection en ligne de fautes ont été développées et évaluées à l'aide de la plateforme développée.

1.4 Contributions

Dans le souci de faciliter la lecture de cette thèse où plusieurs abréviations et acronymes sont utilisés et une répartition d'information étalée sur plusieurs chapitres, un listing des travaux développés est incorporé dans cette introduction. Cette liste comprend essentiellement les points suivants :

- Conception en langage VHDL (Very high speed integrated circuits Hardware Description Language) d'un microcontrôleur 8-bit configurable qui est complètement compatible avec la famille des microcontrôleurs PIC18 de la compagnie Microchip. Ce microcontrôleur, dénommé MCIP-C, permet de réaliser de manière efficace plusieurs types d'implémentations.
- Conception en VHDL d'un tag RFID à base de machine à états finis dénommée G2FSM. Ce tag, compatible avec la norme EPC UHF Gen2, se caractérise par une architecture compacte (faible encombrement) et faible consommation.
- Conception en VHDL d'un deuxième tag conformément aussi à la norme EPC Gen2 mais cette fois-ci à base du microcontrôleur MCIP-C. Ce tag, dénommé G2MC est plus flexible et permet d'ajouter aisément d'autres fonctionnalités optionnelles ou personnalisées conformément au protocole EPC Gen2.
- Conception en VHDL d'un lecteur RFID EPC Gen2 à base du MCIP-C. Le lecteur conçu peut être implémenté sur carte FPGA et permet de faire des opérations de lecture et d'écriture sur les tags UHF. Il est possible aussi de programmer ce lecteur pour réaliser des scénarios de communication d'une manière standard ou personnalisée avec des tags RFID.
- Développement d'une plateforme d'émulation de fautes de types SET (Single Event Transient), SEU (Single Event Upset) et MBU (Multiple Bit Upsets). Cette plateforme utilise un FPGA et permet de réaliser des injections de fautes temporelles ou spatiales d'une manière sélective, exhaustive ou aléatoire (dans le temps et/ou dans l'espace).
- Développement d'une nouvelle technique d'analyse de l'impact des fautes de types SET et SEU sur les tags RFID. Cette technique s'articule sur l'utilisation d'une carte FPGA pour réaliser des émulations de fautes exercées sur une paire de lecteur-tag, et comparer les résultats obtenus avec les résultats d'une autre paire non soumises exemptes de fautes, implémentée sur la même FPGA.

- Réalisation d'une analyse particulière sur l'impact des fautes sur les tags RFID. Cette analyse a permis de démystifier les effets des fautes sur les tags à travers une minutieuse classification de ces effets. Elle a permis également d'identifier les éléments les plus sensibles aux fautes dans le tag particulièrement sur l'aspect de sécurité, ainsi qu'une sérieuse évaluation de la robustesse du tag. Les résultats obtenus fournissent des données importantes qui aident significativement lors du diagnostic des erreurs qui se produisent dans les systèmes RFID.
- Développement et application d'une technique de détection en ligne de fautes dans les tags RFID. Dans ces techniques nous avons employé des assertions matérielles consacrées exclusivement à la détection des fautes et des erreurs. À l'aide de cette instrumentation sur puce, le tag devient capable de surveiller son fonctionnement d'une façon qui lui permet de contrôler sa fiabilité et d'améliorer sa sécurité contre les attaques.

1.5 Organisation du manuscrit

La suite du document est structurée en cinq chapitres :

Des généralités et l'état de l'art autour des principaux domaines traités dans cette thèse englobant le diagnostic des fautes dans les systèmes embarqués, les mécanismes d'instrumentation sur puce et la technologie RFID avec ses différents aspects de fiabilité et de sécurité sont détaillés dans le second chapitre.

Le troisième chapitre décrit les différents systèmes d'expérimentation développés dans le but d'appliquer les approches proposées.

Le quatrième et le cinquième chapitre constituent le cœur du document puisqu'ils présentent les principales contributions de la thèse. Dans le quatrième chapitre nous donnons une analyse d'injection de fautes sur les tags RFID pour l'évaluation de leur sécurité et pour le diagnostic. Enfin, le cinquième chapitre introduit la technique de détection en ligne de fautes appliquée aux tags RFID.

Enfin, une conclusion générale termine ce manuscrit, situe la valeur du travail accompli et présente les perspectives pour en bénéficier des résultats obtenus et certainement explorer d'autres réajustements et approches alternatives.

Chapitre 2 Généralités et l'état de l'art

Dans ce chapitre, nous présentons des généralités et l'état de l'art sur trois domaines technologiques divers auxquels sont liés les travaux de cette thèse : le diagnostic des fautes dans les systèmes embarqués, l'instrumentation sur puce et l'identification par radiofréquence.

2.1 Diagnostic des fautes

Lorsqu'un mauvais fonctionnement survient dans un système, le problème principal à soulever est la détection rapide suivie d'un diagnostic efficace concernant le dysfonctionnement. Il s'agit donc de connaitre le type de faute introduite et définir les tâches à exécuter rapidement pour gérer cette occurrence. Ce genre de défi se pose de manière continue et répétitive pendant le développement et l'utilisation des différents systèmes, ce qui rend le diagnostic très important et essentiel pour produire les solutions appropriées.

L'opération du diagnostic accompagne les différentes phases de mise en œuvre des dispositifs en commençant par la phase de développement jusqu'à leurs utilisations (Figure 2-1). Pendant la phase de fabrication, le diagnostic joue un rôle capital dans l'identification des défauts liés à cette phase, ce qui permet de réaliser des améliorations substantielles au processus de fabrication. Durant la phase de fonctionnement, le diagnostic permet d'identifier les fautes intermittentes permettant ainsi d'exécuter des corrections et garder les dispositifs en état de fonctionnement permanent. Le diagnostic permet également de reconnaître d'autres fautes liées aux imperfections de conception et de fabrication ; ce qui permet d'apporter des améliorations aux prochaines versions des dispositifs, d'une part, et de réaliser des corrections aux dispositifs déjà en fonctionnement notamment des mises à jour du logiciel embarqué, d'autre part. Les méthodes, les outils et

les buts du diagnostic diffèrent alors d'une phase à une autre comme ils diffèrent d'un domaine d'application à un autre [7].

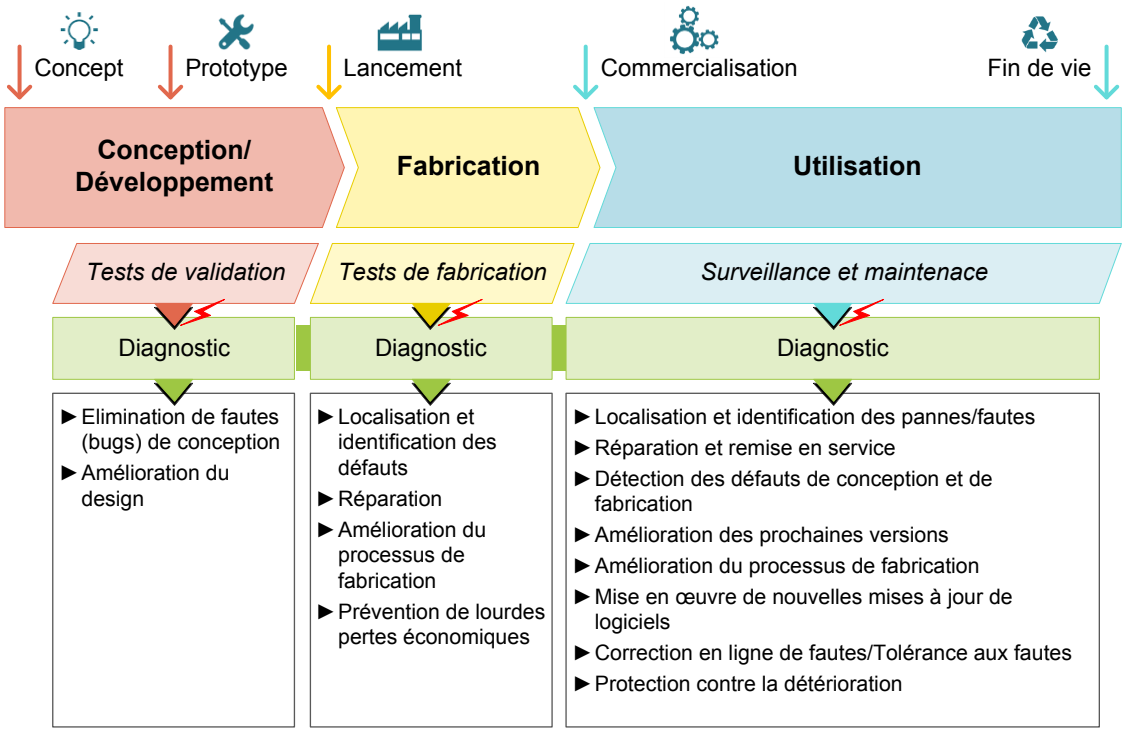


Figure 2-1 Cycle de vie d'un dispositif couvert par le diagnostic.

2.1.1 Terminologie

Étant donné que le champ d'application du diagnostic des fautes est réparti sur plusieurs domaines technologiques différents, la terminologie utilisée n'est pas unique. Les définitions qui suivent sont suggérées par le comité technique IFAC SAFEPROCESS afin d'unifier la terminologie [8] [9].

Faute (*Fault*) : une déviation d'au moins une propriété caractéristique ou une variable du système de son comportement acceptable/habituel/standard.

Défaillance (*Failure*) : une interruption permanente de la capacité d'un système à effectuer une fonction requise dans des conditions de fonctionnement spécifiées.

Dysfonctionnement (*Malfunction*) : une irrégularité intermittente dans l'accomplissement de la fonction désirée d'un système.

Détection de faute (*Fault detection*) : détermination des fautes présentes dans le système et du temps de détection.

Isolation de faute (*Fault isolation*) : détermination du type, de l'emplacement et du temps

de détection d'une faute. Elle suit la détection de fautes.

Identification de faute (*Fault identification*) : détermination de la taille et la variation du temps du comportement d'une faute. Elle suit l'isolement de fautes.

Diagnostic de faute (*Fault diagnosis*) : détermination du type, de la taille, de l'emplacement et du temps de détection d'une faute. Il suit la détection de fautes. Il comprend à la fois l'isolation et l'identification de fautes.

Dans la littérature, il subsiste également d'autres définitions du diagnostic de fautes dont la plus populaire, que nous adoptons d'ailleurs, considère la détection des fautes une partie intégrante du diagnostic [10].

Dans les **systèmes numériques**, un défaut de circuit peut conduire à une faute, une faute peut provoquer une erreur de circuit et une erreur peut provoquer une défaillance du système [11].

Un **défaut (*Defect*)** dans un système électronique est la différence involontaire entre le matériel implémenté et sa conception prévue [12].

Une **erreur (*Error*)** est une instanciation d'une opération incorrecte du système [13].

L'origine des erreurs est multiple : erreurs de conception, erreurs de fabrication, défauts de fabrication ou défaillances physiques. Les **défaillances physiques (*Physical failures*)** se produisent pendant la durée de vie d'un système en raison de l'usure des composants et/ou des facteurs environnementaux. Par exemple, les connecteurs en aluminium à l'intérieur d'un CI s'allongent avec le temps et peuvent se briser en raison de la migration des électrons ou de la corrosion. Les facteurs environnementaux, tels que la température, l'humidité et les vibrations, accélèrent le vieillissement des composants. Le rayonnement cosmique et les particules α peuvent provoquer aussi des fautes à l'intérieur des éléments de mémoire d'un CI [13].

Les erreurs de fabrication, les défauts de fabrication et les défaillances physiques sont collectivement appelés **fautes physiques (*Physical faults*)**. Selon leur stabilité dans le temps, les fautes physiques peuvent être classées comme suit [13] :

- **Permanentes :** présentes en permanence après leurs apparitions ;
- **Intermittentes :** existantes uniquement pendant certains intervalles ;
- **Transitoires :** occurrences uniques causées par des changements temporaires de certains facteurs environnementaux.

En général, les fautes physiques ne permettent pas un traitement mathématique direct des tests et des diagnostics. La solution est de traiter les **fautes logiques (*Logical faults*)** qui forment une représentation convenable de l'effet des fautes physiques sur le fonctionnement du système. Une faute est détectée en observant l'erreur correspondante provoquée [13].

2.1.2 Les approches de diagnostic

Cette partie dresse un état de l'art des méthodes de diagnostic dans le domaine des systèmes embarqués. Les approches proposées dans la littérature sont définies comme étant des approches de diagnostic embarqué, mais peu d'entre elles ont réellement été mises en place sur ce type de système [14].

Les approches de diagnostic proposées essayent d'être de plus en plus innovantes et performantes, car les méthodes traditionnelles telles que les AMDEC (Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité), les arbres de diagnostic… etc. ne sont plus suffisantes. Les approches de diagnostic diffèrent assez largement selon les connaissances mises en jeu. Trois grandes classes se distinguent : l'approche à base de connaissances, l'approche à base de modèles, et l'approche à base de données [15] (Figure 2-2). Cette classification ne prétend pas être exhaustive, mais permet de classer la majorité des travaux sur le diagnostic.

2.1.2.1 Approche à base de connaissances

L'approche à base de connaissances (*Knowledge Based Approach*) est très peu exploitée pour un diagnostic embarqué et est surtout utilisée comme complément en diagnostic hors-ligne [16] [17]. Le principe d'une telle approche est d'associer les causes et les symptômes. Elle peut s'exprimer sous divers formalismes : dictionnaires de pannes, Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité (AMDEC), systèmes à base de règles, arbres de défaillances… etc.

Les AMDEC ont été employées pour la première fois à partir des années 1960 dans le domaine aéronautique pour la sécurité des avions [18]. Cette méthode permet une analyse systématique et très complète, composant par composant, de tous les modes de défaillances possibles et de leurs effets sur le système [19]. La démarche consiste à définir le système, ses fonctions et ses composants. Ensuite, l'ensemble des modes de défaillances des composants doit être établi. L'ensemble des causes susceptibles de provoquer ces

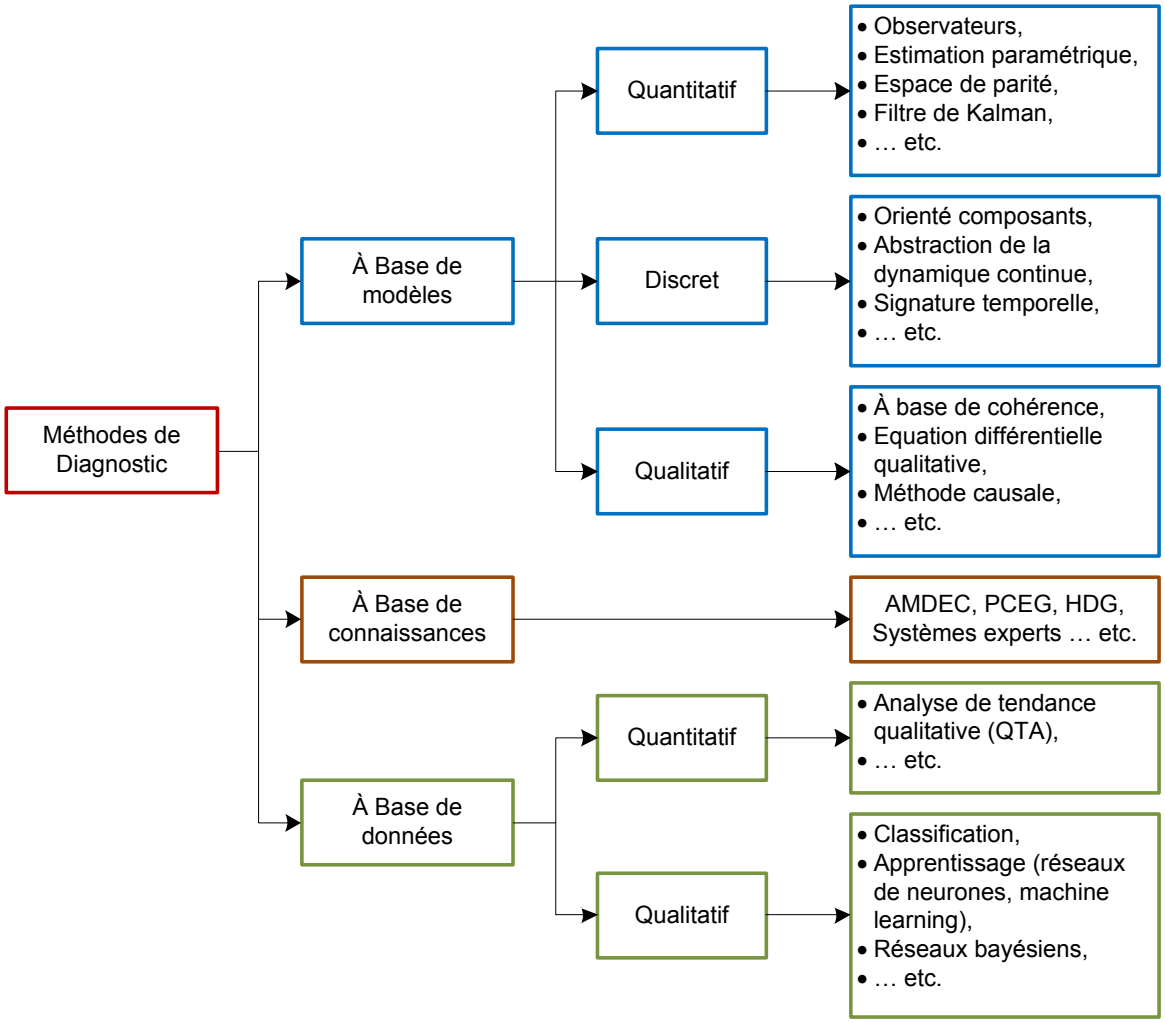


Figure 2-2 Classification des méthodes de diagnostic.

défaillances est déterminé. Finalement, les effets sur le système sont étudiés pour chaque mode de défaillance et chaque cause. Le tout est présenté sous forme d'un tableau récapitulatif. Les AMDEC sont donc utilisées de façon déductive dans un objectif de diagnostic, c'est-à-dire que l'on démarre à partir des effets observés pour remonter aux causes de défaillances possibles [20]. Cette approche est très efficace grâce à ses relations de causes à effets. Néanmoins, les limites d'une telle approche résident dans le fait que les défaillances doivent être définies à priori. Le recensement préalable de ces défaillances ne peut pas être exhaustif et nécessite une longue expérience. Le coût de construction des AMDEC est donc très élevé, car cette construction peut être longue et très difficile à mettre en œuvre. En outre, toute évolution ou modification du système implique une réécriture du tableau construit.

Un arbre de défaillances est une méthode qui permet de déterminer les chemins critiques dans un système [19]. Elle définit les différentes combinaisons possibles d'événements qui entraînent l'apparition d'un événement non désirable et unique. Cette approche est représentée par une structure arborescente (un arbre) dont le sommet est l'événement indésirable. Il est établi sous forme d'un diagramme logique. Les causes immédiates qui produisent l'événement indésirable sont hiérarchisées par des symboles ET et OU logiques. De cette façon, l'arbre est créé au fur et à mesure pour atteindre des événements dits élémentaires. Le problème de ce type d'approche est leur forte dépendance aux erreurs du concepteur. De plus, comme pour les AMDEC, les défaillances doivent être déterminées a priori et l'évolution du système nécessite une reconstruction de l'arbre de défaillance.

Dans les systèmes à base de règles, nous retrouvons les systèmes experts [21]. Ce sont des approches qui consistent à reproduire les connaissances et le raisonnement d'un expert dans l'accomplissement d'une tâche. La connaissance utilisée dans ces systèmes est représentée par un ensemble de règles qui s'enchaînent pour simuler le raisonnement de l'expert. Ces règles décrivent les relations fonctionnelles entre les différents composants et les relations causales entre les défaillances et leurs effets. Un moteur d'inférence est ensuite utilisé pour établir un diagnostic à partir des connaissances contenues dans la base de connaissances. Nous retrouvons deux méthodes de raisonnement. Le chaînage avant qui, à partir d'une déviation, permet d'activer certaines règles dont les conséquences deviennent de nouveaux faits qui activent de nouvelles règles. Le processus s'arrête lorsqu'il n'y a plus de règles à activer. Le chaînage arrière quant à lui permet, à partir d'un fait, de rechercher toutes les règles dont il est la conclusion. Les antécédents de ces règles deviennent des faits à partir desquels le raisonnement se poursuit. Dans le domaine automobile, ce système a été largement utilisé dans le but d'un diagnostic plus systématique et plus précis, car en plus de localiser les composants défaillants et leurs causes, le système propose des actions pertinentes pour corriger les défaillances [22].

La difficulté des systèmes experts est de pouvoir formaliser sous forme de règles toutes les propriétés d'un système et par conséquent de l'expérience que nous pouvons en avoir [23]. De plus, comme pour les autres approches, les systèmes experts sont basés sur la connaissance des défaillances du système. Ils restent également très dépendants du système et par conséquent nécessitent la réécriture des règles et la réacquisition de la connaissance des dysfonctionnements du système [24].

Les méthodes à base de connaissances, ne font pas l'objet d'étude sur les systèmes embarqués. Leur forte dépendance aux systèmes, qui lorsqu'il s'agit de systèmes embarqués peuvent évoluer régulièrement, rend difficile leur utilisation. Nous pouvons remarquer que ces approches nécessitent une connaissance préalable de l'ensemble des défaillances qui peuvent survenir sur le système, ce qui les rend très efficaces. Néanmoins, ces approches sont principalement utilisées pour un diagnostic hors-ligne, après détection d'un symptôme, afin de localiser les composants défaillants et la nature de leur défaillance. Elles sont peu adaptées pour la détection et la localisation en ligne des défaillances.

2.1.2.2 Approche à base de modèles

Les approches à base de modèles (*Model-Based Diagnosis*) dans un objectif de diagnostic embarqué sont très largement répandues dans la littérature scientifique. L'approche à base de modèles exploite un modèle du fonctionnement explicite du système à diagnostiquer. Elle fonde la détection des défaillances sur la constatation de discordances entre le comportement prédit par le modèle et le comportement réellement observé, et la localisation sur le recoupement des groupes de composants impliqués dans chacune de ces discordances.

Selon la connaissance acquise sur le système, il est possible de définir deux approches différentes : l'approche FDI (Fault Detection and Isolation) issue de la communauté d'automatique et dont le principe repose sur des modèles quantitatifs, et l'approche DX issue de la communauté de l'intelligence artificielle (IA) et dont le principe repose sur des modèles qualitatifs.

2.1.2.2.1 L'approche DX

La méthode, dite de diagnostic à base de modèles, ou encore diagnostic à partir des principes premiers, a vu le jour au milieu des années 70 et a été formalisée au début des années 80. Un nombre croissant de travaux a été mené depuis et cette problématique est devenue un domaine de recherches à part entière de l'IA. Les articles les plus marquants jusqu'à 1991 sont regroupés dans [25] [26] [27]. Une synthèse pertinente de ce domaine est présentée dans l'ouvrage de P. Dague [28].

Les connaissances incluses dans les modèles utilisés décrivent la structure (connexions entre composants) du système à diagnostiquer et son comportement (en termes du comportement des différents composants intervenant dans le système). Seules les

connaissances structurelles sont spécifiques au système en question ; les connaissances comportementales, idéalement issues des lois de la physique, sont génériques et ne dépendent que du domaine (électronique, thermodynamique, hydraulique, mécanique… etc.).

Ce qui distingue cette approche est qu'il n'est pas nécessaire de raisonner sur des modèles de fautes, mais sur les modèles de bon comportement pour déterminer si le système décline de celui-ci. Plus besoin d'anticiper les défauts ou les dysfonctionnements pouvant affecter un système et leurs effets pour pouvoir les détecter et les localiser : modéliser le comportement correct est suffisant. L'idée fondamentale est de comparer le comportement réel du système tel qu'il peut être observé par l'intermédiaire de capteurs et son comportement attendu tel qu'il est prédit grâce aux modèles de bon comportement. Toute incohérence entre les observations et les prédictions déduites des modèles est interprétée comme la manifestation d'un dysfonctionnement, c'est-à-dire la présence d'un ou plusieurs défauts. Ceci est bien sûr conditionné par le fait que les modèles soient corrects, c'est-à-dire qu'ils représentent réellement le comportement du système.

Conflits et diagnostics : La présence de défauts est détectée par l'observation d'une incohérence entre les observations et les prédictions. Cette incohérence met en évidence qu'un composant ou plusieurs, utilisés pour faire de la prédiction, ne suivent plus le modèle de bon comportement. L'un d'entre eux est nécessairement défectueux. L'ensemble de ces composants forme un conflit. La localisation se fait en recoupant les différents conflits obtenus à partir des observations sur le système et des prédictions faites à partir des modèles.

Ensuite, l'objectif est de rétablir la cohérence en faisant des hypothèses sur le fonctionnement des composants. Un diagnostic de l'ensemble des composants qui sont supposés en fautes permet de rétablir la cohérence avec les observations. C'est pourquoi le terme de diagnostic à base de cohérence est également employé. Finalement, lorsqu'il y a plusieurs diagnostics, un raffinement des candidats pourra être utilisé.

Modèles de fautes : Dans certains cas, la connaissance de modèles de fautes permet d'enrichir le diagnostic dans la mesure où elle nous permet d'aller plus loin que la simple localisation, c'est-à-dire l'identification. Aussi, des extensions du formalisme furent proposées pour permettre d'utiliser des modèles de faute : GDE+ [29] et Sherlock [25] sont les travaux les plus marquants.

Aux deux modes de comportement correct et incorrect utilisés précédemment s'ajoutent les modèles des modes de fautes connus, considérés deux à deux exclusifs, ainsi qu'un mode inconnu sans modèle associé qui représente tous les modes de faute non anticipés.

Ainsi, un conflit représente une incohérence entre les observations et les prédictions faites à partir des assignations de modes comportementaux à certains composants. Un diagnostic est le rétablissement de la cohérence par assignation de modes comportementaux à tous les composants du système. La génération des diagnostics étant un problème NP-difficile [14], diverses techniques de localisation furent proposées [30] pour ne générer que les diagnostics préférés selon certains critères (diagnostics les plus probables [31], diagnostics explicatifs [32] [26]. Nous retrouvons ces travaux, si l'objectif est l'embarqué dans [33].

2.1.2.2.2 L'approche FDI

L'application de l'automatique dans un objectif de détection et de diagnostic est apparue dans les années 70 [34]. Plusieurs articles de synthèse sont disponibles, notamment [35], [36] et [28].

Les modèles sont construits à partir des lois fondamentales (physiques… etc.), et sont décrits par des modèles analytiques. Le principe de détection consiste en la génération d'indicateurs de défauts, appelés résidus. Les systèmes embarqués font appel à des modèles aussi bien linéaires que non linéaires.

La surveillance comprend deux phases :

- le calcul des résidus
- l'évaluation des résidus

Les résidus représentent la différence entre le comportement réel du système et celui prédit par le modèle. Dans le cas nominal, la valeur du résidu est nulle ou non nulle selon la présence de défaillances. L'expression des résidus est obtenue à priori, comme une fonction des grandeurs observables (phase de calcul des résidus). Elle est ensuite évaluée en utilisant les mesures (phase d'évaluation des résidus). Une procédure de décision, prenant en compte les caractéristiques statistiques des bruits de mesure et les erreurs de modélisation est en général nécessaire pour évaluer les résidus, c'est-à-dire décider si l'écart par rapport à zéro est significatif.

La génération des résidus à partir des modèles analytiques fait l'objet de nombreuses études. Deux grandes approches se distinguent. La première consiste à surveiller les variables utilisées dans la description du système qui peuvent évoluer au cours du temps. Il s'agit des approches basées sur l'estimation des sorties et celle de l'espace de parité. L'autre approche consiste à estimer les paramètres structuraux du système. Ces paramètres sont généralement constants. Par exemple, dans les systèmes embarqués, les techniques de filtrage de Kalman sont largement employées. Le filtre de Kalman est un filtre récursif qui estime les paramètres structuraux ou de sorties d'un système à partir de mesures incomplètes et bruitées comme c'est généralement le cas dans les systèmes embarqués [37] [38].

Une autre approche utilisant le filtre de Kalman est appelée *"interacting multiple models"*. Cette technique consiste à utiliser un banc de filtres de Kalman où chaque filtre représente un mode de fonctionnement du système. Cela permet d'estimer l'état du système sous une réduction significative du bruit. Une probabilité pour chaque modèle est calculée pour indiquer le mode courant [39] [40].

Le filtre de Kalman s'applique pour des systèmes linéaires. Or, les systèmes actuels ne sont pas linéaires. Une variante du filtre de Kalman a été développée, appelée filtre de Kalman étendu, pour les systèmes non linéaires, mais linéarisables localement. Nous retrouvons ce genre d'approche dans les systèmes aéronautiques [41]. D'autres approches et notamment le filtrage particulaire ont été mises en œuvre pour les systèmes non linéaires et non linéarisables. Elles sont basées sur des simulations et des techniques statistiques [42] [43].

Pour la localisation, différents types de résidus sont générés : les résidus structurés et les résidus directionnels. Les résidus structurés sont conçus pour que chaque résidu soit sensible à un sous-ensemble de fautes et insensible aux autres. Aussi, quand une faute survient, certains résidus répondent et d'autres restent à zéro : ceci constitue la signature de la faute. L'ensemble des signatures prédéfinies pour les différentes fautes à considérer constitue la matrice de signatures. Les résidus directionnels sont conçus de telle sorte que, lorsqu'une faute survient, le vecteur de résidus soit confiné suivant une direction particulière de l'espace des résidus.

Notons qu'en pratique, l'objectif est de structurer ou d'orienter au mieux un ensemble de résidus en minimisant/maximisant la sensibilité des résidus par rapport à divers sous-

ensembles de fautes. Par ailleurs, il est préférable de rendre les signatures de faute insensibles aux perturbations. Ainsi, cela revient à résoudre un problème d'optimisation global de manière à construire des résidus sensibles aux défaillances de manière structurée, et robustes vis-à-vis des perturbations et incertitudes du modèle.

Il se peut également que les résidus soient corrélés entre eux. Pour remédier à ce problème, la méthode du maximum de vraisemblance généralisée est usuellement employée. Il s'agit d'une technique qui, sous l'hypothèse que les variables ont une distribution connue, permet d'estimer les paramètres d'un modèle (d'une équation ou d'un système, linéaire ou non) avec des restrictions sur des paramètres (coefficients, matrices de variances ou covariances) ou non [44].

Le point fort de cette approche, qui la distingue des autres, est qu'elle peut accomplir la détection et, dans une large mesure, la localisation de fautes en se fondant uniquement sur un modèle de bon fonctionnement du système, à l'exclusion de toute connaissance préalable sur les fautes ou les modes de défaillances. Les modèles de fautes ne sont pas nécessaires (pour le diagnostic purement fondé sur la cohérence, qui permet la localisation de fautes), mais ceux qui existent peuvent être efficacement pris en compte (extension au diagnostic abductif, qui permet l'identification des fautes et l'explication des symptômes).

2.1.2.3 Approche à base de données

La complexité que l'on retrouve dans les systèmes surveillés que ce soit en aéronautique, dans le ferroviaire ou l'automobile, rend difficile l'obtention de modèles. Il a donc été développé des techniques ne faisant pas appel à la connaissance des modèles mais se basant sur des données précédemment recueillies.

L'approche à base de données procède par apprentissage numérique et classification en exploitant les données existantes, à l'exclusion de toute forme de modélisation (analytique, symbolique ou autre) et peut aussi se fonder sur des techniques d'apprentissage et de classification symboliques (apprentissage par similarité).

Ces approches permettent d'associer un ensemble de mesures (continues ou discrètes) effectuées sur le système à des états de fonctionnement connus. Elles permettent d'avoir une relation d'un espace caractéristique vers un espace de décision, de façon à minimiser le risque de mauvaise classification. Une première technique est une technique classique de discrimination basée sur les outils de la probabilité. Cette technique peut se montrer insuffisante car elle suppose une connaissance a priori de tous les états de fonctionnement

et ne prend pas en compte l'évolution du système. D'autres techniques de discrimination reposent sur l'intelligence artificielle. Ces techniques ont l'avantage de ne pas se baser sur les connaissances a priori des états de fonctionnement mais plutôt sur une phase d'apprentissage. Les techniques de reconnaissance des formes par réseaux de neurones [45] [46] et de reconnaissance des formes par la logique floue [47] [48] [49] sont celles qui sont les plus utilisées.

Les réseaux de neurones sont des outils de l'intelligence artificielle, capables d'effectuer entre autres des opérations de classification. Leur fonctionnement est inspiré par les principes de fonctionnement des neurones biologiques. Leur principal avantage par rapport aux autres outils est leur capacité d'apprentissage et de généralisation de leurs connaissances à des entrées inconnues. Le processus d'apprentissage est une phase très importante pour qu'une classification puisse se faire avec succès. Plusieurs types de réseaux de neurones et plusieurs algorithmes d'apprentissage existent dans la littérature. Une des qualités de ce type d'outil est son adéquation pour la mise au point de systèmes de surveillance modernes, capables de s'adapter à un système complexe avec reconfigurations multiples. L'expert humain joue un rôle très important dans ce type d'application. Toute la phase d'apprentissage supervisé du réseau de neurones dépend de son analyse des modes de fonctionnement du système. Chaque mode doit être caractérisé par un ensemble de données recueillies sur le système. A chaque mode est associée une expertise faite par l'expert. Cette association (ensemble de données - modes de fonctionnement) est apprise par le réseau de neurones. Après cette phase d'apprentissage, le réseau de neurones associe les classes représentant les modes de fonctionnement aux formes d'entrée caractérisées par les données du système [50] [51].

En plus de leur utilisation pour l'estimation de classes, les réseaux de neurones sont également largement utilisés pour l'estimation de paramètres. L'objectif est d'établir une estimation de paramètres afin de voir s'ils correspondent au bon fonctionnement du système [52] [53] [54].

2.2 Instrumentation sur puce

Réussir le test et la validation des SoC sophistiqués actuels nécessite des solutions sophistiquées, en particulier des outils capables de fournir des contrôles et des observations avancés dans le système embarqué. Les solutions portées jusqu'à aujourd'hui se basent toutes sur l'instrumentation sur puce (On-Chip Instrumentation) qui constitue un moyen

indispensable utilisé dans la quasi-totalité des CI actuels. L'instrumentation sur puce consiste à ajouter à un design des blocs matériels, hors de ses spécifications, dont le but est de produire des fonctions de test, débogage, diagnostic, surveillance… etc. Cependant, le terme « instrumentation sur puce » est utilisé ordinairement pour se référer à l'instrumentation sur puce pour le débogage. On distingue deux types d'instrumentation sur puce : l'instrumentation post-silicium (post-silicon) et l'instrumentation pré-silicium (pre-silicon).

2.2.1 Instrumentation sur puce post-silicium

Pour ce type d'instrumentation, des blocs matériels (circuit) dédiés à l'instrumentation sont fabriqués avec le CI final, afin de satisfaire des besoins d'instrumentation post-silicium. Ce type d'instrumentation est couramment utilisé dans le domaine de test où pratiquement tous les CI sont équipés avec un circuit additif dédié principalement pour le test de fabrication. Les normes IEEE-1491 (Boundary Scan) et IEEE-1500 sont souvent utilisées dans les circuits de test des CI [11].

Durant la dernière décennie, plusieurs solutions efficaces ont été développées autour de l'instrumentation sur puce pour satisfaire les besoins de débogage des SoC modernes. En effet, au fur et à mesure que la modélisation et l'analyse d'une conception passent de la simulation au matériel, il survient une perte de visibilité et d'accès aux signaux internes de la conception. En simulation, tous les signaux, variables et paramètres de modélisation sont disponibles pour la visualisation, fournissant ainsi un environnement d'analyse riche, indépendamment des autres limitations. Une fois la conception est matérialisée, que ce soit dans le système d'émulation ou dans le silicium final, un problème de débogage se pose à cause de la diminution de la quantité des signaux contrôlables et observables à travers les broches d'E/S du système (Figure 2-3). Dans cet environnement matériel, l'instrumentation sur puce augmente significativement la quantité de visibilité en temps réel et le contrôle de la conception au prix de l'ajout de blocs d'analyse à la conception [55].

Le débogage post-silicium permet la vérification des systèmes dans leurs conditions réelles afin de recouvrir les bugs non détectés dans la phase de conception, c'est pour cette raison que plusieurs solutions à base de l'instrumentation sur puce ont été proposées pour le débogage post-silicium. La Figure 2-4 présente un exemple pratique d'une instrumentation sur puce pour le débogage post-silicium dédié aux SoC. Cette solution, appelée MCDS (Multi Core Debug Solution), utilise la compression de données pour

augmenter la contrôlabilité et l'observabilité dans les SoC [56].

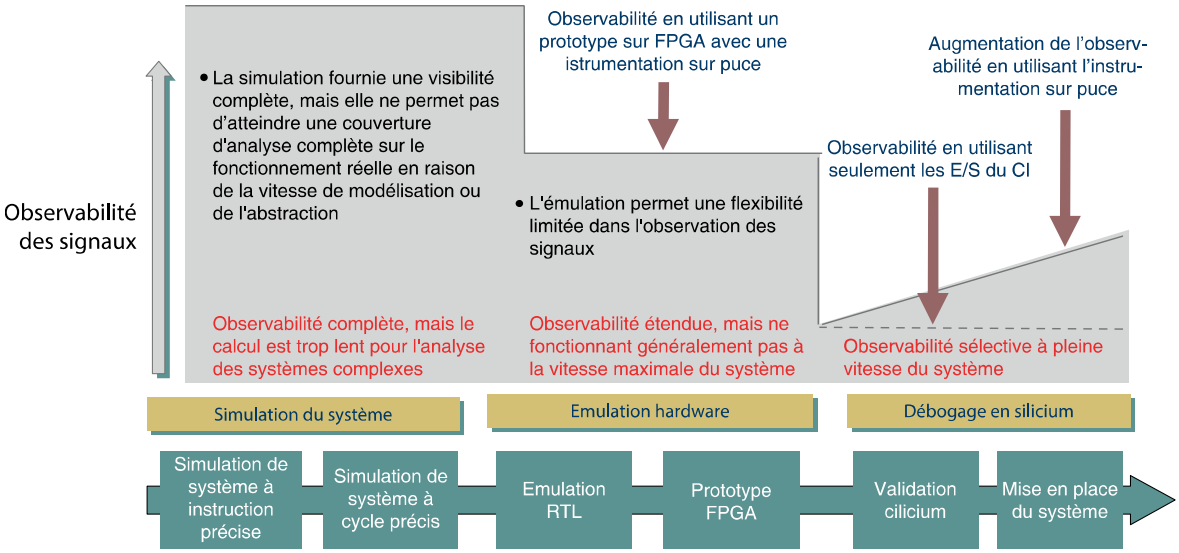


Figure 2-3 Observabilité au cours des étapes de conception [55].

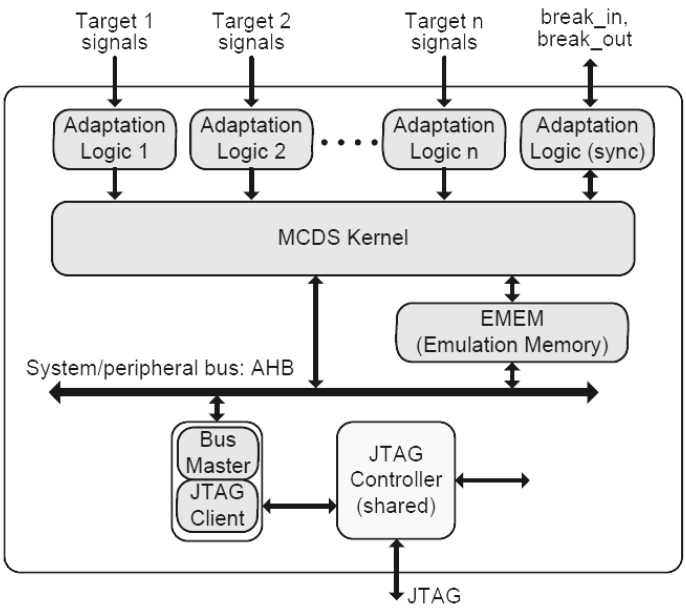


Figure 2-4 Schéma blocs du système de débogage MCDS avec l'accès JTAG [56].

Parmi les travaux de recherche ayant ciblé l'instrumentation sur puce post-silicium on peut citer celui de [57] et de [58] où les auteurs proposent des Infrastructures-IP pour la détection et la correction de fautes dans les SoC. D'autres travaux proposent des circuits d'instrumentation multi tâches qui permettent à la fois le test, le diagnostic et le débogage [59] [60].

2.2.2 Instrumentation sur puce pré-silicium

Ce type d'instrumentation utilise des FPGA pour émuler le système à instrumenter. La logique programmable fournit alors un excellent moyen pour réaliser différents types d'instrumentations comme le débogage, mesure de performance, analyse de fautes… etc. La contrôlabilité et l'observabilité des signaux dans ce cas sont assez grandes et adaptables selon le besoin (Figure 2-3) [55].

2.3 Identification par radiofréquence – RFID

L'utilisation de la technologie d'identification par radiofréquence (RFID) est devenue de plus en plus répandue dans plusieurs domaines industriels. Cette technologie est idéale pour l'identification et le suivi d'objets, tout en utilisant une simple antenne à faible coût attachée à l'élément en question. La liaison par radiofréquence permet alors la lecture automatique de données pour laquelle il existe maintenant plusieurs normes dédiées. Le progrès considérable de la RFID notamment dans sa normalisation a contribué significativement dans son déploiement intensif dans de nombreuses applications telles que la gestion de stock, contrôle d'accès, clés à puces, cartes bancaires, passeports biométriques, et l'internet des objets. Malgré cette évolution remarquable, le déploiement de la RFID présente toujours quelques inconvénients qui concernent principalement les aspects de sécurité.

2.3.1 Principe de fonctionnement

Un système RFID est constitué généralement de trois éléments (Figure 2-5) :

2.3.1.1 Tag

Ce terme provient du mot anglais *tag* qui signifie étiquette, et *to tag*, cocher, marquer [61]. Les tags RFID (appelés aussi **étiquettes** ou **transpondeurs**) sont des éléments qui contiennent couramment une mémoire dans laquelle sont stockées des données numériques. Ces données peuvent être lues ou modifiées par un lecteur via une liaison RF (sans contact). Il existe plusieurs types de tags (Figure 2-6) ; certains types utilisent des batteries pour fonctionner, tandis que d'autres (représentent le cas le plus courant) ne contiennent aucune batterie. Pour faire fonctionner ces derniers, les tags récupèrent de l'énergie à partir des ondes électromagnétiques émises par le lecteur.

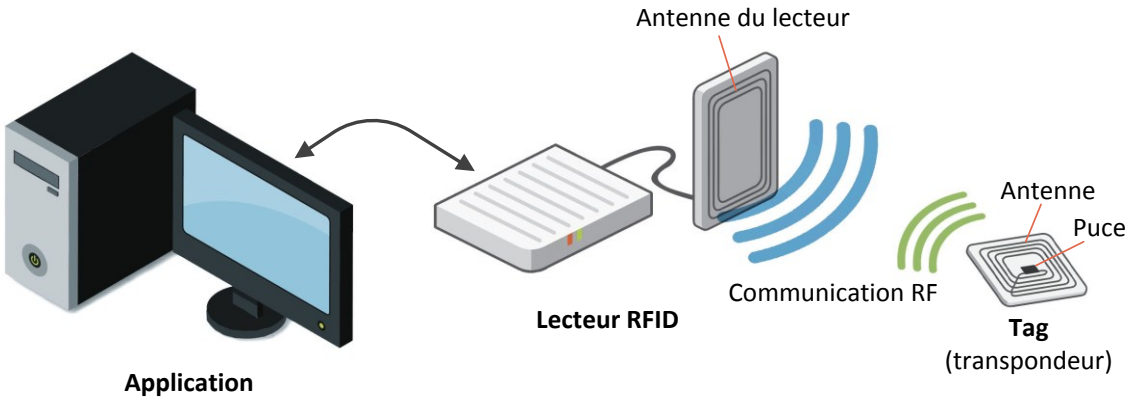


Figure 2-5 Représentation d'un système RFID.

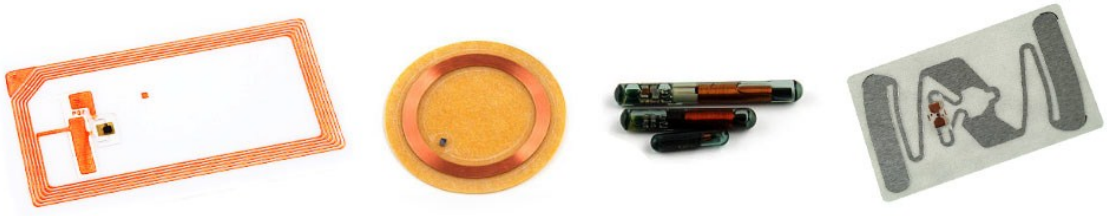


Figure 2-6 Quelques types de tags RFID.

Selon le moyen de communication que dispose le tag et qui l'utilise pour assurer la liaison descendante (du tag vers le lecteur) ainsi que la façon à laquelle il est alimenté, on distingue trois catégories de tags [62] [63] : tags passifs, tags semi-passifs et tags actifs.

2.3.1.1.1 Tags passifs

Ils rétrodiffusent (*backscatter*) les ondes électromagnétiques provenant du lecteur, c'est la seule façon de communiquer avec le lecteur. En d'autres termes, ils n'ont aucun émetteur RF à bord de sorte qu'ils ne peuvent pas créer leurs propres signaux RF. Les tags passifs sans batterie utilisent le signal entrant provenant du lecteur pour fournir l'énergie à la puce intégrée. 'Sans batterie' et 'passive' sont deux caractéristiques différentes du tag, mais malheureusement elles sont souvent confondues.

2.3.1.1.2 Tags semi-passifs

Les tags semi-passifs disposent d'une batterie pour alimenter la circuiterie interne, des capteurs ou des actionneurs connectés. Cette source d'alimentation n'est pas utilisée pour créer un signal RF quelconque, car le tag est toujours passif (uniquement rétrodiffusion du signal provenant du lecteur).

2.3.1.1.3 Tags actifs

Les tags actifs ont leur propre émetteur RF intégré. Ils peuvent alors générer leurs propres signaux RF sans le besoin de rétrodiffuser le signal arrivant du lecteur. Comme la création d'un signal RF nécessite beaucoup d'énergie, les tags actifs ont souvent une alimentation interne. Cela signifie qu'ils sont souvent confondus avec les tags à batteries.

2.3.1.2 Lecteur

Le lecteur RFID, (appelé aussi **interrogateur**) est utilisé pour communiquer avec les tags présents dans sa portée. Il peut alors interroger les tags pour effectuer des opérations de lecture, ou parfois des opérations d'écriture, de leurs contenus. Un lecteur est capable de gérer une grande quantité de tags en même temps. Pour satisfaire la grande variété des applications, il existe des lecteurs RFID mobiles (portables), comme il existe des lecteurs pour des applications dans des lieux fixes ce qui représente le cas le plus courant. Il existe aussi plusieurs formes et tailles de lecteurs dont certains possèdent des écrans et peuvent être exploités directement par l'utilisateur. La majorité des lecteurs RFID sont équipés avec des interfaces de communication pour pouvoir se brancher aux ordinateurs.

2.3.1.3 Logiciel d'application

Comme la majorité des systèmes actuels, les systèmes RFID ont besoin d'un logiciel d'application pour faire fonctionner l'ensemble du système. Le logiciel est utilisé pour gérer un ou plusieurs lecteurs afin d'effectuer des opérations d'identification des tags et/ou des opérations de lecture et de modification de leurs contenus. En parallèle, le logiciel permet la sauvegarde et l'analyse de toutes les données.

2.3.2 Classification des systèmes RFID

Établir une classification complète de la technologie RFID est difficile étant donné le nombre de caractéristiques qui doivent être considérées pour définir un tag. C'est uniquement lors de la conception d'une application RFID qu'il est possible de définir les besoins technologiques et, par la suite, d'identifier le type de tag le plus approprié. Si nous voulons conserver une vue d'ensemble des systèmes RFID, nous devons rechercher des fonctionnalités qui peuvent être utilisées pour différencier un système RFID d'un autre (Figure 2-7).

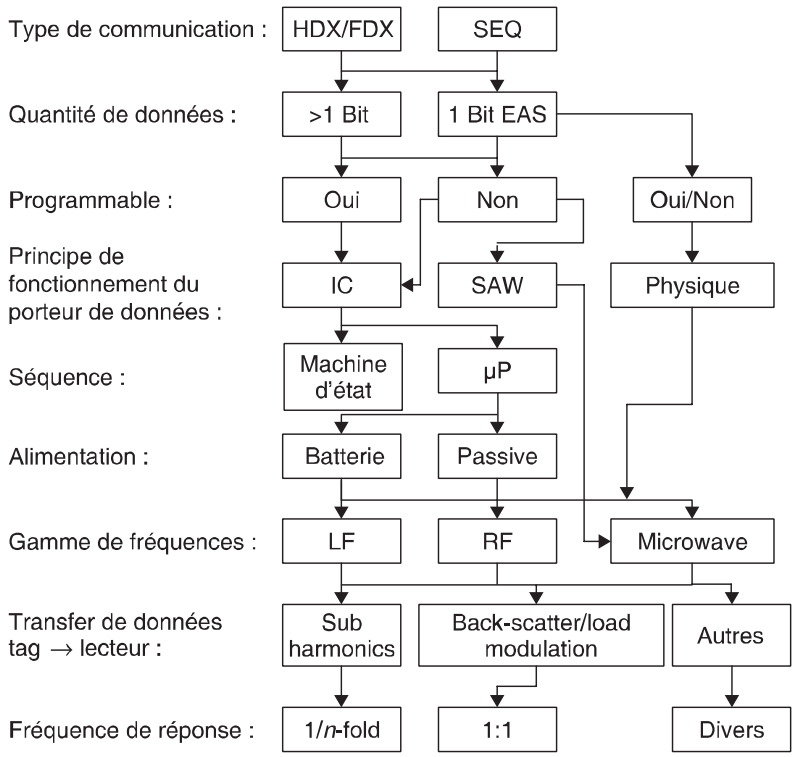


Figure 2-7 Différentes caractéristiques des systèmes RFID [62].

Il existe donc plusieurs manières de classifier les systèmes RFID. La classification la plus répandue et communément admise est la classification en fonction de la fréquence de l'onde utilisée pour les communications lecteurs/tags. En effet, cette fréquence est directement liée aux phénomènes physiques mis en jeu pour réaliser la communication [61]. Le Tableau 2-1 donne les différentes fréquences utilisées en RFID.

Tableau 2-1 Fréquences retenues et/ou autorisées en RFID.

Ondes radiofréquences		**Fréquences retenues et/ou autorisées en RFID**	
De 30 à 300 kHz	LF	Basses fréquences	< à 135 kHz
De 3 à 30 MHz	HF	Hautes fréquences	13,56 MHz
De 300 à 3000 MHz	UHF	Ultra-hautes fréquences	433 MHz, de 860 à 960 MHz et 2,45 GHz
De 3 à 30 GHz	SHF	Super-hautes fréquences	5,8 GHz

Dans cette étude, nous nous intéressons particulièrement aux systèmes RFID qui fonctionnent dans la bande UHF. En effet, les utilisations actuelles de la RFID qui nécessitent l'étude de la sûreté de fonctionnement des systèmes sont principalement dans ce domaine de fréquences.

Les applications utilisant ces fréquences sont soit des applications critiques liées au nucléaire, au ferroviaire ou encore à l'aéronautique ; soit des applications de logistique, mettant en œuvre une grande quantité de tags et dont le but est de minimiser les

interventions humaines. Une défaillance dans le système RFID d'une application logistique va nécessiter une intervention humaine coûteuse en temps et en argent.

2.3.3 Protocoles et standards de la RFID

L'Organisation internationale de normalisation (*International Standards Organization* – ISO) et EPCglobal sont deux organisations qui travaillent ensemble pour approuver les normes et les protocoles afin de fournir des spécifications universelles pour la RFID. En créant des normes mondiales, ces organisations favorisent l'adoption mondiale de la RFID. Les protocoles ratifiés définissent les méthodes de communication approuvées avec l'interface radio en conjonction avec la fréquence de fonctionnement, la bande passante du canal, le saut de fréquence… etc.

La plupart des tags RFID et des codes à barres qui contiennent un code de produit sont régis par des normes et des directives créées par EPCglobal. Cette organisation a créé le format standard pour le numéro **EPC** (Electronic Product Code), qui comprend un entête, un identifiant EPC unique et une valeur de filtre. L'organisation a également développé les normes pour les tags de Classe 1 Génération 2 (**EPC Gen2**), qui ont été ratifiées par l'ISO pour devenir ISO 18000-6C.

Les classes des tags RFID définies par EPCglobal sont approuvées par l'ISO et l'Organisation mondiale du commerce (*World Trade Organization* – OMC). Les tags RFID UHF utilisent le protocole d'interface radio ISO 18000, un protocole développé pour décrire les spécifications de communication des lecteurs et des tags mises en œuvre pour encourager l'adoption universelle. L'importance de cela réside dans les règlements mis en place dans le cadre du protocole décrivant les méthodes de communication approuvées entre le tag et le lecteur. Le Tableau 2-2 présente les 7 parties du protocole 18000 et les domaines de fréquence qu'elles affectent [61] [64].

Tableau 2-2 Les standard ISO 18000 (spécifications de l'interface radio).

Code du standard	Description
ISO 18000-1	Vocabulaires et définitions
ISO 18000-2	Communication inférieure à 135 KHz
ISO 18000-3	Communication à 13,56 MHz
ISO 18000-4	Communication à 2,45 GHz
ISO 18000-5	Communication à 5,8 GHz
ISO 18000-6	Communication à 860-960 MHz
ISO 18000-7	Communication à 433 MHz

2.3.4 Le standard EPC Gen2 (ISO 18000-6C)

Le standard **EPC UHF Class-1 Generation-2** (**EPC Gen2**) est produit par l'organisation EPCglobal et décrit les spécifications de communication entre les lecteurs et les tags selon le principe ITF (Interrogator Talks First). Les systèmes RFID ITF sont caractérisés par le fait que le tag ne rétrodiffuse aucune donnée sauf à la demande du lecteur (l'interrogateur). L'EPC Gen2 définit également trois parties liées à la communication lecteur-tag : le codage de données, la modulation et la gestion des collisions [65] [66].

- Encodage (*Bit-coding*) :
  - Du lecteur vers le tag (**liaison montante**, *Reader→Tag*, *forward link* ou *down-link*) : encodage par intervalles d'impulsions (Pulse Interval Encoding − **PIE**).
  - Du tag vers le lecteur (**liaison descendante**, *Tag→Reader*, *backward link*, ou *up-link*) : encodage **FM0** (Bi-Phase Space Coding) ou **Miller** (Miller Encoded Subcarrier).
- Type de modulation :
  - Du lecteur vers le tag : **ASK** (Amplitude-Shift Keying).
  - Du tag vers le lecteur : **ASK** et/ou **PSK** (Phase-Shift Keying).
- Gestion de collision par le lecteur : l'algorithme Q [65].

2.3.4.1 Encodage et modulation

2.3.4.1.1 Liaison montante (R→T)

Un lecteur RFID peut communiquer avec un ou plusieurs tags par liaison RF en utilisant la modulation ASK. Un lecteur peut lancer un cycle de détection des tags disponibles à sa portée ou une détection sélective (recherche de tags prédéfinis). Chaque cycle de détection est initié par une commande '*Query*' et représente cependant une **tournée d'inventaire** (*inventory round*).

A) Encodage de données

Le lecteur utilise l'encodage PIE pour effectuer la liaison montante (R→T). La Figure 2-8 présente les deux symboles de cet encodage. Le '*Tari*' représente l'intervalle du temps de référence qui est équivalent à la durée du symbole de la donnée '0'. La durée du *Tari* peut varier de 6,25 µs jusqu'à 25 µs. Durant chaque tournée d'inventaire, le lecteur utilise un

débit de données constant.

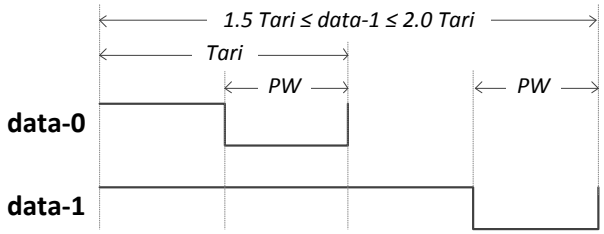


Figure 2-8 Les symboles de l'encodage PIE.

B) Préambule et synchronisation de trame de la liaison montante

Chaque commande envoyée par le lecteur commence par un préambule (*preamble*). Son utilisation permet de synchroniser la réception de données. La Figure 2-9 montre les formes du préambule et de la 'synchronisation de trame' (*frame-sync*) utilisées dans la liaison montante. Chaque trame d'une commande *Query* commence toujours par un préambule, alors que les trames des autres commandes commencent par *frame-sync*.

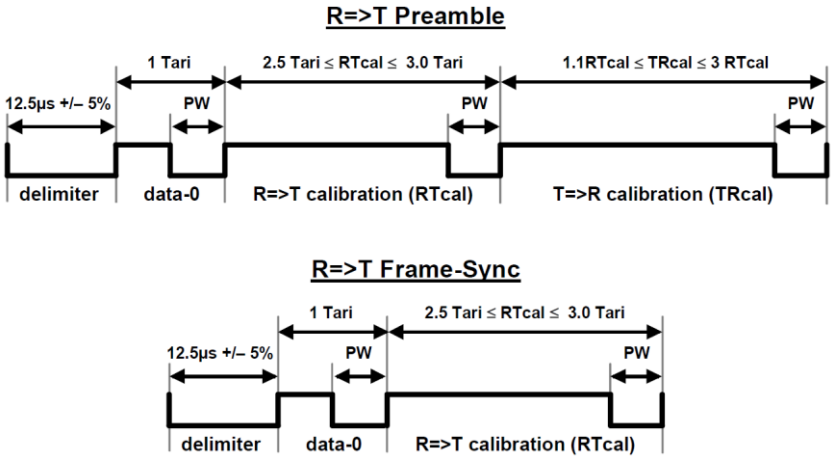


Figure 2-9 Préambule et 'synchronisation de trame' de la liaison montante.

RTcal (Reader-to-Tag calibration symbol) est utilisée par les tags pour calculer la largeur de la donnée '1' où *RTcal* est égale à la largeur de la donnée '0' plus la largeur de la donnée '1' ($RTcal = 0_{length} + 1_{length}$). Or, la largeur de la donnée '0' est égale au *Tari*. D'autre part, *TRcal (Tag-to-Reader calibration symbol)* est utilisée aussi par les tags, mais cette fois-ci pour calculer le débit des symboles de la liaison descendante (*backscatter link frequency* − BLF). Ce débit est défini par l'Équation (2.1) [65]:

$$BLF = \frac{DR}{TRcal} \quad (2.1)$$

Où *DR* (*divide ratio*) est un paramètre de 1 bit défini par la commande *Query* (*DR* = 8 ou 64/3). La transmission de données commence toujours par le bit MSB en liaison montante et descendante.

2.3.4.1.2 Liaison descendante (T→R)

Le tag communique avec le lecteur en utilisant la modulation de rétrodiffusion (*backscatter modulation*), dans laquelle le tag change le coefficient de réflexion de son antenne entre deux états en fonction des données envoyées. Le tag peut alors utiliser la modulation ASK ou PSK pendant la rétrodiffusion.

A) Encodage de données

Les types d'encodage utilisés par le tag durant la rétrodiffusion sont FM0 ou Miller. Le lecteur précise dans la commande *Query* (par le paramètre *M*) le type d'encodage qui devrait être utilisé par le tag. Pour *M* = '00', l'encodage FM0 est sélectionné, et pour les autres valeurs de *M*, l'encodage Miller est sélectionné. La Figure 2-10 montre la forme des symboles pour l'encodage FM0.

Figure 2-10 Les symboles de l'encodage FM0.

B) Préambule et fin de séquences

Comme dans le cas du lecteur, chaque séquence de trames rétrodiffusées par le tag commence par un préambule. La Figure 2-11 illustre les deux formes de préambule dans le cas de FM0. La forme du préambule est sélectionnée par le lecteur à travers le paramètre *TRest*, défini dans la commande *Query*. Également, chaque séquence de trames se termine par un bit de donnée '1' en plus (*dummy data-1*), comme le montre la Figure 2-12.

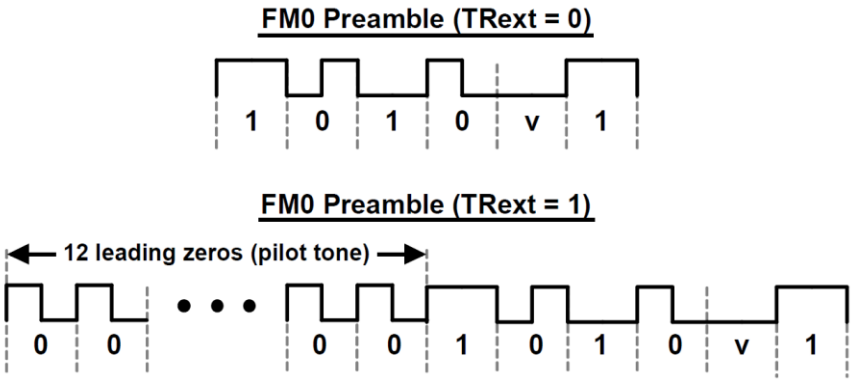


Figure 2-11 Préambule de FM0.

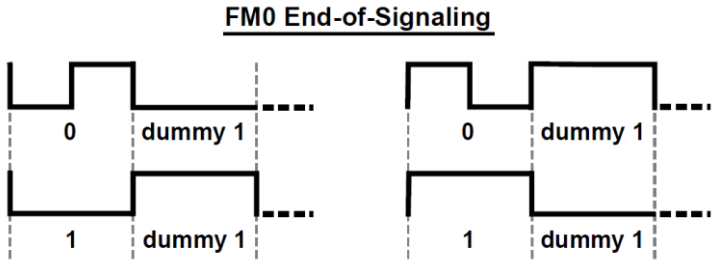


Figure 2-12 Fin de transmissions de FM0.

2.3.4.2 Vérification de trames

Le lecteur et le tag utilisent le CRC (contrôle de redondance cyclique, ou *cyclic-redundancy check* en anglais) pour vérifier la validité de certaines trames. Le protocole EPC Gen2 utilise deux types de CRC : CRC-16 et CRC-5. Pour générer un CRC-16, un tag ou un lecteur doit d'abord générer le précurseur CRC-16 indiqué dans le Tableau 2-3, puis faire le complément à un pour former le CRC-16. Le CRC-5 se calcule suivant la définition donnée dans le Tableau 2-4.

Tableau 2-3 Précurseur CRC-16.

Type de CRC	Longueur	Polynôme	Preset	Résidu
ISO/IEC 13239	16 bits	$x^{16} + x^{12} + x^5 + 1$	$FFFF_h$	$1D0F_h$

Tableau 2-4 Définition du CRC-5.

Type de CRC	Longueur	Polynôme	Preset	Résidu
–	5 bits	$x^5 + x^3 + 1$	01001_2	00000_2

2.3.4.3 Commandes du lecteur et réponses du tag

Le lecteur RFID peut communiquer avec un ou plusieurs tags à travers l'envoi de quelques trames de commandes. A leur tour, les tags retournent des réponses dans des cas précis. Chaque trame de commande est composée de trois parties : un **préambule**, un **code de commande** et un ou plusieurs **paramètres**. Quelques commandes contiennent une quatrième partie dédiée à un **code de vérification** (CRC-16 ou CRC-5). Un exemple sur la commande *Query* est donné par la Figure 2-13. Le code de cette commande est '1000' ; ses paramètres sont *DR*, *M*, *TRext*, *Sel*, *Session*, *Target*, et *Q* ; son code de vérification est le CRC-5. Dans le cas où le tag répond à cette commande, il répond par RN16 (16 bits). Un exemple de réponse est donné à la Figure 2-14.

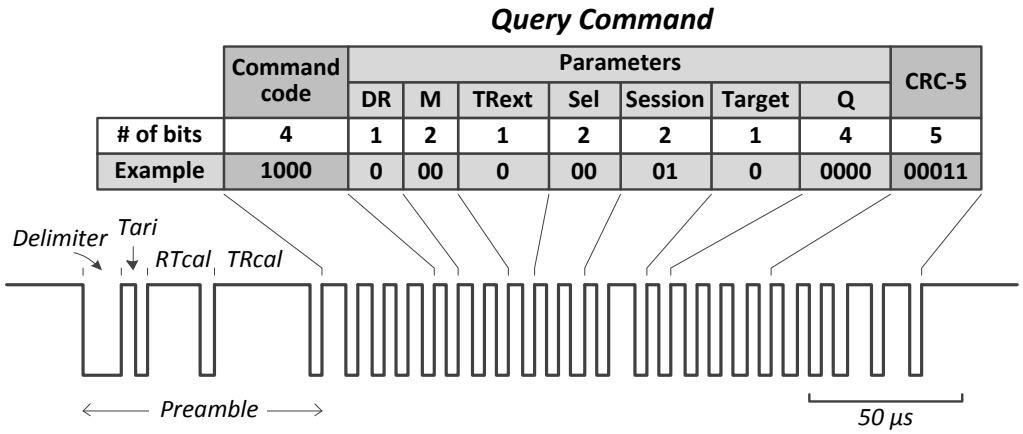


Figure 2-13 Exemple d'une commande *Query*.

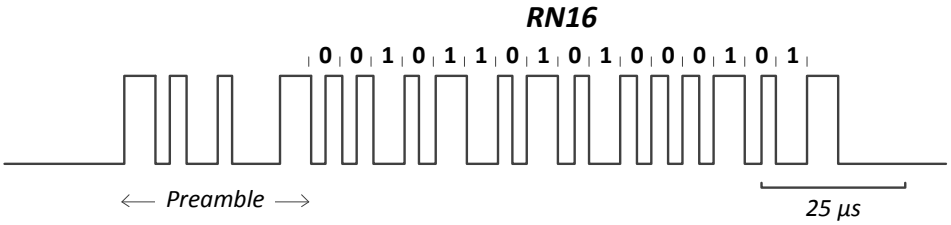


Figure 2-14 Exemple de réponse de tag (*RN16*).

Le Tableau 2-5 donne la liste des commandes selon le protocole EPC Gen2 [65]. Quelques commandes sont obligatoires (*mandatory*), c'est-à-dire que tous les lecteurs et les tags compatibles avec ce protocole doivent supporter ces commandes, alors que les autres sont optionnelles.

Tableau 2-5 Les commandes de EPC Gen2.

Commande	Code	Longueur	Obligatoire?	Protection
QueryRep	**0**0	4	Oui	Longueur fixe
ACK	**0**1	18	Oui	Longueur fixe
Query	**10**00	22	Oui	CRC-5
QueryAdjust	**10**01	9	Oui	Longueur fixe
Select	**10**10	> 44	Oui	CRC-16
NAK	**110**00000	8	Oui	Longueur fixe
Req_RN	**110**00001	40	Oui	CRC-16
Read	**110**00010	> 57	Oui	CRC-16
Write	**110**00011	> 58	Oui	CRC-16
Kill	**110**00100	59	Oui	CRC-16
Lock	**110**00101	60	Oui	CRC-16
Access	**110**00110	56	Non	CRC-16
BlockWrite	**110**00111	> 57	Non	CRC-16
BlockErase	**110**01000	> 57	Non	CRC-16
BlockPermalock	**110**01001	> 66	Non	CRC-16

2.3.4.4 Etats du tag

Tous les tags conformes à EPC Gen2 intègrent les sept états suivants : *Ready*, *Arbitrate*, *Reply*, *Acknowledged*, *Open*, *Secured* and *Killed*. Lorsqu'il est alimenté, un tag non 'tué' (*not killed*) se met dans l'état *Ready*. Ensuite, le tag peut changer son état suivant les commandes reçues par le lecteur. Un digramme d'état détaillé est donné dans l'Annexe A. La Figure 2-15 illustre un exemple d'accès à un tag et la lecture de son code EPC, le tag change son état de *Ready* à *Reply* quand il envoie un *RN16*, ensuite il change son état à *Acknowledged* quand il envoie son EPC, et ainsi de suite.

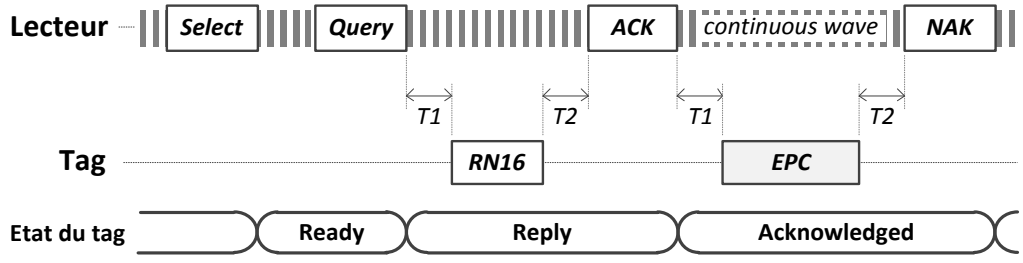


Figure 2-15 Exemple de communication entre un lecteur et un tag.

2.3.4.5 Mémoire du tag

Les tags compatibles avec EPC Gen2 intègrent une mémoire non volatile organisée en quatre banques comme le montre la Figure 2-16 [65] :

- **Mémoire réservée** (Reserved) : contient deux mots de passe : le mot de passe d'accès (*access password*) qui doit être délivré par le lecteur pour accéder à l'état sécurisée (*Secured*) du tag ; et '*killed password*' qui permet de 'tuer' le tag (le faire entrer dans l'état *Killed*).
- **Mémoire EPC** : contient le code EPC (Electronic Product Code) du tag.
- **Mémoire TID** : contient l'identifiant du tag (Tag ID) qui comprend un identifiant de classe d'allocation sur 8 bits conformément à ISO/IEC 15963, ainsi que des informations sur les commandes et les options supportées.
- **Mémoire d'utilisateur** (User) : c'est une mémoire optionnelle réservée à l'utilisateur.

La lecture ou le changement du contenu de la mémoire sont possibles à travers les commandes *Read* et *Write*. Cependant, la lecture et/ou l'écriture peuvent être verrouillées, le verrouillage se fait à travers la commande *Lock*.

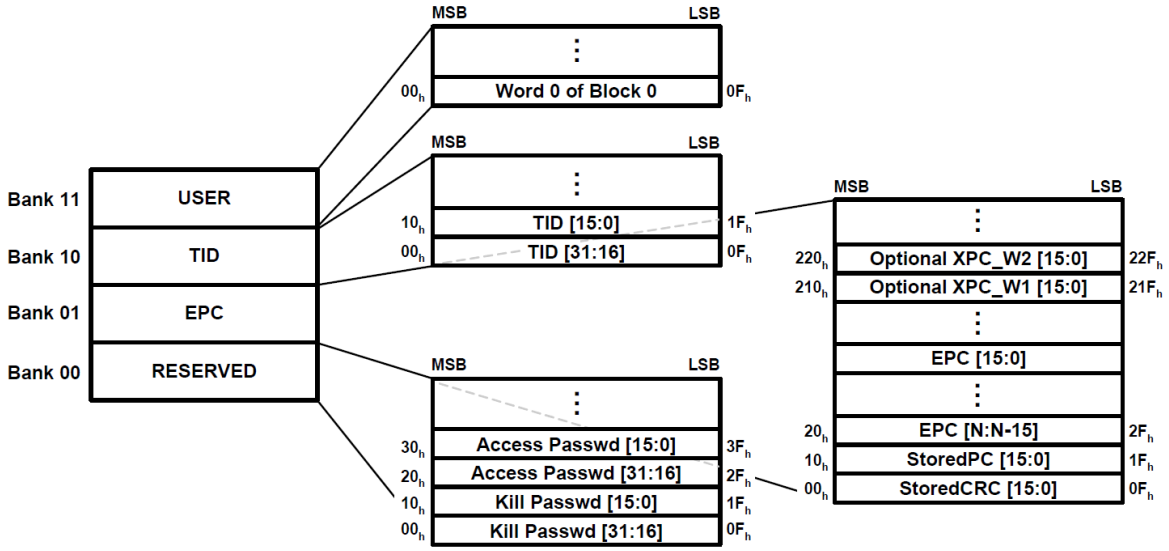


Figure 2-16 Organisation de la mémoire du tag [65].

2.3.5 Mécanisme de sureté de fonctionnement d'un système RFID UHF

2.3.5.1 Définitions

La sureté de fonctionnement est une propriété importante à évaluer afin de déterminer si le

système est suffisamment fiable pour être utilisé dans telle ou telle application. Elle est caractérisée par cinq points principaux, appelés attributs de la sûreté de fonctionnement [67] [68] :

1) **La disponibilité :** c'est l'aptitude d'un composant ou d'un système à délivrer le service attendu à un instant t ;
2) **La fiabilité :** c'est l'aptitude d'un composant ou d'un système à fonctionner pendant un intervalle de temps $[t_0, t]$;
3) **La maintenabilité :** c'est l'aptitude d'un composant ou d'un système à être réparé ;
4) **La sécurité :** c'est l'aptitude d'un composant ou d'un système à ne pas conduire à des accidents catastrophiques, en terme humain, matériel ou financier ;
5) **La sécurité-immunité :** c'est l'aptitude d'un composant ou d'un système à garantir l'intégrité des données qu'il contient et à en empêcher l'accès ou les manipulations non autorisées (c'est-à-dire à garantir la confidentialité de ces données).

La Figure 2-17 illustre les relations existantes entre les différents attributs de la sûreté de fonctionnement [69]. En effet, la fiabilité a un impact direct sur la disponibilité, si la fiabilité est basse, il va y avoir beaucoup de défauts, et ainsi diminution de la disponibilité. De plus, il est connu qu'il y a souvent une corrélation entre l'apparition d'un défaut et l'occurrence d'un accident, la fiabilité impacte donc aussi la sécurité. Une maintenabilité inadaptée, dans le cas des systèmes réparables, peut compromettre la disponibilité du système car le nombre de défauts peut augmenter et le temps de réparation peut être long. En plus, si le nombre de défauts augmente, la maintenabilité augmente par conséquence le nombre possible d'accidents, diminuant par la même occasion la sécurité. Enfin, en général, la sécurité a un impact néfaste sur la disponibilité. En effet, dans la majorité des cas, la sécurité impose un service dégradé en présence de défauts pour minimiser le risque de panne, ce qui aura pour effet la diminution de la disponibilité ; par exemple, l'arrêt d'un moteur suite à un défaut. Néanmoins, dans certains cas, la nécessité de la sécurité impose une forte disponibilité comme dans le cas d'un système de freinage de voiture ou des systèmes de commandes d'un avion.

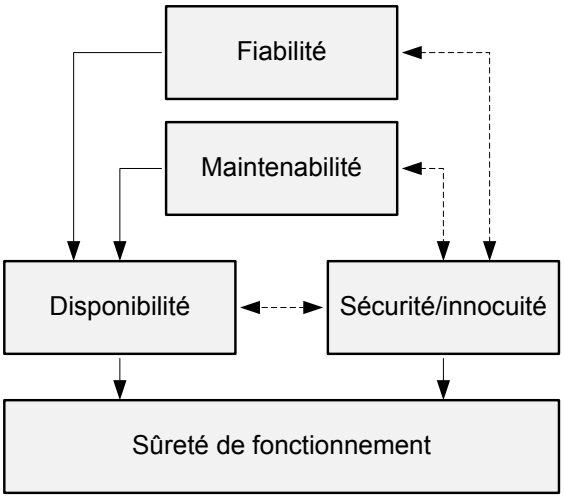


Figure 2-17 Relation entre les attributs de la sûreté de fonctionnement.

2.3.5.2 Questions de sécurité dans le cas de la RFID

Nous considérons qu'un tag UHF délivre un service correct si celui-ci obéit scrupuleusement aux exigences du protocole EPC Gen2. Une défaillance est soit la conséquence d'un tag non conforme au standard cité ou que son architecture ne constitue pas un système robuste. Dans cette thèse, nous ciblons la fiabilité et la sécurité de la partie digitale du tag. Dans la suite nous décrivons les problèmes potentiels liés à la fiabilité de la partie digitale du tag RFID UHF.

Lorsqu'une faute se produit dans un système, elle peut engendrer des erreurs, et par la suite elle peut causer une défaillance de service (Figure 2-18). Comme la fonctionnalité du tag est basée sur un séquencement correct d'un certain nombre d'états, la relation précédente peut se traduire par une transition incorrecte entre états. Par exemple, l'occurrence d'une ou plusieurs fautes dans les registres d'un tag peut causer sa transition à un état sécurisé (erreur), et par la suite engendrer une délivrance de ses données sécurisées (défaillance).

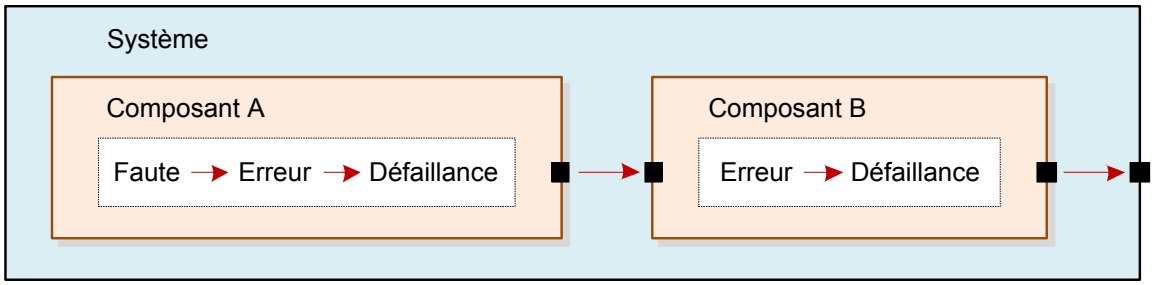


Figure 2-18 Propagation d'erreurs dans les systèmes numériques.

2.3.5.3 Approches utilisées

Afin d'améliorer la sûreté de fonctionnement d'un système, plusieurs moyens sont envisageables. Ces moyens peuvent être classés en quatre catégories [68] :

1) Elimination des fautes. Cette catégorie regroupe les techniques visant à réduire le nombre et la gravité des fautes :
   - Validation de conception
   - Analyse de testabilité
   - Test logiciel
   - Test de production
2) Tolérance aux fautes. Cette catégorie regroupe les mécanismes permettant au système de continuer à assurer un service malgré la présence de fautes :
   - Redondance
   - Test en ligne
   - Diagnostic en ligne
   - Reconfiguration
3) Prévision des fautes. Cette catégorie a pour objet l'estimation de la présence et des conséquences des fautes :
   - Simulation de fautes
   - Evaluation d'attributs
4) Prévention de fautes. Cette catégorie regroupe les moyens dont le but est d'empêcher l'introduction de fautes dans le système :
   - Conception particulière
   - Redondance matérielle
   - Redondance logicielle
   - Contrôle qualité

L'étude de la sûreté de fonctionnement est dépendante des applications. En effet, toutes les applications n'attachent pas la même importance à ses différents attributs. Ainsi, des moyens pour améliorer la sûreté de fonctionnement doivent être étudiés pour une

application particulière. Les systèmes RFID n'échappent pas à une étude de la sûreté de fonctionnement dépendant de l'application, prenant en compte ses caractéristiques intrinsèques et les besoins de l'utilisateur. Les auteurs dans [70] ont identifié les différents paramètres permettant d'évaluer la fiabilité et la disponibilité des systèmes RFID. Le principal paramètre est la capacité des tags à être identifié. En effet, la fonction principale d'un système RFID est la lecture de l'identifiant contenu dans le tag. De plus, cette capacité est directement impactée par les principaux facteurs engendrant une baisse de fonctionnement : la capacité du tag à récupérer l'énergie envoyée par le lecteur et la capacité du lecteur à envoyer de l'énergie aux tags ; la géométrie et l'architecture des antennes et des tags ; les protocoles… etc.

De plus, les systèmes RFID sont des systèmes embarqués, contenant à la fois du matériel et du logiciel qui fonctionnent en étroite collaboration. Ainsi, comme indiqué dans [71], il est important de prendre en compte les interactions existant entre le logiciel et le matériel : les défaillances de l'un doivent être gérées par l'autre. Ainsi, le logiciel peut venir aider au test et au diagnostic des équipements matériels pouvant tomber en panne. Il existe beaucoup d'études concernant l'amélioration de la sécurité des systèmes RFID. En effet, les systèmes RFID manipulent une grande quantité de données pouvant porter atteinte à la vie privée [72]. L'exemple le plus courant est le paiement sans contact. Comme les données échangées concernant les données bancaires, cette application demande une grande sécurité. En effet, les transactions pouvant être espionnées, il est donc important de protéger ces informations car elles renseignent sur les habitudes des personnes. Des personnes mal intentionnées peuvent alors s'en servir pour traquer les utilisations des usagers. Cette sécurité apparaît clairement nécessaire pour ce genre de systèmes (contrôle d'accès, paiement sans contact… etc.) mais elle est tout aussi nécessaire pour la gestion de stock. En effet, lorsqu'un magasin utilise la RFID pour gérer son stock, chaque produit est associé à un tag contenant des informations permettant d'identifier le type de produit, le fabricant ou encore son prix. Ainsi, tous les produits que le client souhaite acheter peuvent être identifiés lors du passage en caisse. Mais, une fois hors du magasin, pour préserver la vie privée, il est important que les tags ne soient plus lisibles, afin d'empêcher quiconque de voir ce que le client a acheté sans son accord. De ce fait, de nombreuses études existent et proposent des solutions, aussi bien au niveau applicatif [73], middleware [74] [75], lecteur [76] [77] ou bien tag [78] [79].

2.4 Conclusion

Trois aspects techniques liés aux travaux de cette thèse sont présentés dans ce chapitre : le diagnostic des fautes, l'instrumentation sur puce et la RFID. Le diagnostic est un domaine vaste et très important pour augmenter la fiabilité et la sécurité des systèmes. Le développement de nouveaux mécanismes de diagnostic embarqué pour les futurs systèmes est primordial afin d'équiper ces systèmes d'une intelligence particulière. Le test et la validation des systèmes complexes actuels sont accomplis grâce à l'utilisation de l'instrumentation sur puce. Ce type d'instrumentation possède des avantages multiples qui peuvent être employés dans le développement de nouvelles solutions de diagnostic embarqué. La RFID commence à s'imposer et se frayer des chemins dans plusieurs applications de grande importance. Cependant, son usage nécessite toujours de nouvelles solutions pour traiter les différentes problématiques qui concernent essentiellement l'aspect de la sécurité. L'utilisation de l'instrumentation sur puce, pour des fins de diagnostic dans les systèmes RFID, constitue l'objectif majeur de cette thèse.

Chapitre 3 Plateformes d'expérimentation développées

Afin d'appliquer les approches qui ont été consignées pour cette thèse, il était recommandé de commencer par développer des plateformes d'expérimentation qui permettraient d'appliquer d'une manière efficiente nos approches. Dans ce cadre, nous avons développé plusieurs conceptions et plateformes ayant nécessité des efforts considérables au niveau structurel et validation. Dans ce chapitre, nous présentons les différents systèmes développés qui comprennent un microcontrôleur configurable, un circuit de tag RFID à base de microcontrôleur, un circuit de tag RFID à base de machine d'état, un circuit de lecteur RFID et un système d'injection de fautes.

3.1 MCIP−C : Microcontrôleur 8-bit configurable

MCIP (Micro-Controller IP ou Mezzah-Chemali IP) est un microcontrôleur (*Microcontroller Unit* − MCU) sous forme d'IP développé en langage VHDL. Il a été élaboré en premier temps dans le cadre du projet de fin d'études d'ingéniorat [80] ; ensuite, il a été implémenté sur carte FPGA et testé dans une application réelle durant les travaux de Magister [6]. Le MCIP à une architecture Harvard 8-bit complètement compatible avec les microcontrôleurs de la famille PIC18 de Microchip [81] [82]. Une version allégée du MCIP est disponible en téléchargement sur le site web du OpenCores [83]. Les principales caractéristiques de cet IP sont comme suit :

- Architecture RISC (Reduced Instruction Set Computer) avec pipeline,
- Jeux d'instructions de 72 instructions,
- Bus d'instruction sur 16 bits et bus de données sur 8 bits,
- Mémoire de programme paramétrable jusqu'à 2 Mo,

- Mémoire de données paramétrable jusqu'à 4 Ko,
- Multiplicateur matériel 8x8,
- Watchdog et mode de veille,
- 5 sources d'interruption avec priorité,
- 2 Timers paramétrables 8/16 bits,
- 4 Ports d'entrée/sortie de 8 bits.

Nous avons utilisé MCIP dans les travaux de cette thèse pour développer une plateforme d'expérimentation RFID [84] ainsi que pour développer de nouvelles techniques de détection de fautes et de diagnostic en ligne [85] [2]. Pour utiliser MCIP dans ce but, il était primordial de le reconstituer pour le rendre configurable et plus flexible à supporter plusieurs types d'implémentation et d'instrumentation. Dès lors, la hiérarchie du MCIP a été rétablie et quelques améliorations ont été apportées sur certains modules, ce qui a permis d'aboutir à un MCU configurable davantage qu'à l'origine et que nous avons dénommé MCIP−C (C pour Configurable).

Nous aurions pu utiliser un MCU de type *open source* pour répondre à nos besoins (plusieurs exemplaires sont téléchargeables sur opencores.org), mais nous avons choisi d'utiliser MCIP-C vu que :

- Nous maîtrisons parfaitement ses différentes parties. Ainsi, il est plus simple et plus pratique de développer des applications et réaliser des instrumentations autour de MCIP−C qu'avec d'autres MCU non maîtrisés.
- Nous avons débogué et testé MCIP−C dans plusieurs applications, ce qui prouve son bon fonctionnement. Cela garantit la réussite de l'utilisation de MCIP−C par rapport aux autres MCU *open source* dont le bon fonctionnement n'est pas garanti et la flexibilité n'est pas assurée.
- Utiliser un MCU *open source* dans le but de réaliser nos objectifs d'instrumentation pour le diagnostic exige une maitrise profonde de ce MCU. L'utilisation de MCIP−C permet d'éviter de confronter ces types de difficultés et d'économiser un temps précieux.
- MCIP est un MCU 8-bit *low-power* adapté aux applications des systèmes embarqués communicants notamment pour la RFID.
- MCIP est compatible avec le PIC18 qui est non seulement un microcontrôleur très

populaire mais classé premier dans les ventes mondiales des microcontrôleurs 8 bits en 2006.

3.1.1 Constitution du MCIP−C

Afin de produire un MCU configurable, nous avons reproduit les composants du MCIP suivant un flot hiérarchique qui comprend trois types de composants (Figure 3-1) : composants de base (*Basic components*), composants configurables (*Configurable components*) et les périphériques (*Peripherals*). Les composants de base comprennent tous les composants fondamentaux du MCU et qui ne nécessitent aucune modification pour toute configuration désirée. Tandis que les composants configurables comprennent d'autres composants fondamentaux du MCU mais qui nécessitent d'être configurables. Ces derniers composants incluent le Contrôleur d'interruption, le Décodeur d'adresse et les Contrôleurs de la mémoire de données et la mémoire de programme ; leurs caractéristiques de configuration permettent de faciliter l'instanciation des différents périphériques et d'utiliser des mémoires de programmes et de données de tailles variées suivant les applications ciblées.

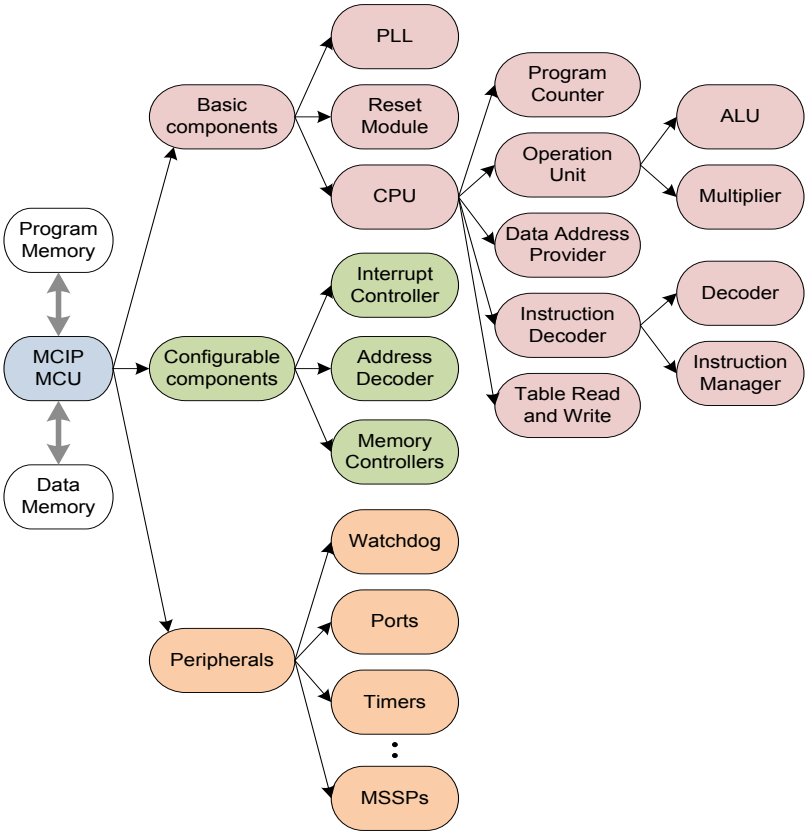


Figure 3-1 Flot de conception hiérarchique du MCIP−C.

Cette méthode de partitionnement des composants a permis de produire un MCU plus flexible comme l'illustre le diagramme de blocs de la Figure 3-2. Tous les composants fondamentaux sont alors regroupés dans le noyau de l'IP (MCIP−C core). Les mémoires de programme et de données ainsi que les blocs de périphériques peuvent être personnalisés selon l'application ciblée ensuite instanciés au noyau de l'IP. On distingue trois types de périphériques, des périphériques complètement compatibles avec PIC18 (ex. MSSP), périphériques compatibles avec PIC18 mais qui ont été personnalisés (ex. SPI Master), et de nouveaux périphériques absents de l'architecture du PIC18 (ex. PS/2).

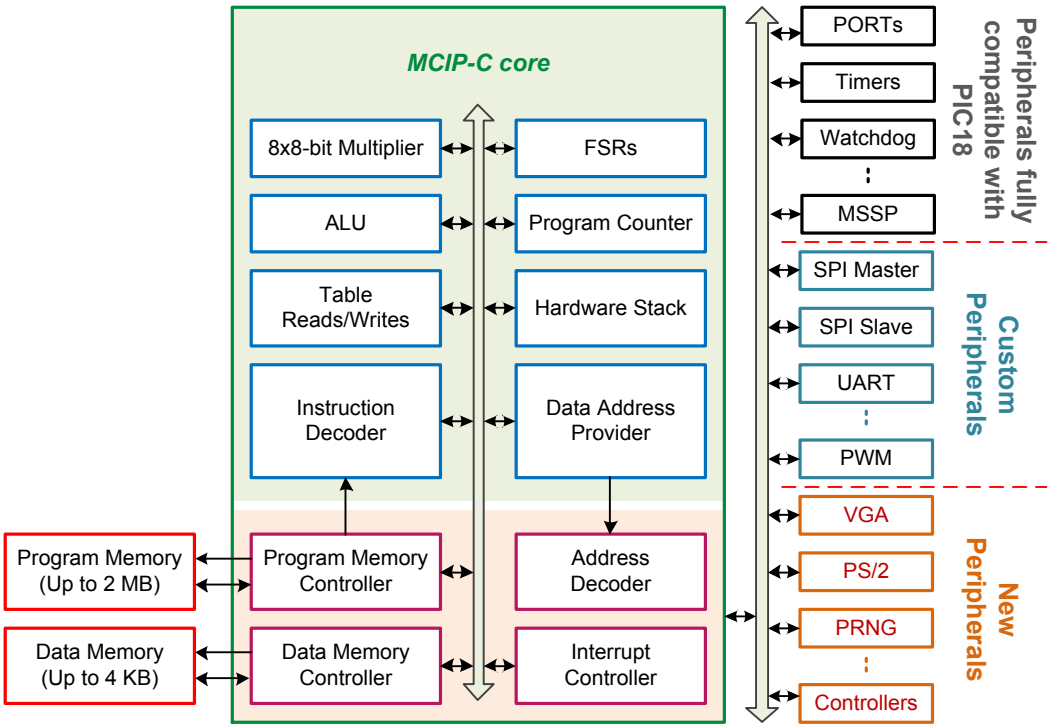


Figure 3-2 Diagramme de blocs du MCIP−C.

Pour instancier un périphérique au noyau du MCIP−C, il suffit, premièrement, de configurer les adresses SFR (Special Function Registers) de ce périphérique dans le Décodeur d'adresse (Address Decoder), puis configurer les interruptions correspondantes dans le Contrôleur d'interruption (Interrupt Controller) et finalement connecter ce périphérique aux bus du noyau. Le concepteur peut donc personnaliser des périphériques ou développer de nouveaux périphériques non disponibles dans les MCU de PIC18 et les instancier aisément au MCIP−C. Cette fonctionnalité représente une amélioration importante, car elle permet d'accroître l'efficacité des applications en réduisant des parties du logiciel embarqué (*firmware*) et les implémenter en hardware comme périphériques.

Une autre amélioration apportée au MCIP−C permet au CPU d'exécuter une instruction (un cycle d'exécution) en une période d'horloge, alors qu'en mode de fonctionnement normal (conformément au PIC18), un cycle d'exécution dure 4 périodes d'horloge. Cette amélioration permet donc de multiplier par quatre la vitesse d'exécution (1 M instructions/MHz au lieu de 0,25 M instructions/MHz) [84]. De nombreux autres périphériques ont été développés notamment des contrôleurs SPI Master/Slave, UART, PS/2, VGA, PWM… etc.

3.1.2 Implémentation sur FPGA

L'implémentation du MCIP−C (sans périphériques) sur différents types de FPGA de la marque Xilinx a donné les résultats présentés dans le Tableau 3-1. Concernant l'implémentation sur Spartan-3E (XC3S500E), MCIP−C a occupé 260 bascules (*flip-flops*) et 1242 LUT (Look-Up Table) ce qui correspond respectivement à 2 % et 13 % des ressources logiques du FPGA. Par ailleurs, la fréquence de fonctionnement peut atteindre 213 MHz sur Kintex-7.

Tableau 3-1 Implémentation du MCIP−C sur différents types d'FPGA de Xilinx.

FPGA	Flip-Flops	LUTs	Slices	F max
Spartan3E (xc3s500e)	260 (2%)	1242 (13%)	694 (14%)	56 MHz
Spartan6 (xc6slx9)	262 (2%)	978 (17%)	384 (26%)	111 MHz
Virtex5 (xc5vlx50t)	256 (1%)	790 (2%)	314 (4%)	156 MHz
Kintex7 (xc7k70t)	255 (1%)	855 (2%)	333 (3%)	213 MHz

3.2 G2MC : tag RFID à base de MCIP−C

La caractéristique de configuration du MCIP−C est exploitée dans la conception d'une nouvelle architecture de tag RFID où de nouveaux modules, tels que "Transmitter" et "Receiver", sont développés et ajoutés au MCIP−C en tant que périphériques (Figure 3-3). Plusieurs fonctionnalités du tag sont donc intégrées dans ces périphériques, ce qui a permis de réduire considérablement la partie *firmware* du tag et de la rendre facile à élaborer. Le tag développé, dénommé G2MC, est alors aussi flexible, permettant d'ajouter toute fonctionnalité optionnelle ou personnalisée conformément au protocole EPC UHF Gen2 [65].

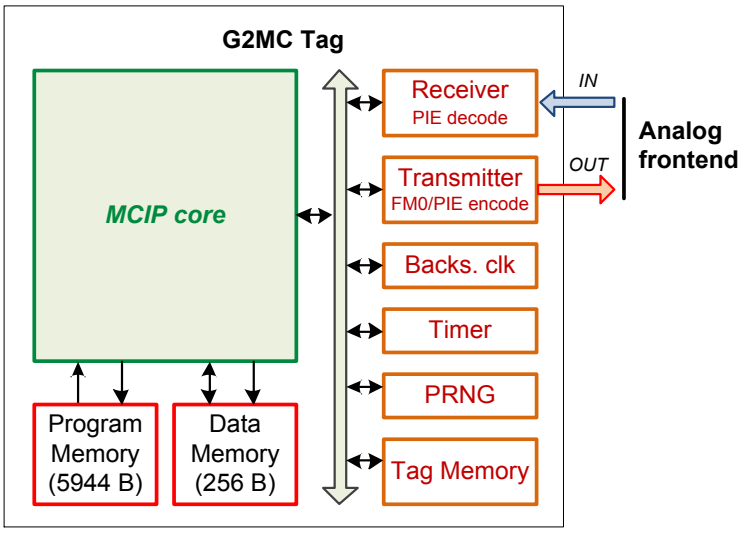


Figure 3-3 Diagramme de blocs du tag G2MC.

Comme le montre la Figure 3-3, six périphériques sont développés en VHDL et ajoutés au MCIP−C pour former la partie hardware du tag. Les modules "Receiver" et "Transmitter" constituent les principaux modules qui sont chargés de la génération et la réception des trames RF. Les fonctions des autres périphériques sont :

- "Backscatter clk" : génère le signal d'horloge nécessaire pour la génération des trames des réponses ;
- "Timer" : utilisé pour le comptage des périodes de temps ;
- "PRNG" (Pseudo-Random Number Generator) : un générateur de nombre aléatoire ;
- "Tag Memory" : inclut les quatre banques de mémoire du tag "Reserved memory", "EPC memory", "TID memory" et "User memory" conformément au protocole EPC Gen2 [65].

Le diagramme de blocs de la Figure 3-4 donne une explication simplifiée du fonctionnement du "Receiver" qui est contrôlé à travers le registre RXCON. Un signal d'interruption est généré à chaque fois qu'une partie de trame est reçue. L'état du "Receiver" peut être vérifié à partir du registre RXSTAT qui donne des informations sur le déroulement de réception d'une trame.

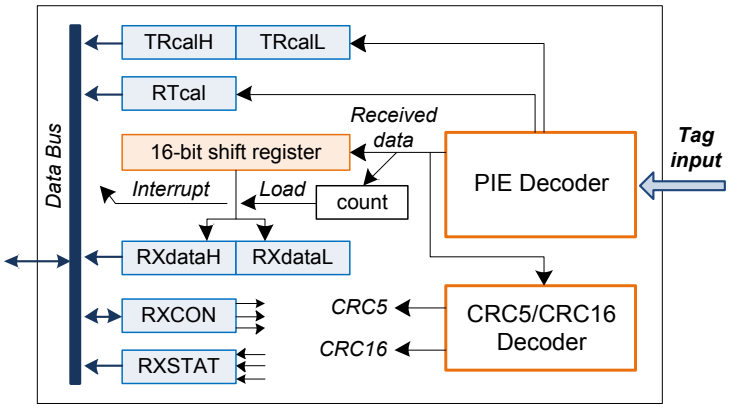


Figure 3-4 Diagramme du module "Receiver".

Le firmware du tag est développé en utilisant le langage C en conjonction avec l'assembleur pour produire un programme optimal. Ce dernier est organisé en 20 fonctions en plus du programme principal ("main") et la routine d'interruptions (Figure 3-5). Les fonctions "Frame reception", "Open state" et "Secured state" occupent un espace de mémoire plus grand que les autres fonctions vu qu'elles intègrent la majorité des fonctionnalités du tag. Notant que les fonctions "Open state" et "Secured state" comprennent presque les mêmes fonctionnalités donc elles peuvent être regroupées. Nous avons gardé ces deux fonctions séparées pour pouvoir distinguer les différents états du tag et préserver une flexibilité dans la détection des fautes et dans l'application des techniques de diagnostic et d'amélioration de sécurité. La taille totale du programme obtenu est de 5944 octets utilisant seulement 28 octets de RAM. Afin d'optimiser la consommation d'énergie, le mode veille est employé extensivement dans le programme ; ainsi, durant une communication entre G2MC et un lecteur, le CPU du G2MC reste en mode veille environ 90 % du temps.

L'implémentation de G2MC sur un FPGA de type Virtex-5 (xc5vlx50t) a donné les résultats présentés dans le Tableau 3-2. Selon les résultats obtenus, le nombre des cellules de registres (*slice registers*) occupées est de 674 cellules, tandis que le nombre des cellules LUT est de 3475 cellules, ce qui représente en total 12 % des ressources de l'FPGA. Notant que cette implémentation inclut le *firmware* du tag qui lui seul occupe environ 7 % des ressources de l'FPGA. Une implémentation physique du G2MC sur une carte FPGA de type LX9 MicroBoard est présentée à la Figure 3-6. Une petite carte de *Front-end* analogique (réalisée par le laboratoire de conception et d'intégration des systèmes (LCIS) de l'Institut polytechnique (INP) de Grenoble) est employée avec une antenne afin de

pouvoir communiquer avec G2MC à travers un lecteur RFID externe. En utilisant ce montage, nous pouvons employer G2MC avec toutes ses fonctionnalités comme tout autre tag UHF ordinaire.

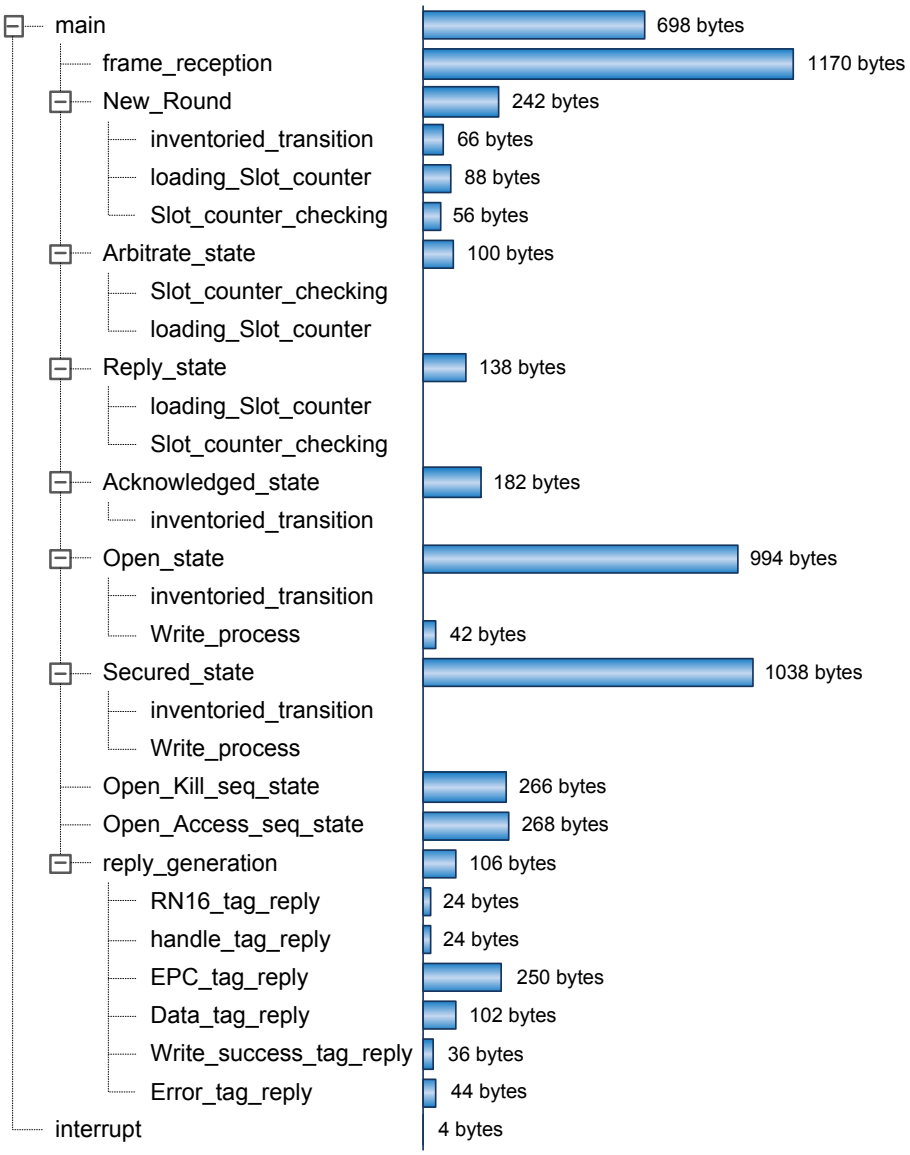


Figure 3-5 Organisation du *firmware* du G2MC avec représentation de la taille de chaque fonction.

Tableau 3-2 Implémentation du G2MC sur FPGA de type Virtex-5 (xc5vlx50t).

Type de logique	Utilisée	Disponible	Pourcentage
Cellule de registres	674	28 800	2%
Cellule de LUT	3475	28 800	12%
DSP	1	48	2%

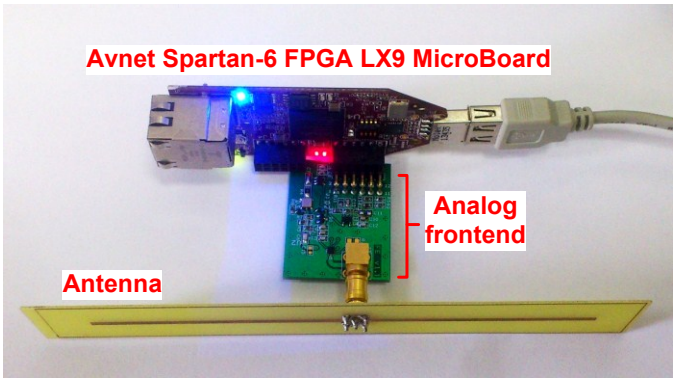


Figure 3-6 Implémentation physique du G2MC.

3.3 G2FSM : tag RFID à base de machine à états finis

Dans le cadre de cette thèse, une deuxième architecture de tag RFID, dénommée G2FSM, a été aussi développée conformément au protocole EPC Gen2. Contrairement au tag G2MC présenté précédemment qui est basé sur un MCU, cette architecture de tag est basé sur une machine à états finis (FSM) [86] [87] donc elle ne contient pas de programme embarqué. Cette architecture a été développée en collaboration avec le laboratoire LCIS - INP Grenoble.

3.3.1 Architecture du G2FSM

La Figure 3-7 montre le diagramme de blocs du G2FSM qui comprend quatre parties principales : "Côté réception", "Côté transmission", "Tag FSM", et les "Périphériques". La fréquence du signal d'horloge utilisé par ce tag est de 2,56 MHz.

La totalité des différents modules de G2FSM a été développée en langage VHDL, la structuration du tag présentée à la Figure 3-7 a donc simplifié l'élaboration du code entier du tag tout en gardant une bonne flexibilité dans l'architecture et dans le code. Le tag résultant intègre évidemment tous les états possibles (*Ready*, *Arbitrate*, *Reply*, *Acknowledged*, *Open*, *Secured* et *Killed*) conformément au protocole EPC Gen2 [65], il intègre aussi toutes les commandes fondamentales du protocole (*Select*, *Query*, *QueryAdjust*, *QueryRep*, *ACK*, *NAK*, *Req_RN*, *Read*, *Write*, *Kill* et *Lock*) en plus de la commande *Access* qui permet d'accéder à l'état sécurisé du tag en délivrant un mot de passe. L'implémentation du G2FSM sur FPGA de type Virtex-5 (xc5vlx50t) a donné une occupation de surface de 1059 cellules (*slices*), ce qui représente 3,67 % de la surface totale de l'FPGA.

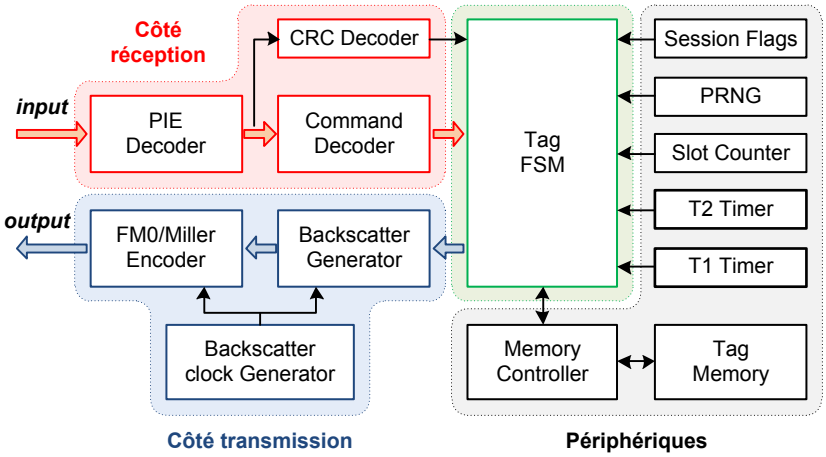


Figure 3-7 Diagramme de blocs du G2FSM.

3.3.2 Côté réception

Cette partie du tag est responsable de la réception et du décodage des trames reçues. L'entrée du tag est connectée au module "PIE decoder" qui se charge de la détection des trames ainsi que de l'extraction de l'information véhiculée par la trame y compris RTcal (calibration R→T), TRcal (calibration T→ R) et les données numériques relatives à la commande transmise par le lecteur. Chaque symbole décodé de la trame est ensuite envoyé aux modules "Commande Decoder" et "CRC Decoder". Cependant, chaque commande reçue est divisée en trois parties : le code de commande, les paramètres et le CRC (CRC-5 ou CRC-16). Le "Command Decoder" reconnaît la commande reçue et identifie sa longueur afin de sauvegarder cette commande et vérifier en conséquence sa validité. À son tour, le "CRC Decoder" vérifie le code CRC de la commande reçue, excepté pour les commandes qui n'incluent pas de code CRC, dans ce cas le "CRC Decoder" demeure désactivé.

3.3.3 Machine d'états

La machine d'états du G2FSM est incorporée dans le module "Tag FSM" (Figure 3-7). Ce module est responsable du traitement de toute commande reçue par le tag. De ce fait, il constitue l'élément principal du G2FSM qui contrôle son fonctionnement intégral. La machine d'états intégrée dans ce module est organisée en sept états principaux comme indiqué précédemment. La Figure 3-8 montre ces états et les différentes transitions possibles entre eux. Chaque état principal contient plusieurs sous-états. Le nombre de sous-états diffère d'un état principal à un autre et atteint en totalité 90 sous-états. À titre d'exemple, la Figure 3-8 montre également 11 sous-états de l'état principal "Ready" et le

diagramme de transitions correspondant. Les quatre sous-états "NewR_0" à "NewR_3" sont concernés par la gestion de l'arrivée d'une nouvelle commande *Query* qui indique le début d'une nouvelle tournée d'inventaire (*new inventory round*).

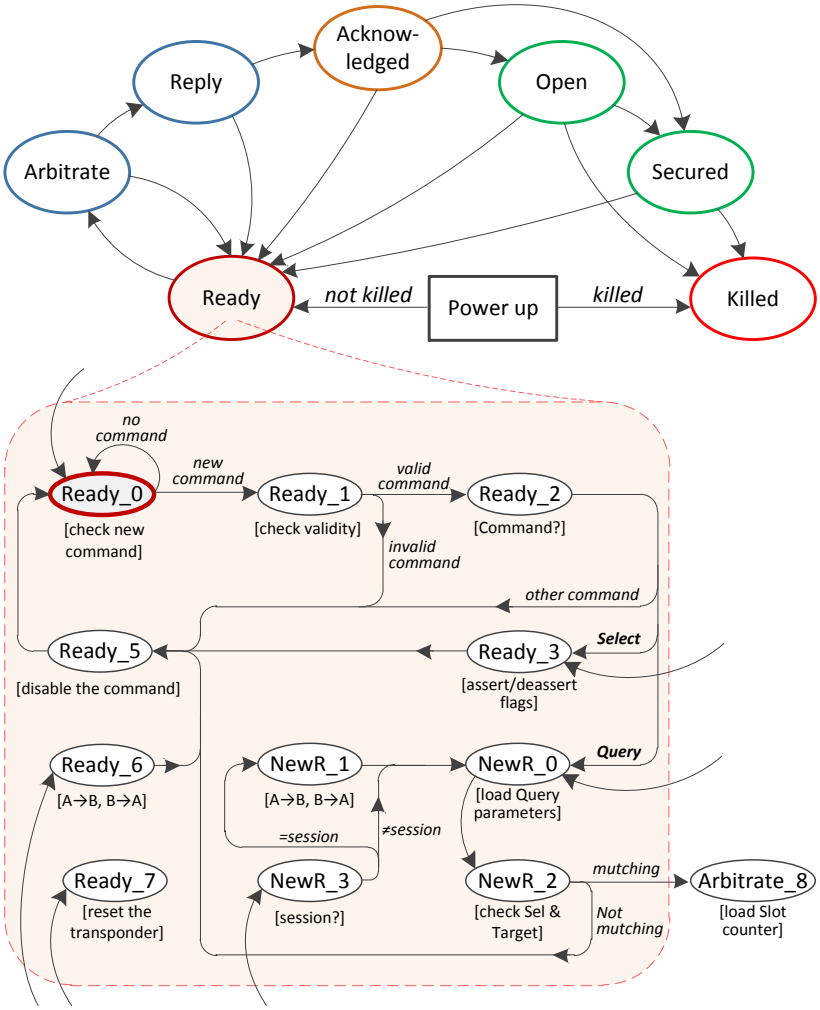


Figure 3-8 Diagramme d'états simplifié du G2FSM.

Selon la description de la machine d'état (FSM) du G2FSM donnée précédemment, il est évident que l'encodage de cette FSM joue un rôle important dans la robustesse et la fiabilité vu qu'il gère tout le fonctionnement du tag. Pour analyser l'impact de l'encodage du FSM sur le tag, les états principaux du FSM sont codés sur trois bits et les sous-états sont codés sur cinq bits, ce qui représente le nombre minimum de bits permettant l'encodage de la FSM complète suivant les spécifications que nous avons définies. Une illustration du fonctionnement de la FSM est donnée à la Figure 3-9, les transitions entre les états du FSM sont contrôlées par une multitude de signaux de 250 bits de longueur.

Selon chaque état, un décodage est effectué par une logique combinatoire pour générer différents signaux qui commandent et contrôlent le fonctionnement du tag.

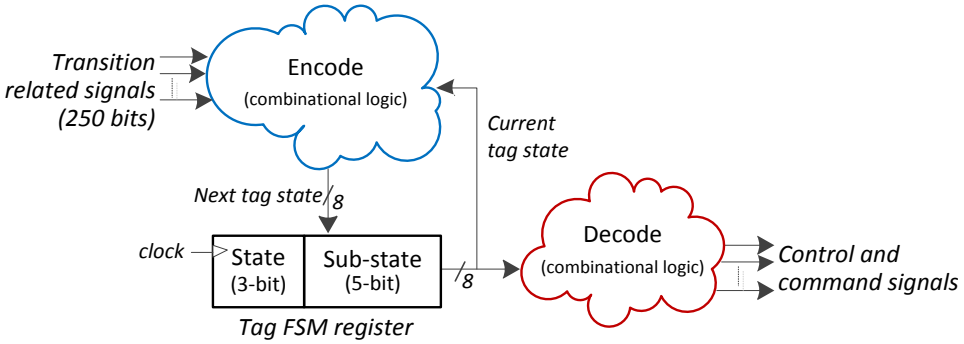


Figure 3-9 Fonctionnement de la machine d'état du G2FSM.

3.3.4 Périphériques

Plusieurs modules sont utilisés par le Tag FSM pour réaliser ses tâches, ces modules sont considérés comme des périphériques du tag (Figure 3-7). Nous y trouvons :

1) "Session flags" : contient les flags des sessions inventoriées en plus du "Selcted flag" (SL flag) [65] ;
2) "PRNG" : c'est un générateur de nombre pseudo-aléatoire de 16 bits ;
3) "Slot Counter" : contient le "Slot counter" du tag et sa fonctionnalité conformément au protocole ;
4) "T1 Timer" : utilisé pour mesurer le temps séparant la transmission du lecteur et la réponse du tag à un acknowledge attendu (T2, voir Figure 2-15) ;
5) "Memory Controller" et "Tag Memory" : la mémoire est composée de "Reserved memory", "EPC memory" (EPC-96bit), "TID memory" et "Used memory" (jusqu'à 128 mots).

3.3.5 Côté transmission

La partie transmission du tag est chargée de la génération de la rétrodiffusion du tag (*tag backscatter*). Comme la fréquence du lien T→R varie selon les paramètres du TRcal délivré par le lecteur, "Backscatter clock Generator" génère l'horloge nécessaire au processus de l'encodage. Le bloc "Backscatter Generator" génère séquentiellement les données de la réponse du tag à l'entrée du "FM0/Miller Encoder" qui, à son tour, produit le préambule et effectue l'encodage des données afin de former la trame de rétrodiffusion.

3.3.6 Implémentations et tests

Nous avons synthétisé le code VHDL du G2FSM à l'aide de deux outils différents : Xilinx ISE 14.7 et Cadence Encounter 14.11. Nous donnons dans l'Annexe B un rapport généré par Encounter et qui détaille l'occupation de surface du G2FSM. Ce rapport est obtenu en utilisant la bibliothèque des cellules standards de TSMC 0.18 µm où G2FSM a occupé 3036 cellules [88]. Le fonctionnement du G2FSM est testé aussi en utilisant deux simulateurs différents : Xilinx ISim et Cadence NCsim. La Figure 3-10 montre les résultats de simulation sous le simulateur ISim où toutes les transitions entre les différents états du tag (de l'état *Ready* jusqu'à l'état *Killed*) sont testées avec succès (conforme au transitions du diagramme d'états, Annexe A).

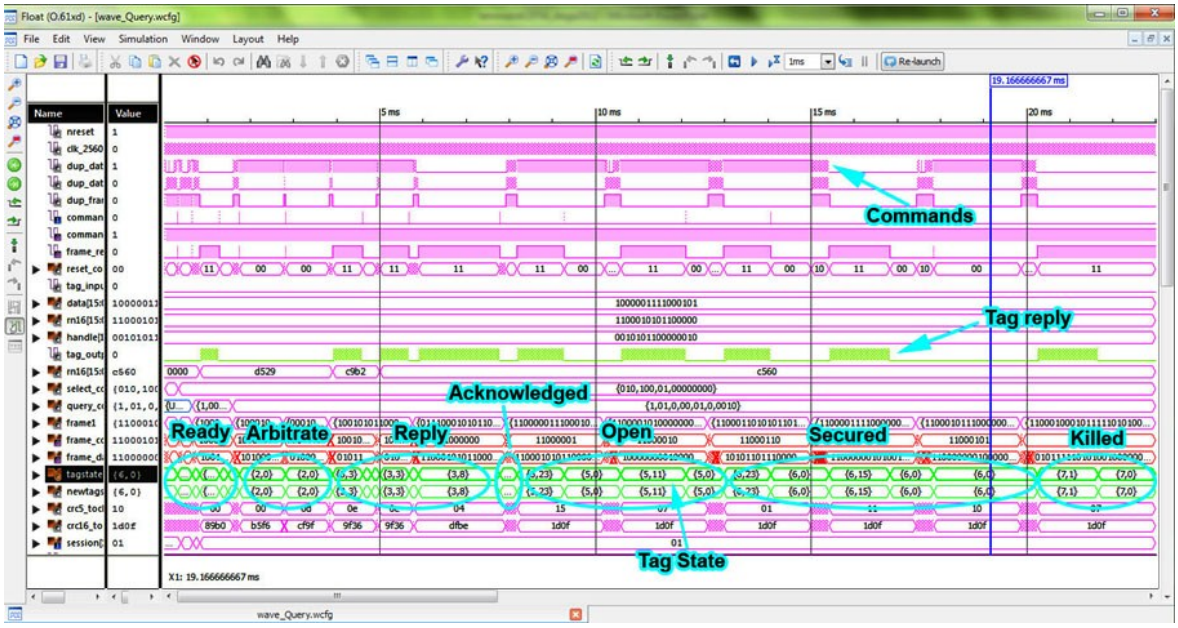


Figure 3-10 Simulation du fonctionnement global du G2FSM.

3.4 G2MC-Reader : lecteur RFID à base de MCIP−C

3.4.1 Développement du G2MC−Reader

Dans le cadre des travaux réalisés, nous avons aussi conçu en VHDL un module d'un lecteur RFID UHF (dénommé G2MC−Reader). En suivant la même méthodologie que pour le développement du G2MC, la conception du lecteur est effectuée par l'élaboration de deux périphériques supplémentaires : le "Transmitter" et le "Receiver" (Figure 3-11).

Trois autres périphériques de MCIP−C sont utilisés : "I/O Port", pour brancher un afficheur LCD ; "Timer0", pour les temporisations ; et "UART", pour établir une interface

de communication série. Plusieurs fonctionnalités du tag sont implémentées en VHDL pour simplifier au maximum la partie logicielle du tag. De ce fait, les fonctions de l'encodage PIE, de génération du préambule, et de calcule du CRC-5/CRC-16 sont toutes implémentées en VHDL et incluses dans le module "Transmitter" ; cela a permis de faciliter considérablement la génération de trames à travers le logiciel du lecteur.

Par ailleurs, l'effort de conception est porté sur la partie réception vu que le processus de décodage est toujours plus compliqué que celui de l'encodage. En effet, seulement le décodage de FM0 est intégré dans la version actuelle du lecteur. L'implémentation de ce dernier a occasionné une occupation de surface de 6 % sur FPGA Virtex-5 XCVLX50T.

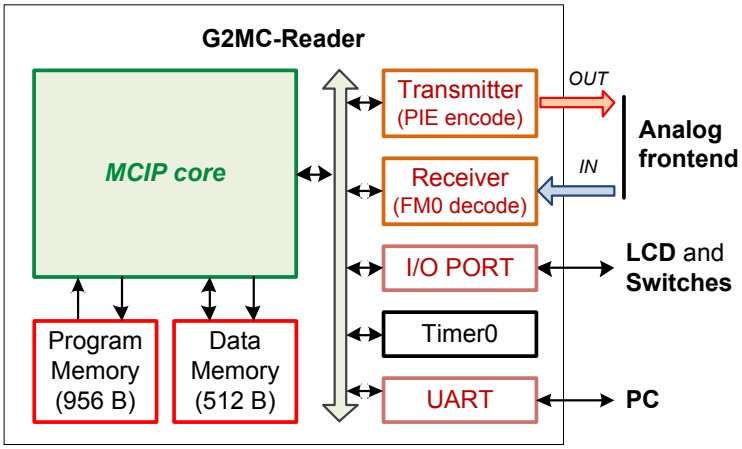


Figure 3-11 Diagramme de blocs du lecteur RFID développé (G2CM−Reader).

3.4.2 Implémentation et test des modules RFID développés

Les deux modules RFID développés à base de MCIP−C (G2MC et G2MC−Reader) ont été validés par simulation et par une implémentation physique sur une carte FPGA. En effet, nous avons employé une seule carte pour implémenter les deux modules en utilisant une communication câblée entre les deux. Après l'implémentation physique des deux modules sur la carte FPGA (Digilent Genesys), la communication entre G2MC−Reader et G2MC a parfaitement fonctionné et toutes les opérations sous-adjacentes telles que l'interrogation et l'identification ont été validées. Le système implémenté constitue une plateforme d'émulation RFID très utile. La Figure 3-12 montre le montage de l'implémentation réalisée, un oscilloscope est employé pour visualiser les signaux de communication entre le circuit du lecteur (G2MC−Reader) et celui du tag (G2MC). La Figure 3-13 montre également les signaux capturés lors de la communication où le lecteur envoie des commandes qui permettent de lire l'EPC du tag et d'y accéder (le mettre dans l'état *Open*).

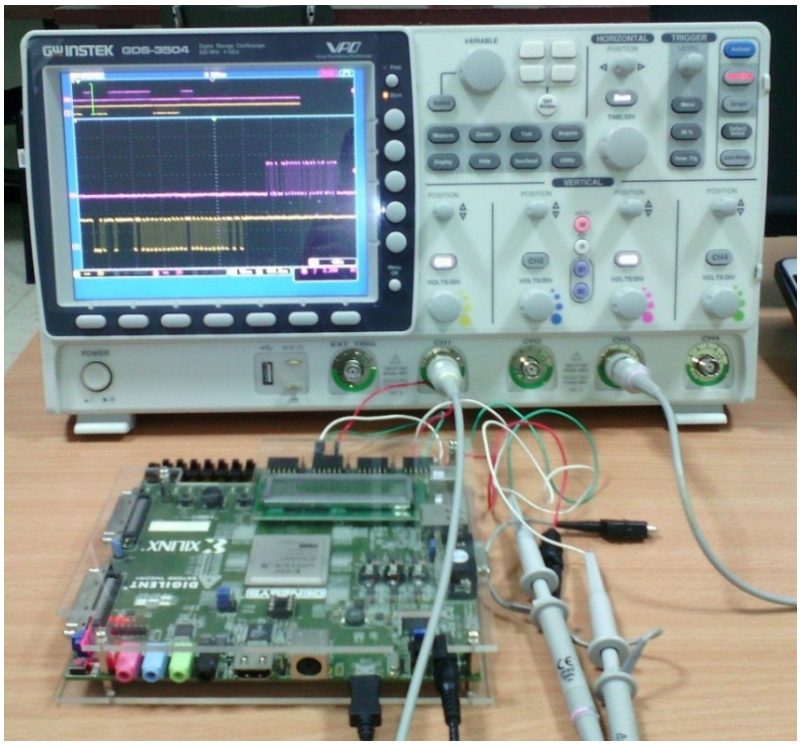


Figure 3-12 Test d'implémentation du G2MC avec G2MC-Reader sur carte FPGA Digilent Genesys.

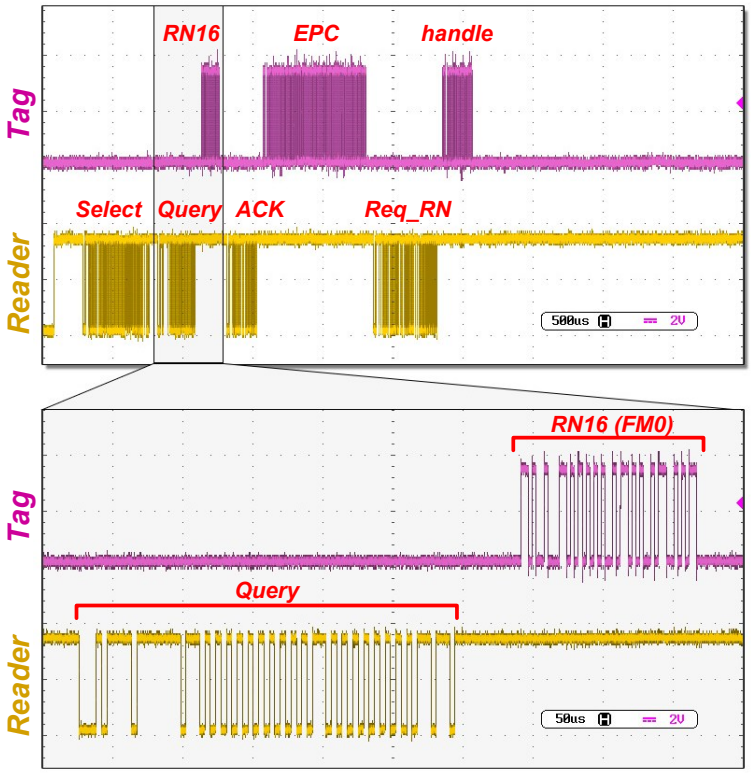


Figure 3-13 Signaux visualisés durant la communication entre le tag (G2MC) et le lecteur (G2MC-Reader).

3.5 Système configurable d'injection de fautes

3.5.1 Méthodes et techniques d'injection de fautes

Les techniques d'injection de fautes (*Fault injection*) se sont largement imposées ces dernières années pour répondre aux différents problèmes qui accompagnent la progression et le déploiement des VLSI. L'injection de fautes constitue un moyen efficace pour évaluer la robustesse, la fiabilité et la sécurité des systèmes liés aux applications critiques (*safety critical applications*) [89]. Parmi les utilisations dans ce cadre, on trouve l'évaluation de la sécurité des systèmes sécurisés (qui intègrent généralement des fonctions de cryptographie) contre les attaques par fautes. Cela est abordé dans plusieurs travaux de recherches et à travers plusieurs moyens expérimentaux [90] [91]. Par ailleurs, l'injection de fautes est bien exploitée dans beaucoup d'autres domaines comme [92] [93] :

- Détermination de la couverture des mécanismes de détection d'erreurs et de récupération,
- Evaluation de l'efficacité des mécanismes de tolérance aux fautes,
- Evaluation de perte de performance,
- Génération de test,
- Diagnostic de fautes.

L'approche d'utilisation de l'injection de fautes pour le diagnostic consiste à établir une base de données autour des effets des fautes injectées sur le système ciblé ; après, cette base de données est exploitée pour identifier quelles sont les fautes qui sont survenues dans le système, et cela à partir des effets observés qui ont été déjà répertoriés dans la base de données.

3.5.1.1 Méthodes d'injection de fautes

Il existe différentes méthodes et techniques d'injection de fautes dans les systèmes électroniques, elles peuvent être classées en trois catégories :

- Injection physique de fautes (*hardware-based*), où le système réel est affecté par des sources physiques externes causant des fautes. Plusieurs moyens peuvent être utilisés comme sources pour entraîner des fautes, par exemple à travers des variations de tension, radiations ioniques, interférences électromagnétiques et inductions optiques.

- Injection de fautes par simulation (*simulation-based*), où un simulateur est utilisé pour simuler des fautes sur un modèle du système ciblé.
- Injection de fautes par émulation (*emulation-based*), où un émulateur de fautes (généralement avec des FPGA) est utilisé pour réaliser des injections de fautes sur le système ciblé.

L'injection physique de fautes produit des résultats réalistes, elle est bien adaptée lorsqu'un prototype du système est déjà disponible. Cependant, cette méthode nécessite généralement des équipements très coûteux et présente des limites dans la contrôlabilité et le suivi des fautes injectées [92]. Par ailleurs, l'injection de fautes à base de simulation est très efficace dans le sens où elle permet de faire des analyses détaillées, de plusieurs types de fautes, sur les systèmes qui sont en phase de conception. Cette méthode peut être exploitée donc lorsqu'un prototype n'est pas disponible. Néanmoins, elle nécessite un temps de simulation très important ce qui la rend moins adaptée aux systèmes complexes.

Aujourd'hui, l'injection de fautes à base d'émulation est considérée comme un moyen qui permet l'accélération des expérimentations d'injection de fautes. L'injection de fautes par émulation utilise des FPGA afin d'émuler le comportement du circuit sous test. Les FPGA modernes incluent des millions de portes logiques et permettent ainsi l'émulation de grands circuits et cela à une vitesse avantageuse. Quelques méthodes d'émulation de fautes ont réussi à atteindre une grande vitesse d'émulation d'environ un million de fautes par seconde [94].

3.5.1.2 Les fautes SET, SEU et MBU

Les fautes de types **SEU** (Single Event Upset) et **SET** (Single Event Transient) représentent des fautes potentielles qui peuvent être les conséquences de l'effet d'une particule lourde (comme la particule α) tombée sur un circuit numérique. La faute SEU est représentée par le changement dans l'état d'un élément de stockage (ex. basculement dans l'état d'une bascule), alors que la faute de type SET est représentée par une impulsion transitoire qui se produit dans les cellules combinatoires. Un intérêt grandissant pour ces deux types de fautes a émergé suite au progrès de la technologie des semi-conducteurs, impliquant des transistors plus petits et une alimentation électrique réduite. Actuellement, l'estimation de la sensibilité aux SEU et SET est une tâche cruciale dans la conception des circuits critiques [95]. Un autre type de faute qui semble avoir le même intérêt est la **MBU** (Multiple Bit Upsets) qui est caractérisée par l'occurrence d'une multitude de fautes SEU

en même temps, ou par la propagation d'une faute SET à plusieurs éléments de mémoire à travers plusieurs chemins combinatoires [96].

3.5.2 Développement du système d'injection de fautes configurable

Dans le cadre des travaux de cette thèse, nous avons développé un système d'injection de fautes afin de satisfaire nos besoins expérimentaux qui impliquent la réalisation de plusieurs tâches d'injection de fautes. Le système développé est destiné à la pratique d'injection de fautes par émulation en utilisant des circuits FPGA. La technique sur laquelle repose notre système est la technique d'injection de fautes à base de modèle HDL [97] [98]. Cette technique consiste à ajouter au modèle VHDL (ou autre langage HDL) du circuit concerné des blocs additionnels qui permettent l'injection de fautes moyennant une sélection de signaux. La méthode d'injection de fautes à base de modèle HDL présente plusieurs avantages parmi lesquels :

- sa simplicité, où elle est accessible pour tous les intervenants et ne nécessite pas d'outils ou matériel spécifiques ;
- elle permet de produire plusieurs types de fautes ;
- elle permet de faire de la simulation de fautes aussi bien que de l'émulation ;
- elle assure une bonne contrôlabilité des fautes injectées.

Par contre, cette méthode présente également certains inconvénients tels que la nécessité de modification du modèle VHDL ; et la non garantie de la similitude des résultats des injections avec les comportements du circuit réel. Pour garder des résultats d'injection valides pour l'implémentation réelle du circuit, il faut s'assurer que le synthétiseur du code VHDL est instruit de faire une translation directe du code VHDL sans simplification. Dans tous les cas, les résultats d'injection de fautes au niveau VHDL restent valides pour une évaluation primitive d'un circuit en cours de développement [98].

La Figure 3-14 montre le principe d'utilisation du système d'injection de fautes développé. Ce système peut être ajouté et instancié comme un bloc dans l'architecture du circuit ciblé. Les signaux concernés par l'injection de fautes (*Data_in*) sont connectés au système d'injection. Une fois activé, le système injecte des fautes dans les bits concernés du vecteur *Data_out* qui représente les signaux défectueux.

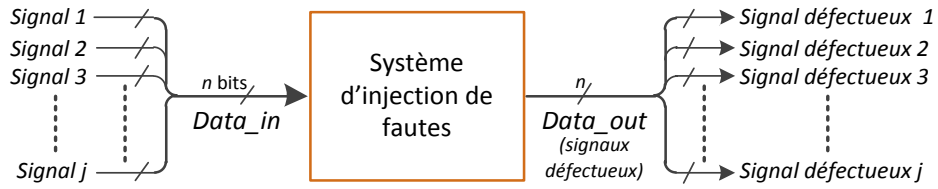


Figure 3-14 Principe d'utilisation du système d'injection de fautes dans un circuit.

L'élaboration du système d'injection de fautes est effectuée en utilisant le VHDL. Le système résultant est flexible et configurable afin de permettre de supporter plusieurs types de fautes, différentes tailles de signaux et différents modes d'injection (comme le mode aléatoire). Les caractéristiques de ce système sont comme suit :

- Types de fautes supportées : faute **permanente** (faute de collage à '1' ou à '0'), faute **transitoire** (**SET**) et faute **SEU**.
- Possibilité d'injecter plusieurs fautes en même temps (faute permanente multiple, SET multiple (**MBT** − Multiple Bit Transients [99]), et SEU multiple (**MBU**)).
- Possibilité de combiner plusieurs types de fautes en même temps.
- Possibilité de réaliser des injections aléatoires dans l'espace et/ou dans le temps.
- Possibilité de réaliser des injections ciblées dans l'espace (sur des signaux précis) et/ou dans le temps (dans des cycles précis).

La Figure 3-15 illustre le diagramme de blocs du système développé. Ce dernier est composé de trois parties principales : "**Injection time controller**" (Contrôleur de temps d'injection), "**Injection space controller**" (Contrôleur d'espace d'injection), et "**Injection logic**" (Logique d'injection).

Le bloc "Injection time controller" spécifie **quand** les fautes seront injectées ; le bloc "Injection space controller" spécifie **où** les fautes seront injectées ; tandis que le bloc "Injection logic" **assume l'injection** de faute au niveau des signaux concernés. Plus de détails sur le fonctionnement de ces trois blocs sont donnés dans les prochaines sections.

3.5.2.1 Contrôleur de temps d'injection (ITC)

Comme décrit précédemment, ce bloc indique le moment de l'injection de la faute en délivrant le signal *injection_enable* (Figure 3-15). La Figure 3-16 illustre la structure générale d'ITC, on constate qu'il intègre deux **comparateurs** et deux **LFSR** (Linear Feedback Shift Register). Ces derniers sont utilisés avec le premier comparateur (en haut)

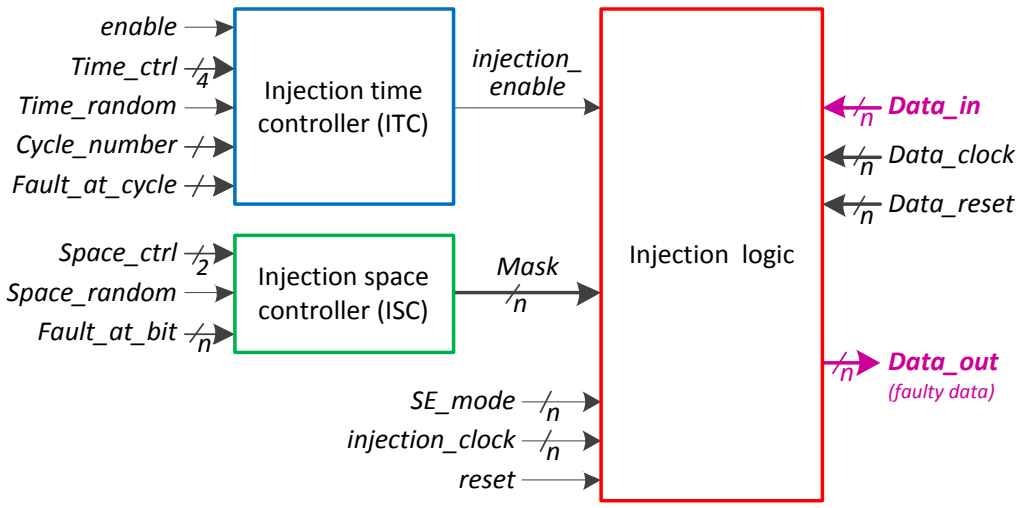


Figure 3-15 Diagramme de blocs du système d'injection de fautes.

pour implémenter la fonction *'time-random'* (temps-aléatoire) conformément à l'approche présentée dans [100]. Les LFSR sont utilisés pour la génération de nombres pseudo-aléatoires. L'emploi d'autres mécanismes de génération de nombre aléatoire est possible mais l'utilisation des LFSR constitue la méthode la plus simple. La fonction *time-random* est activée quand le paramètre *Time_random* = '1' ; dans ce cas, le paramètre *Time_ctrl* (de 4 bits) définit le taux des injections. Le *Time_ctrl* précise cependant le nombre de bits qui doivent être comparés entre les deux LFSR, l'injection est activée dans le cas où les bits comparés entre les deux LFSR sont égaux.

Selon les deux paramètres *Time_random* et *Time_ctrl*, l'ITC peut fonctionner en trois modes différents : (1) Mode de **faute permanente** ; (2) Mode de **temps-aléatoire** ; et (3) Mode d'**injection au cycle**.

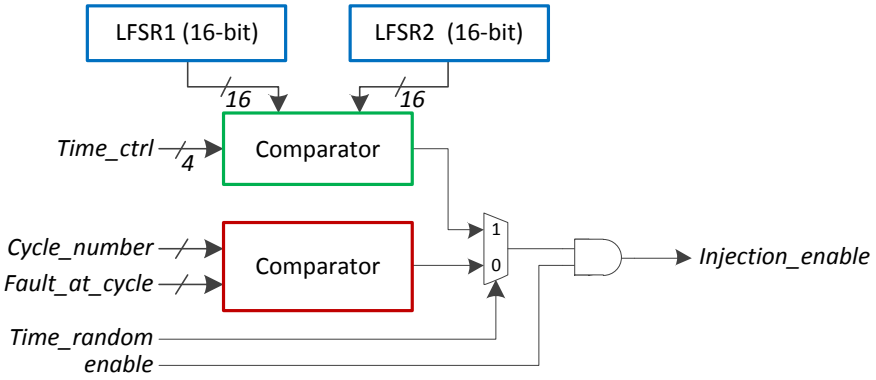


Figure 3-16 Diagramme de blocs du contrôleur de temps d'injection (ITC).

3.5.2.1.1 Mode de faute permanente

Ce mode est sélectionné quand le paramètre *Time_random* = '1' et le paramètre *Time_ctrl* = '0000' (Figure 3-16). Cette dernière valeur instruit le premier comparateur de ne faire aucune comparaison entre les deux LFSR (comparaison de '0' bit), c'est-à-dire que l'injection est toujours activée (taux d'injection en fonction du temps est de 100 %) en caractérisant ainsi la faute permanente. Dans ce mode, le signal *Injection_enable* (signal d'activation de l'injection) est donc toujours égal à '1' (injection activée) dans le cas où, bien sûr, le signal *enable* est égal à '1'. Ce dernier représente un signal externe utilisé pour l'activation globale de l'injection.

3.5.2.1.2 Mode de temps-aléatoire

Ce mode est sélectionné quand *Time_random* = '1' et *Time_ctrl* a une valeur entre '0001' et '1111'. Le fonctionnement de ce mode est basé sur l'utilisation des deux registres LFSR. Pour mieux comprendre cette approche, nous donnons ici quelques détails sur le principe de fonctionnement des LFSR.

Un LFSR est un **registre à décalage à rétroaction linéaire**. La Figure 3-17 illustre un LFSR à *n* bits, la boucle de rétroaction effectue une combinaison de XOR des différents bits du registre avant d'introduire le résultat dans la première bascule. Cette méthode permet de générer des séquences binaires pseudo-aléatoires. Les séquences générées dépendent de l'état initial du registre ainsi que du choix des bits participant à la boucle de rétroaction qui sont définis par l'ensemble des valeurs de a_i. Ces valeurs représentent ce qu'on appelle le **polynôme de rétroaction**. Certains choix de ce polynôme permettent la génération d'un maximum de séquences (2^n-1 séquences pour un registre de *n* bits) [60]. La Figure 3-18 donne à titre d'exemple un cycle de 7 différentes séquences générées par un LFSR à 3 bits de polynôme égale à x^3+x^2+1 (a_1 = '0', a_2 = '1' et a_3 = '1').

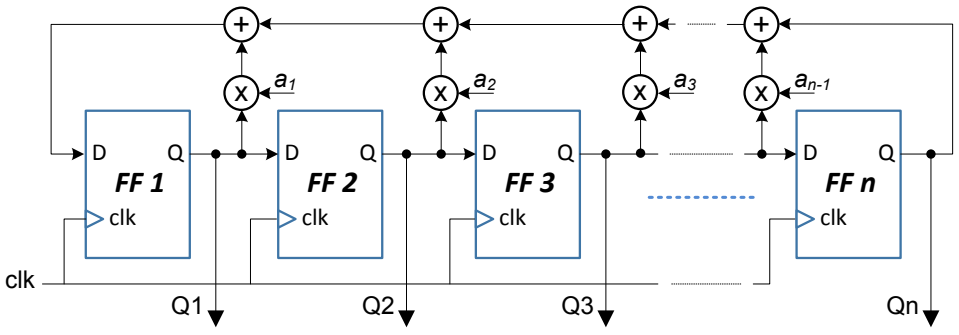
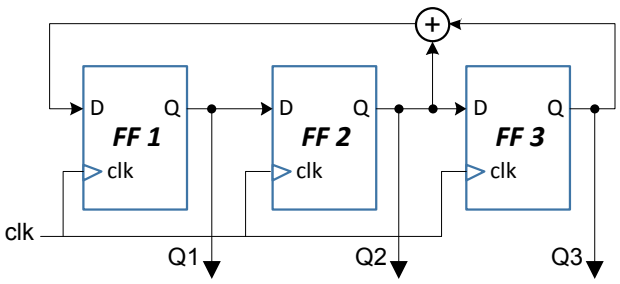


Figure 3-17 Représentation d'un LFSR à *n* bits.



t	$Q_1 Q_2 Q_3$ (t)	$Q_1 Q_2 Q_3$ (t+1)
1	1 1 1	0 1 1
2	0 1 1	0 0 1
3	0 0 1	1 0 0
4	1 0 0	0 1 0
5	0 1 0	1 0 1
6	1 0 1	1 1 0
7	1 1 0	1 1 1
8	1 1 1	0 1 1

Figure 3-18 Exemple de compteur pseudo-aléatoire à 3 bits.

La Figure 3-19 donne l'organigramme du mécanisme d'activation de l'injection en mode 'temps-aléatoire'. Pour plus d'efficacité, il est recommandé d'utiliser deux polynômes différents pour les deux LFSR. La valeur de Time_ctrl configure le nombre de bits des deux séquences qui doivent être comparées, l'injection est alors activée dans le cas où les bits comparés sont égaux comme le montre l'exemple de la Figure 3-19. La comparaison entre les deux séquences des LFSR n'est pas effectuée de façon directe (bit1 avec bit1, bit2 avec bit2… etc.) mais l'ordre des bits est mélangé (ex. bit1 avec bit4, bit2 avec bit13, bit3 avec bit1… etc.) pour avoir plus d'efficacité. Le taux des injections aléatoires (TIa) dépend de Time_ctrl et défini par l'équation (3.1). Dans le mécanisme d'injection pseudo-aléatoire utilisé, le taux d'injection diffère légèrement de celui des injections aléatoires. Les mesures effectuées par simulation du taux d'injection pseudo-

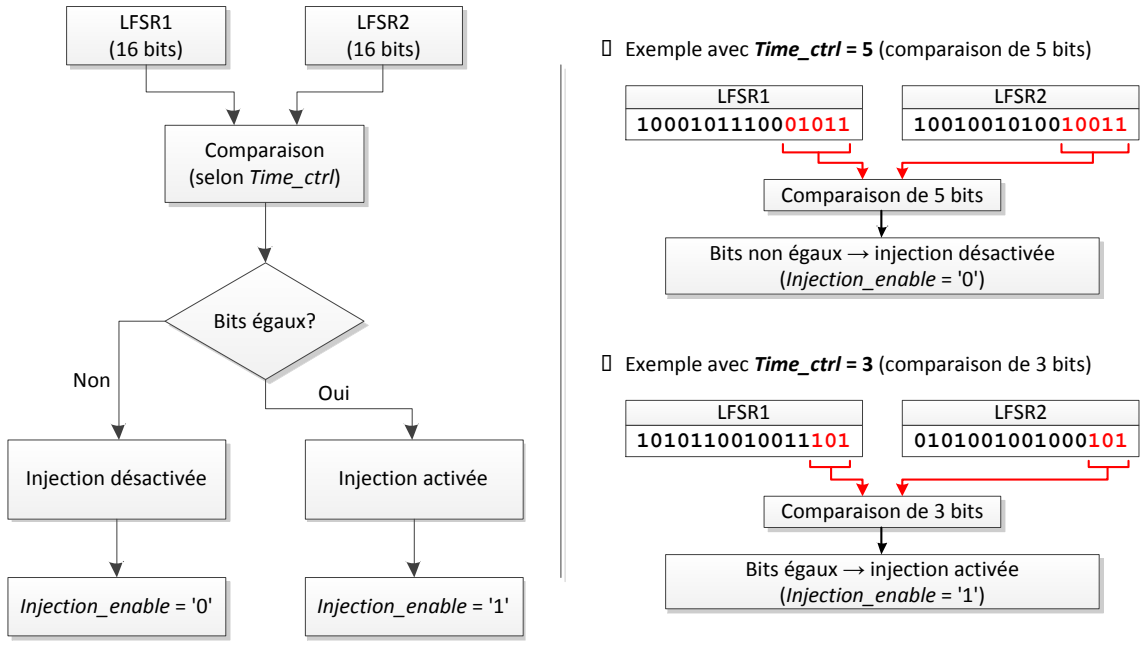


Figure 3-19 Organigramme de fonctionnement du mode 'temps-aléatoire' (avec des exemples).

aléatoire (TIp-a) sont donnés par le Tableau 3-3 et la Figure 3-20 qui présentent également le taux d'injection aléatoire théorique. Ces résultats dépendent des caractéristiques des LFSR.

$$TI_a = \frac{1}{2^{(Time\_ctrl)}} \quad (3.1)$$

Tableau 3-3 Taux d'injection aléatoire (TI_a) et pseudo-aléatoire ($TI_{p\text{-}a}$) en fonction de *Time_ctrl*.

Time_ctrl	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
TI_a (%)	100	50	25	12,50	6,25	3,12	1,56	0,78	0,39	0,19	0,098	0,049	0,024	0,012	0,006	0,003
$TI_{p\text{-}a}$ (%)	100	62	34	17,89	9,16	4,69	2,37	1,13	0,57	0,28	0,15	0,064	0,038	0,015	0,009	0,006

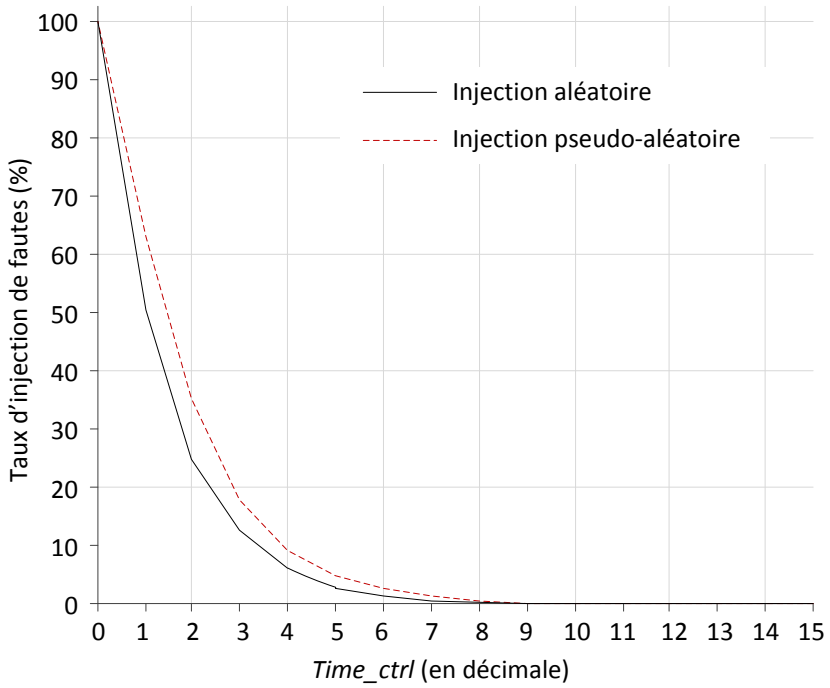


Figure 3-20 Taux d'injection de fautes en fonction du *Time_ctrl*.

3.5.2.1.3 Mode d'injection au cycle

Ce mode est sélectionné quand le mode aléatoire est désactivé (*Time_random* = '0'), il permet d'activer l'injection dans un cycle précis sélectionné par le vecteur *Fault_at_cycle* (Figure 3-16), le numéro du cycle en cours est donné en parallèle par le vecteur *Cycle_number*. Le deuxième comparateur de l'ITC active l'injection dans le cas où *Fault_at_cycle* = *Cycle_number*. Ce mode permet de réaliser des injections exhaustives dans le temps en injectant une faute dans chaque cycle d'horloge durant plusieurs cycles (une activation par cycle).

3.5.2.2 Contrôleur d'espace d'injection (ISC)

Ce module spécifie le lieu où les fautes doivent être injectées, il délivre donc le masque (*Mask*) qui indique les bits concernés par l'injection. La Figure 3-21 illustre le diagramme de blocs d'ISC, il fonctionne également selon deux modes différents : (2) Mode de **faute au niveau bit** (*fault at bit*) et (2) Mode d'**espace-aléatoire**.

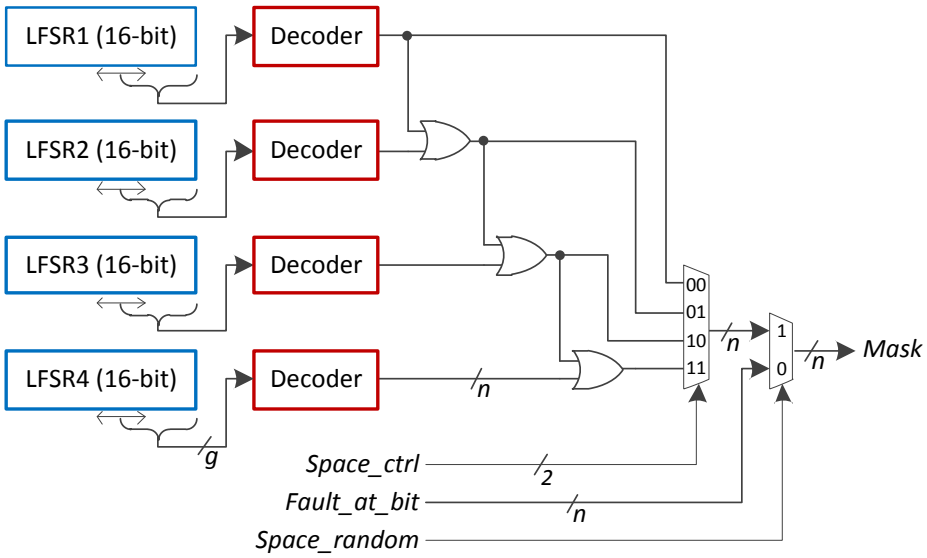


Figure 3-21 Diagramme de blocs du contrôleur d'espace d'injection (ISC).

3.5.2.2.1 Mode de faute au niveau bit

Ce mode est sélectionné quand *Space_random* = '0'. Dans ce cas, le masque (*Mask*) reçoit la valeur définie par le vecteur *Fault_at_bit* (*Mask* = *Fault_at_bit*) qui précise les bits auxquels les fautes seront injectées.

3.5.2.2.2 Mode d'espace aléatoire

Ce mode est choisi quand *Space_random* = '1'. Il permet de désigner aléatoirement des bits pour l'injection. Cependant, il est possible d'injecter une ou plusieurs fautes en même temps grâce au mécanisme présenté par la Figure 3-21. Ce mécanisme contient plusieurs étages de LFSR-Décodeur, chaque étage active un seul bit parmi un vecteur de *n* bits d'une manière pseudo-aléatoire. Le paramètre *Space_ctrl* est utilisé pour définir le nombre maximal de bits à activer en même temps. Dans le cas présenté dans la Figure 3-21 où *Space_ctrl* possède 2 bits de largeur, il est possible d'activer jusqu'à 4 bits en même temps.

La Figure 3-22 illustre un exemple (obtenu par simulation) du nombre de bits activés dans un masque de 25 bits de largeur et pour différentes valeurs de *Space_ctrl*. Le nombre de bits utilisés de chaque LFSR est égal à 5 bits ($g = 5$), ce qui permet de supporter un masque de 32 bits ($2^5 = 32$) de largeur au maximum. Chaque décodeur active un seul bit dans le masque selon la valeur des 5 bits du LFSR correspondant, il y aura donc des valeurs (de 25 jusqu'à 31) qui ne donnent aucune activation par les décodeurs. En plus, certaines valeurs de deux LFSR ou plus peuvent être identiques par coïncidence, ce qui génèrera la même activation. Cela explique la raison pour laquelle le nombre de bits activés varie d'un cycle d'horloge à un autre pour la même valeur de *Space_ctrl* (Figure 3-22).

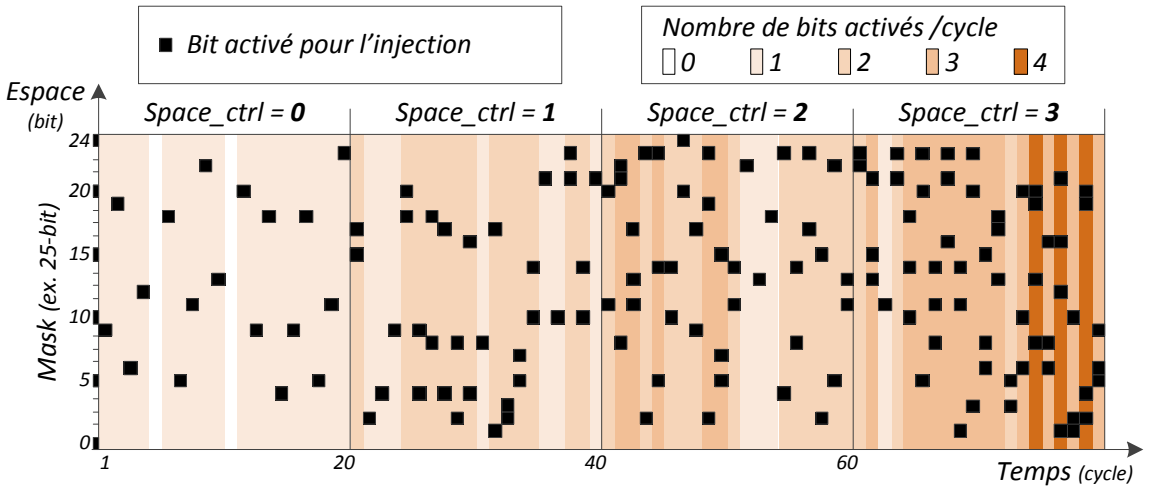


Figure 3-22 Nombre de bits activés par cycle d'horloge en fonction du *Space_ctrl*.

3.5.2.3 Logique d'injection

Cette partie est concernée par la réalisation des injections de fautes en fonction du masque (*Mask*) et du signal d'activation (*Injection_enable*) délivrés par les deux autres blocs (ITC et ISC). La Figure 3-23 illustre la structure de la logique d'injection, elle est basée sur l'utilisation des tranches d'injection de faute configurables (CFIS - Configurable Fault Injection Slice). Nous avons conçu la CFIS d'une manière qui permet d'injecter plusieurs types de fautes à la fois. Plusieurs CFIS sont insérées selon le besoin pour supporter le vecteur de signaux destiné à l'injection de fautes (*Data_in*), chaque bit de ce vecteur étant relié à un CFIS qui délivre ainsi le bit défectueux qu'il lui correspond. Le masque est utilisé pour sélectionner les bits concernés par l'injection des défauts comme le montre la Figure 3-23.

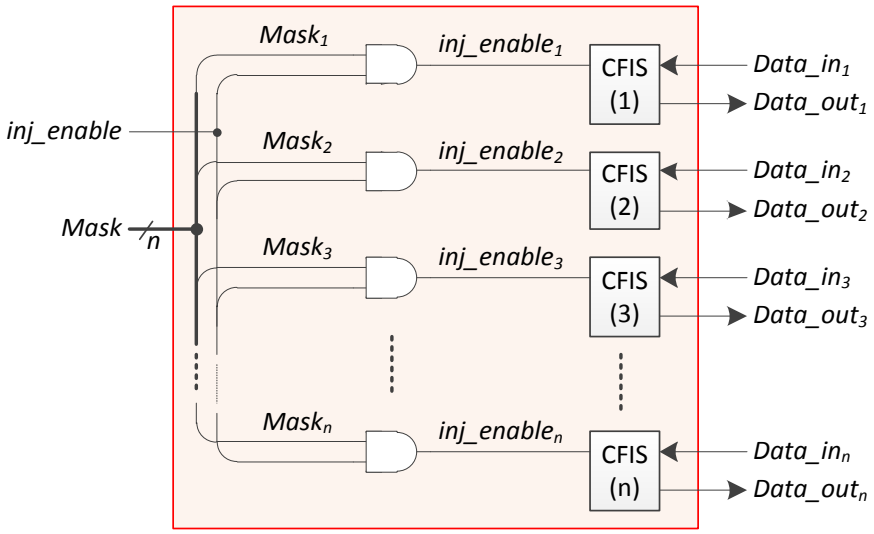


Figure 3-23 Structure de la logique d'injection de fautes.

La CFIS est composée de deux bascules et d'une logique combinatoire, comme le montre la Figure 3-24. La bascule FF1 (la bascule principale) est utilisée pour activer l'injection de faute pour au moins une période de l'horloge d'injection (*inj_clock*), elle est donc mise à '1' lorsque le signal d'entrée *inj_enable* est activé. L'utilisation de cette bascule est essentielle, car elle isole la logique combinatoire située dans le contrôleur de temps d'injection (ITC) et qui délivre le signal *inj_enable*. Ainsi, pour chaque activation d'injection par ce contrôleur, la faute est injectée dans le cycle suivant de l'horloge. La CFIS peut être configurée pour injecter soit une faute de collage (à '0' ou à '1'), une faute SET ou une faute SEU.

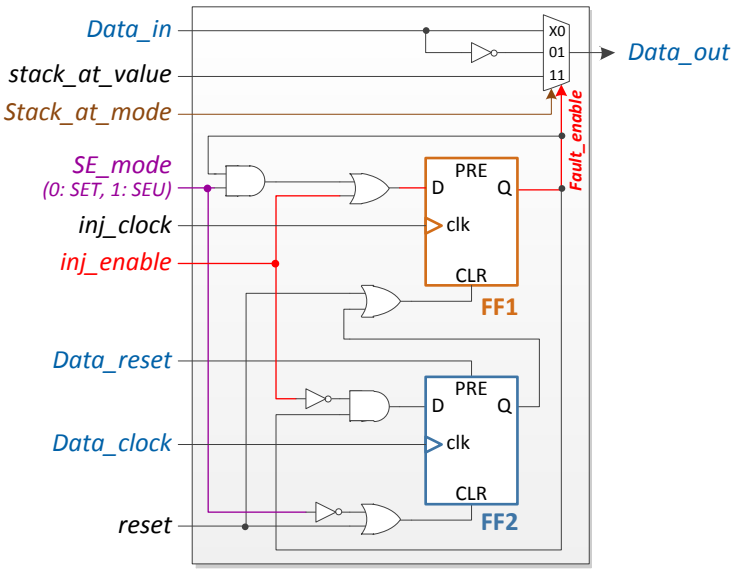


Figure 3-24 Composition du CFIS (*Configurable Fault Injection Slice*).

3.5.2.3.1 Mode de collage

Ce mode est sélectionné lorsque *Stack_at_mode* = '1' et *SE_mode* = '0', il permet de réaliser une faute de collage à '0' ou de collage à '1' selon le besoin. La valeur de collage est donnée par le signal *stack_at_value*. Quand l'injection de faute est activée, *Data_out* reçoit la valeur de *stack_at_value* indépendamment de la valeur de *Data_in*. La Figure 3-25 montre comment utiliser la CFIS dans le mode de collage.

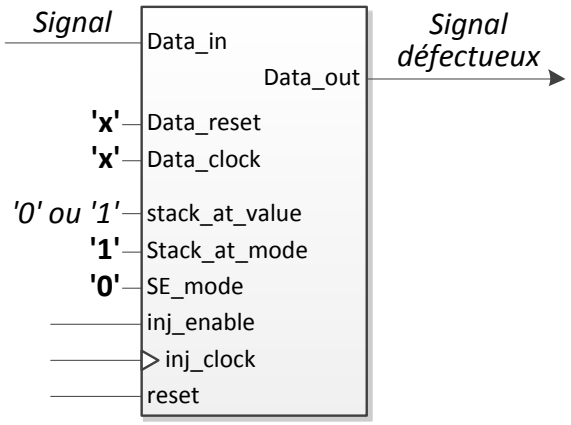


Figure 3-25 Utilisation du CFIS dans le mode de collage.

3.5.2.3.2 Mode SET

Le mode SET est sélectionné lorsque *Stack_at_mode* = '0' et *SE_mode* = '0'. Dans ce cas, la bascule FF1 active l'injection de faute dans *Data_in* (inversion de bit) durant un cycle d'horloge et pour chaque activation du signal *inj_enable*. Cela permet de générer une SET (faute transitoire) d'une durée d'une période de l'horloge *inj_clock*. La bascule FF2 n'a pas d'effet dans ce mode, elle est utilisée seulement dans le mode SEU. La Figure 3-26 illustre l'utilisation du CFIS dans le mode SET.

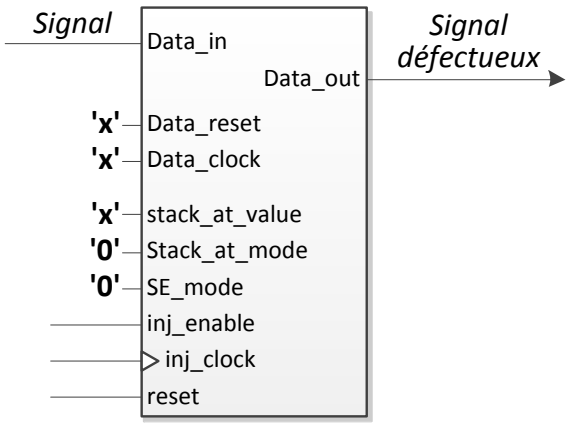


Figure 3-26 Utilisation du CFIS dans le mode SET.

3.5.2.3.3 Mode SEU

Le mode SEU est sélectionné quand *Stack_at_mode* = '0' et *SE_mode* = '1'. Dans ce mode aussi, pour chaque activation du signal *inj_enable*, FF1 active l'injection de faute (inversion de bit). Par contre la deuxième bascule (FF2) est exploitée seulement dans ce mode pour réinitialiser la bascule FF1, ce qui désactive l'injection de faute en cours. Cette action survient lorsque l'un des deux évènements suivants survient : (1) lors de l'arrivé du prochain front de *Data_clock* ou (2) lorsque le signal *Data_reset* est activé, ce qui permet de réaliser une faute SEU. Pour utiliser la CFIS dans le cas de bascule équipée d'un signal d'activation d'horloge (*clock enable*), il est possible d'ajouter ce signal à la bascule FF2 ou même utiliser le *clock gating* comme une autre alternative. En mode SEU, il est nécessaire de délivrer les signaux d'horloge et de reset de la bascule correspondante comme il est illustré dans la Figure 3-27.

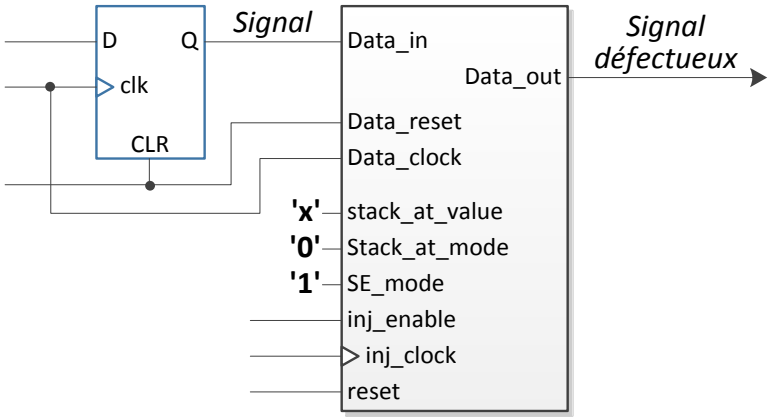


Figure 3-27 Utilisation du CFIS dans le mode SEU.

3.6 Conclusion

Dans ce chapitre, nous avons présenté les différents systèmes développés et utilisés dans le cadre des travaux de cette thèse. Ces systèmes sont développés en VHDL et comprennent un microcontrôleur configurable (MCIP−C), un circuit de tag RFID à base de microcontrôleur (G2MC), un circuit de tag RFID à base de machine d'état (G2FSM), un circuit de lecteur RFID (G2MC−Reader), et un système d'injection de fautes. Tous ces composants ont été utilisés dans le développement des approches proposées dans cette thèse à l'instar d'autres travaux antérieurs. Ainsi, MCIP−C a été utilisé dans des travaux de fin d'études de Master notamment dans [101] [102] [103]. G2FSM a été utilisé également dans un travail de Master pour une implémentation sur ASIC [88]. De plus, G2FSM a été

utilisé dans une thèse de Doctorat [104] et dans d'autres travaux connexes [105] [106] [87] [107] [108]. De la même manière, les composants développés présentés dans ce chapitre peuvent bien être exploités dans de futurs travaux de recherche et de développement.

Chapitre 4 Analyse d'injection de fautes sur tag RFID

Comme les tags sont des éléments électroniques, ils sont de nature vulnérables aux perturbations externes ainsi qu'aux attaques par injection de fautes (*fault attacks*) [4]. À cet effet, l'étude de la tolérance aux fautes des tags et l'évaluation de leur sécurité sont des tâches essentielles dans plusieurs cas d'application. Les techniques d'analyse des effets de fautes sur les circuits électroniques représentent ainsi un outil indispensable qui permet d'évaluer la robustesse des tags et d'identifier les parties les plus vulnérables qui le composent.

Le but du travail présenté dans ce chapitre est d'effectuer une analyse sur les effets des fautes SEU et SET sur les tags RFID afin d'évaluer la robustesse, identifier les fautes ayant un impact sur la sécurité et discerner les parties les plus vulnérables du tag. En effet, plus de latitude de changements avec moins de compromis est possible dans les étapes de conception antérieures. L'amélioration de la sécurité devrait résulter de l'amélioration et de la perfection de la conception du tag, ou à travers des actions de contre-mesures appropriées sur les parties les plus vulnérables aux fautes dans le circuit du tag. Faire une étude autour des effets des fautes sur les tags permet aussi d'améliorer le diagnostic de fautes dans les tags en particulier, et dans les systèmes RFID en général.

Dans le domaine de la RFID, lorsqu'on considère une seule communication entre un lecteur et plusieurs tags durant un tour d'inventaire, une infinité de scénarios de communication est possible du point de vue d'échange des commandes-réponses entre le lecteur et les tags. En effet, pour un scénario donné, plusieurs paramètres peuvent être supportés dont le calibrage lecteur-tag (RTcal), le calibrage tag-lecteur (TRcal) et le type de codage utilisé dans la communication descendante (T→R, encodage FM0 ou Miller).

Ainsi, l'injection de fautes basée sur l'émulation constitue un moyen efficace qui permet d'effectuer des expérimentations d'injection de fautes étendues couvrant plusieurs scénarios et impliquant plusieurs compositions de paramètres.

Dans ce chapitre, nous présentons une nouvelle plateforme d'émulation de fautes pour les systèmes RFID. Cette plateforme sert à effectuer des analyses d'injection de fautes de types SEU et SET sur les tags UHF compatibles au standard EPC Gen2. Cependant, cette approche est également applicable à d'autres standards RFID. Nous présentons aussi les résultats des expérimentations d'injection de fautes que nous avons menées sur le circuit du tag G2FSM et qui ont été valorisés par une publication [109].

Il est important de rappeler que le but de l'injection de fautes basée sur l'émulation est d'évaluer la sensibilité aux fautes SEU d'un design d'ASIC (Application-Specific Integrated Circuit) et non pas d'un design pour FPGA. Les effets des SEU apparaissent dans les éléments mémoire des circuits, qui sont les mêmes dans l'ASIC et dans le prototype FPGA [94] [110]. En ce qui concerne les expérimentations des SET que nous avons menées et qui sont présentées dans ce chapitre, nous avons considéré seulement les SET qui affectent les sorties des bascules du tag préservant ainsi des résultats valables pour l'ASIC. En outre, nous avons considéré que les SET appliqués durant suffisamment de temps de manière à affecter le circuit, sans prendre en compte les différents aspects qui accompagnent la faute SET comme détaillé et discuté dans plusieurs travaux de recherche [111] [112].

4.1 Travaux rattachés et similaires

L'analyse de fautes dans les systèmes électroniques a toujours été considérée comme un domaine majeur comme l'indique l'énorme quantité de travaux et de discussions couvrant ce sujet durant les décennies qui ont suivi l'évolution des technologies de l'électronique. Cependant, un nombre très limité de travaux ont couvert les systèmes RFID dans ce domaine. La plupart de ces travaux se sont concentrés sur la pratique des attaques par canal latéral (*side-channel attacks*) visant les tags RFID à travers plusieurs moyens incluant les interférences électromagnétiques et les inductions optiques [4] [113] [5] [91], tandis que d'autres ont pratiqué des techniques d'injection de faute à base de contact [114] [4]. Les deux méthodes ont démontré que les attaques par injection de fautes peuvent effectivement briser les fonctions de cryptage embarquées et induire plusieurs types d'erreurs dans les tags RFID. Généralement, ces approches évaluent la sécurité des tags préfabriqués contre

les attaques par fautes, mais elles ne permettent pas de discerner ou d'identifier les éléments faibles dans les architectures de ces tags ni de mener des améliorations guidées dans ces architectures. Il est important d'évaluer les tags à l'étape de conception afin de garantir des conceptions sécurisées avant d'entamer l'étape de fabrication comme nous le considérons à travers l'analyse d'injection de fautes présentée dans ce chapitre.

Une méthode d'évaluation de la tolérance aux fautes des tags RFID est présentée dans [105] où les auteurs ont utilisé un FPGA pour implémenter le tag à expérimenter avec un mécanisme d'injection de fautes en plus d'un *front-end* analogique et un lecteur externe. Nous avons utilisé des méthodes identiques dans [87] [86] pour expérimenter des injections de fautes sur le tag G2FSM. Cependant, ces approches fournissent des informations limitées sur les effets des fautes injectés comme elles ne permettent pas de faire des analyses approfondies. Un autre travail présenté dans [115] propose également une plate-forme FPGA (DemoTag) pour tester les conceptions matérielles numériques des tags RFID et analyser l'impact des attaques par fautes sur ces conceptions sans pour autant introduire aucun moyen pour concrétiser ce travail.

Dans notre approche, nous avons utilisé un seul FPGA pour implémenter le tag à étudier (G2FSM) avec le lecteur correspondant (G2MC-Reader) pour obtenir un système RFID complet ; donc ce système est prêt pour une gestion et un suivi efficaces de ces scénarios de communication. Par conséquent, le comportement du système RFID est contrôlé et surveillé d'une manière approfondie.

4.2 Emulation de fautes sur tag RFID

Le processus d'analyse de fautes (*fault analysis*) consiste à vérifier la réponse du circuit à analyser en présence de fautes, en comparant les comportements de ce circuit avec et sans la présence de fautes. La Figure 4-1 présente le système d'émulation de fautes basé sur FPGA que nous avons développé [109]. Les fautes sont injectées dans le circuit du tag expérimenté pendant son fonctionnement, et les effets de ces fautes sont observés et analysés. Un deuxième circuit de tag, identique au premier, est maintenu sans fautes (*fault-free*) pour fournir des moyens de comparaison comme recommandé dans les techniques de test [13]. Le lecteur ajouté au système a le rôle de communiquer avec le tag. De ce fait, l'étude des communications RFID effectives est possible avec une grande flexibilité dans l'édition des scénarios de communication tout en évitant la génération des stimuli. D'autre part, inclure le lecteur dans le système d'émulation permet l'observation et le suivi des

erreurs liées à la communication (par exemple les erreurs d'encodage et les erreurs à tolérer).

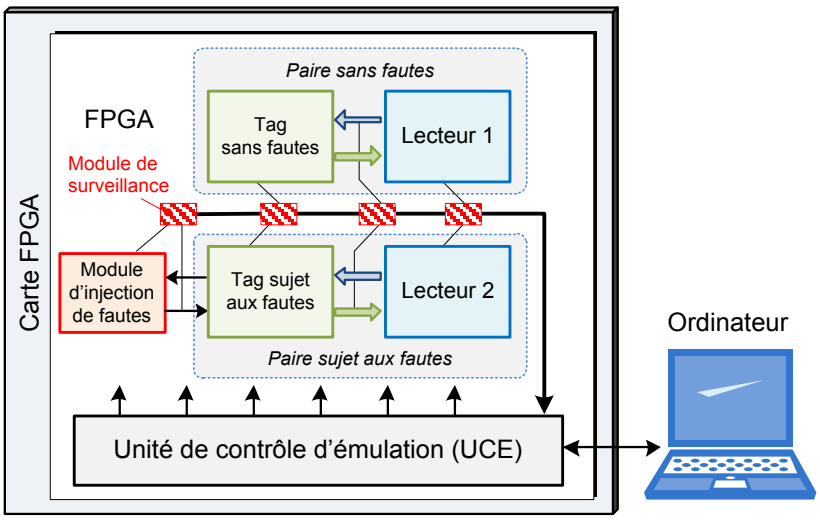


Figure 4-1 Plateforme RFID développée pour l'émulation de fautes.

Pour effectuer la comparaison et le suivi des effets des fautes injectées, deux circuits de lecteurs RFID identiques (Lecteur 1 et Lecteur 2) sont ajoutés au système pour constituer **deux paires lecteur-tag** : une première **paire sujette aux fautes** et l'autre **paire sans fautes** (Figure 4-1). Cette structure de lecteur-tag, communique de manière autonome, présente des conditions de comparaison appropriées et permet ainsi un contrôle et un suivi complets de toutes les opérations RFID. La solution adoptée exclut la communication RF par l'implémentation des deux lecteurs et des deux tags sur une seule FPGA. Cela simplifie considérablement le processus d'émulation en maintenant des résultats d'analyse valides puisque le comportement du tag est volontairement maintenu non affecté. Notons que l'obtention des mêmes conditions de test en utilisant des lecteurs externes est une tâche extrêmement difficile, car elle nécessite une isolation RF et une synchronisation sévère entre les deux paires.

Comme le montre la Figure 4-1, le système contient également des blocs supplémentaires : un **module d'injection de fautes**, une **unité de contrôle d'émulation** (**UCE**) et des **modules de surveillance** spécifiques. Ces derniers surveillent les deux paires lecteur-tag et toute communication survenue afin d'effectuer des comparaisons. De plus, les modules de surveillance sont chargés de fournir des informations sur les fautes injectées et leurs effets sur le tag. Le système global qui en résulte est configurable, autonome et donc capable d'effectuer une campagne complète d'émulation de fautes de

types **SEU** ou **SET** sans aucune exigence d'interaction externe. Le processus d'émulation peut être accéléré en utilisant la fréquence FPGA maximale. La plate-forme fonctionne en interaction avec un ordinateur hôte pour charger les paramètres d'émulation, acquérir et analyser les résultats d'émulation produits.

L'approche proposée offre la possibilité d'étudier n'importe quelle architecture de tag conforme aux différents protocoles RFID. Pour étudier une architecture d'un tag donnée, nous devrions utiliser un circuit de lecteur correspondant qui utilise un protocole conforme. Dans nos expérimentations sur le protocole **EPC Gen2**, nous avons expérimenté le circuit du tag **G2FSM** (décrit dans la Section 3.3) ; comme nous avons employé le circuit du lecteur **G2MC-Reader**, décrit dans la Section 3.4, pour former les paires lecteur-tag. Par ailleurs, le système d'injection de fautes présenté dans le chapitre précédent est utilisé dans notre plateforme d'émulation pour pouvoir réaliser différentes compagnes d'injection de fautes de types SEU et SET sur le tag.

L'opération de l'émulation de fautes est réalisée sous la supervision complète de l'unité de contrôle d'émulation (UCE), cette unité contrôle tous les composants du système et synchronise l'ensemble du processus d'émulation. De ce fait, l'UCE organise la progression de chaque cycle d'émulation, collecte les résultats d'émulation et finalement transfère les résultats à l'ordinateur hôte. La communication avec l'ordinateur hôte est réalisée via le port série. Cependant, un *buffer* de transmission (TX buffer) est employé pour sauvegarder et envoyer les résultats d'un cycle d'émulation vers l'ordinateur (Figure 4-2). Un autre *buffer* (de réception, RX buffer) intégré sert à stocker des paramètres d'émulation qui peuvent être chargés à partir de l'ordinateur. L'UCE intègre également un circuit spécifique permettant d'effectuer une émulation exhaustive dans l'espace et dans le temps. Cela signifie qu'il est possible de réaliser automatiquement, par une seule activation d'émulation, une injection de faute à chaque cycle d'horloge de l'émulation (temps exhaustif) et pour tout bit du vecteur concerné (espace exhaustif). Le mode exhaustif défini fonctionne séparément pour le temps et l'espace de sorte que nous pouvons les utiliser séparément ou combiner les deux en même temps. Tous les paramètres d'émulation, y compris l'activation du mode exhaustif, le mode d'injection de fautes (SEU ou SET), la durée du cycle d'émulation et d'autres paramètres sont chargeables via le port série.

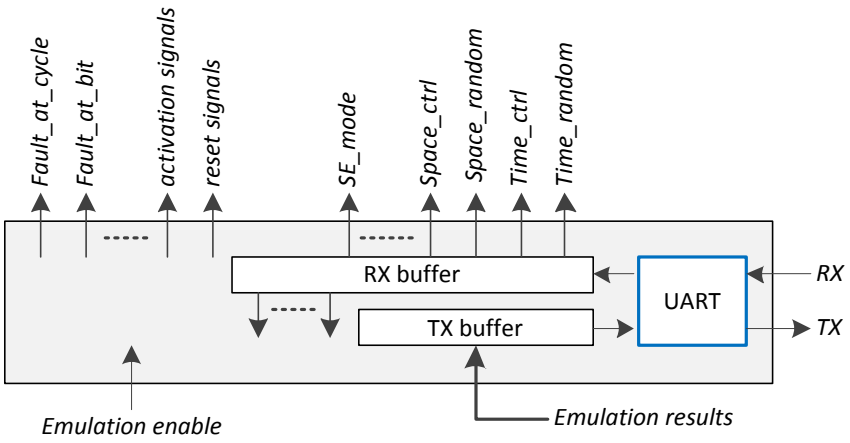


Figure 4-2 Représentation simplifiée du contrôleur d'émulation (UCE).

L'ordinateur hôte utilisé est équipé d'une application spécifique qui permet de charger les paramètres d'émulation dans la carte FPGA d'une part, et d'acquérir et d'analyser les résultats d'émulation produits d'autre part. Cette application génère alors plusieurs fichiers de sortie qui incluent les résultats acquis à partir de la carte d'émulation, la classification des fautes injectées et les analyses avancées sur les résultats obtenus. Le développement de cette application est effectué en langage C++ sous Microsoft Visual Studio. La partie qui réalise les opérations de classification et d'analyse dans cette application peut être implémentée dans le FPGA de la plateforme, mais nous avons préféré les mener à l'aide de l'ordinateur pour avoir plus de flexibilité dans la collecte des résultats, la gestion des fichiers de sortie et l'analyse des données. Cette manière permet également de sauvegarder et de réanalyser les résultats obtenus lors des expérimentations précédentes sans avoir le besoin de refaire ces expérimentations à nouveau.

En ce qui concerne l'implémentation sur FPGA, nous avons utilisé l'outil Xilinx ISE 14.7 pour élaborer le système d'émulation entier. Une campagne d'injection de fautes peut être lancée une fois que le système complet soit chargé sur FPGA et après avoir introduit les paramètres des fautes à émuler via le port série. Une campagne implique plusieurs cycles d'émulation. Dans chaque cycle, il y aura une injection de faute, une classification de cette faute et une analyse des erreurs générées. Les détails concernant le processus d'émulation sont donnés dans la section 4.3.

4.3 Processus d'émulation

Dans la plateforme d'émulation proposée, un cycle d'émulation se déroule en quatre phases (Figure 4-3) :

1) Au début, l'UCE initialise les lecteurs, les tags et les circuits de surveillance.

2) Ensuite, l'UCE active les deux paires lecteurs-tag pour qu'ils communiquent ensemble durant une période assez suffisante pour l'accomplissement d'un **scénario prédéfini**. Pendant cette période et selon les paramètres d'injection chargés, les fautes sont effectivement introduites dans le tag concerné. Les circuits de surveillance vérifient et enregistrent tous les événements importants simultanément.

3) Quand la période d'émulation expire, l'UCE sauvegarde les résultats obtenus dans le buffer de transmission.

4) Finalement, l'UCE transfère les données des résultats à l'ordinateur hôte.

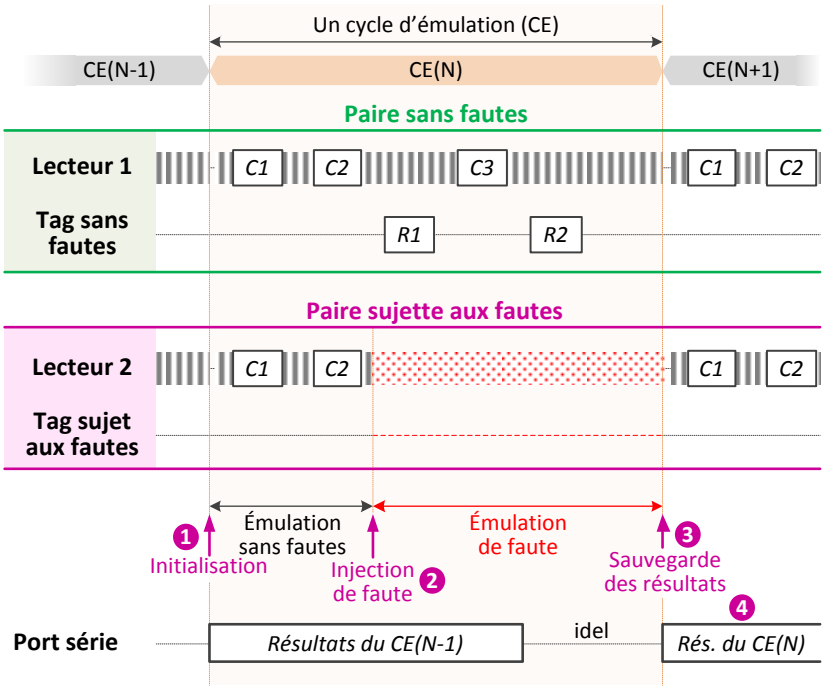


Figure 4-3 Processus d'émulation de fautes.

Généralement, une expérimentation nécessite plusieurs cycles d'émulation. Un pipeline est utilisé pour diminuer le temps d'émulation ; ainsi, les phases 1 et 2 se déroulent en parallèle avec les phases 3 et 4 comme le montre la Figure 4-3.

Une classification est effectuée pour chaque faute injectée selon son effet sur le déroulement du scénario de communication et selon les erreurs induites. Une classification en cinq catégories a été donc établie :

1) **Silent (S) :** scénario de communication achevé sans aucune détection d'erreur.

2) **Frame alteration (FA) :** scénario de communication achevé en présence d'altérations de trame, mais sans influences de rupture.

3) **Latent (L) :** scénario de communication achevé, mais accompagné d'un contenu de mémoire modifié qui à son tour peut produire un échec (*failure*) dans les prochains cycles.

4) **Latent with frame alteration (L&FA) :** similaire à Latent, mais en présence d'altérations de la trame.

5) **Failure (F) :** scénario non achevé.

Il est important de souligner que dans le cas général, une classification typique ne comporte que trois catégories principales : Silent (masqué), Latent, et Failure [116] [117]. Cependant, dans nos expérimentations sur la RFID, de nouvelles situations concernant l'altération des trames se sont présentées mais ne sont pas répertoriées dans la classification existante. Ceci nous a amené à introduire deux nouvelles catégories qui sont les FA et L&FA. Cette nouvelle classification a en effet permis une meilleure compréhension du comportement du tag comme le démontrent les résultats présentés par la suite.

4.4 Expérimentations

La plateforme développée supporte des analyses approfondies sur les impacts des fautes SEU et SET sur les tags RFID. Dans nos expérimentations, nous avons adopté un scénario de communication typique tel qu'illustré par la Figure 4-4. Cette figure détaille les étapes

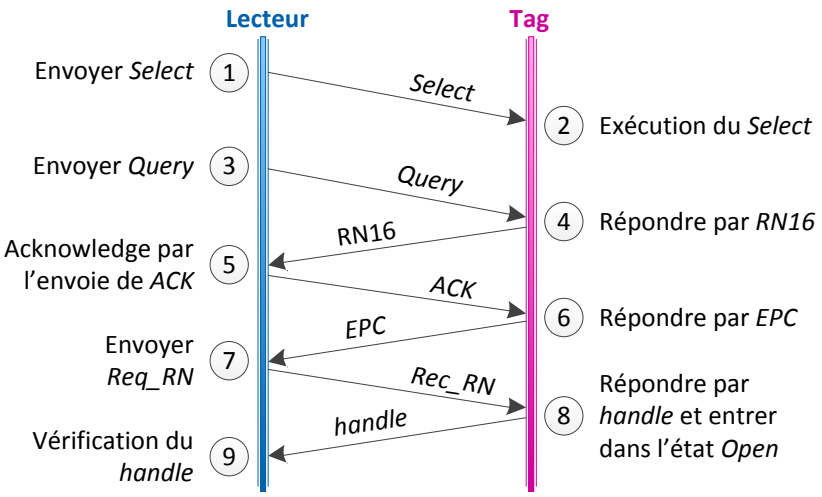


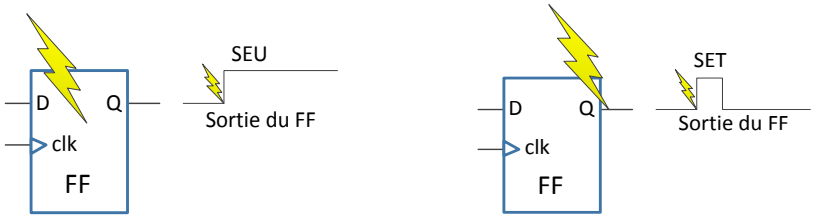
Figure 4-4 Le scénario de communication expérimenté.

de commandes/réponses échangées entre le lecteur et le tag [65]. Pour ce scénario, un cycle d'émulation dure 8000 périodes d'horloge du tag alors que 7600 périodes sont suffisantes pour accomplir le scénario. Notons que la durée du cycle d'émulation doit être supérieure au temps requis pour l'accomplissement du scénario. Cela permet d'avoir assez de temps pour détecter d'éventuelles activités, non prévues du tag, dont les durées surpassent celle du scénario.

Dans ce travail, nous avons réalisé deux expérimentations pour les SEU et les SET séparément :

1) Dans la première expérimentation, une injection exhaustive de fautes SEU (Figure 4-5, a) est réalisée où une faute SEU est injectée pour chaque bascule (*Flip-Flop* − FF) du tag et ceci à chaque cycle d'horloge. En total, 314 bascules ont fait l'objet de 2 071 772 SEU où chaque bascule est concernée par l'injection de 6598 SEU. L'opération en entier incluant l'introduction et l'analyse des fautes s'exécute automatiquement.

2) Dans la deuxième expérimentation, nous avons effectué deux injections de fautes SET pour chaque sortie de bascule (Figure 4-5, b) et ceci à chaque cycle d'horloge. Ainsi, une SET d'une durée d'une demi-période d'horloge a été injectée exhaustivement avant le front actif de l'horloge ; et une deuxième injection SET a été réalisée après le front actif.



(a) (b)

Figure 4-5 Illustration de l'effet d'une faute sur la sortie d'une bascule :(a) effet du SEU, (b) effet du SET.

Comme mentionné ci-dessus, les deux expériences d'injection de fautes ont ciblé tous les modules qui incluent des bascules. Ainsi, les modules "Tag Memory" et "Memory Controller" ne sont pas impliqués, car ils n'incluent aucune bascule. Egalement, lorsque les fautes injectées produisent les mêmes résultats quand le tag demeure dans les cycles de repos (*idle cycles*), les résultats d'injection d'un cycle sont donc représentatifs de l'ensemble des cycles de repos.

4.5 Résultats des expérimentations et discussion

Les expérimentations décrites précédemment ont été réalisées sur une carte Digilent Genesys équipée d'un FPGA Virtex-5 (XC5VLX50T). L'intégralité du système a consommé 69 % des ressources de la carte FPGA. Le module d'injection de fautes et l'UCE (Figure 4-1) ont consommé 10 982 slices, ce qui représentant 38 % de la surface utilisée du FPGA. Initialement, dans la première expérimentation, 110 minutes ont été nécessaires pour accomplir l'émulation de 2 071 772 SEU en utilisant l'horloge de fonctionnement réelle du tag (2,56 MHz, 3,22 ms/cycle d'émulation). Comme le nombre de SET émulés dans la deuxième expérimentation est le double, la durée de cette expérimentation a par conséquent doublé (220 minutes). Cependant, nous sommes parvenus à réduire le temps d'émulation en augmentant la fréquence d'horloge de l'émulation jusqu'à la fréquence d'implémentation maximale permise sans altérer les résultats d'émulation.

La fréquence maximale possible pour notre implémentation est de 29 MHz, permettant d'obtenir une durée d'expérimentation de 10 minutes pour la première expérimentation et de 20 minutes pour la deuxième. La durée totale de la campagne d'injection de fautes (6 215 316 SEU et SET) en incluant l'analyse des résultats obtenus est de 30 minutes. Notons que la durée d'expérimentation obtenue est spécifique au scénario de la Figure 4-4 et que cette durée devrait s'allonger avec l'augmentation du nombre de fautes injectées lors de scénario d'une plus longue durée. Cependant, la plateforme d'émulation proposée ainsi que la classification de fautes restent adaptées à tout autre scénario.

La Figure 4-6 illustre un exemple d'émulation de faute SEU. Les signaux exposés représentent la communication des deux paires lecteur-tag, la paire sujette aux fautes et la paire sans fautes, pendant l'émulation des SEU ; ces signaux sont obtenus via des pins de la carte FPGA et visionnés à l'aide d'un oscilloscope numérique. La faute SEU injectée dans cet exemple a causé un échec d'accomplissement du scénario (un *'failure'*, corruption de trame) comme le montre la Figure 4-6. La numérotation des actions dans cette figure montre la progression d'un cycle d'émulation conformément au scénario détaillé dans la Figure 4-4.

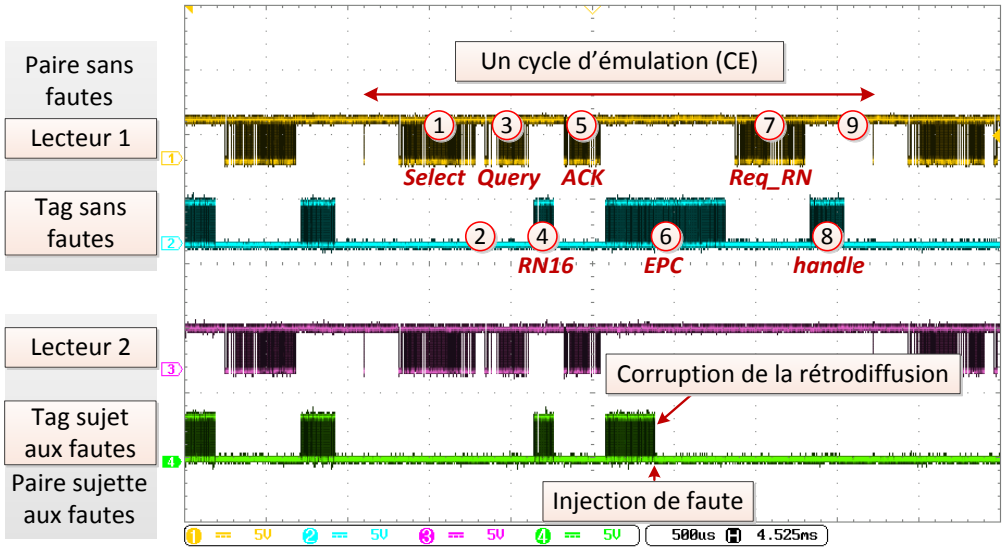


Figure 4-6 Signaux de communication des deux paires lecteur-tag durant l'émulation de SEU.

4.5.1 Résultats de l'émulation SEU

Les résultats de l'émulation des SEU sont résumés dans le Tableau 4-1. Les détails concernant le nombre des SEU injectées et leurs classifications résultantes sont présentés pour chaque bloc du tag G2FSM. L'analyse de ces résultats montre que les effets des fautes diffèrent d'un bloc (ou partie) à un autre et reflètent en fait les fonctions appropriées intégrées dans chaque bloc. Par exemple, un nombre important de ***'failures'*** (33 %) s'est produit dans le bloc "Tag FSM" (voir Figure 3-7), car il est chargé de contrôler toutes les autres parties du tag. Inversement, le nombre des *'failures'* détectées dans les blocs désignés comme périphériques est très limité (inférieur à 5 %), ce qui s'accorde avec leurs activités mineures au cours du mode active du tag.

D'autre part, le nombre des SEU ***'silent'*** est considérable et atteint, respectivement, 71,95 % et 75,85 % pour le côté réception et le côté transmission. Plusieurs signaux contenus dans ces deux parties sont initialisés lorsqu'ils ne sont pas utilisés, de sorte que l'effet SEU disparaît à la suite de cette initialisation, ce qui donne alors un nombre élevé de SEU *'silent'*. Pour les SEU classées sous la catégorie de ***'frame alteration'*** (altération de trame), leur nombre est limité (6,21 %) et concentré principalement dans le côté transmission qui est chargé de la génération des trames. Les SEU qui produisent latence (***latent***) et latence avec altération de trame (***latent & frame alt.***) sont concentrées dans les périphériques (31,32 % et 26,84 % respectivement) en accord avec les changements relatifs qui se passent dans les blocs "PRNG", "Slot Counter" et "Session Flags".

Au total, le taux des SEU causant '*failures*' est aux alentours de 16 %, le taux des SEU *latents* (L et L & FA) est un peu plus élevé (22,15 %), alors que plus de 60 % des SEU émulées ont un effet silencieux (*silent*).

Tableau 4-1 Résultats de l'émulation des SEU.

	Bloc	Nombre de FF	Nombre de SEU injectées	Classification des SEU				
				Failure (F)	Latent & frame alt. (L&FA)	Latent (L)	Frame alt. (FA)	Silent (S)
Côté réception	PIE Decoder	42	277 116	38 523	9 952	59 252	306	169 083
	Command Decoder	68	448 664	62 402	18 244	31 277	1	336 740
	CRC Decoder	21	138 558	17 692	0	4 800	0	116 066
	Total	**131**	**864 338**	**118 617**	**28 196**	**95 329**	**307**	**621 889**
				13,72%	**3,26%**	**11,03%**	**0,04%**	**71,95%**
Périphériques	PRNG	16	105 568	0	95 456	10 112	0	0
	Slot Counter	15	98 970	15	0	70 350	0	28 605
	T2 Timer	6	39 588	8	0	0	0	39 580
	T1 Timer	12	79 176	13 388	163	38	2 005	63 582
	Session Flags	5	32 990	1 906	0	31 084	0	0
	Total	**54**	**356 292**	**15 317**	**95 619**	**111 584**	**2 005**	**131 767**
				4,30%	**26,84%**	**31,32%**	**0,56%**	**36,98%**
Côté transmission	Backs. clock	5	32 990	4 599	22 926	66	1 542	3 857
	Backs. Generator	40	263 920	25 370	216	255	19 575	218 504
	FM0/Miller Encoder	15	98 970	17 615	5	0	3 448	77 902
	Total	**60**	**395 880**	**47 584**	**23 147**	**321**	**24 565**	**300 263**
				12,02%	**5,85%**	**0,08%**	**6,21%**	**75,85%**
	Tag FSM	**69**	**455 262**	**150 577**	**25 770**	**78 900**	**2 567**	**197 448**
				33,07%	**5,66%**	**17,33%**	**0,56%**	**43,37%**
Total		**314**	**2 071 772**	**332 095**	**172 732**	**286 134**	**29 444**	**1 251 367**
				16,03%	**8,34%**	**13,81%**	**1,42%**	**60,40%**

La Figure 4-7 présente les détails concernant la classification des fautes SEU où chaque ligne horizontale correspond à une bascule distincte du tag. Une ligne entière caractérise le nombre total de SEU émulées pour chaque bascule (une injection dans chaque cycle d'horloge ce qui représente 6598 SEU par ligne). Cette figure donne une idée sur la distribution globale des effets des SEU et permet d'identifier clairement les bascules les plus critiques.

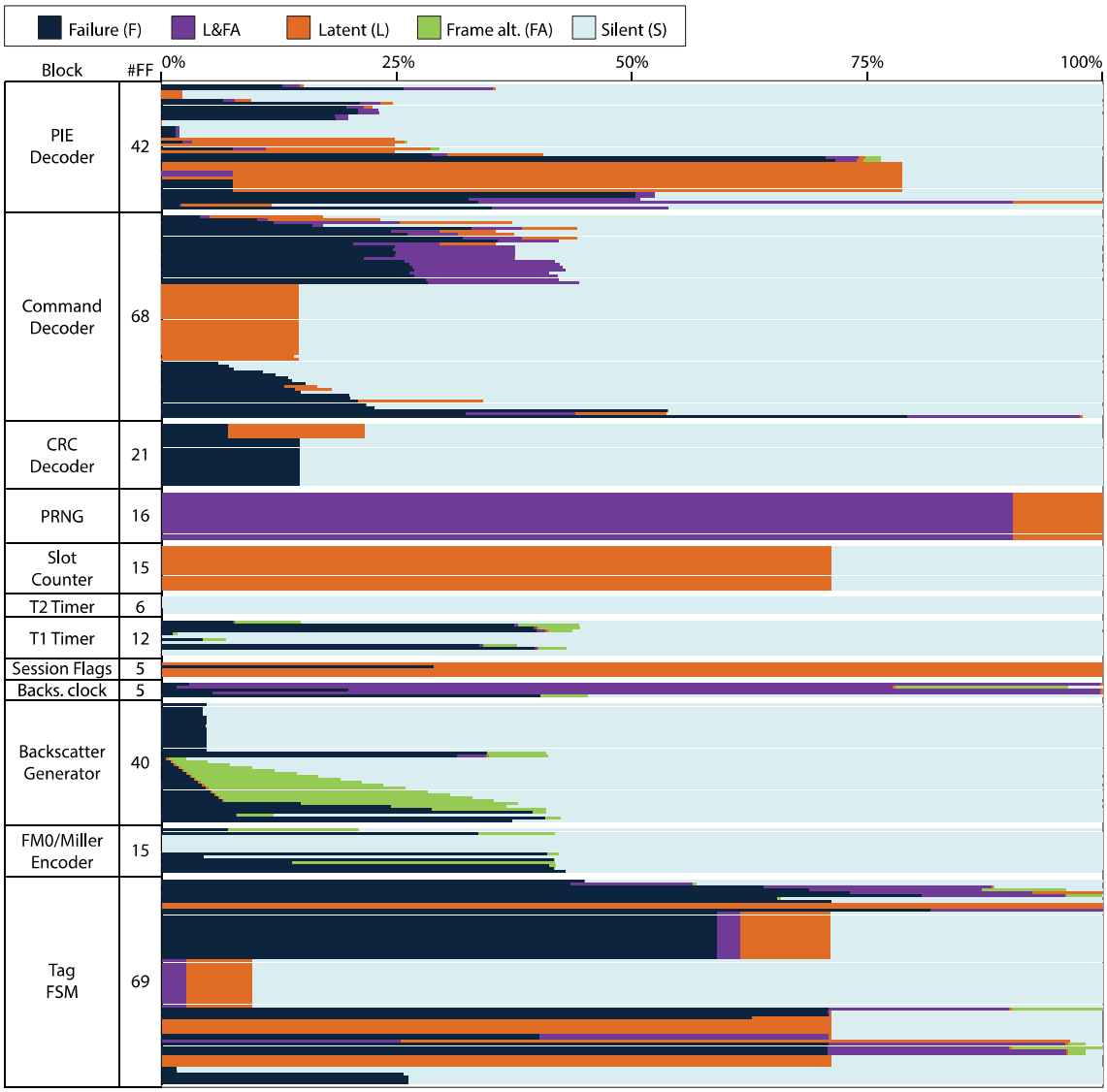


Figure 4-7 Classification des SEU injectées pour chaque bascule du tag.

Les bascules qui produisent des '*failures*' sont concentrées dans le bloc "Tag FSM". Ces bascules représentent les huit bits du registre d'encodage du FSM, une bascule concernée par la génération du signal de *reset* du système, le registre RN16 (16 bits), les bascules qui stockent les paramètres de la commande *Query* (*D*, *M*, et *TRext*) [65], et finalement les quatre bits MSB (Most Significant Bit) du registre qui stocke la valeur de la calibration T→R (TRcal). Concernant les fautes responsables de l'occurrence des altérations de trames, ils apparaissent en forme de triangle (avec une couleur verte) dans le bloc "Backscatter Generator". Les bascules concernées sont celles du registre à décalage de 16 bits qui fournit les données des trames envoyées ; le nombre d'altérations de trame augmente linéairement du LSB (Least Significant Bit) au MSB, d'où l'explication de la forme triangulaire.

Une majorité des SEU *'Silent'* détectées est liée aux bascules suivantes : les bascules du "T2 Timer", quatre bascules dans le "T1 Timer", les deux bascules LSB du "PW Counter" situé dans le "PIE Decoder", et six bascules qui sont chargées de l'encodage Miller dans le bloc "FM0/Miller Encoder" où seul le codage FM0 est utilisé dans le scénario expérimenté.

4.5.2 Résultats de l'émulation de SET

Comme décrit plus haut pour cette expérimentation, deux fautes SET sont injectées dans chaque cycle d'horloge et pour chaque bascule. La Figure 4-8 (a) présente la classification des SET injectées avant le front actif d'horloge, tandis que la Figure 4-8 (b) présente la classification des SET injectées après le front actif d'horloge. La somme de chaque catégorie de fautes ainsi que son pourcentage pour les deux cas sont donnés en haut de la figure. Dans le premier cas, le résultat est presque analogue à l'émulation de SEU pour la majorité des blocs à l'exception des blocs "Command Decoder", "CRC Decoder", et "Backscatter Generator" où les fautes '*Silents*' augmentent de 7 % puisque ces blocs utilisent différentes horloges générées par les blocs "PIE Decoder" and "Backscatter clock Generator". Dans le second cas où les SET sont injectés après le front actif d'horloge primaire du système, seulement 2 % des fautes injectées ont un effet non '*silent*'. Ce faible taux concerne 13 bascules (voir Figure 4-8, b) et s'explique par le fait que ces bascules sont, en général, utilisées comme des signaux de *reset*.

4.6 Analyse des erreurs

Nous avons effectué une analyse des erreurs survenues au cours de l'expérimentation des SEU en se basant sur les données fournies par les deux lecteurs et les modules de surveillance inclus dans le système d'émulation. Le Tableau 4-2 résume les erreurs déduites que nous avons divisées en deux catégories : erreurs majeures et erreurs mineures, en fonction de leur impact sur la sécurité et la fiabilité du tag. Notons qu'une faute peut produire plusieurs erreurs en même temps.

Les erreurs mineures ont un impact limité sur le tag. Par exemple, pour l'erreur "trame altérée valide", les trames incluent soit des erreurs d'encodage non destructives ou bien présentent des délais sans pour autant corrompre le scénario ou interrompre la communication. Les erreurs mineures restantes peuvent être recouvrées lors de nouvelles interrogations par le lecteur ou après le lancement d'un nouveau tour d'inventaire.

(a) (b)

Figure 4-8 Classification des SET pour chaque bascule du tag : (a) SET injectées avant le front actif d'horloge, (b) SET injectées après le front actif d'horloge.

Tableau 4-2 Sommaire des erreurs produites pour l'expérimentation de SEU.

	Erreurs	Nombre	Proportion	Catégorie de faute
Majeure	Entrer dans l'état *Secured*	8	1/258 971	F
	Entrer dans l'état *Killed*	0	0%	F
	Changement du contenu de la mémoire du tag	0	0%	F
Mineure	Trame altérée valide	202 176	9,76%	L&FA/FA
	Corruption de trame	68 143	3,29%	F
	Trame invalide	40 875	1,97%	F
	Echec d'identification	157 795	7,62%	F
	Sortie du tour (*round*) en cours	61 368	2,96%	F
	Changement des paramètres d'inventaire	34 926	1,69%	F/L/L&FA
	Changement des paramètre de *Query*	107 085	5,17%	F/L/L&FA

Par ailleurs, la présence d'erreurs majeures, bien que leur nombre soit faible (huit au total), est préjudiciable du moment que la sécurité du tag et les données qu'il contient sont affectées. L'erreur "entrer dans l'état *Secured*" est une menace sérieuse pour la sécurité du tag vu qu'elle peut, involontairement, amener à l'accès à des données sécurisées ou à leur altération. Notons que les erreurs "entrer dans l'état *Killed*" et "changement du contenu de la mémoire du tag" sont considérées comme des erreurs majeures, mais n'ont cependant pas été détectées à l'issue de l'injection exhaustive de fautes que nous avons effectuée.

La Figure 4-9 illustre le lien entre une sélection de cinq erreurs reportées dans le Tableau 4-2 et l'ensemble des bascules du tag (opération de filtration de la Figure 4-7). Cette représentation permet d'identifier efficacement les bascules critiques : chaque fois qu'une bascule est responsable de l'occurrence de plusieurs types d'erreurs, cette bascule est considérée comme critique. Ainsi, comme le montre la Figure 4-9, FF245 à FF252 (le registre FSM) et FF255 (signal de *reset* général du système) représentent les bascules les

Figure 4-9 Détails de cinq erreurs sélectionnées d'après le Tableau 4-2.

plus vulnérables dans le module Tag FSM. Parmi ces bascules, FF251 et FF252 sont les bascules les plus critiques vu qu'elles concernent les huit SEU qui causent l'erreur "entrer dans l'état *Secured*" évoquée précédemment.

Les bascules critiques sont plus facilement identifiables quand nous opérons une superposition de toutes les erreurs répertoriées dans le Tableau 4-2 pour chaque bascule ; le résultat est illustré par la Figure 4-10. Dans cette figure, plus la bascule est de couleur foncée, plus celle-ci est sujette à plus d'erreurs. Par conséquent, les bascules décrites précédemment (huit bascules du registre FSM) et la bascule du reset global apparaissent par une couleur plus sombre. Par ailleurs, la Figure 4-10 révèle d'autres bascules critiques, moins affectées que celles du registre FSM, situées dans les blocs "PIE Decoder" et "Command Decoder".

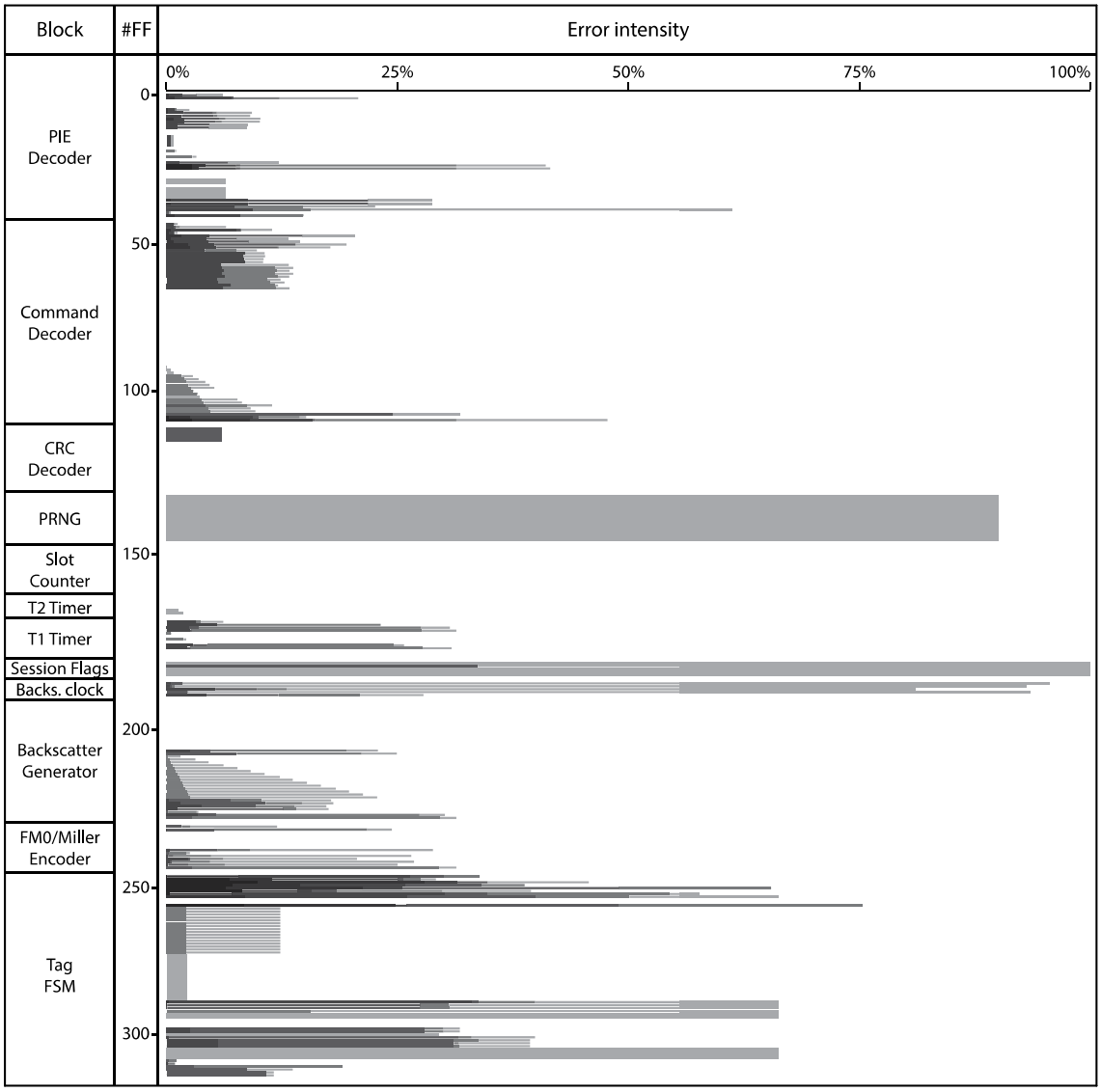


Figure 4-10 Intensité des erreurs pour chaque bascule du tag.

L'utilité des expérimentations réalisées est l'identification des fautes causant des erreurs majeures. La suite logique de cette identification est une tâche essentielle, qui consiste à éradiquer ces points faibles. De ce fait, il est recommandé de revoir la conception des parties sensibles du tag ou alternativement, créer des moyens de protection. L'augmentation de la longueur d'encodage du FSM, ainsi que l'ajout des circuits de surveillance spécifiques pour empêcher les fausses transitions du FSM peuvent également être considérés comme des alternatives appropriées.

4.7 Conclusion

Nous avons présenté dans ce chapitre une nouvelle méthode d'émulation de fautes basée sur FPGA. Le but de cette approche est d'effectuer des études approfondies sur les effets des fautes sur les circuits des tags RFID et évaluer la robustesse et la sécurité de ces derniers. La réalisation de ces études permet de fournir des résultats utiles qui peuvent être exploités dans le domaine du diagnostic des systèmes RFID et dans l'amélioration des futures conceptions de ces systèmes. Dans ce cadre, nous avons réalisé des expérimentations d'injections intensives de fautes SEU et SET sur G2FSM qui représente un circuit de tag compatible au protocole EPC Gen2. Cette approche est également applicable à d'autres protocoles RFID. Les expérimentations réalisées sur G2FSM ont démontré que 60,40 % des SEU ont été tolérées, 16,03 % des SEU ont causé des défaillances, tandis que 23,57 % des SEU ont produit des altérations de trame et/ou une latence dans les éléments de mémoire.

Une classification des erreurs produites à travers les résultats du système d'émulation a permis d'évaluer les impacts des fautes SEU et SET sur le tag. Seulement 8 SEU sur 2 071 772 ont provoqué des erreurs majeures forçant le tag expérimenté à entrer involontairement dans des états sécurisés. Notre système a non seulement identifié les bascules concernées par les erreurs majeures, mais aussi les bascules les plus sensibles provoquant, par exemple, un grand nombre de défaillances. L'expérimentation a également démontré comment dériver des actions de conception appropriées pour une éventuelle amélioration de la sécurité RFID à faible coût. Les résultats obtenus et la stratégie de classification des erreurs produites peuvent être utilisés pour améliorer le diagnostic dans la RFID.

Chapitre 5 Détection en ligne de fautes via des assertions

Dans ce chapitre, nous présentons une nouvelle approche de surveillance du fonctionnement des tags RFID afin de permettre la mise en place des fonctions de diagnostic RFID avancées, ainsi que le renforcement de la sécurité des tags contre les attaques par fautes. L'approche que nous proposons consiste à ajouter aux circuits des tags une instrumentation sur puce pour surveiller et sauvegarder simultanément leurs comportements erronés. L'instrumentation sur puce ajoutée est essentiellement composée d'assertions matérielles exclusivement dédiées à la détection en ligne de fautes [86]. Les modules développés dans les travaux de cette thèse, notamment le tag G2FSM et le module d'injection aléatoire de fautes présentés dans le Chapitre 3, sont utilisés pour l'application de cette approche.

Dans les prochaines sections de ce chapitre nous décrivons les principes d'utilisation des assertions dans le domaine de la vérification et le domaine de détection en ligne de fautes. Nous expliquons ensuite l'approche proposée et nous présentons la procédure de son implémentation et les résultats expérimentaux obtenus.

5.1 Utilisation d'assertions pour la vérification

C'est un fait que les conceptions VLSI deviennent de plus en plus complexes avec le temps. Pour maintenir un avantage concurrentiel sur le marché, les concepteurs mettent encore plus de portes logiques sur une seule puce. Typiquement, une telle conception complexe est formée en intégrant des blocs réutilisés ou des IP. Cependant, la réutilisation conduirait à un plus grand défi dans la vérification. Ces blocs fonctionnent correctement dans les anciennes conceptions, mais peuvent ne pas fonctionner lorsqu'ils sont intégrés à

nouveau dans une conception différente. La nouvelle conception peut violer alors les propriétés des blocs réutilisés. Pour s'assurer que les nouvelles conceptions répondent aux exigences spécifiées, toute violation de conception doit être repérée et corrigée par vérification.

Pour répondre aux défis de la vérification fonctionnelle, la **vérification à base d'assertion** (Assertion-Based Verification − **ABV**) est devenue une méthodologie courante et très exploitée dans la vérification des conceptions matérielles (*hardware designs*) aussi bien que dans l'ingénierie logicielle [118]. L'ABV fournit alors des fonctionnalités efficaces pour la **détection des bugs** et s'intègre parfaitement aux flux de conception et de vérification existants. Avec les langages d'assertion standardisés, bien supportés par les outils de vérification, les **assertions** peuvent apporter des avantages immédiats à la plupart des projets de vérification.

5.1.1 C'est quoi une assertion ?

Une assertion est une **déclaration** qu'une certaine **propriété** d'une conception doit être **vérifiée**. Si une assertion est **violée**, cela est **rapporté** instantanément. Une assertion est déclarée généralement par quelques lignes de code. Par exemple, la déclaration d'une assertion en VHDL se fait comme suit :

```
Assert <expression booléenne>
Report <message>
Severity <niveau de sévérité>;
```

Si l'**expression booléenne** est violée (pendant la simulation), le **message** est affiché sur l'écran avec le **niveau de sévérité**. Le niveau de sévérité peut être : *note*, *warning*, *error*, ou *failure*.

Le Listing 5-1 donne un exemple d'une assertion employée dans le *testbench* de l'ALU (unité arithmétique et logique) du MCIP-C (voir la section 3.1). À chaque application d'un nouveau vecteur de test aux entrées de l'ALU, l'assertion listée vérifie le résultat de l'opération effectuée par l'ALU (donné par le signal `s`) qui doit être égale à la valeur attendue (`exp_s`). Si le résultat n'est pas correct (une violation), durant la simulation du *testbench*, le message donné dans le listing est affiché avec le niveau de sévérité (`error`) ainsi que le moment de la violation. Par l'emploi de cette assertion, nous introduisons un moyen de vérification automatique et efficace qui facilite la détection et l'isolation des bugs.

Listing 5-1 Exemple d'assertion utilisée dans le *testbench* de l'ALU du MCIP.

```
assert s = exp_s
report "ERROR: output << result >> differs from the expected
        value"
severity error;
```

Pour pouvoir vérifier des propriétés assez complexes telles que la vérification d'une interface qui doit suivre un protocole donné, il existe des modules d'assertion spécifiques appelés *checkers* (vérificateurs) où chacun est conçu pour la vérification d'une propriété spécifique. Les *checkers* sont généralement regroupés dans des bibliothèques dédiées pour la vérification à base d'assertion.

5.1.2 Vérification à base d'assertion

La vérification à base d'assertion (*Assertion Based Verification* − ABV) est une méthodologie pour améliorer l'efficacité d'un environnement de vérification. L'ABV peut être donc utilisée très efficacement comme une technique de débogage dans le processus de vérification des conceptions. La Figure 5-1 montre le principe de l'ABV. Les concepteurs utilisent des assertions pour contrôler des propriétés spécifiques dans leurs conceptions ; ensuite, à travers la simulation, la vérification formelle et/ou l'émulation, les assertions indiquent si ces propriétés sont implémentées correctement. Lors de la mise en œuvre des assertions pour une conception donnée, le document de spécification

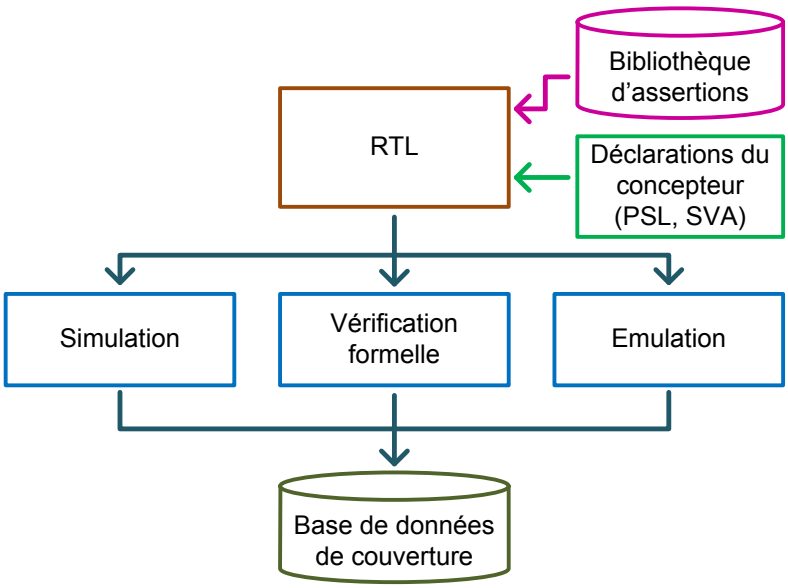


Figure 5-1 Méthodologie de la vérification à base d'assertions (ABV).

fonctionnelle de cette conception fonctionne en tant que source d'entrée et infère une liste de propriétés en convertissant (par code) ces propriétés en assertions.

5.1.3 Les langages standards d'assertions et les "Checkers"

Les langages d'assertion standard les plus courants sont le SVA (SystemVerilog Assertions) et le PSL (Property Specification Language). Ces langages permettent de définir les spécifications fonctionnelles des conceptions, y compris les suppositions, les obligations et les invariants. L'OVL (Open Verification Library) n'est pas un langage, mais une bibliothèque de *checkers* prédéfinis élaborés en VHDL et/ou en Verilog.

5.1.3.1 SystemVerilog Assertions (SVA)

Le SVA ajoute au Verilog traditionnel des fonctionnalités pour spécifier les assertions d'un système. En outre, les assertions peuvent être utilisées pour fournir une couverture fonctionnelle et générer des stimuli d'entrée pour la validation. Il y a deux sortes d'assertions : immédiate et concourante. Pour garantir l'efficacité du langage, le SVA fournit également les fonctions de déclaration des assertions de sorte à pouvoir lier aisément les modules des assertions (les *checkers*) aux modules de conception [118].

5.1.3.2 Property Specification Language (PSL)

Le PSL est un langage distinct spécialement conçu pour fonctionner avec de nombreux HDL et leurs couches d'expression. Par conséquent, le PSL ne peut pas être écrit directement en tant qu'une partie des HDL. Cependant, les propriétés PSL peuvent être attachées aux modèles HDL en utilisant des directives de liaison, et les outils peuvent prendre en charge l'inclusion de PSL dans les HDL [119].

5.1.3.3 Open Verification Library (OVL)

L'OVL est une bibliothèque composée d'un ensemble de *checkers* d'assertions prêts à être utiliser dans l'ABV pour vérifier des propriétés spécifiques dans les designs conçus en VHDL ou en Verilog. La dernière version d'OVL (OVL 2.8.1, téléchargeable à [120]) contient 55 *chekers* en total dont la majorité peut être utilisée en langage PSL et SVA. Parmi ces *checkers*, il y a seulement 10 *checkers* qui sont synthétisables et qui peuvent être donc utilisés en VHDL ou en Verilog. Le Tableau 5-1 donne la liste des *checkers* accessibles en VHDL et en Verilog [121].

Tableau 5-1 Liste des *checkers* OVL synthétisables.

Type de *checker*	VHDL		Verilog	
	Disponibilité	Synthétisable	Disponibilité	Synthétisable
ovl_always	•	•	•	•
ovl_cycle_sequence	•	•	•	•
ovl_implication	•	•	•	•
ovl_never	•	•	•	•
ovl_never_unknown	•	•	•	•
ovl_never_unknown_async	•	•	•	
ovl_next	•	•	•	•
ovl_one_hot	•	•	•	•
ovl_range	•	•	•	•
ovl_zero_one_hot	•	•	•	•

Les *checkers* d'OVL sont adaptés pour couvrir les problèmes potentiels de conception. Dans d'autres cas où il y a un besoin d'un *checker* d'assertions spécifique qui n'existe pas dans la bibliothèque, une combinaison de deux ou trois *checkers* d'assertions peut produire l'assertion envisagée. Il est également possible de combiner une assertion OVL avec une logique HDL supplémentaire pour vérifier le comportement souhaité.

Les *checkers* d'assertion OVL sont partitionnés selon cinq classes [121] :

- Assertions combinatoires − comportement vérifié avec la logique combinatoire.
- Assertions à 1 cycle − comportement vérifié dans le cycle en cours.
- Assertions à 2 cycles − comportement vérifié pour les transitions du cycle en cours au cycle prochain.
- Assertions à n cycles − comportement vérifié pour les transitions sur un nombre fixe de cycles.
- Assertions liées à des événements − le comportement est vérifié entre deux événements.

Chaque *checker* d'assertion OVL possède son propre ensemble de paramètres et son propre port d'entrées/sorties. Les paramètres suivants sont généralement communs à tous les *checkers* : *severity_level*, *property_type*, *msg*, *coverage_level*, *clock_edge*, *reset_polarity*, et *gating_type*. On donne comme exemple le *checker* "ovl_one_hot" illustré dans la Figure 5-2 qui donne aussi les paramètres de ce *checker* ainsi que la classe à

laquelle il appartient (Assertions à 1 cycle) [121]. Le *checker* "ovl_one_hot" vérifie, à chaque front actif de l'horloge, que dans le vecteur *test_expr* il y a toujours un seul bit qui est mis à '1' (one hot). Si une violation est survenue, le signal *fire* est activé. L'Annexe C présente un sommaire abrégé sur les autres *ckeckers* OVL avec leurs descriptions, paramètres et types.

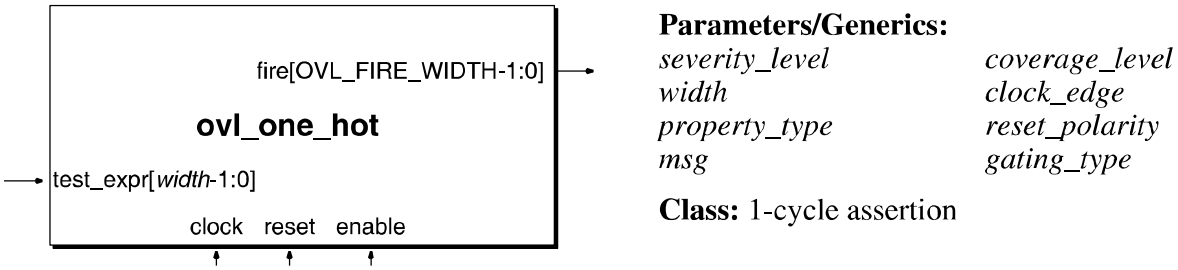


Figure 5-2 Représentation du checker "ovl_one_hot".

5.2 Détection en ligne de fautes à base d'assertions

Bien que les assertions soient utilisées pour la vérification des conceptions, plusieurs recherches suggèrent l'utilisation des assertions pour des usages de post-silicium comme le débogage après fabrication, le test en ligne, et la conception de systèmes tolérants aux fautes [122] [123] [124]. Etant donné que les assertions ne peuvent pas être implémentées directement en matériel, car elles sont écrites au moyen d'un langage haut niveau et destinées à être utilisées en simulation, plusieurs approches ont proposé des solutions pour synthétiser les assertions définies en langages PSL ou SVA afin de pouvoir les implémenter en matériel [122] [123]. Par ailleurs, il existe peu de générateurs de *checkers* qui permettent à la fois une génération efficace et une implémentation matérielle des *checkers* [125] [126].

Suivant ce domaine, nous avons utilisé des assertions pour la détection en ligne de fautes au niveau du tag RFID [127] [86]. Des *checkers* OVL standards ont été choisis pour élaborer ces assertions en bénéficiant de la portabilité et l'efficacité d'OVL comme démontré dans [128]. Mettre efficacement une multitude d'assertions OVL matérielles dans un circuit permet de surveiller constamment son activité. Lorsque le circuit fonctionne correctement, aucune violation n'est produite ni détectée. Une fois qu'une violation est détectée par un *checker* d'assertions, son signal d'erreur est activé indiquant la présence de cette violation, ce qui signifie qu'une ou plusieurs fautes sont produites à l'intérieur du circuit. Les sections suivantes donnent plus de détail sur cette approche.

5.3 Auto-surveillance en ligne appliquée aux tags RFID

Nous proposons à travers cette approche de réaliser une instrumentation sur puce dans les tags RFID afin de permettre un diagnostic avancé dans les systèmes RFID en surveillant et sauvegardant les comportements erronés des tags, d'une part, et d'augmenter la sécurité des tags contre les attaques par fautes, d'autre part [86]. L'instrumentation sur puce consiste à ajouter au circuit du tag une infrastructure composée essentiellement de *checkers* d'assertions matériels. Ces *checkers* permettent la détection en ligne des fautes qui surviennent dans le circuit du tag. Une partie de ces *checkers* est sélectionnée à partir de la bibliothèque OVL, alors que d'autres sont conçus spécifiquement pour surveiller les transitions de quelques machines d'état du tag. Les fautes détectées par les *checkers* sont comptées et enregistrées dans la mémoire du tag. Ensuite, la lecture des informations enregistrées est possible à travers un lecteur RFID (par RF) et par conséquent un diagnostic approprié pourrait être opéré.

L'approche que nous proposons présente plusieurs avantages envers les systèmes RFID qu'on peut résumer par les points suivants [86] :

- Le circuit d'infrastructure ajouté délivre le nombre de fautes détectées dans le tag, si ce nombre n'est pas nul, cela indique que soit le tag est défectueux, ou soit il existe des causes externes qui ont influencé le fonctionnement du tag.
- Les méthodes existantes qui permettent la détection des tags défectueux et le diagnostic des défaillances survenant dans les composants des systèmes RFID sont basées essentiellement sur des données délivrées par les lecteurs RFID [129] [130]. Cependant, l'approche que nous proposons fournit plus d'informations concernant le nombre de comportements incorrects détectés du tag permettant ainsi un meilleur diagnostic pour les systèmes RFID.
- La présente approche permet la détection des fautes produites durant les attaques par fautes, cela permet d'effectuer plusieurs actions parmi lesquelles : − Sécuriser le tag contre ce type d'attaques en réinitialisant les fonctions du tag lorsqu'un comportement susceptible est détecté. Cette action constitue un moyen efficace pour empêcher les attaquants de capturer toute donnée utile ou d'introduire des modifications non souhaitées dans la mémoire du tag. − L'enregistrement des comportements incorrects du tag dans sa mémoire permettrait au propriétaire du tag d'accéder aux informations enregistrées et par la suite d'identifier si le tag a été soumis à des attaques par fautes.

- L'intégration de capteurs dans les tags RFID semi-passifs est largement utilisée dans les réseaux de capteurs sans fil (*Wireless Sensor Network* - WSN) [131]. Afin de maintenir une application WSN en bon fonctionnement, il est essentiel de détecter les nœuds de capteurs défaillants et d'adopter un mécanisme de diagnostic efficace pour traiter les défaillances détectées, puis prendre les mesures appropriées pour continuer à utiliser les bonnes parties du réseau [132]. Notre approche permet de mieux reconnaître les nœuds capteurs défaillants ou perturbés, d'améliorer les conditions de travail des WSN et permet aussi le développement de nouvelles méthodes de diagnostic améliorées.

5.4 Implémentation de l'approche d'auto-surveillance pour tag RFID

Nous avons utilisé le tag G2FSM (Figure 3-7) pour implémenter l'approche proposée. De ce fait, une infrastructure a été ajoutée au circuit du tag pour surveiller son fonctionnement. Cette infrastructure se compose de 13 *checkers* distribués sur l'étendue du tag comme le montre la Figure 5-3 et détaillé dans le Tableau 5-2.

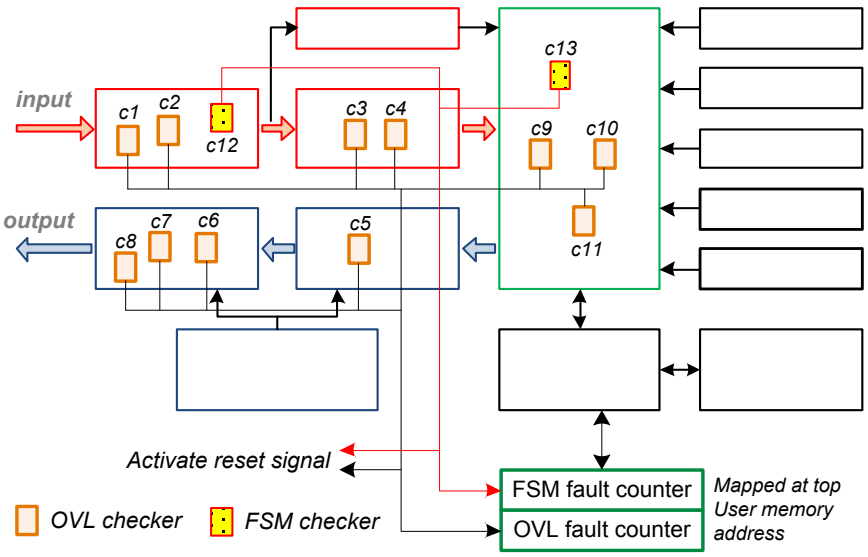


Figure 5-3 Contrôle du fonctionnement du tag utilisant des assertions matérielles.

Parmi les *checkers* utilisés, 11 sont des *checkers* d'assertions OVL définis de façon à couvrir le fonctionnement global du tag et contrôler leurs principaux signaux. Généralement, ces signaux synchronisent la réception et l'émission des trames, et dirigent aussi les transitions de quelques machines d'états du tag. Par exemple, le fonctionnement du tag est organisé en trois phases : l'attente ou la réception de trame, le traitement de

trame, et enfin la transmission de réponse ; les signaux principaux qui organisent ces phases sont contrôlés par le *checker c9 "ovl_one_hot"* pour détecter tout chevauchement de phase.

Tableau 5-2 Types des *checkers* utilisés.

Type de *checker*	Nombre d'assertions	Nom d'assertion (Figure 5-3)
ovl_always	3	c2, c4, et c5
ovl_implication	4	c6, c7, c8, et c10
ovl_next	1	c1
ovl_one_hot	1	c9
ovl_zero_one_hot	2	c3 et c11
FSM *checker*	2	c12 et c13

Cependant, les *checkers* synthétisables de la bibliothèque OVL n'incluent pas tous les aspects de contrôlabilité que nous voulons couvrir dans le tag. Par conséquent, nous avons développé deux autres *checkers (c12* et *c13)* (appelés *FSM checkers*) afin de surveiller deux machines à état (FSM) pertinentes. Comme illustré par la Figure 5-3, le premier *FSM checker (c12)* est implémenté dans le bloc "PIE Decoder" ; son rôle est de surveiller les transitions du "Symbol FSM" (la machine à état utilisée pour le décodage des symboles des trames reçues) afin de détecter toute fausse transition dans cette machine d'état. Le diagramme de la Figure 5-4 illustre toutes les transitions correctes ou fausses possibles de "Symbol FSM".

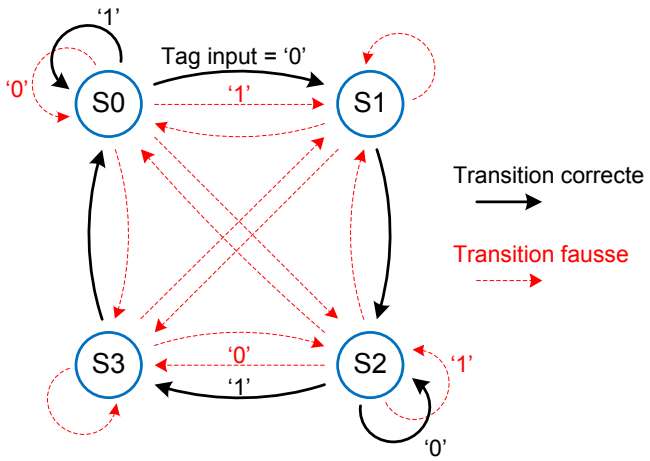


Figure 5-4 Diagramme de transition du "Symbol FSM".

L'utilisation du "Symbol FSM" est essentielle pour le fonctionnement du bloc "PIE Decoder" du fait qu'il l'exploite dans le décodage de chaque symbole reçu afin de déterminer s'il s'agit d'un bit de '0', un bit de '1' , *Tari*, *RTcal*… etc. (voir la section 2.3.4). La Figure 5-5 représente une séquence de transitions du "Symbol FSM" selon la séquence d'entrée du tag. Pour assurer une bonne réception de chaque trame, *c12* détecte toutes les fausses transitions, y compris la vérification des conditions de transition (Figure 5-6) ; ceci permet donc, avec l'utilisation des vérificateurs OVL *c1* et *c2*, de détecter plusieurs comportements erronés du module de réception "PIE Decoder" (Figure 5-3).

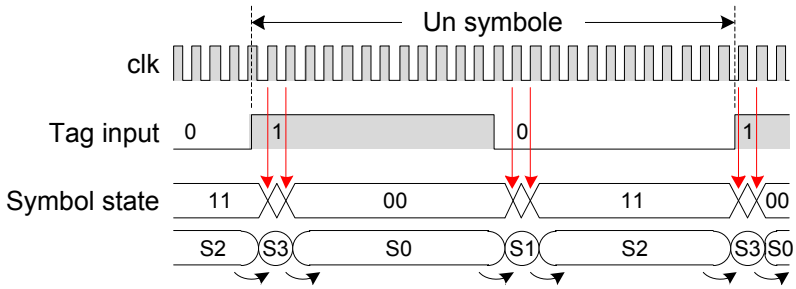


Figure 5-5 Séquence du "Symbol FSM".

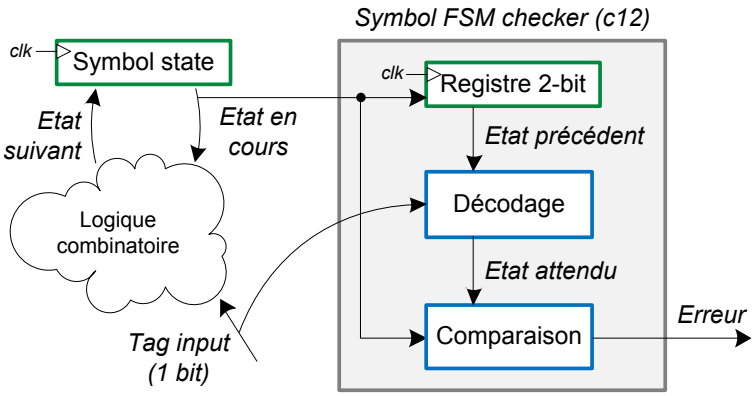


Figure 5-6 Fonctionnement du "*Symbol FSM checker*".

Le deuxième *FSM checker (c13)* est dédié à la surveillance des transitions de la principale machine à état du tag ("Tag FSM", voir la section 3.3.3). En effet, la surveillance de cette machine à états a une importance capitale vu qu'elle contrôle une large partie du tag. La Figure 5-7 illustre l'activité du *checker c13*. L'état précédent du tag est sauvegardé dans un registre à 8 bits à chaque période d'horloge, ensuite, cet état est décodé pour déterminer les états attendus. Enfin, un signal d'erreur est activé si l'état courant du tag ne correspond pas aux états attendus. Contrairement au *checker c12*, la

vérification des conditions de transition d'état n'est pas ajoutée dans le *checker c13* vu qu'il y a plusieurs signaux d'une longueur de 250 bits qui contrôlent les transitions du "Tag FSM" (Figure 5-7). Notons que l'ajout de l'option de vérification des conditions de transition augmente considérablement la taille du *checker*. Cependant, le taux de couverture d'erreur de ce *checker* reste élevé (96,42 %).

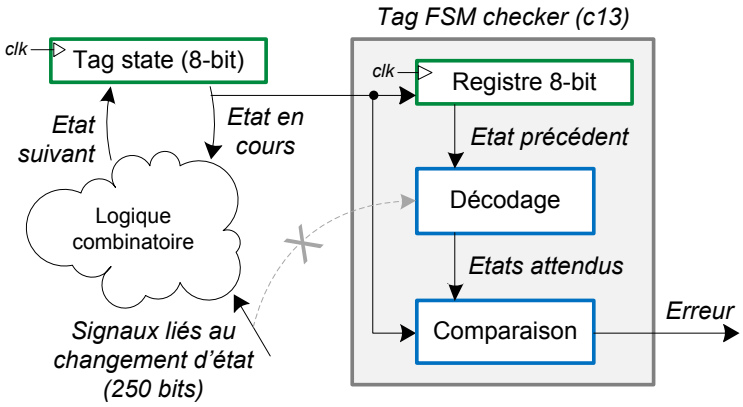


Figure 5-7 Activité du "*Tag FSM checker*".

Comme nous l'avons expliqué précédemment, le circuit de surveillance (l'infrastructure ajoutée au tag) délivre le nombre de fautes qu'il détecte et active, aussitôt, un signal de *reset* dans le cas où il détecte une faute. Par conséquent, deux registres de comptage, *"FSM fault counter"* et *"OVL fault counter"*, sont insérés pour compter chaque faute détectée par chaque type de *checkers* (Figure 5-3). Ces registres sont mappés à l'adresse haute de la mémoire "User Memory" du tag pour permettre la lecture et la modification de leurs contenus. Le signal du *reset* est généré si une faute est détectée par n'importe quel *checker* pour éviter tout comportement non désiré du tag. Aussi, nous avons opté pour l'implémentation des deux registres distincts pour pouvoir distinguer les fautes détectées par chaque type de *checkers*. Cependant, on peut utiliser un ou plusieurs compteurs selon la procédure de diagnostic désirée.

Les surfaces d'occupation du tag G2FSM avant et après l'ajout de l'infrastructure de détection en ligne de fautes sont données dans le Tableau 5-3. Les résultats obtenus sont basés sur une implémentation sur un FPGA Xilinx Spartan-3E XC3S500E. Ces résultats démontrent que l'occupation des LUT a augmenté de 15,23 % après l'ajout de l'infrastructure. Notons que l'occupation des cellules *(slices)* FPGA par le tag avant et après l'ajout de l'infrastructure est respectivement de 13 % et 15 %.

Tableau 5-3 Occupation de surface par le tag sur FPGA de type Spartan-3E avant et après l'implémentation de l'infrastructure de détection on-line de fautes.

Circuit	Occupation de surface	
	LUT	Flip-Flop
Tag sans l'infrastructure	1195	373
L'infrastructure ajoutée	182	46
Tag avec l'infrastructure	1377	419
Augmentation de surface	15,23%	12,33%

5.5 Expérimentation et résultats

Afin de procéder à l'évaluation de l'approche de surveillance proposée, nous avons utilisé le montage expérimental illustré par la Figure 5-8. Celle-ci est composée des trois éléments suivants :

1) Xilinx Spartan-3E Starter Kit : carte FPGA utilisée pour implémenter le circuit du tag.

2) Module Front-end : connecté aux pins d'entrée/sortie de la carte FPGA. Il permet l'envoi et la réception de signaux UHF à travers son antenne. La carte FPGA avec le front-end forment un émulateur de tag RFID UHF.

3) LinkSprite RFID UHF Gen2 reader : utilisé pour interroger l'émulateur de tag et réaliser des opérations d'identification, de lecture, et d'écriture. Il est connecté au PC par un câble USB pour avoir la possibilité de l'utiliser à travers son software.

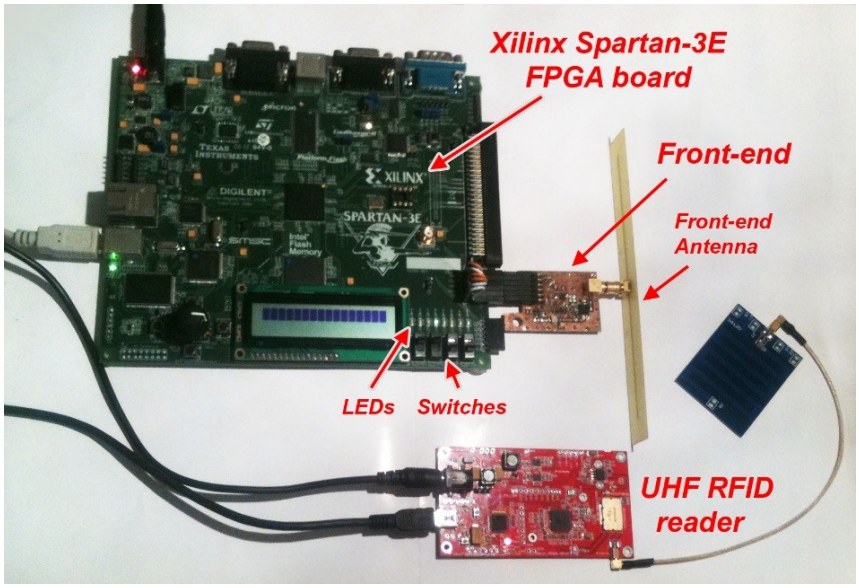


Figure 5-8 Montage expérimental utilisant une carte FPGA, un *Front-end* et un lecteur RFID.

Pour évaluer la capacité de détection de fautes de l'infrastructure proposée, nous avons utilisé le module d'injection de fautes présenté dans la section 3.5 pour réaliser des injections de fautes au niveau des principaux signaux du tag qui sont connectés directement aux entrées des *checkers*. La Figure 5-9 donne un aperçu sur le fonctionnement du module d'injection de fautes et son utilisation sur les signaux ciblés qui ont une largeur de 24 bits en total.

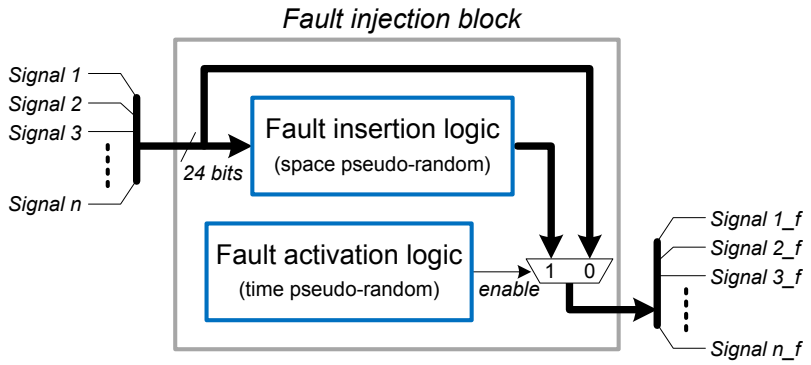


Figure 5-9 Fonctionnement général du bloc d'injection de fautes.

Après l'ajout du module d'injection de fautes, l'implémentation du système résultant est accomplie suivant la procédure standard d'implémentation sur FPGA en commençant par synthétiser le code VHDL jusqu'au chargement du fichier de configuration généré sur la carte FPGA en utilisant l'outil Xilinx ISE.

L'évaluation du circuit de surveillance a été effectuée à travers certaines expérimentations qui ont débouché sur 2051 fautes injectées. Une expérimentation commence par le lancement d'un processus de balayage (*scan*) des tags par le lecteur RFID pendant plusieurs seconds. L'injection de fautes est effectuée automatiquement avec un taux de 0,006 % durant la communication et lorsque le tag est en état de fonctionnement. Au moment de l'injection des fautes, seulement une faute est injectée aléatoirement sur un bit parmi les signaux ciblés en inversant sa valeur. Après un certain temps (défini d'une façon qui ne permet pas aux compteurs des fautes détectées d'atteindre leur limite), le processus du balayage est arrêté et les résultats sont collectés de deux manières différentes :

a) Le nombre de fautes détectées par les *checkers* est obtenu par le lecteur RFID en lisant les données stockées aux adresses 14 et 15 (en décimal) de "User Memory" comme le montre la Figure 5-10.

b) Le nombre total de fautes injectées et le nombre effectif des fautes détectées sont obtenus à travers les huit LED de la carte. Ainsi, nous utilisons les switches de la carte pour multiplexer les données qui apparaissent sur les LED.

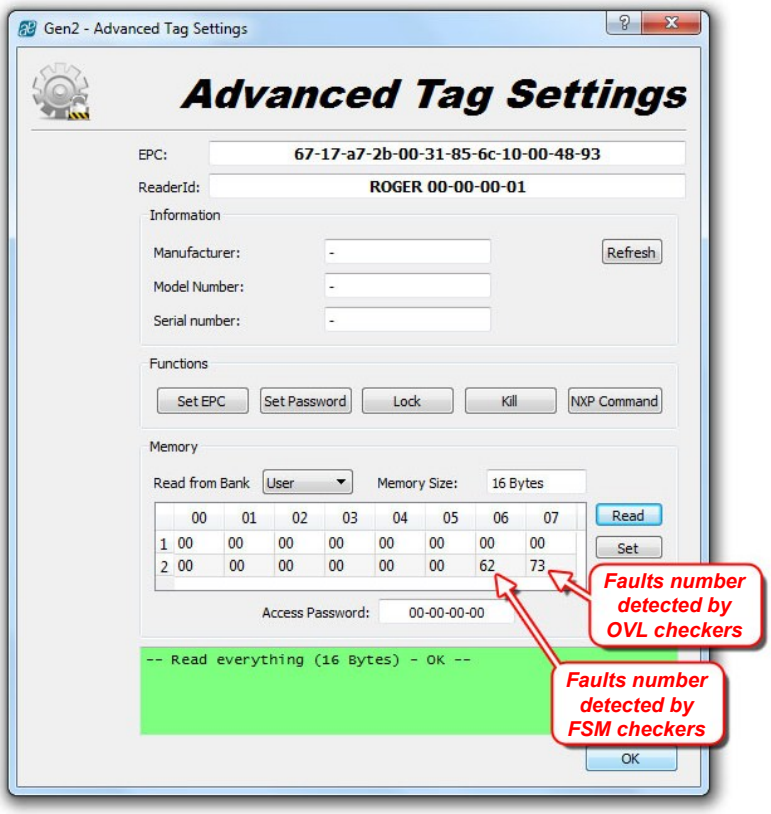


Figure 5-10 Lecture des nombres de fautes détectées utilisant le lecture RFID.

Cette phase d'expérimentation a été effectuée en désactivant l'option de *reset* généré par le circuit de surveillance en cas de détection de faute. Cela permet d'observer le comportement du tag d'un côté et de constater toutes les fautes détectées par les *checkers*. Les résultats obtenus sont exposés dans le Tableau 5-4. Nous constatons que le nombre total des fautes détectées par les *checkers* est de 3358, sachant qu'une faute injectée peut être l'objet d'une multitude de détections. C'est pour cette raison que nous exposons sur le même tableau le nombre effectif des fautes détectées par les *checkers* qui se situe aux alentours de 73 % des 2051 fautes injectées en total. Autrement dit, le nombre effectif des fautes détectées représente le nombre de fautes injectées qui n'ont pas été ratées par les *checkers*. Le pourcentage que nous venons de décrire représente la couverture de fautes pour tous les *checkers*, sachant que cette couverture diffère d'un *checker* à un autre. À noter également que certaines fautes non détectées n'ont pas affecté le fonctionnement du tag (elles sont masquées), ce qui les rend indétectables par les *checkers*.

Tableau 5-4 Efficacité de détection de fautes.

Type de *checker*	Fautes injectées	Fautes détectées	Fautes détectées effectivement	
			Nombre	Pourcentage
FSM checkers	2051	1542	1011	49,29%
OVL checkers		1816	492	23,98%
Total		**3358**	**1503**	**73,28%**

L'implémentation de l'approche proposée a engendré l'augmentation de la surface du tag de 15 %, du fait de l'ajout de l'infrastructure de contrôle, pour atteindre un taux de couverture de fautes de 73 % pour le cas étudié. Ce taux peut être augmenté par l'ajout de *checkers* supplémentaires, mais avec un désavantage qui réside dans l'augmentation de la surface du circuit. En effet, il convient de chercher un meilleur compromis entre le taux de couverture de fautes et l'augmentation de surface résultante.

5.6 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche d'auto-surveillance pour les tags RFID. Cette approche est basé sur l'ajout d'une infrastructure au circuit du tag afin de lui permettre de surveiller son propre fonctionnement de façon continue. L'intérêt d'intégrer cette fonction est d'élaborer, d'un côté, des moyens de diagnostic améliorés pour les systèmes RFID et de renforcer, d'un autre côté, la sécurité des tags contre les attaques par fautes. L'infrastructure ajoutée comprend plusieurs blocs d'assertions (des *checkers*) employés pour la détection en ligne de fautes. Le nombre de fautes détectées est sauvegardé dans la mémoire interne du tag pour pouvoir le consulter en utilisant un lecteur RFID.

L'approche proposée a été implémentée et testée en utilisant la plateforme FPGA d'émulation de tag (G2FSM). Les capacités de détection de fautes de cette approche sont évaluées à travers des expérimentations d'injection aléatoire de fautes. Les résultats expérimentaux ont démontré la validité de l'approche de sorte que 73,28 % des fautes injectées ont été détectées.

Conclusion et perspectives

Durant plusieurs années, l'utilisation de l'instrumentation sur puce a permis de fournir des solutions quasi-efficaces aux problèmes générés par l'augmentation de la complexité des systèmes électroniques, assurant de bon degré de sureté de fonctionnement. Ce type d'instrumentation est employé sous diverses formes et dans plusieurs domaines comme le test, le débogage, le diagnostic et la surveillance des CI.

Dans cette thèse, nous avons exploité l'instrumentation sur puce sous deux différentes formes pour produire des solutions liées aux recommandations et impératifs des systèmes RFID.

En ce qui concerne la première solution développée, nous avons conçu une instrumentation sur puce basée sur FPGA pour mesurer les effets des fautes SEU et SET sur le fonctionnement des tags RFID en évaluant ainsi la robustesse et la sécurité de ces derniers. L'instrumentation développée permet, à travers l'injection de fautes par émulation et l'analyse de leurs effets, de déterminer les parties les plus sensibles aux fautes dans les circuits des tags étudiés ainsi que relever les erreurs potentielles qui peuvent affecter la sécurité et la fiabilité des tags. Cette approche vise à garantir la fiabilité et la sécurité des architectures des tags à la phase de conception et cela à travers l'amélioration des conceptions ou encore en les équipant avec de mécanismes de protection.

Des expérimentations d'émulation intensives de fautes SEU et SET ont été réalisées sur chaque bascule d'un tag RFID (G2FSM) compatible avec le protocole EPC Gen2. Ces expérimentations ont démontré que 60,40 % des SEU ont été tolérées, 16,03 % des SEU ont causé des défaillances, tandis que 23,57 % des SEU ont produit des altérations de trame ou une latence dans les éléments de mémoire. Une classification des erreurs produites a permis d'évaluer les impacts des fautes SEU et SET sur le tag.

L'expérimentation a également démontré comment dériver des actions de conception appropriées pour une éventuelle amélioration de la sécurité RFID à faible coût. Les résultats obtenus et la stratégie de classification des erreurs produites peuvent être utilisés pour améliorer le diagnostic dans la RFID.

Dans la deuxième approche, nous avons proposé une nouvelle approche d'instrumentation sur puce post-silicium pour les tags RFID. Cette approche consiste à ajouter aux tags un mécanisme d'auto-surveillance sous forme de circuit d'infrastructure qui permet la détection en ligne de fautes. L'intérêt d'intégrer cette solution est, d'un côté, d'élaborer des moyens de diagnostic améliorés pour les systèmes RFID et, d'un autre côté, renforcer la sécurité des tags contre les attaques par fautes. Le nombre de fautes détectées est sauvegardé dans la mémoire interne du tag et reste consultable par le bais d'un lecteur RFID.

L'approche proposée a été implémentée et testée sur le tag étudié dans la première approche (G2FSM) en utilisant la plateforme d'émulation à base de FPGA. Les capacités de détection de fautes de cette approche ont été évaluées à travers des expérimentations d'injection aléatoire de fautes. Les résultats expérimentaux ont démontré la validité de l'approche de sorte que 73,28 % des fautes injectées ont été détectées.

L'application et la validation des solutions proposées n'ont été possibles que grâce aux différents modules développés dans ce but. Ces modules, développés en VHDL, comprennent un microcontrôleur configurable (MCIP−C), un circuit de tag RFID à base de microcontrôleur (G2MC), un circuit de tag RFID à base de machine d'état (G2FSM), un circuit de lecteur RFID (G2MC−Reader), et un système d'injection de fautes. Tous ces composants peuvent être exploités dans de futurs travaux de recherche et de développement.

Les travaux réalisés dans cette thèse relèvent plusieurs perspectives :

Dans le cadre des travaux de cette thèse, nous avons développé une autre solution d'auto-surveillance et de protection dédiée aux tags à base de MCU, cette solution a été implémentée sur le tag G2MC, mais que nous n'avons pas encore validé. C'est pour cette raison qu'elle ne figure pas dans ce manuscrit. Nous envisageons donc d'évaluer très prochainement cette approche afin de la valider comme une solution efficace pour les tags à base de MCU.

L'analyse d'injection de fautes que nous avons réalisée (présentée dans le Chapitre 4)

est menée sur G2FSM qui représente un tag à base de machine d'états. Il est recommandé de réaliser une analyse identique mais cette fois-ci sur G2MC et de faire par la suite une comparaison entre les deux architectures pour déduire laquelle est la plus robuste et la mieux adaptée aux applications sécurisées.

Bien que l'utilisation des FPGA nous a permis de valider les deux conceptions des tags et les approches développées, il reste que l'implémentation sur ASIC produira des résultats plus réalistes. Dès lors, il serait hautement avantageux de cibler des implémentations sur ASIC pour valoriser encore davantage les solutions proposées dans cette thèse.

Enfin, nous considérons que la traduction de toutes les étapes de conception suivies pour la conception de notre microcontrôleur MCIP sous forme d'un support didactique (interactions hardware-software au niveau instruction) serait très bénéfique à toute personne qui voudrait s'initier à l'art de conception de processeurs et circuits numériques avancés.

Nous considérons aussi que la bonne qualité de notre plateforme de développement sur FPGA, fournira un élan de taille à la conception de nouveaux produits et brisera certainement la crainte de toucher aux noyaux sacrés des circuits modernes.

Bibliographie

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Ieee Micro*, 2005, vol. 25, no. 6, pp. 10-16.

[2] I. Mezzah and H. Chemali, "Diagnostic dans les systèmes embarqués," in *JDoc'11*, Setif, May 2011.

[3] C. T. Huang, S. J. Wang, W. L. Wang, and Y. S. Wang, "Construction of an online rfid enabled supply chain system reliability monitoring model," in *International Symposium on Computer, Consumer and Control (IS3C)*, 2012, pp. 626-629.

[4] M. Hutter, J.-M. Schmidt, and T. Plos, "RFID and Its Vulnerability to Faults," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Berlin, Heidelberg, 2008, pp. 363–379.

[5] M. Hutter, S. Mangard, and M. Feldhofer, "Power and EM Attacks on Passive 13.56MHz RFID Devices," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Berlin, Heidelberg, 2007, pp. 320-333.

[6] I. Mezzah, *Etude et conception d'un microcontrôleur IP sur FPGA*. Sétif, Mémoire de Magister, Département d'électronique, Université de Sétif, 2008.

[7] R. Isermann, *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*.: Springer Science & Business Media, 2011.

[8] R. Isermann and P. Balle, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control engineering practice*, 1997, vol. 5, no. 5, pp. 709-719.

[9] K. Patan, *Artificial neural networks for the modelling and fault diagnosis of technical processes*.: Springer, 2008.

[10] P. Campolucci, A. Uncini, F. Piazza, and B. D. Rao, "On-line learning algorithms for locally recurrent neural networks," *IEEE transactions on neural networks*, 1999, vol. 10, no. 2, pp. 253-271.

[11] L. T. Wang, C. E. Stroud, and Touba, N. A., *System-on-Chip Test Architectures*. Burlington, USA: Elsevier, 2008.

[12] M. Stanisavljević, A. Schmid, and Y. Leblebici, *Reliability of Nanoscale Circuits and Systems: Methodologies and Circuit Architectures*.: Springer Science & Business Media, 2010.

[13] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital systems testing and testable design*. New York: Computer science press, 1990, vol. 2.

[14] S. Soldani, *Vers le diagnostic embarqué de défaillances dans les systèmes à événements discrets: application au domaine automobile*.: Doctoral dissertation, Institut National des Sciences Appliquées de Toulouse (INSA Toulouse), 2008.

[15] V. Venkatasubramanian, R., Yin, K. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & chemical engineering*, 2003, vol. 27, no. 3, pp. 293-311.

[16] R. R. Murphy and D. Hershberger, "Classifying and recovering from sensing failures in autonomous mobile robots," in *National Conference on Artificial Intelligence*, 1996, pp. 922-929.

[17] Y. Papadopoulos, "Model-based system monitoring and diagnosis of failures using statecharts and fault trees," *Reliability Engineering & System Safety*, 2003, vol. 81, no. 3, pp. 325-341.

[18] J. L. Recht, *Failure mode and effect*.: National Safety Council, 1966.

[19] A. Villemeur, *Sureté de fonctionnement des systèmes industriels: fiabilité, facteurs humains, informatisation*.: Eyrolles, 1988.

[20] G. Zwingelstein, *Diagnostic des défaillances, théorie et pratique pour les systèmes industriels*.: Hermès, 1995.

[21] B. G. Buchanan and E. H. Shortliffe, *Rule Based Expert Systems : The Mycin Experiments of the Stanford Heuristic Programming Project*.: Addison-Wesley, 1984.

[22] H. L. Gelgele and K. Wang, "An expert system for engine fault diagnosis: development and application," *Journal of Intelligent Manufacturing*, 1998, vol. 9, no. 6, pp. 539-545.

[23] Y. Papadopoulos and J. A. McDermid, "Automated safety monitoring: A review and classification of methods," *International journal of COMADEM*, 2001, vol. 4, no. 4, pp. 14-32.

[24] R. Davis and W. Hamscher, "Model-based reasoning: Troubleshooting," *Exploring artificial intelligence*, 1988, vol. 8, pp. 297-346.

[25] J. De Kleer and B. C. Williams, "Diagnosis with Behavioral Modes," *IJCAI*, 1989, vol. 89, pp. 1324-1330.

[26] J. De Kleer, A. K. Mackworth, and R. Reiter, "Characterizing diagnoses and systems," *Artificial intelligence*, 1992, vol. 56, no. 2-3, pp. 197-222.

[27] L. Console and G. Friedrich, "Introduction," *Annals of Mathematics and Artificial Intelligence*, 1994, vol. 11, no. 1, pp. 1-10.

[28] B. Dubuisson, *Diagnostic, intelligence artificielle et reconnaissance des formes*.: Hermès science publications, 2001.

[29] P. Struss and O. Dressler, ""Physical Negation" Integrating Fault Models into the General Diagnostic Engine," *IJCAI*, 1989, vol. 89, pp. 1318-1323.

[30] J. De Kleer, "Focusing on Probable Diagnoses," *AAAI*, 1991, vol. 91, pp. 842-848.

[31] J. De Kleer, "Using crude probability estimates to guide diagnosis," *Artificial intelligence*, 1990, vol. 45, no. 3, pp. 381-391.

[32] L. Console and P. Torasso, "Integrating models of the correct behavior into abductive diagnosis," in *9th European Conference on Artificial Intelligence*, Stockholm, 1990, pp. 160-166.

[33] G. Steinbauer and F. Wotawa, "Detecting and locating faults in the control software of autonomous mobile robots," in *IJCAI*, 2005, pp. 1742-1743.

[34] H.H. Rosembrock, *State-space and multivariable theory*.: John Wiley and Sons, 1970.

[35] H. L. Gelgele and K. Wang, "An expert system for engine fault diagnosis: development and application," *Journal of Intelligent Manufacturing*, 1998, vol. 9, no. 6, pp. 539-545.

[36] R. Isermann, "Supervision, fault-detection and fault-diagnosis methods—an introduction," *Control engineering practice*, 1997, vol. 5, no. 5, pp. 639-652.

[37] P. Li et al., "Estimation of railway vehicle suspension parameters for condition

monitoring," *Control engineering practice*, 2007, vol. 15, no. 1, pp. 43-55.

[38] R. Washington, "On-board real-time state and fault identification for rovers," in *IEEE International Conference on Robotics and Automation*, vol. 2, San Francisco, USA, 2000, pp. 1175-1181.

[39] Y. Zhang and X. R. Li, "Detection and diagnosis of sensor and actuator failures using interacting multiple-model estimator," in *36th IEEE Conference on Decision and Control*, vol. 5, San Diego, USA, 1997, pp. 4475-4480.

[40] I. Hwang, J. Hwang, and C. Tomlin, "Flight-mode-based aircraft conflict detection using a residual-mean interacting multiple model algorithm," in *The AIAA guidance, navigation, and control conference*, Austin, Texas, USA, 2003, pp. 21-28.

[41] C. Hajiyev and F. Caliskan, "Sensor and control surface/actuator failure detection and isolation applied to F-16 flight dynamic," *Aircraft Engineering and aerospace technology*, 2005, vol. 77, no. 2, pp. 152-160.

[42] P. Li and R. Goodall, "Model-based condition monitoring for railway vehicle systems," in *Control 2004 Conference*, 2004.

[43] V. Verma, G. Gordon, R. Simmons, and S. Thrun, "Real-time fault diagnosis [robot fault diagnosis]," *IEEE Robotics & Automation Magazine*, 2004, vol. 11, no. 2, pp. 56-66.

[44] G. Gómez, G. Campion, M. Gevers, and P. Willems, "A case study of physical diagnosis for aircraft engines," in *American Control Conference*, vol. 4, 2000 , pp. 2383-2387.

[45] C. M. Bishop, *Neural networks for pattern recognition*.: Oxford university press, 1995.

[46] B. D. Ripley, *Pattern recognition and neural networks*.: Cambridge university press, 2007.

[47] L.A. Zadeh, "Fuzzy sets," *Information and Control*, 1965, vol. 8, no. 3, pp. 338–353.

[48] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *In Readings in Fuzzy Sets for Intelligent Systems*, 1993, pp. 387-403.

[49] R. R. Yager and D. P. Filev, "Essentials of fuzzy modeling and control," *New York*, 1994, vol. 388,.

[50] A. J. Sharkey, G. O. Chandroth, and N. E. Sharkey, "Acoustic emission, cylinder pressure and vibration: A multisensor approach to robust fault diagnosis," in *IEEE-*

INNS-ENNS International Joint Conference on Neural Networks, vol. 6, Como, Italy, 2000, pp. 223-228.

[51] H. Poulard, *Statistiques et réseaux de neurones pour un système de diagnostic: application au diagnostic de pannes automobiles*.: Doctoral dissertation, Université Paul Sabatier-Toulouse III, 1996.

[52] A. Debiolles, L. Oukhellou, and P. Aknin, "Combined use of partial least squares regression and neural network for diagnosis tasks," in *17th International Conference on Pattern Recognition*, vol. 4, Cambridge, UK, 2004, pp. 573-576.

[53] Y. M. Chen and M. L. Lee, "Neural networks-based scheme for system failure detection and diagnosis," *Mathematics and Computers in Simulation*, 2002, vol. 58, no. 2, pp. 101-109.

[54] A. Schwarte and R. Isermann, "Neural network applications for model based fault detection with parity equations," *IFAC Proceedings Volumes*, 2002, vol. 35, no. 1, pp. 205-210.

[55] N. Stollon, *On-Chip Instrumentation: Design and Debug for Systems on Chip*. USA: Springer Science and Business Media, 2011.

[56] J. Braunes and R. G. Spallek, "A High-Level Language and Compiler to Configure the Multi-Core Debug Solution (MCDS)," in *International Conference on Advances in System Testing and Validation Lifecycle*, Porto, Portugal, 2009.

[57] P. Bernardi, M. Rebaudengo, and F. L. Vargas, "A New Hybrid Fault Detection Technique for Systems-on-a-Chip," *IEEE Transaction on Computers*, 2006, vol. 55, no. 2, pp. 185-198.

[58] P. Bernardi, L.M. Bolzani Poehls, M. Grosso, and M. Sonza Reorda, "A Hybrid Approach for Detection and Correction of Transient Faults in SoCs," *IEEE Transaction on Dependable and Secure Computing*, 2010, vol. 7, no. 4, pp. 439-445.

[59] P. Bernardi, M. Grosso, M. Rebaudengo, and M. Sonza Reorda, "Exploiting an infrastructure-intellectual property for systems-on-chip test, diagnosis and silicon debug," *IET Comput. Digit. Tech.*, 2010, vol. 4, no. 2, pp. 104-113.

[60] M. Abramovici, "In-System Silicon Validation and Debug," *IEEE Design & Test of Computers*, 2008, vol. 25, no. 3, pp. 216-223.

[61] D. Paret, *RFID en ultra et super hautes fréquences: UHF-SHF: Théorie et mise en oeuvre*.: Dunod, 2008.

[62] K. Finkenzeller, *RFID handbook: fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*.: John

Wiley & Sons, 2010.

[63] centrenational-rfid.com. [Online]. (2017, December) http://www.centrenational-rfid.com/features-of-rfid-tags-article-19-gb-ruid-202.html

[64] Suzanne Smiley. RFID Insider Inc. [Online]. (2017, December) https://blog.atlasrfidstore.com/uhf-rfid-tag-communications-protocols-standards

[65] "EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID, Protocol for Communications at 860 MHz – 960 MHz," Version 1.2.0 October 2008.

[66] EPC/RFID. [Online]. (2017, December) https://www.gs1.org/epc-rfid

[67] G. Fritz, *Simulation de fautes pour l'évaluation du test en ligne de systèmes RFID*. Doctoral dissertation, Grenoble, 2012.

[68] J. C. Laprie et al., *Guide de la Sûreté de Fonctionnement*, 2nd ed.: Cépaduès, 1996.

[69] L. Cauffriez, J. Ciccotelli, B. Conrard, and M. Bayart, "Design of intelligent distributed control systems: a dependability point of view," *Reliability Engineering & System Safety*, 2004, vol. 84, no. 1, pp. 19-32.

[70] G. Lodewijks, H. P. Veeke, and A. L. De La Cruz, "Reliability of RFID in logistic systems," in *IEEE International Conference on Service Operations and Logistics, and Informatics*, Shanghai, China, 2006, pp. 971-976.

[71] J. C. Knight, "Dependability of embedded systems," in *24th International Conference on Software Engineering*, Orlando, FL, USA, 2002, pp. 685-686.

[72] T. Karygiannis, B. Eydt, G. Barber, L. Bunn, and T. Phillips, "Guidelines for securing radio frequency identification (RFID) systems," *NIST Special publication*, 2007, vol. 80, pp. 1-154.

[73] W. Mallouli, F. Bessayah, A. Cavalli, and A. Benameur, "Security rules specification and analysis based on passive testing," in *Global Telecommunications Conference*, New Orleans, LO, USA, 2008, pp. 1-6.

[74] D. Geng, J. Wen, and W. Tian, "A scheme to enhance security of RFID middleware," in *International Conference on Communications, Circuits and Systems (ICCCAS)*, Chengdu, China, 2010, pp. 147-149.

[75] M. D. Nguyen, G. Fritz, O. E. K. Aktouf, V. Beroulle, and D. Hély, "Towards middleware-based fault-tolerance in RFID systems," in *13th European Workshop on Dependable Computing*, Pise, Italy, 2011, pp. 49-52.

[76] N. Park, H. Lee, H. Kim, and D. Won, "A security and privacy enhanced protection scheme for secure 900MHz UHF RFID reader on mobile phone," in *IEEE Tenth International Symposium on Consumer Electronics*, St. Petersburg, Russia, 2006, pp. 1-5.

[77] J. L. Ma, X. Wei, and J. Liu, "Security Analysis of RFID Based on Multiple Readers," in *Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Kaohsiung, Taiwan, 2007, pp. 265-268.

[78] J. Park, J. Na, and M. Kim, "A practical approach for enhancing security of EPCglobal RFID Gen2 tag," in *Future Generation Communication and Networking*, Jeju, South Korea, 2007, pp. 436-441.

[79] C. Piao, Z. Fan, C. Yang, and X. Han, "Research on RFID security protocol based on grouped tags and re-encryption scheme," in *IEEE International Conference on Wireless Communications, Networking and Information Security*, 2010.

[80] I. Mezzah and L. Boudjlida, *Conception d'un microcontrôleur IP (MCIP)*. Sétif, Mémoire d'ingénieurat, Département d'électronique, Université de Sétif, 2005.

[81] I. Mezzah, H. Chemali, and S. Mezzah, "Développement d'un Microcontroleur IP en VHDL et Implémentation sur FPGA Spartan-3E," in *2nd Embedded System Conference*, Bordj El-Bahri, Alger, May 2009.

[82] I. Mezzah, H. Chemali, and S. Mezzah, "Design of a structured IP microcontrolle," in *International Conference on Applied Informatics (ICAI 09)*, Bordj Bou Arreridj, Novembre 2009.

[83] I. Mezzah. MCIP open, OpenCores. [Online]. (2017, December) https://opencores.org/project,mcip_open

[84] I. Mezzah, H. Chemali, S. Mezzah, O. Kermia, and O. Abdelmalek, "MCIP: High configurable 8-bit microcontroller IP-core," in *Science and Information Conference (SAI)*, London, UK, 2015, pp. 1387-1390.

[85] I. Mezzah and H. Chemali, "Conception d'une infrastructure-IP pour le contrôle de fiabilité des systèmes sur puce," in *JDoc'12*, Setif, Avril 2012.

[86] I. Mezzah et al., "Assertion based on-line fault detection applied on UHF RFID tag," in *8th International Design and Test Symposium (IDT)*, Marrakesh, Maroc, December 2013.

[87] O. Abdelmalek, D. Hely, V. Beroulle, and I. Mezzah, "An UHF RFID Emulation Platform with Fault Injection and real time Monitoring capabilities," in *8th International Design and Test Symposium (IDT)*, Marrakesh, Maroc, December

2013.

[88] H. Sellani and B. Mouloud, *Implémentation d'un transpondeur RFID UHF sur circuit intégré*.: Mémoire de Master, Université de Blida, 2016.

[89] L. Sterpone, *Electronics system design techniques for safety critical applications*.: Springer Science & Business Media, 2008, vol. 26.

[90] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology—CRYPTO'97*, London, UK, 1997, pp. 513-525.

[91] A. Barenghi et al., "Exploring the feasibility of low cost fault injection attacks on sub-threshold devices through an example of a 65nm AES implementation," in *International Workshop on Radio Frequency Identification: Security and Privacy Issues*, Berlin, Heidelberg, 2011, pp. 48–60.

[92] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, 1997, vol. 30, no. 4, pp. 75-82.

[93] H. Cox and J. Rajski, "A method of fault analysis for test generation and fault diagnosis," *IEEE transactions on computer-aided design of integrated circuits and systems*, 1988, vol. 7, no. 7, pp. 813-833.

[94] C. López-Ongil, M. García-Valderas, M. Portela-García, and L. Entrena, "Autonomous fault emulation: a new FPGA-based acceleration system for hardness evaluation," *IEEE Transactions on Nuclear Science*, 2007, vol. 54, pp. 252–261.

[95] R. Velazco, S. Rezgui, and R. Ecoffet, "Predicting error rate for microprocessor-based digital architectures through CEU (Code Emulating Upsets) injection," *IEEE Transactions on Nuclear Science*, 2000, vol. 47, no. 6, pp. 2405-2411.

[96] R. A. Reed et al., "Heavy ion and proton-induced single event multiple upset," *IEEE Transactions on Nuclear Science*, 1997, vol. 44, no. 6, pp. 2224-2229.

[97] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," in *Twenty-Fourth International Symposium on Fault-Tolerant Computing - FTCS-24*, Austin, TX, USA, 1994, pp. 66-75.

[98] T. A. Delong, B. W. Johnson, and J. A. Profeta, "A fault injection technique for VHDL behavioral-level models," *IEEE Design & Test of Computers*, 1996, vol. 13, no. 4, pp. 24-33.

[99] R. R. Some et al., "Fault injection experiment results in space borne parallel application programs," in *Aerospace Conference Proceedings*, vol. 5, Big Sky, MT, USA, 2002.

[100] P. K. Lala, "Transient and permanent fault injection in VHDL description of digital circuits," *Circuits and Systems*, 2012, vol. 3, no. 2.

[101] S. Boujait and Amokranne A., *Développement des éléments embarqués (HW/SW) d'un robot mobile à base de carte FPGA*.: Mémoire de Master, Ecole Nationale Supérieure de Technologie (ENST) - Rouiba, 2014.

[102] R. Kaid-Youcef and N. Taghirint, *Réalisation d'un système de commande sans fil par interface LabVIEW pour un robot mobile autonome*.: Mémoire de Master, Ecole Nationale Supérieure de Technologie (ENST) - Rouiba, 2014.

[103] M. Otmane, *Conception de la partie transmission d'un lecteur RFID UHF à base de FPGA*.: Mémoire de Master, Ecole Nationale Supérieure de Technologie (ENST) - Rouiba, 2014.

[104] O. Abdelmalek, *Design and prototyping of robust architectures for UHF RFID Tags*. Doctoral dissertation, Université Grenoble Alpes, 2016.

[105] O. Abdelmalek, D. Hely, and V. Beroulle, "Emulation based fault injection on UHF RFID transponder," in *17th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*, Warsaw, Poland, 2014, pp. 254-257.

[106] D. Hely, V. Beroulle, and A. Omar, "Triggering Hardware Trojans in EPC C1G2 RFID Tags," in *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, Avignon, France, 2013.

[107] O. Abdelmalek, D. Hély, and V. Beroulle, "Fault tolerance evaluation of RFID tags," in *15th Latin American Test Workshop-LATW*, Fortaleza, Brazil, 2014.

[108] O. Abdelmalek, D. Hély, and V. Beroulle, "EPC Class 1 GEN 2 UHF RFID tag emulator for robustness evaluation and improvement," in *8th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, Abu Dhabi, UAE, 2013, pp. 20-24.

[109] I. Mezzah, H. Chemali, and O. Kermia, "Emulation-based fault analysis on RFID tags for robustness and security evaluation," *Microelectronics Reliability*, 2017, vol. 69, pp. 115-125.

[110] A. Ejlali and S. G. Miremadi, "Error propagation analysis using FPGA-based SEU-fault injection," *Microelectronics Reliability*, 2008, vol. 48, no. 02, pp. 319–328.

[111] M. G. Valderas, L. Entrena, R. F. Cardenal, C. L. Ongil, and M. P. García, "SET emulation under a quantized delay model," *Journal of Electronic Testing*, 2009, vol. 25, no. 01, pp. 107–116.

[112] G. Wirth, F. L. Kastensmidt, and I. Ribeiro, "Single event transients in logic circuits—load and propagation induced pulse broadening," *IEEE Transactions on Nuclear Science*, 2008, vol. 55, no. 06, pp. 2928–2935.

[113] T. Plos, "Susceptibility of UHF RFID tags to electromagnetic analysis," in *Topics in Cryptology–CT-RSA*, 2008, pp. 288–300.

[114] M. Hutter, J. M. Schmidt, and T. Plos, "Contact-based fault injections and power analysis on RFID tags," in *European Conference on Circuit Theory and Design - ECCTD*, Antalya, Turkey, 2009, pp. 409–412.

[115] T. Plos et al., "Semi-Passive RFID Development Platform for Implementing and Attacking Security Tags," *International Journal of RFID Security and Cryptography*, 2012, vol. 01, pp. 16–24.

[116] P. Civera, L. Macchiarulo, M. Rebaudengo, M. S. Reorda, and M. Violante, "Exploiting FPGA-based techniques for fault injection campaigns on VLSI circuits," in *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, San Francisco, CA, USA, 2001, pp. 250–258.

[117] M. Ebrahimi, A. Mohammadi, A. Ejlali, and S. G. Miremadi, "A fast, flexible, and easy-to-develop fpga-based fault injection technique," *Microelectronics Reliability*, 2014, vol. 54, no. 05, pp. 1000–1008.

[118] Z. Ren and H. Al-Asaad, "Overview of Assertion-Based Verification and its Applications," in *Int'l Conf. Embedded Systems, Cyber-physical Systems, & Applications*, 2016.

[119] *Property Specification Language Reference Manual V1.1*. Accellera Organisation, Napa, CA, USA, 2004.

[120] Download OVL (Open Verification Language). [Online]. (2014) http://accellera.org/downloads/standards/ovl

[121] Accellera Systems Initiative Inc., Ed., *Accellera Standard OVL V2 - Library Reference Manual*. Napa, CA, USA, march 2014, accellera.org.

[122] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in *8th International Symposium on Quality Electronic Design*, Washington - USA, 2007, pp. 613-620.

[123] M. Riazati, S. Mohammadi, Afzali-Kusha A., and Z. Navabi, "Improved Assertion Lifetime via Assertion-Based Testing Methodology," in *International Conference on Microelectronics*, Dhahran, Saudi Arabia, 2006, pp. 48-51.

[124] P. Kubalik, P. Fiser, and H. Kubatova, "Fault tolerant system design method based on self-checking circuits," in *12th IEEE InternationalOn-Line Testing Symposium*, Lake Como, Italy, 2006.

[125] M. Boule and Z. Zilic, "Incorporating efficient assertion checkers into hardware emulation," in *International Conference on Computer Design*, San Jose, CA, USA, 2005, pp. 221-228.

[126] K. Morin-Allory and D. Borrione, "Proven correct monitors from PSL specifications," in *Design, Automation and Test in Europe, DATE'06*, Munich, Germany, 2006.

[127] I. Mezzah, H. Chemali, V. Beroulle, and O. Abdelmalek, "Utilisation des blocs OVL pour la détection de fautes en ligne et application au tag RFID," in *JDoc'13*, Setif, 2013.

[128] M. R. Kakoee, M. Riazati, and S. Mohammadi, "Enhancing the testability of RTL designs using efficiently synthesized assertions," in *9th International Symposium on Quality Electronic Design*, San Jose, CA, USA, 2008, pp. 230-235.

[129] G. Fritz, V. Beroulle, M. D. Nguyen, and D. Hély, "RFID system on-line testing based on the evaluation of the tags Read-Error-Rate," *Journal of Electronic Testing*, 2011, vol. 27, no. 03, pp. 267-276.

[130] R. Kheddam and I. Parissis, "Online monitoring and diagnosis of RFID readers and tags," in *20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, Split, Croatia, 2012, pp. 1-9.

[131] A. Ruhanen et al., "Sensor-enabled RFID tag handbook," vol. 33546, 2008.

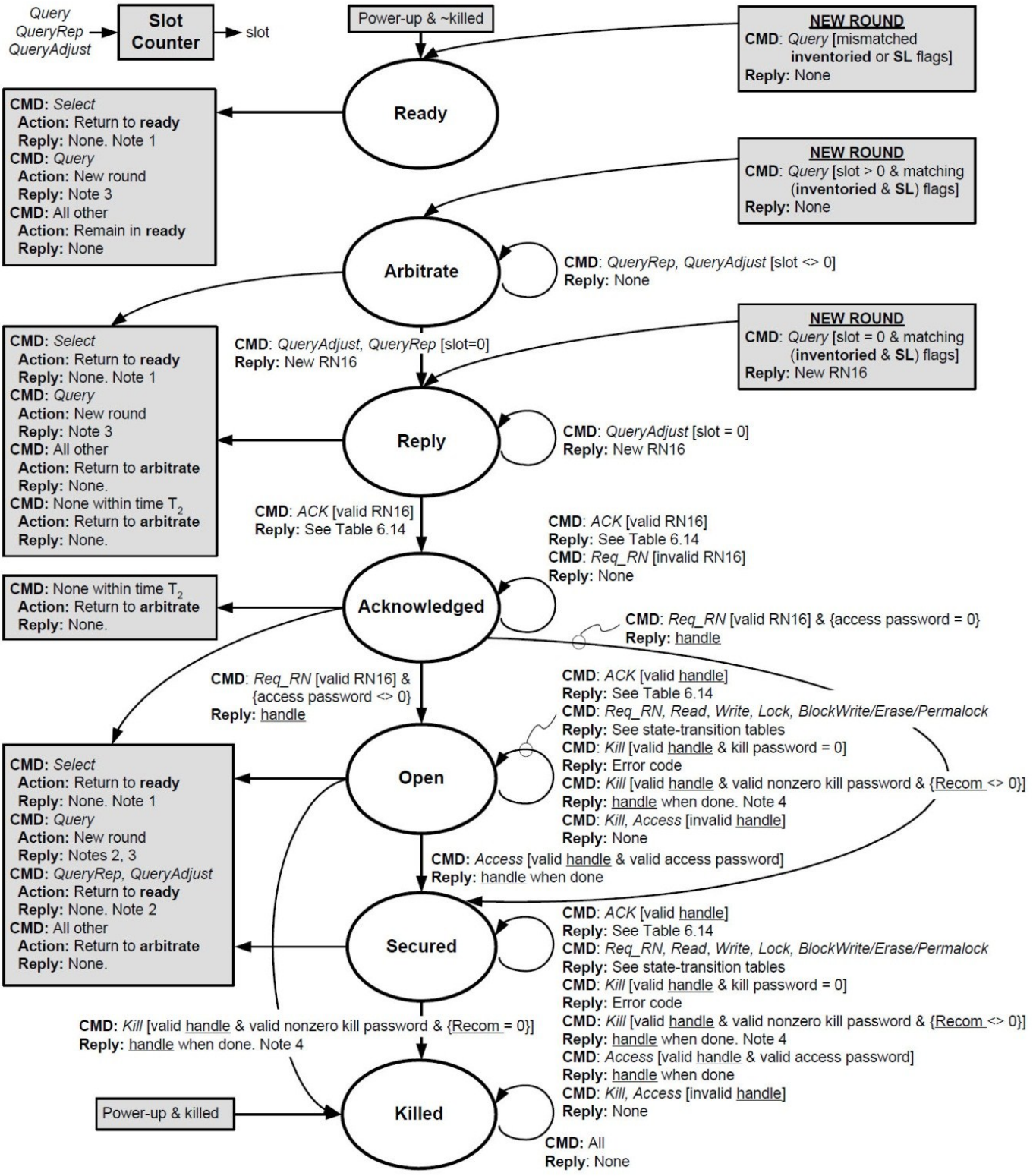
[132] M. Aliouat, Z. Aliouat, and M. Naidja, "Adaptative nodes diagnosis and recovery for Wireless Sensor Networks," in *IEEE Symposium on Computer Applications and Industrial Electronics*, Kota Kinabalu, Malaysia, 2012, pp. 256-261.

Liste des publications

- Mezzah I., Chemali H., Kermia O., “Emulation-based fault analysis on RFID tags for robustness and security evaluation,” in *Microelectronics Reliability*, Volume 69, February 2017, pp. 115-125.
- Mezzah I., Chemali H., Mezzah S., Kermia O., & Abdelmalek O., “MCIP: High configurable 8-bit microcontroller IP-core,” in *Science and Information Conference (SAI)*, London, July 2015, pp. 1387-1390, IEEE.
- Mezzah I., Kermia O., Chemali H., Abdelmalek O., Beroulle V., & Hély D., “Assertion based on-line fault detection applied on UHF RFID tag,” in *8th International Design and Test Symposium (IDT)*, Marrakesh, December 2013, pp. 1-5, IEEE.
- Abdelmalek O., Hely D., Beroulle V., & Mezzah I., “An UHF RFID Emulation Platform with Fault Injection and real time Monitoring capabilities,” in *8th International Design and Test Symposium (IDT)*, Marrakesh, December 2013, pp. 1-2, IEEE.
- Mezzah I., Chemali H., Beroulle V., & Abdelmalek O., “Utilisation des blocs OVL pour la détection de fautes en ligne et application au tag RFID,” in *JDoc’13*, Setif, April 2013.
- Mezzah I., & Chemali H., “Conception d’une infrastructure-IP pour le contrôle de fiabilité des systèmes sur puce,” in *JDoc’12*, Setif, April 2012.
- Mezzah I., & Chemali H., “Diagnostic dans les systèmes embarqués,” in *JDoc’11*, Setif, May 2011.
- Mezzah I., Chemali H., & Mezzah S., “Design of a structured IP microcontroller,” in *International Conference on Applied Informatics (ICAI 09)*, Bordj Bou Arreridj, November 2009.
- Mezzah I., Chemali H., & Mezzah S., “Développement d’un Microcontroleur IP en VHDL et Implémentation sur FPGA Spartan-3E,” in *2nd Embedded System Conference*, Bordj El-Bahri, Algiers, May 2009.

Annexe A

Diagramme d'état détaillé du Tag selon EPC Gen2 [65]



NOTES

1. *Select*: Assert/deassert **SL** or set **inventoried** to *A* or *B*.
2. *Query*: *A*→ *B* or *B*→ *A* if the new session matches the prior session; otherwise no change to the **inventoried** flag.
   QueryRep/QueryAdjust: *A*→ *B* or *B*→ *A* if the session matches the prior *Query*; otherwise, the command is invalid and ignored by the Tag.
3. *Query* starts a new round and may change the session. Tags may go to **ready**, **arbitrate**, or **reply**.
4. If a Tag does not implement recommissioning then the Tag treats nonzero Recom bits as though Recom = 0.

Annexe B

Rapport de synthèse du G2FSM sous Cadence Encounter 14.11

```
====================================================================
  Generated by:           Encounter(R) RTL Compiler RC14.11 - v14.10-s012_1
  Generated on:           May 03 2016  01:35:47 pm
  Module:                 TagRFID
  Technology libraries:   tcb018gbwp7twc 270
                          tpd018nvwc 280a
  Operating conditions:   WCCOM (balanced_tree)
  Wireload mode:          segmented
  Area mode:              timing library
====================================================================

       Instance          Cells  Cell Area  Net Area  Total Area    Wireload
-------------------------------------------------------------------------------------
TagRFID                   3036      87372         0       87372 ZeroWireload (S)
  DataMemory               465      26432         0       26432 ZeroWireload (S)
  CommandeDecoder          386       9262         0        9262 ZeroWireload (S)
  BackscatterGenerator     183       6340         0        6340 ZeroWireload (S)
    CRC16_trans             52       1515         0        1515 ZeroWireload (S)
  PIEdecoder               284       6302         0        6302 ZeroWireload (S)
  SlotCounter              117       2793         0        2793 ZeroWireload (S)
  FM0_MillerEncoder        117       2656         0        2656 ZeroWireload (S)
    Miller_module           39        921         0         921 ZeroWireload (S)
    FM0_module               9        251         0         251 ZeroWireload (S)
  MemoryController          92       2070         0        2070 ZeroWireload (S)
  ReplyClockGenerator      111       1910         0        1910 ZeroWireload (S)
  T1_TimerT                103       1796         0        1796 ZeroWireload (S)
  PRNGenerator              33       1355         0        1355 ZeroWireload (S)
  FlagSessionModule         54       1279         0        1279 ZeroWireload (S)
  CRCdecoder                29       1225         0        1225 ZeroWireload (S)
    Inst_crc_16             20        906         0         906 ZeroWireload (S)
    Inst_crc_5               9        320         0         320 ZeroWireload (S)
  T2_Timer                  24        533         0         533 ZeroWireload (S)

 (S) = wireload was automatically selected
```

Annexe C

OVL Quick Reference

TYPE	NAME	PARAMETERS	PORTS	DESCRIPTION
Single-Cycle	**assert_always**	#(severity_level, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must always hold
Two Cycles	**assert_always_on_edge**	#(severity_level, **edge_type**, property_type, msg, coverage_level)	(clk, reset_n, **sampling_event**, **test_expr**)	test_expr is true immediately following the specified edge (edge_type: 0=no-edge, 1=pos, 2=neg, 3=any)
n-Cycles	**assert_change**	#(severity_level, **width**, **num_cks**, **action_on_new_start**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**)	test_expr must change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	**assert_cycle_sequence**	#(severity_level, **num_cks**, **necessary_condition**, property_type, msg, coverage_level)	(clk, reset_n, **event_sequence**)	if the initial sequence holds, the final sequence must also hold (necessary_condition: 0=trigger-on-most, 1=trigger-on-first, 2=trigger-on-first-unpipelined)
Two Cycles	**assert_decrement**	#(severity_level, **width**, **value**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	if test_expr changes, it must decrement by the value parameter (modulo 2^width)
Two Cycles	**assert_delta**	#(severity_level, **width**, **min**, **max**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	if test_expr changes, the delta must be >=min and <=max
Single Cycle	**assert_even_parity**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must have an even parity, i.e. an even number of bits asserted
Two Cycles	**assert_fifo_index**	#(severity_level, **depth**, **push_width**, **pop_width**, property_type, msg, coverage_level, **simultaneous_push_pop**)	(clk, reset_n, **push**, **pop**)	FIFO pointers should never overflow or underflow
n-Cycles	**assert_frame**	#(severity_level, **min_cks**, **max_cks**, **action_on_new_start**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**)	test_expr must not hold before min_cks cycles, but must hold at least once by max_cks cycles (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	**assert_handshake**	#(severity_level, **min_ack_cycle**, **max_ack_cycle**, **req_drop**, **deassert_count**, **max_ack_length**, property_type, msg, coverage_level)	(clk, reset_n, **req**, **ack**)	req and ack must follow the specified handshaking protocol
Single-Cycle	**assert_implication**	#(severity_level, property_type, msg, coverage_level)	(clk, reset_n, **antecedent_expr**, **consequent_expr**)	if antecedent_expr holds then consequent_expr must hold in the same cyle
Two Cycles	**assert_increment**	#(severity_level, **width**, **value**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	if test_expr changes, it must increment by the value parameter (modulo 2^width)
Single-Cycle	**assert_never**	#(severity_level, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must never hold
Single-Cycle	**assert_never_unknown**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **qualifier**, **test_expr**)	test_expr must never be an unknown value, just boolean 0 or 1
Combinatorial	**assert_never_unknown_async**	#(severity_level, **width**, property_type, msg, coverage_level)	(reset_n, **test_expr**)	test_expr must never go to an unknown value asynchronously, it must remain boolean 0 or 1
n-Cycles	**assert_next**	#(severity_level, **num_cks**, **check_overlapping**, **check_missing_start**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**)	test_expr must hold num_cks cycles after start_event holds
Two Cycles	**assert_no_overflow**	#(severity_level, **width**, **min**, **max**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	if test_expr is at max, in the next cycle test_expr must be >min and <=max
Two Cycles	**assert_no_transition**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**, **start_state**, **next_state**)	if test_expr==start_state, in the next cycle test_expr must not change to next_state
Two Cycles	**assert_no_underflow**	#(severity_level, **width**, **min**, **max**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	if test_expr is at min, in the next cycle test_expr must be >=min and <max
Single-Cycle	**assert_odd_parity**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must have an odd parity, i.e. an odd number of bits asserted
Single-Cycle	**assert_one_cold**	#(severity_level, **width**, **inactive**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must be one-cold i.e. exactly one bit set low (inactive: 0=also-all-zero, 1=also-all-ones, 2=pure-one-cold)
Single-Cycle	**assert_one_hot**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must be one-hot i.e. exactly one bit set high
Combinatorial	**assert_proposition**	#(severity_level, property_type, msg, coverage_level)	(reset_n, **test_expr**)	test_expr must hold asynchronously (not just at a clock edge)
Two Cycles	**assert_quiescent_state**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **state_expr**, **check_value**, **sample_event**)	state_expr must equal check_value on a rising edge of sample_event (also checked on rising edge of `OVL_END_OF_SIMULATION)
Single-Cycle	**assert_range**	#(severity_level, **width**, **min**, **max**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must be >=min and <=max
n-Cycles	**assert_time**	#(severity_level, **num_cks**, **action_on_new_start**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**)	test_expr must hold for num_cks cycles after start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
Two Cycles	**assert_transition**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**, **start_state**, **next_state**)	if test_expr changes from start_state, then it can only change to next_state
n-Cycles	**assert_unchange**	#(severity_level, **width**, **num_cks**, **action_on_new_start**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**)	test_expr must not change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	**assert_width**	#(severity_level, **min_cks**, **max_cks**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must hold for between min_cks and max_cks cycles
Event-bound	**assert_win_change**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**, **end_event**)	test_expr must change between start_event and end_event
Event-bound	**assert_window**	#(severity_level, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**, **end_event**)	test_expr must hold after the start_event and up to (and including) the end_event
Event-bound	**assert_win_unchange**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **start_event**, **test_expr**, **end_event**);	test_expr must not change between start_event and end_event
Single-Cycle	**assert_zero_one_hot**	#(severity_level, **width**, property_type, msg, coverage_level)	(clk, reset_n, **test_expr**)	test_expr must be one-hot or zero, i.e. at most one bit set high

PARAMETERS

severity_level
- `OVL_FATAL
- `OVL_ERROR
- `OVL_WARNING
- `OVL_INFO

property_type
- `OVL_ASSERT
- `OVL_ASSUME
- `OVL_IGNORE

msg descriptive string

USING OVL

+define+OVL_ASSERT_ON

+define+OVL_MAX_REPORT_ERROR=1

+define+OVL_INIT_MSG

+define+OVL_INIT_COUNT=<tbench>.ovl_init_count

+libext+.v+.vlib

-y <OVL_DIR>/std_ovl

+incdir+<OVL_DIR>/std_ovl

DESIGN ASSERTIONS

Monitors internal signals & Outputs

Examples
- * One hot FSM
- * Hit default case items
- * FIFO / Stack
- * Counters (overflow/increment)
- * FSM transitions
- * X checkers (assert_never_unknown)

INPUT ASSUMPTIONS

Restricts environment

Examples
- * One hot inputs
- * Range limits e.g. cache sizes
- * Stability e.g. cache sizes
- * No back-to-back reqs
- * Handshaking sequences
- * Bus protocol

الملخص

تعرف الأنظمة المضمنة في السنوات الأخيرة انتشارا كثيفا خاصة مع التزايد الكبير في أعداد الأجهزة المتصلة التي من المتوقع أن تصل إلى مئات المليارات في المستقبل القريب. يواجه هذا النمو تحديات معتبرة تتعلق بموثوقية وأمن هذه الأنظمة، الأمر الذي يستدعي المزيد من الحاجة إلى جعل هذه الأنظمة أكثر ذكاء و ذلك فيما يخص تحسين قدرتها على ضمان مستويات عالية جدا من الأمن و الموثوقية في الأداء. ساعدت تقنية القياس على الرقائق على تطوير حلول فعّالة في مجالات اختبار الرقائق الإلكترونية وكشف أخطاء التصميم. يتركز عمل هذه الأطروحة على تطوير طرق جديدة وتقنيات قياس على الرقائق بما يسمح للأنظمة الإلكترونية الحصول على بعض الذكاء الذي يمكنها من كشف الأخطاء خلال التشغيل والقيام بعملية التشخيص لضمان مستوى إستثنائي لموثوقية التشغيل و مراقبة مستمرة للموثوقية عموما. تم تطبيق هذا النهج على أنظمة التعرف بموجات الراديو (RFID) التي تواجه تحديات كبيرة فيما يخص الجانب الأمني. قمنا في هذا الإطار بتصميم وسائل تجريبية عدة تتضمن متحكم (MCU)، بطاقتا RFID، قارئ RFID و نظام حقن الأخطاء و ذلك باستخدام دارات FPGA. استعملت هذه الوسائل في إجراء بعض تجارب حقن الأخطاء على دارة بطاقة RFID. ساعدت هذه التجارب على فهم تأثير الأخطاء على سلامة بطاقات RFID. تم أيضا تطوير تقنية جديدة تساعد على كشف الأخطاء خلال التشغيل و ذلك باستخدام وحدات مراقبة، الأمر الذي ساهم في تحسين السلامة و مراقبة الموثوقية.

الكلمات الدالة: الأنظمة المضمنة ؛ RFID ؛ حقن الأخطاء ؛ الأمن ؛ الموثوقية ؛ التشخيص ؛ FPGA

Résumé

Les systèmes embarqués connaissent ces dernières années un déploiement intense notamment avec la super croissance du nombre des objets connectés qui devrait atteindre des centaines de milliards dans un futur proche. Cette croissance est confrontée à des défis colossaux concernant la fiabilité et la sécurité de ces systèmes ; ce qui accentue, de plus en plus, la nécessité de concevoir des systèmes plus intelligents dans le sens où ils sont capables d'assurer des degrés très élevés de sécurité et de sureté de fonctionnement. L'instrumentation sur puce a permis continuellement de produire des solutions effectives dans les domaines de test et de débogage des circuits électroniques. Les travaux de cette thèse se focalisent sur le développement de nouveaux moyens et techniques d'instrumentation sur puce qui permettent aux systèmes électroniques d'avoir une certaine intelligence qui leur permettent d'effectuer des tâches de détection de fautes et de diagnostic en ligne, assurant ainsi une haute sureté de fonctionnement et un contrôle de fiabilité permanent. Cette approche est appliquée à un système embarqué communicant connu comme étant la nouvelle tendance en matière d'identification et d'authentification d'objet ou de personne, en l'occurrence la RFID (Radio Frequency Identification). L'invasion de cette dernière, dans le quotidien, soulève de grandes interrogations et craintes de la part des consommateurs et des autorités vis-à-vis de sa fiabilité et de son respect de la vie privée. Pour ce faire, une plateforme d'expérimentation a été développée qui comprend un microcontrôleur configurable, deux architectures de tag RFID, un lecteur RFID et une plateforme d'injection de fautes. Des analyses approfondies d'injection de fautes de différents types, réalisées sur des tags RFID en utilisant des FPGA, ont permis de démystifier l'impact des fautes sur la sécurité des tags. Des techniques de détection en ligne de fautes ont été aussi développées par l'utilisation des blocs d'assertion permettent ainsi d'améliorer la sécurité et de contrôler la fiabilité de ces systèmes.

Mots-clés: Systèmes embarqués ; RFID ; Injection de fautes ; Sécurité ; Fiabilité ; Diagnostic ; FPGA.

Abstract

Recently, Embedded Systems have experienced an intensive expansion due to the sustained growth of the number of connected things which is expected to reach hundreds of billions in a near future. This growth is facing tremendous challenges regarding reliability and system security. These challenges emphasize the necessity of designing more reliable systems able to ensure high level security and dependability. On chip instrumentation has been the means to produce continuously effective solutions for test area and for debugging electronic circuits. The proposed work targets the development of new on-chip instrumentation means and technics enabling electronic systems to perform autonomous online diagnostic and fault detection tasks achieving a high safety level and a permanent reliability monitoring. This approach is applied on a communicating embedded system known as a new trend in human and things identification and authentication, namely RFID (Radio Frequency Identification). Our daily life is, indeed, overwhelmed of these new gadgets which, increasingly, raise deep questions and concerns about their reliability and privacy. To that end, an experimentation platform has been developed that includes a configurable microcontroller, two RFID tag architectures, an RFID reader and a fault injection platform. Several types' faults injection thorough analysis, carried out on RFID tag using FPGA, allowed enlightening faults impact on tags. Online fault detection technics have been, as well, developed on the basis of assertion blocks in order to enhancing security and to monitor the reliability of these systems.

Keywords: Embedded systems ; RFID ; Fault injection ; Security ; Reliability ; Diagnosis ; FPGA.