Conception de matériel salutaire pour lutter contre la contrefaçon et le vol de circuits intégrés

Cédric Marchand

LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

UNIVERSITÉ DE LYON

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

edsis 488
SCIENCES • INGÉNIERIE • SANTÉ

Thèse

pour obtenir le grade de docteur

de l'Université de Lyon

Discipline : **Microélectronique**

Équipe : **Systèmes Embarqués Sécurisés**

Conception de matériel salutaire pour lutter contre la contrefaçon et le vol de circuits intégrés

---

Cédric MARCHAND

La soutenance de thèse est prévue pour le 24 novembre 2016 avec le jury suivant

Directeur de thèse :	
Lilian Bossuet	**LaHC, France**
Président du jury :	
Gildas Avoine	**IRISA, France**
Rapporteurs :	
Sylvain Guilley	**TELECOM ParisTech, France**
Arnaud Tisserand	**IRISA, France**
Examinateurs :	
Viktor Fischer	**LaHC, France**
Julien Francq	**Airbus Defence & Space - CyberSecurity, France**
Lionel Torres	**LIRMM, France**

Remerciement

Le parcours menant à un doctorat n'est pas linéaire et chaque étape du parcours étudiant est marquée par des événements qui façonnent peu à peu le projet professionnel de chacun. Pour ma part, je n'aurais pas imaginé m'engager dans une thèse lorsque j'ai commencé une école d'ingénieur en électronique. En effet, j'ai découvert la recherche seulement en deuxième année et ce fut pour moi une étape importante de mon parcours. Je souhaite tout d'abord remercier Jean-Baptiste Rigaud et Julien Francq qui m'ont encadré lors de deux stages du cycle d'ingénieur et grâce à qui j'ai découvert le monde de la recherche. Ensuite, je donne toute ma gratitude à mon directeur de thèse, Lilian Bossuet, qui est venu me chercher dans un petit bureau de chez Airbus Defense & Space Cybersecurity pour me proposer un sujet de thèse. Cette rencontre faite presque par hasard, en fin de journée, s'est aujourd'hui transformé en une véritable collaboration et je suis très heureux d'avoir étudié et travaillé sous sa direction.

Les trois années de doctorat que j'ai passé à Saint-Étienne ont pour moi été une expérience très enrichissante et très agréable. Je remercie pour cela l'ensemble des membres de l'équipe SESAM (systèmes embarqués sécurisés et architectures matérielles). J'ai également une pensée pour l'équipe CERG (Cryptographic Engineering Research Group) de l'université George Mason aux États-Unis. En particulier, je voudrais remercier le Docteur Kris Gaj qui m'a permis de faire un séjour de trois mois dans son équipe de recherche.

Enfin, je souhaiterai remercier toutes les personnes qui m'ont soutenu et qui ont cru en moi. En particulier, je pense à mes parents qui m'ont toujours aidé et soutenu. Je pense également à Claudine Charrondière qui m'a beaucoup aidé et encouragé lors de la rédaction, et sans qui mon séjour à Saint-Étienne n'aurait pas été ce qu'il a été.

Résumé

Le vol et la contrefaçon touchent toutes les sphères industrielles de nos sociétés. En particulier, les produits électroniques représentent la deuxième catégorie de produits la plus concernée par ces problèmes. Parmi les produits électroniques les plus touchés, on retrouve les téléphones mobiles, les tablettes, les ordinateurs mais aussi des éléments bien plus basiques comme des circuits analogiques ou numériques et les circuits intégrés. Ces derniers sont au cœur de la plupart des produits électroniques et un téléphone mobile peut être considéré comme contrefait s'il possède ne serait-ce qu'un seul circuit intégré contrefait. Le marché de la contrefaçon de circuits intégrés représente entre 7 et 10% du marché total des semi-conducteurs, ce qui implique une perte d'au moins 24 milliards d'euros en 2015 pour les entreprises concevant des circuits intégrés. Ces pertes pourraient s'élever jusqu'à 36 milliards d'euros en 2016. Il est donc indispensable de trouver des solutions pratiques et efficaces pour lutter contre la contrefaçon et le vol de circuits intégrés.

Le projet SALWARE, financé par l'Agence Nationale de la Recherche et par la Fondation de Recherche pour l'Aéronautique et l'Espace, a pour but de lutter contre le problème de la contrefaçon et du vol de circuits intégrés et propose l'étude et la conception de matériels salutaires (ou *salwares*). En particulier, l'un des objectifs de ce projet est de combiner astucieusement plusieurs mécanismes de protection participant à la lutte contre la contrefaçon et le vol de circuits intégrés, pour construire un système d'activation complet. L'activation des circuits intégrés après leur fabrication permet de redonner leur contrôle au véritable propriétaire de la propriété intellectuelle.

Dans ce manuscrit de thèse, nous proposons l'étude de trois mécanismes de protection participant à la lutte contre la contrefaçon et le vol de circuits intégrés. Dans un premier temps, nous étudierons l'insertion et la détection de *watermarks* dans les machines à états finies des systèmes numériques synchrones. Ce mécanisme de protection permet de détecter un vol ou une contrefaçon. Ensuite, une fonction physique non-clonable basée sur des oscillateurs en anneaux dont les oscillations sont temporaires est implantée et caractérisée sur FPGA. Ce mécanisme de protection permet d'identifier un circuit grâce à un identifiant unique créé grâce aux variations du processus de fabrication des circuits intégrés. Enfin, nous aborderons l'implantation matérielle d'algorithmes légers de chiffrement par bloc, qui permettent d'établir une communication sécurisée au moment de l'activation d'un circuit intégré.

Abstract

Counterfeiting and theft affects all industrial activities in our society. Electronic products are the second category of products most concerned by these issues. Among the most affected electronic products, we find mobile phones, tablets, computers as well as more basic elements such as analog and digital circuits or integrated circuits. These are the heart of almost all electronic products and we can say that a mobile phone is counterfeited if it has at least one counterfeit integrated circuit inside. The market of counterfeit integrated circuit is estimated between 7 and 10% of the global semi-conductors market, which represents a loss of at least 24 billion euros for the lawful industry in 2015. These losses could reach 36 billion euros in 2016. Therefore, there is an absolute necessity to find practical and efficient methods to fight against counterfeiting and theft of integrated circuits.

The SALWARE project, granted by the French "Agence Nationale de la Recherche" and by the "Fondation de Recherche pour l'Aéronautique et l'Espace", aims to fight against the problem of counterfeiting and theft of integrated circuits. For that, we propose to design salutary hardwares (*salwares*). More specifically, we propose to cleverly combine different protection mechanisms to build a complete activation system. Activate an integrated circuit after its manufacturing helps to restore the control of integrated circuits to the true owner of the intellectual property.

In this thesis, we propose the study of three different protection mechanisms fighting against counterfeiting and theft of integrated circuits. First, the insertion and the detection of watermark in the finite state machine of digital and synchronous systems will be studied. This mechanism helps to detect counterfeit or theft parts. Then, a physical unclonable function based on transcient effect ring oscillator is implemented and characterized on FPGA. This protection mechanism is used to identify integrated circuit with a unique identifier created thanks to the extraction of entropy from manufacturing process variations. Finally, we discuss the hardware implementations of lightweight block ciphers, which establish a secure communication during the activation of an integrated circuit.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

ADN :	Acide désoxyribonucléique
AES :	Algorithme standard de chiffrement symétrique (*Advanced Encryption Standard*)
AGMA :	Alliance américaine contre le marché gris et la contrefaçon (*Alliance for gray market and counterfeit abatement*)
ALM :	Module logique adaptable (*Adaptative logic module*)
ANR :	Agence Nationale de la Recherche
ASIC :	Circuit électronique spécialisé (*Application-Specific Integrated Circuit*)
BRAM :	Bloc de mémoire vive (*Block of random access memory*)
CLB :	Bloc logique configurable (*Configurable logic block*)
CMOS :	Type de transistor : *Complementary Metal Oxide Semiconductor*
Code QR :	Code bar à réponse rapide (*Quick Response Code*)
DES :	Algorithme de chiffrement standard (*Data Encryption Standard*)
DSP :	Processeur de signal numérique (*Digital Signal Processor*)
***efuse* :**	Fusible électrique (*Electronic fuse*)
EM :	Électromagnétique
FRAE :	Fondation de Recherche pour l'Aéronautique et l'Espace
FSM :	Machine à états finie (*Finite State Machine*)
FPGA :	Circuit logique programmable (*Field Programmable Gate Array*)
GAO :	Cour des comptes américaine (*Government Accountability Office*)
GDSII :	Format de fichier binaire utilisé pour représenter le dessin physique d'un circuit intégré
GE :	Porte équivalente (*Gate Equivalent*)
GFN :	Réseau de Feistel généralisé (*Generalized Feistel Network*)
HCI :	Type de dégradation touchant les transistors NMOS (*Hot Carrier Injection*)
HD :	Distance de Hamming
IC :	Circuit intégré (*Integrated Circuit*)
IP :	Propriété intellectuelle (*Intellectual Property*)
LAB :	Tableau de bloc logique (*Logic Array Block*)

LSFR : Registre à décalage à rétroaction linéaire (*Linear feedback shift register*)
LUT : Table de correspondance (*Look Up Table*)
LUTx : Table de correspondance à x entrées.
MSB : Bit de poids le plus fort (*Most Significant Bit*)
MUXxtoy : Multiplexeur sélectionnant x bits parmi y
NBTI : Type de dégradation touchant les transistors PMOS (*Negative-Bias Temperature Instability*)
NIST : *National Institute of Standards and Technology*
NMOS : Type de transistor : *N-type Metal-Oxide-Semiconductor*
OTP : Mémoire programmable une seule fois (*One Time Programmable memory*)
PMOS : Type de transistor : *P-type Metal-Oxide-Semiconductor*
PUF : Fonction physique non-clonable (*Physical Unclonable Function*)
qsf : Fichier de paramètres quartus (*Quartus Setting File*)
RFID : Radio-identification (*Radio Frequency Identification*)
RO : Oscillateur en anneau (*Ring Oscillator*)
ROM : Mémoire morte (*Read Only Memory*)
Sbox : Tableau de substitution (*Substitution Box*)
SEM : Microscope électronique à balayage (*Scanning electron microscope*)
SPN : Réseaux de substitution et de permutation (*Substitution and Permutation Network*)
SRAM : Mémoire vive statique (*Static Random Access Memory*)
SVM : Machine à vecteurs de support (*Support Vector Machine*)
TEM : Microscope électronique en transmission (*Transmission Electron Microscope*)
TERO : Oscillateur en anneau à effet transitoire (*Transient Effect Ring Oscillator*)
TRNG : Générateur de nombres vraiment aléatoires (*True Random Numbers Generator*)
ucf : Fichier de contraintes utilisateur (*User Constraints File*)
USB : Bus universel en série (*Universal Serial Bus*)
VHDL : Langage de description matérielle (*Very high speed integrated circuit Hardware Descritption Language*)

Introduction

Contexte

Dans ce manuscrit de thèse, nous nous intéressons à la lutte contre le vol et la contrefaçon des circuits intégrés. La notion de vol est assez concrète et facile à appréhender. En revanche, celle de contrefaçon est beaucoup plus floue. Pourtant, la contrefaçon est massivement présente dans nos sociétés. Elle touche toutes les sphères industrielles depuis que l'Homme fait du commerce et produit des biens de consommation. Des exemples de contrefaçons sont présents dès l'Antiquité et le musée de la contrefaçon de Paris [1] possède notamment des jarres de vin romaines portant un sceau contrefait. La notion de contrefaçon s'est développée conjointement avec l'apparition des premières lois sur la propriété intellectuelle durant la Révolution française. En 1791, un premier texte esquisse la notion de propriété intellectuelle pour les auteurs de pièces de théâtre. En 1793, une loi élargi le concept à tous les auteurs en leur accordant le droit de *«jouir, durant leur vie entière, du droit exclusif de vendre, faire vendre, distribuer leurs ouvrages dans le territoire de la République et d'en céder la propriété en tout ou partie»* [18]. Près d'un siècle plus tard, les révolutions industrielles font apparaître la notion de marque, et la contrefaçon évolue en même temps que l'industrie se développe. Une définition commune de la contrefaçon actuellement utilisée est la suivante : *«La contrefaçon se définit comme la reproduction, l'imitation ou l'utilisation totale ou partielle d'une marque, d'un dessin, d'un brevet, d'un logiciel ou d'un droit d'auteur, sans l'autorisation de son titulaire, en affirmant ou laissant présumer que la copie est authentique.»* [2]

Aujourd'hui, la mondialisation et le libéralisme favorisent la création de contrefaçons dans tous les secteurs. En effet, beaucoup d'entreprises font appel à des pays en voie de développement pour fabriquer leurs produits. La délocalisation et l'externalisation de la production permet de réduire les coûts de fabrication des produits des entreprises, ce qui leur permet d'augmenter leur marge. Malheureusement, ces points positifs vont de paire avec la contrefaçon, qui a fortement augmenté ces dernières années.

Les rapports successifs du département américain de la sécurité intérieure [126] montre l'évolution des saisies de produits contrefaits : entre 2001 et 2015, le nombre d'objets contrefaits saisis par les douanes américaines a été multiplié par 800 avec un nombre de saisies passant d'un peu plus de 3 000 à près de 30 000 [126]. Dans les rapports datés de 2010 à 2015, on constate une nette augmentation du nombre de saisies d'articles électroniques contrefaits. En effet, il est passé de 2 252 en 2010, ce qui représentait 9% du nombre total de saisies, à 5 326 en 2015, ce qui représente

1. `https://musee-contrefacon.com/origine-des-collections/`
2. `http://www.insee.fr/fr/methodes/default.asp?page=definitions/contrefacon.htm`

18% des saisies. La douane française a classé les différents types de produits en fonction du nombre d'articles contrefaits saisis durant l'année 2013 [32]. Dans ce classement, les téléphones mobiles sont classés sixième avec 293 190 articles saisis, et les équipements électriques, électroniques et informatiques sont classés neuvième avec 98 515 articles saisis. Il est donc primordial de trouver des solutions pratiques et réalistes afin que l'industrie de l'électronique puisse combattre le vol et la contrefaçon de ses produits.

Parmi les produits électroniques visés, on trouve des produits finis comme des téléphones mobiles [32], des tablettes ou des ordinateurs, mais aussi des éléments bien plus petits comme des composants analogiques ou des circuits intégrés. Ainsi, on comprend bien que la contrefaçon est aujourd'hui un réel problème pour tout les secteurs d'activités. La Figure 1 schématise l'ensemble des produits ciblés par la contrefaçon. Dans ce manuscrit de thèse, nous nous intéressons uniquement au problème de la contrefaçon, du vol et de la copie illégale des circuits intégrés (en rouge sur la Figure 1).

FIGURE 1 – Cibles de la contrefaçon et positionnement du projet SALWARE.

Dans le passé, une entreprise qui concevait des circuits intégrés était financièrement capable de les produire. Cependant, la diminution de la taille des transistors ainsi que la complexité croissante des circuits ont provoqué une augmentation de leur coût de production. Par exemple, la transition de la technologie CMOS 32 nm à la technologie 28 nm s'est accompagnée d'une augmentation de 40% du coût de production d'un *wafer* de 300 mm de diamètre et d'une augmentation de 30%

pour la production d'un *wafer* de 450 mm de diamètre [23]. À cause de ce phénomène, de plus en plus d'entreprises qui conçoivent des circuits intégrés ne sont plus capables de les fabriquer elles-mêmes et préfèrent donc externaliser la fabrication de leurs produits. Ainsi, le nombre d'entreprises sans moyen de production (dites *fabless*) a fortement augmenté, ce qui a fait évoluer le paysage industriel de la micro-électronique. Une conséquence de cette évolution est le changement dans la manière de concevoir les circuits électroniques. Cette conception est aujourd'hui basée sur la réutilisation et l'intégration de composants virtuels (connus sous l'acronyme *IP* pour *Intellectual Property* en anglais) dans des systèmes de plus en plus complexes. Nous renouvelons de plus en plus souvent les appareils électroniques que nous utilisons dans la vie de tous les jours (ordinateurs, téléphones portables, tablettes, *etc*). Cela a pour conséquence de réduire le temps de développement des produits électroniques, qui doivent être mis sur le marché de plus en plus rapidement. Toutes ces évolutions soumettent l'industrie de la micro-électronique à une ardente compétition dans laquelle la contrefaçon occupe une part importante.

Dans ce contexte, le projet SALWARE, financé par l'Agence Nationale de la Recherche et par la Fondation de Recherche pour l'Aéronautique et l'Espace, a pour but de lutter contre le problème de la contrefaçon et du vol de circuits intégrés et propose d'étudier et de concevoir des matériels salutaires (ou *salwares*). Le but d'un *salware* est d'apporter une protection contre le vol, la copie illégale et la contrefaçon d'un circuit intégré, contrairement aux matériels malveillants (connus sous le nom *malwares*) dont le but est d'aider une activité malveillante (vol d'identifiants ou de clés secrètes, blocage à distance, *etc*). Ainsi, l'expression *matériel salutaire* fait référence à un système matériel qui est inséré dans un circuit intégré ou un composant virtuel afin de fournir des informations sur la propriété intellectuelle du circuit et de permettre son activation (ou son blocage) à distance. Pour cela, un *salware* doit être difficilement détectable ou difficile à contourner.

Les travaux présentés dans ce manuscrit se concentrent sur l'étude et la mise en place de trois mécanismes de protection développés dans le cadre du projet SALWARE : le *watermarking*, les fonctions physiques non clonables et l'implantation d'algorithmes de chiffrement ultra-légers. Bien que différents, ces trois mécanismes de protection peuvent être astucieusement combinés pour apporter un très haut niveau de sécurité à un circuit intégré.

Contributions

Les principales contributions des travaux présentés dans ce manuscrit de thèse sont :

- la conception d'une méthode de vérification de *watermark* basée sur l'analyse du canal auxiliaire de la consommation de puissance. Un banc de test dédié à cette méthode de vérification a été développé sur Matlab afin d'automatiser la prise de mesures et l'analyse des données au sein du même environnement ;
- la conception et la caractérisation d'une fonction physique non clonable (PUF).

Cette PUF, qui est basée sur des oscillateurs en anneau dont les oscillations sont temporaires (TERO), a été implantée dans deux familles différentes de FPGA (*Field Programmable Gate Array*) : Xilinx et Altera. Un banc de caractérisation dédié au PUF a été développé sur Octave/Matlab. Il est entièrement automatisé et permet de faire une caractérisation en température ou en tension de six cartes FPGA simultanément ;

- la première implantation matérielle de l'algorithme de chiffrement par bloc LILLIPUT sur FPGA. Deux stratégies (parallèle et série) ont été utilisées pour implanter cet algorithme et huit implantations différentes sont comparées pour ASIC et FPGA. De plus, la comparaison de ces implantations matérielles a été effectuée avec celles de trois autres algorithmes légers de chiffrement par bloc : KLEIN, LED et KTANTAN.

Organisation du manuscrit

Le premier chapitre détaille le modèle de menaces ciblant la propriété intellectuelle du concepteur de circuits intégrés et décrit précisément les différents mécanismes de protection, qui ont été proposés dans la littérature. Ce chapitre présente également le projet SALWARE et le schéma de protection contre la contrefaçon et le vol de circuits intégrés qu'il propose.

Le Chapitre 2 présente une méthode d'insertion de *watermark* et une méthode de vérification basée sur le canal auxiliaire de la consommation de puissance. Le banc de test qui a été développé pour expérimenter la méthode de vérification est également présenté dans ce chapitre. Enfin, deux expériences, qui valident le fonctionnement de la méthode de vérification et celui de la méthode d'insertion, sont décrites. Leurs résultats sont présentés à la fin du chapitre.

Le Chapitre 3 présente la conception et la caractérisation du TERO-PUF sur deux cibles FPGA différentes. Le banc de caractérisation en température et en tension qui a été développé y est entièrement décrit. Enfin, les différentes métriques qui sont utilisées pour caractériser et analyser le TERO-PUF sont présentées ainsi que les résultats de la caractérisation.

Le Chapitre 4 présente l'algorithme de chiffrement léger LILLIPUT. Les implantations parallèles et série de cet algorithme y sont détaillées. De plus, une étude complète de la fonction de gestion de clé est présentée et montre que le partage de ressources n'est pas nécessairement la meilleure solution sur FPGA. Enfin, les résultats des implantations sont comparés à la fois pour ASIC et pour FPGA.

Le Chapitre 5 présente la comparaison de l'implantation de LILLIPUT avec les implantations matérielles de trois autres algorithmes légers de chiffrement par bloc : KLEIN, LED et KTANTAN. Dans ce chapitre, seules les implantations sur FPGA sont comparées pour les deux stratégies (parallèle et série).

Chapitre 1

Les mécanismes de protection des circuits intégrés

Les circuits intégrés suivent un cycle de vie complexe, composé de plusieurs étapes et faisant intervenir plusieurs acteurs. Malheureusement, un certain nombre de menaces, pesant sur la propriété intellectuelle du concepteur, apparaît tout au long de la vie de ces circuits. Parmi ces menaces, le vol, la copie illégale et la contrefaçon sont aujourd'hui devenus de réels problèmes, autant pour les entreprises qui conçoivent les circuits que pour les clients qui les utilisent. Ce chapitre s'intéresse à ces menaces et a pour but de donner une vue d'ensemble du contexte des travaux présentés tout au long de ce manuscrit.

Dans une première partie, un modèle de menaces visant les circuits intégrés est mis en place. Les Sections 1.2, 1.3 et 1.4 présentent quelles sont les tentatives de solutions proposées dans la littérature. Le projet SALWARE [27] est présenté dans la Section 1.5. Enfin, la Section 1.6 donne une vue d'ensemble sur l'implantation d'algorithmes de chiffrement par bloc.

1.1 Menaces existantes sur la propriété intellectuelle du concepteur de circuit

1.1.1 Cycle de vie des circuits intégrés et contrefaçon

La conception et la fabrication des circuits intégrés sont aujourd'hui très complexes et peuvent être composées de plus de 400 étapes [29, 30]. Cependant, la vie d'un circuit intégré ne se limite pas à sa conception et à sa fabrication. En effet, une fois un circuit produit, sa vie comporte un temps d'usage, éventuellement précédé d'un temps de stockage et de distribution. Enfin, la vie du circuit se termine lorsque celui-ci ne fonctionne plus ou lorsque il est remplacé (par une version plus récente par exemple). Toutefois, il est possible de simplifier ce cycle de vie des circuits en quatre grandes étapes : (1) la conception, (2) la production, (3) la distribution et le temps d'usage et 4) la fin de vie. La Figure 1.1 schématise ce cycle de vie simplifié. L'entreprise, sans moyen de production, qui a conçu le circuit intégré doit fournir la description complète de son circuit au producteur du circuit. Cette description peut être sous la forme d'un fichier binaire, par exemple GDSII. La production du circuit en elle-même est très complexe et comporte de très nombreuses étapes que nous pouvons résumer en quatre phases : la fabrication des *wafers*, le test des puces, la mise en boîtier des circuits, et le test des circuits intégrés. Une fois produit, le circuit est distribué ou stocké par des courtiers en composants électroniques avant d'être utilisé. Enfin, lorsque l'utilisation du circuit est terminée, le circuit arrive à la fin de sa vie et les composants qui ne peuvent pas être recyclés sont mis en décharge.

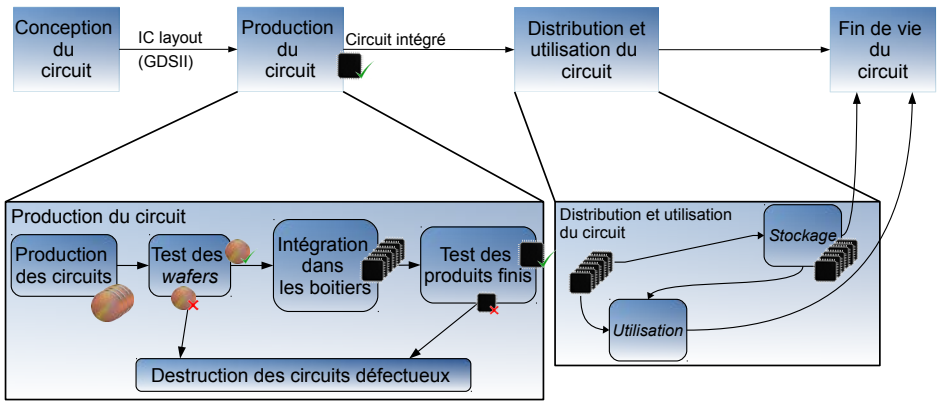

FIGURE 1.1 – Cycle de vie simplifié d'un circuit intégré.

Ce cycle de vie des circuits intégrés fait intervenir de nombreux acteurs, ce qui rend possible le développement de nombreuses menaces pour la propriété intellectuelle du concepteur des circuits intégrés. Dans [69], les auteurs regroupent toutes ces menaces sous la bannière de la contrefaçon, bien que les objectifs de chaque menace puissent être très différents. Dans [57], le vol et la contrefaçon sont regroupés au sein de la même catégorie mais la rétro-ingénierie et les chevaux de Troie

matériels sont mis à part. Dans [140], une classification légèrement plus fine est mise en place avec la séparation de la contrefaçon et du vol de circuit. Ainsi, quatre catégories de menaces sont mises en avant : les chevaux de Troie matériels, le vol par surproduction, la rétro-ingénierie et la contrefaçon. Dans un premier temps, les menaces qui seront détaillées dans cette section (le vol, la contrefaçon et la copie illégale de circuit intégré) seront résumées par le terme « *contrefaçon* ».

Comme dans les autres secteurs d'activités, le marché de la contrefaçon des produits électroniques a pris une ampleur considérable ces dernières années : il a été multiplié par 4 entre 2009 et 2011 et par 700 entre 2001 et 2011 [59]. Le coût de la contrefaçon est aujourd'hui estimé entre 7% et 10% du marché total des semiconducteurs [130], ce qui représente un minimum de 23.4 milliards d'euros en 2015, et pourrait atteindre 34 milliards d'euros en 2016 selon un rapport de la *Global Semiconductor Alliance* [60]. D'autres références estiment le coût de la contrefaçon de produits électroniques à 7.5 milliards de dollars par an pour l'industrie américaine, ce qui représente une perte de 11 000 emplois. En 2010, un rapport de l'*Alliance for Gray Market and Counterfeit Abatement* (AGMA) [48] montre que 83% des incidents liés à la contrefaçon ne sont pas rapportés aux autorités et que seuls 4% des distributeurs de produits électroniques signalent tous les incidents. Le même constat est établi dans [130] et [59]. Le *Government Accountability Office* (GAO), équivalent américain de la Cour des comptes française, déclare clairement dans un rapport de février 2016 la nécessité d'améliorer le taux de déclaration des incidents liés à la contrefaçon [110]. En 2012, ce bureau indépendant a mené une expérience afin de montrer l'importance de ce problème : il a créé une entreprise fictive dans le but d'acheter des composants électroniques pour des applications militaires. Le résultat de cette expérience montre que parmi les composants achetés, 100% étaient des contrefaçons et 80% d'entre eux en provenance de Chine [1].

Parmi les cas de contrefaçon rapportés, nombreux sont ceux qui touchent les systèmes militaires [63, 114]. Des circuits contrefaits ont été trouvés dans des hélicoptères de la marine américaine (Seahawk SH-60B, AH-64, CH-46) et dans des avions de l'U.S. Force (C-17, C-103J, C-27J et P-8A) [59, 149]. Un cas de contrefaçon a également été retrouvé dans des systèmes de missile FLIR, développés par la société américaine Raytheon [149]. Ce qui est alarmant est qu'un composant coûtant moins de deux dollars peut compromettre l'intégralité d'un système coûtant plus de dix millions de dollars, comme le déclare un responsable des programmes d'armement américains [149]. Tous ces produits contrefaits peuvent également se retrouver dans l'industrie civile (aviation, transport publique, ...) et ne visent pas uniquement les marchés militaires. Par exemple, des dizaines de personnes ont été blessées par l'explosion de leur téléphone mobile, due à un composant électronique contrefait [130].

Ces exemples montrent bien que tous les types de produits électroniques sont visés par la contrefaçon. En 2011, les circuits les plus touchés sont les circuits analogiques (25.2%), suivis par les microprocesseurs (13.4%), les mémoires (13.1%), les

1. `http://www.journal-aviation.com/actualites/infos-contrefacon.html`

circuits logiques programmables (8.3%) et les transistors (7.6%) [59]. Il est donc indispensable de ne pas se limiter aux circuits numériques et de proposer des solutions applicables pour la majorité des circuits intégrés [27]. Détaillons maintenant les différentes menaces pesant sur le cycle de vie des circuits intégrés.

1.1.2 Les différentes menaces

La contrefaçon peut prendre diverses formes et les auteurs des articles [64, 155] en donnent une vue d'ensemble. À partir de la Figure 1.1 qui montre une version très simplifiée du cycle de vie des circuits intégrés, il est possible d'ajouter et de positionner les différentes menaces qui existent (Figure 1.2).

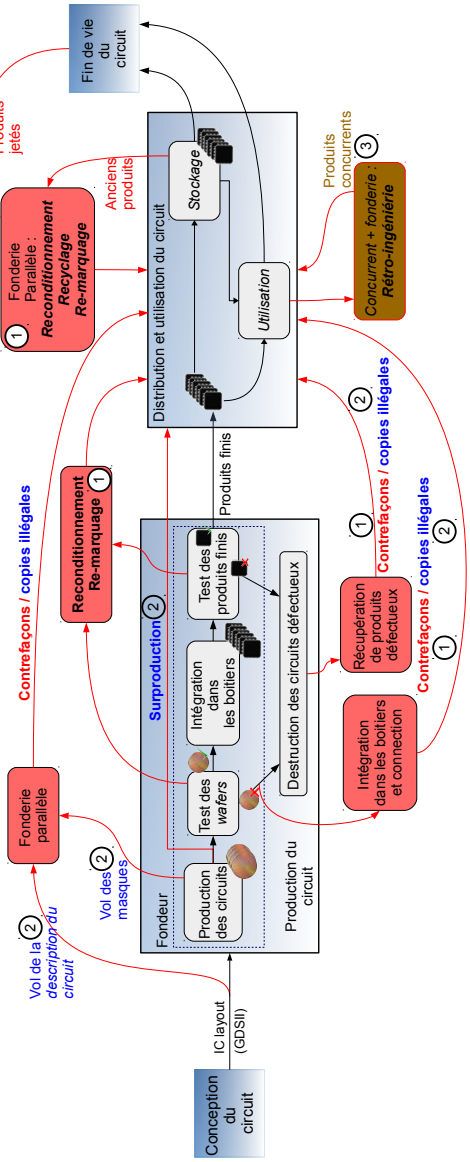

FIGURE 1.2 – Modèle de menaces pesant sur le cycle de vie des circuits intégrés.

Sur la Figure 1.2, trois grandes catégories de menaces (mises en avant par la couleur et un numéro) apparaissent sur le cycle de vie des circuits intégrés : (1) la contrefaçon, (2) le vol ou le clonage de circuits, et (3) la rétro-ingénierie.

1.1.2.1 La contrefaçon

La définition générale de la contrefaçon présentée dans l'introduction de ce manuscrit peut être complétée par des menaces particulières à l'industrie de la microélectronique, comme la remise à neuf de composants usagés ou le changement de boîtier d'un circuit par exemple. Quatre types de contrefaçons peuvent être retrouvés : le changement de boîtier d'une puce (*repackaging*), le changement de marquage d'un circuit intégré (re-marquage), la remise à neuf des circuits usagés (recyclage) et l'imitation d'un circuit intégré (contrefaçon) [155].

Le changement de boîtier est l'une des suites possibles du vol des puces électroniques durant leur fabrication. Comme on peut le voir sur la Figure 1.2, le vol d'une puce peut intervenir à différents moments de la fabrication, depuis le vol de masques de photolithographie, qui permettent de fabriquer le circuit, jusqu'au vol des puces avant conditionnement (pendant le test des *wafers* par exemple). Une autre méthode très utilisée pour voler des puces électroniques est la surproduction. En effet, celle-ci est très difficile à détecter par le concepteur du circuit, qui n'a aucun moyen pour contrôler le nombre d'entités produites une fois qu'il a fourni la description complète du circuit intégré. Ainsi, le contrefacteur obtient une puce strictement identique au circuit qui est produit et peut alors le vendre, en son propre nom, à des prix beaucoup moins importants que le véritable propriétaire du circuit produit. La Figure 1.3 montre un exemple de mémoire Samsung conditionnée dans un boîtier Toshiba [2].

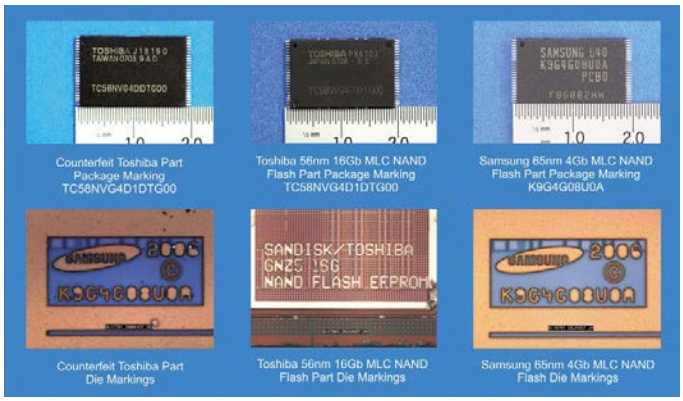

FIGURE 1.3 – Exemple d'un circuit intégré qui a changé de boîtier : le circuit original Toshiba au centre, un circuit d'une technologie plus ancienne de chez Samsung à droite et la contrefaçon du circuit Toshiba avec la puce Samsung à gauche.

2. http://www.eetimes.com/document.asp?doc_id=1281183

La Figure 1.3 montre un exemple de mémoire NAND flash contrefaite par changement de boîtier, on y trouve un circuit original Samsung à droite, un circuit original Toshiba au milieu et on voit que le circuit de gauche possède le boîtier du circuit Toshiba mais que la puce qui est à l'intérieur est celle du circuit Samsung.

Le re-marquage consiste à changer les inscriptions visibles sur le boîtier d'un circuit intégré. Ce marquage comporte généralement le nom du circuit ainsi que le nom et le logo de son concepteur, mais aussi la référence du circuit, son niveau de performance, le type de boîtier, des informations sur sa fabrication (lieu, lot et date de fabrication) et sur ses spécifications (certification militaire ou aéronautique par exemple). La procédure de re-marquage commence par l'effacement (par ponçage et décapage chimique, ou par substance abrasive) ou le camouflage (par peinture ou pose d'un capot supplémentaire) du marquage original. La nouvelle marque est ensuite apposée sur le circuit par marquage laser ou à l'encre. L'impression par jet d'encre est un choix très populaire chez les contrefacteurs car il est peu coûteux, mais il existe d'autres techniques comme le transfert d'encre (*transfer printing*) ou le tamponnage (*stamping*) [82]. De la même manière, plusieurs techniques de marquage laser existent mais les plus utilisées sont les lasers au CO_2 et les lasers au YAG [82]. Deux exemples de circuit re-marqués sont présentés sur la Figure 1.4. À gauche, on voit que l'ancien marquage est encore présent sur le circuit (flèches rouges) [82]. À droite, on peut voir que le marquage du circuit de gauche (grade B) ne correspond pas à celui du circuit original (grade A) [3].

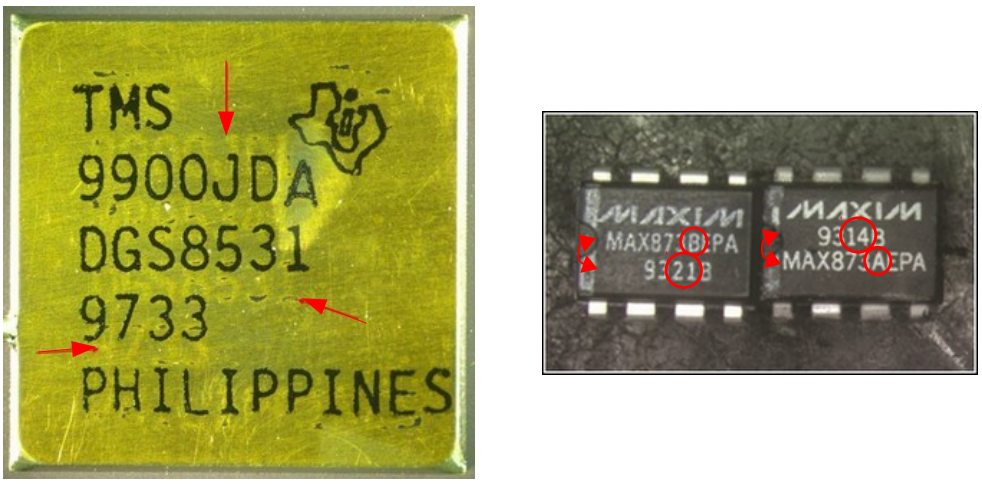

FIGURE 1.4 – Exemples de circuits intégrés re-marqués.

Cette menace fait suite au vol d'un circuit intégré, lors du test final des circuits par exemple. Lors de cette étape de la vie des circuits intégrés, il est possible de déclarer défectueux un circuit parfaitement fonctionnel pour le voler ou encore de

3. https://www.maximintegrated.com/en/app-notes/index.mvp/id/5458

récupérer les circuits mis au rebut pour réel dysfonctionnement. Comme le montrent les exemples de la Figure 1.4 ainsi que [82], ce type de contrefaçon est relativement facile à détecter du fait des imprécisions commises par les contrefacteurs. Le re-marquage de circuits intégrés est toutefois très utilisé en raison de sa facilité de mise en œuvre.

Le recyclage de circuits intégrés consiste à récupérer des composants destinés à la casse (*salvaging*) pour les recycler (*refurbishing*). Même si le recyclage est une excellente chose, la revente de composants usagés comme neufs sans aucune garantie de test, de performance et de longévité est un délit, et ces composants constituent donc des contrefaçons [24]. La plupart du temps, le recyclage des composants électroniques vient d'une décharge de composants (venus illégalement des pays industrialisés [4] [5]). Dans ces décharges, les circuits sont retirés des cartes électroniques sans aucune précaution pour la santé des personnes y travaillant, qui risquent notamment cancers et problèmes neurologiques[4]. Une fois extraits, les composants récupérés sont revendus à l'unité ou ré-assemblés dans d'autres produits électroniques [130], sur Internet [54, 133] ou dans des centres spécialisés (comme ceux de la ville de Shenzen en Chine [6]). Dans le but de paraître neufs, les composants électroniques et circuits intégrés recyclés peuvent être soumis à un changement de boîtier ou à un re-marquage. La Figure 1.5 présente quelques exemples de personnes travaillant dans des décharges et recyclant des composants électroniques.

FIGURE 1.5 – Recyclage de composants électroniques dans des décharges.

Dans cette première partie dédié aux menaces pesant sur le cible de vie des circuits intégrés, nous avons décrit trois différentes façons de créer une contrefaçon d'un circuit : le changement de boîtier (*repackaging*), le re-marquage (*relabeling*) et le recyclage (*salvaging/refurbishing*). La Figure 1.6 résume les trois cas de contrefaçons qui viennent d'être abordés [43].

La Sous-figure 1.6.a montre le circuit original avec la puce, le boîtier ainsi que

4. `http://future.arte.tv/fr/la-tragedie-electronique`
5. `http://boutique.arte.tv/f9742-tragedie_electronique`
6. article du Monde : `https://lc.cx/4Vc5`

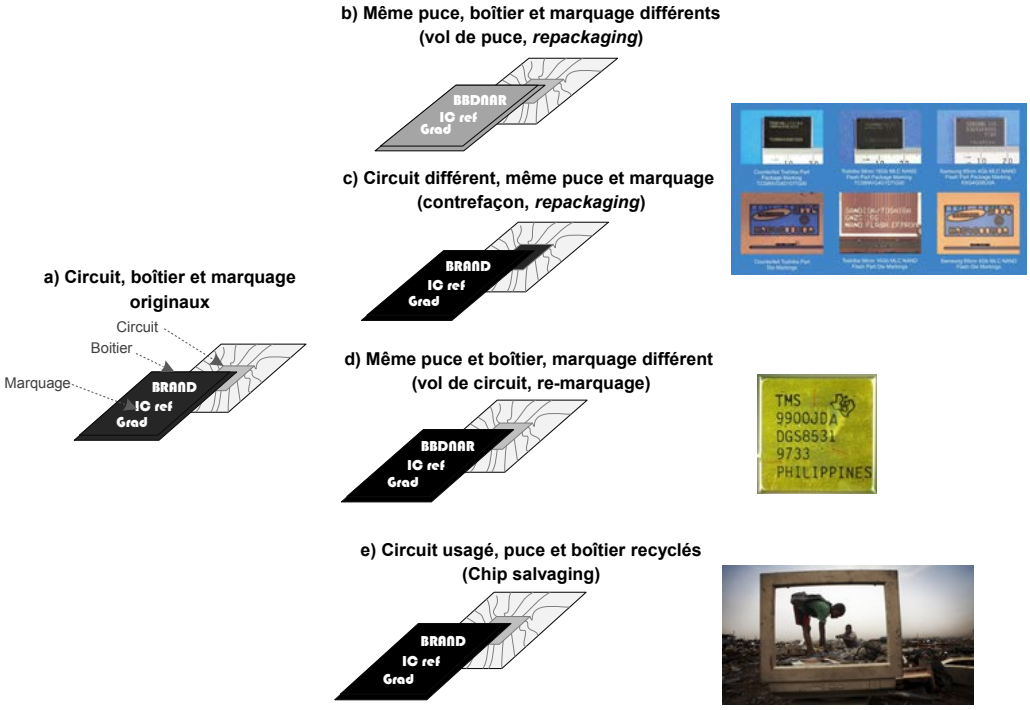

FIGURE 1.6 – Synthèse des différents types de contrefaçons.

le marquage. Dans le premier cas (Sous-figure 1.6.b), la puce électronique est la même que l'originale mais le boîtier ainsi que le marquage ne correspondent pas, c'est donc un cas typique de changement de boîtier. La Sous-figure 1.6.c présente un cas classique de contrefaçon : un puce ressemblant à l'originale est mise dans un boîtier dont l'aspect ressemble à celui d'origine et dont le marquage est proche du marquage du circuit authentique. La Sous-figure 1.6.d montre un exemple de circuit re-marqué. Le même puce électronique est présente dans le boîtier mais le marquage visible est différent. Enfin, la Sous-figure 1.6.e présente un exemple de circuit recyclé.

Ce qu'il faut absolument retenir avec ces quelques exemples de contrefaçon est que le but du contrefacteur est d'imiter à moindre coût un circuit intégré pour en tirer le maximum de profit. Par exemple, un processeur RAD750, blindé contre les radiations et pouvant supporter de très fortes variations de températures coûte environ 200 000 dollars alors que sa version commerciale ne coûte que quelques centaines de dollars [161]. Le re-marquage d'un processeur commercial en processeur ayant la certification spatiale rapporte donc beaucoup d'argent très rapidement et c'est exactement ce que cherchent les contrefacteurs de circuits intégrés.

1.1.2.2 La copie illégale de circuits intégrés

La copie illégale ou le clonage d'un circuit intégré consiste à recopier un circuit à l'identique pour le vendre en son nom. Cette menace peut intervenir à différents moments de la vie d'un circuit intégré comme le montre la Figure 1.2. Dès que le concepteur a envoyé la description de son circuit au fondeur, celle-ci peut être dérobée, copiée et transmise à une fonderie parallèle et illégale qui pourra alors produire des circuits en grand nombre avec des matériaux moins chers et sans avoir à payer le coût de la conception du circuit. Cette fonderie parallèle peut également recevoir les masques de photolithographie, volés lors de la fabrication des circuits. La copie d'un circuit intégré peut aussi être le résultat d'une surproduction (*overbuilding*) [142]. Dans ce cas, le fondeur augmente simplement la quantité de puces à fabriquer au-delà de ce qui est prévu dans le contrat signé avec le concepteur. Cette surproduction est très peu coûteuse et très simple à mettre en place. De plus, le concepteur n'a presque aucun moyen d'agir pour empêcher cette menace puisqu'il a fourni l'intégralité de la description de son circuit.

1.1.2.3 La rétro-ingénierie

La rétro-ingénierie, ou ingénierie inverse est la dernière manière d'attaquer un circuit intégré. Cette méthode d'attaque ne se limite pas au circuit intégré et touche tous les objets manufacturés, au même titre que la contrefaçon. Les objectifs de la rétro-ingénierie sont très nombreux et on peut par exemple citer les suivants [115] :

- comprendre ce que fait le circuit ;
- comprendre comment fonctionne le circuit ;
- comprendre comment le circuit a été conçu ;
- répliquer le circuit en partie ou en totalité ;
- changer les fonctionnalités du circuit ;
- récupérer des informations sensibles comme des clés de chiffrement ou des identifiants ;
- valider et vérifier le circuit.

Les techniques de rétro-ingénierie peuvent s'appliquer à toutes les parties d'un système aussi bien matérielles que logicielles, mais ce manuscrit s'intéresse uniquement à la partie matérielle des circuits intégrés. Dans ce cadre, la rétro-ingénierie peut être effectuée de manière invasive (avec la destruction du circuit) ou non-invasive (sans destruction du circuit).

Une première méthode de rétro-ingénierie non-invasive consiste à analyser le comportement des entrées/sorties d'un circuit afin de comprendre son fonctionnement. Par exemple, ce type d'analyse permet de différencier une machine à états finie de *Moore* ou de *Mealy* [113]. Dans le cas où aucune documentation n'est disponible sur le comportement du circuit, il est aussi possible d'effectuer une attaque de type boîte noire (*black box*) en soumettant le circuit à des entrées aléatoires et en observant les sorties obtenues. Ce type d'attaques peut toutefois s'avérer très long si le circuit analysé possède de nombreux ports d'entrées/sorties.

L'analyse non-invasive d'un circuit électronique peut également être effectuée grâce aux canaux auxiliaires. Ces techniques d'analyses sont très connues dans le domaine de la cryptographie pour attaquer les implantations d'algorithmes de chiffrement. Elles utilisent les canaux qui ne sont pas directement utilisés pour transmettre de l'information, comme la consommation de puissance, le rayonnement électromagnétique ou photonique par exemple. Le canal électromagnétique peut servir à obtenir des informations sur le circuit [15, 25] ou à le cartographier [148].

Pour les techniques de rétro-ingénierie invasive, plusieurs étapes doivent être suivies : la suppression du boîtier et l'extraction de la puce, le découpage hiérarchique des différents niveaux de métallisation de la puce, l'analyse par traitement d'image, la création d'un schéma de câblage des transistors et l'analyse finale du circuit étudié [166]. La complexité des techniques et de la technologie employée pour effectuer ce type de rétro-ingénierie va de pair avec la complexité du circuit à analyser.

Une étape particulièrement cruciale des techniques invasives est le découpage des différents niveaux de métallisation de la puce. Cette étape peut nécessiter une pré-analyse faite à partir d'une coupe de la puce par exemple. De plus, l'analyse par image des couches peut nécessiter différentes technologies de microscopie selon la complexité du circuit, allant du simple microscope optique [125] au microscope électronique de type SEM (*scanning electron microscope*) ou TEM (*transmission electron microscope*) [166].

La rétro-ingénierie n'est pas seulement une attaque visant les circuits intégrés. Elle peut également être considérée comme un moyen de protection et de détection de copies illégales [161, 166] ou de chevaux de Troie matériels [158].

Dans cette première section, nous avons détaillé les menaces qui pèsent sur le cycle de vie des circuits intégrés. Ce cycle faisant intervenir de nombreux acteurs dont les objectifs sont parfois très différents, la contrefaçon et le vol de circuit y sont au cœur, et il est impératif de trouver des solutions réalistes et durables afin de protéger la propriété intellectuelle du concepteur de circuit. Le Tableau 1.1 précise quels sont les acteurs du cycle de vie des circuits qui participent aux différentes menaces.

Tableau 1.1 – Les acteurs du cycle de vie des circuits intégrés et les différentes menaces.

	rétro-ingénierie		contrefaçon			vol de circuit	
	invasive	non-invasive	ré-emballage	re-marquage	recyclage	surproduction	vol de circuit
Concepteur							
Fabricant		X	X	X		X	X
Testeur			X				X
Concurrent	X	X					
Courtier en composants			X	X	X		
Collecteur de déchets					X		

On peut remarquer sur le Tableau 1.1 que les entités impliquées dans le plus de menaces sont le fabriquant et le courtier en composants. Cela s'explique par le fait

que ces deux acteurs sont incontournables dans le cycle de vie des circuits intégrés. La prochaine section s'intéresse aux mécanismes de protection qui ont été proposés pour lutter contre la contrefaçon et le vol de circuits intégrés.

1.2 Classification des mécanismes de protection

Face à toutes ces menaces, la communauté scientifique ainsi que les entreprises travaillent afin de proposer des mécanismes de protection. Les différents mécanismes de protection qui ont été proposés appartiennent à deux grandes catégories : les mécanismes de protection destructifs et ceux qui ne le sont pas. Les mécanismes destructifs utilisent généralement les techniques de la rétro-ingénierie afin d'identifier les circuits contrefaits ou volés [161]. Parmi les mécanismes non destructifs, on trouve deux sous-catégories : les mécanismes actifs et les mécanismes passifs. Chacun de ces deux groupes peut être divisé en plusieurs branches selon le type de mécanismes de protection.

Les mécanismes de protection non destructifs qui ont été proposés pour lutter contre le vol et la contrefaçon des circuits intégrés sont nombreux et plusieurs articles en proposent une vue d'ensemble [43, 92, 93, 94]. À partir de ces études, il est possible d'en extraire une classification, présentée dans la Figure 1.7.

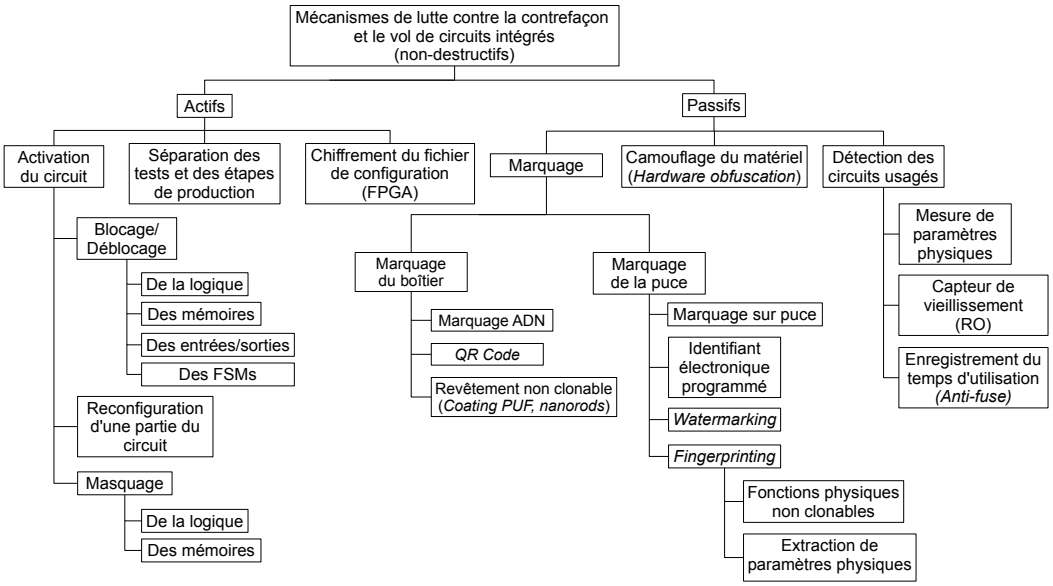

FIGURE 1.7 – Classification des différents mécanismes de protection.

Les mécanismes actifs ont pour principal objectif de redonner au concepteur du circuit le dernier rôle avant l'utilisation des circuits intégrés. Les mécanismes passifs visent la plupart du temps l'identification des circuits intégrés et permettent donc de détecter les contrefaçons ou les circuits volés. Dans cette thèse, seuls certains mécanismes liés à la lutte contre le vol et la contrefaçon sont étudiés. Toutefois, les

deux prochaines sections détaillent les différents mécanismes de protection présentés dans la Figure 1.7. La Section 1.3 s'intéresse aux mécanismes de protection actifs et la Section 1.4 aux mécanismes passifs.

1.3 Mécanismes de protection actifs

La plupart des mécanismes de protection actifs visent à redonner le dernier mot au véritable propriétaire des circuits intégrés en leur permettant de les activer avant leur utilisation. Ainsi, ces mécanismes de protection permettent de lutter contre la surproduction ou le vol de circuits. Parmi ces mécanismes, on trouve donc les systèmes permettant l'activation des circuits intégrés, mais aussi des protocoles de fabrication et des tests qui font intervenir différentes fonderies. Dans le cas particulier des systèmes sur FPGA, un mécanisme de protection très utilisé est le chiffrement des fichiers de configuration.

1.3.1 Activation des circuits intégrés

Cette première catégorie de mécanismes de protection des circuits intégrés contre le vol et la contrefaçon a pour but de permettre l'activation des circuits avant leur utilisation. Plusieurs techniques peuvent être envisagées dont le blocage du circuit ou la reconfiguration dynamique d'une partie du circuit.

1.3.1.1 Blocage et déblocage d'une partie du circuit

Le blocage des circuits intégrés consiste à rendre une partie ou la totalité du circuit non-fonctionnelle avant son déblocage. Plusieurs parties peuvent être bloquées : la logique, les mémoires ou les entrées/sorties par exemple. Le déblocage des fonctionnalités du circuit passe généralement par l'acquisition d'une clé d'activation. De ce fait, certains articles étudiant ce mécanisme de protection utilisent les termes chiffrement ou camouflage [3, 134, 135, 142] pour parler de mécanismes de blocage et déblocage des circuits intégrés. Pourtant, ils n'utilisent ni cryptographie, ni techniques de camouflage (obfuscation), ce qui rend ces termes trompeurs. Toutefois, tous les mécanismes de protection liés au blocage et déblocage des fonctionnalités du circuit nécessitent un protocole d'envoi de la clé d'activation, qui requiert une forte sécurité où la cryptographie doit occuper une part importante [4, 78]. Ces mécanismes de protection permettent notamment de lutter contre des menaces telles que la surproduction et le vol des circuits intégrés.

Blocage des parties combinatoires et séquentielles

La première solution consiste à bloquer la machine à états finie d'un système en rajoutant une nouvelle machine à états finie dite de blocage. Cette solution a été proposée pour la première fois dans [3]. La machine à états finie supplémentaire est généralement ajoutée au niveau des états de démarrage de la machine originale, ce

qui permet de créer une phase d'activation du circuit lors de sa mise en tension. La Figure 1.8 illustre cette solution.

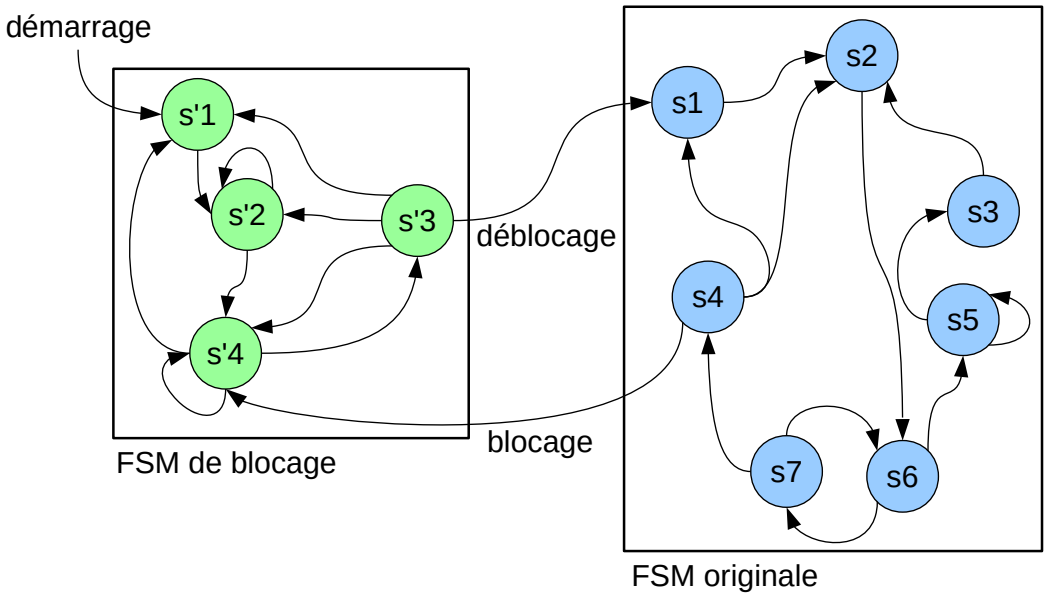

FIGURE 1.8 – Exemple d'ajout d'une machine à états finie de blocage.

La suite d'états à appliquer sur la machine à états finie de blocage (à gauche de la Figure 1.8) est appelée clé d'activation et permet de rendre le circuit fonctionnel. Il est également possible d'améliorer la sécurité de ce mécanisme en couplant la clé d'activation avec une fonction unique aléatoire (appelée *Random unique block*) [5]. Cette fonction crée un identifiant unique pour chaque circuit en tirant parti des délais de propagation et des paramètres physiques uniques de chaque circuit. Ce bloc peut aussi utiliser une fonction physique non-clonable pour créer l'identifiant secret [92]. Plus de détails concernant l'identification des circuits et les fonctions physiques non-clonables sont présentés dans la Section 1.4.2.1.

Plus récemment, un autre schéma de blocage des machines à états finies a été proposé [84]. Il utilise la synchronicité de la machine à états finie d'un circuit pour permettre l'activation du circuit. Dans [84], le schéma proposé permet d'accéder à un état connu depuis n'importe quel état de la machine, en appliquant une séquence spécifique à l'entrée du circuit (clé d'activation). L'avantage de cette méthode est qu'il n'est pas nécessaire d'ajouter des états à la machine. De plus, toute tentative de suppression de cette étape d'activation entraîne la destruction des fonctionnalités du circuit puisque le mécanisme de protection est intrinsèquement inclus dans le système.

La machine à états finie n'est pas la seule partie sur laquelle il est possible d'agir pour bloquer un circuit intégré. En effet, il est également possible de rendre

certaines fonctionnalités inutilisables, en ajoutant des portes *xor* sur les chemins non critiques [142]. L'ajout de ces portes permet d'inverser ou non les signaux de sortie d'une fonction et donc de la rendre inutilisable. Toutefois, la conséquence de l'ajout de portes *xor* n'est pas l'arrêt du fonctionnement du circuit mais son dysfonctionnement : on parle donc de masquage de la logique. D'autres techniques d'ajout de portes au sein d'un circuit ont également été proposées dans [45, 134, 135]. Dans [45], la méthode d'insertion de portes utilise la construction et la réduction d'un graphe pour choisir où et quelles portes insérer.

Blocage des mémoires et des entrées/sorties du circuit

La deuxième partie d'un circuit qu'il est possible de bloquer pour rendre inutilisable un circuit intégré est sa mémoire [44]. En effet, le blocage de l'accès à la mémoire d'un circuit rend toute opération impossible. Par exemple, il est possible de bloquer l'écriture de données dans la mémoire du circuit [70, 183] ou leur lecture [9, 183]. Dans le cas d'un système complexe, le blocage de la mémoire rend toute opération impossible si la portion de mémoire bloquée se situe au niveau des registres d'opérations par exemple. Ainsi, sans avoir au préalable activé le circuit et débloqué l'accès à la mémoire, il est impossible de se servir du circuit. Malheureusement, très peu de travaux traitent spécifiquement de cette problématique. En revanche, il est tout à fait possible d'utiliser les méthodes de blocage de la logique présentées dans la section précédente pour arriver à bloquer la mémoire d'un circuit. Par exemple, il est possible de modifier la machine à états finie du circuit pour inclure ce type de blocage.

De même, il est possible de bloquer les ports d'entrées ou de sorties d'un circuit pour le rendre inutilisable [12]. Ici encore, peu de travaux traitent spécifiquement du blocage des entrées/sorties d'un système mais le déni de service apporté par les solutions proposées pour bloquer la logique d'un circuit se traduit généralement par le blocage de l'envoi des données ou par l'envoi de résultats erronés [45, 134, 135]. Dans [143], c'est le bus de transmission de données qui est spécifiquement visé, ce qui a pour conséquence le blocage à la fois de la mémoire et des entrées/sorties de chaque bloc présent dans le circuit.

1.3.1.2 Reconfiguration dynamique d'une partie du circuit

Un autre mécanisme actif de protection des circuits intégrés contre la contrefaçon et le vol consiste à ajouter une partie reconfigurable aux circuits. Pour un attaquant, il sera impossible d'utiliser le circuit s'il ne possède pas le fichier de configuration de la partie reconfigurable. L'idée principale de ce mécanisme est donc de rendre reconfigurable une partie indispensable du circuit (bus de communication, séquenceur de l'unité arithmétique, ...) [14, 100]. Pour rendre une partie d'un circuit reconfigurable, il est par exemple possible d'acheter un IP (*intellectual property*) FPGA et de l'inclure dans un ASIC. Une méthode permettant de sélectionner quelle est la meilleure partie à reconfigurer a été proposée dans [14]. Comme

l'ajout d'une telle partie reconfigurable peut augmenter très fortement le coût de fabrication du circuit, il est impératif de sélectionner la surface la plus petite possible qui a l'impact le plus important sur le circuit. Les bus de communication sont généralement un bon choix car ils permettent de facilement bloquer l'intégralité d'un circuit.

1.3.2 Séparation des étapes de production et des tests

La séparation des tests et des étapes de production des circuits intégrés a pour but de protéger les circuits contre le vol et la surproduction. La première idée est de séparer la production des circuits et leurs tests afin de protéger le concepteur du circuit contre le vol. De plus, la fonderie ne possédant pas les procédures permettant de tester les circuits, le vol des *wafers* est inutile et ne permettra pas de créer des contrefaçons [46].

Dans le but d'aller encore plus loin dans la protection de la propriété intellectuelle du concepteur de circuit, il est également possible de séparer les différentes étapes de la production des circuits intégrés. Cela consiste à produire les différentes couches d'un circuit sur différentes lignes de production (dans la même usine de production) [83] ou dans différentes usines [81, 137]. L'idée principale de la séparation des étapes de production est d'empêcher le vol des masques de photolithographie ou de le rendre inutile. Chaque fonderie impliquée dans la fabrication d'un circuit ne se voit alors recevoir que les informations utiles aux étapes de fabrication dont elle est chargée.

1.3.3 Chiffrement du fichier de configuration (FPGA)

Dans le cas particulier des FPGA *(Field Programmable Gate Array)*, le mécanisme de protection le plus employé et étudié est le chiffrement des fichiers de configuration. Dans ce cas, la sécurité de toute la chaine de configuration des FPGA est à prendre en compte [8, 164]. Les premiers travaux de recherche concernant cette problématique ont été proposés en 2003 [26] afin d'améliorer la sécurité du flot de configuration, tout en laissant le choix à l'utilisateur de chiffrer ou non le fichier de configuration, grâce à un algorithme de chiffrement symétrique. Pour cela, la configuration partielle et dynamique des FPGA est utilisée. Toutefois, la mise en place de la solution proposée dans [26] implique l'implantation d'une unité de contrôle complexe et la configuration des FPGA est également plus lente.

Depuis, plusieurs travaux proposent de configurer dynamiquement une partie d'un FPGA à l'aide de fichiers de configuration chiffrés. Dans [173, 184], le système de configuration partielle proposé utilise un microprocesseur câblé dans le FPGA (Microblaze) pour recevoir le nouveau fichier de configuration, qui est alors envoyé à un module cryptographique pour être déchiffré avant que la reconfiguration n'ait lieu.

Beaucoup d'autres travaux proposent d'autres protocoles de reconfiguration d'une partie d'un FPGA [35, 36, 51, 74]. Aujourd'hui, les fabricants de FPGA

(Xilinx, Altera et Microsemi) proposent ce type de protection directement dans la chaine de développement FPGA, en laissant le choix à l'utilisateur de chiffrer ou non le fichier de configuration. La résistance de ces chaines de développement sécurisées a été testée dans [117, 118, 154].

Dans cette section, nous avons succinctement détaillé les différents mécanismes de protection actifs permettant de lutter contre la contrefaçon et le vol de circuits intégrés. Tous ces mécanismes redonnent au concepteur du circuit le dernier mot dans la chaine de production. Celui-ci peut alors contrôler le nombre de circuits produits. De plus, les différents mécanismes actifs peuvent être astucieusement combinés pour créer un système de licence, périodique ou permanente, dans le but d'activer un circuit. Ce type de techniques a été proposé pour améliorer la sécurité des composants virtuels [47, 103], souvent utilisés dans des systèmes plus complexes. Dans le cas des circuits intégrés complets, le terme de licence n'est pas spécifiquement employé mais on peut très bien assimiler la clé d'activation des systèmes de blocage au mot licence.

De plus, la combinaison de la reconfiguration partielle d'un circuit et du blocage et déblocage du circuit peut conduire à concevoir un système permettant une mise à jour des fonctionnalités d'un circuit et ainsi créer une licence temporaire ou une version d'évaluation. Toutefois, tous les mécanismes de protection actifs permettant de lutter contre la contrefaçon et le vol de circuits intégrés requièrent l'identification, ainsi qu'une communication fortement sécurisée. L'identification d'un circuit fait partie des mécanismes de protection passifs, au même titre que la détection des circuits recyclés ou le camouflage du matériel. Ces mécanismes sont détaillés dans la prochaine section.

1.4 Mécanismes de protection passifs

Les mécanismes de protection passifs regroupent les techniques qui permettent d'identifier les circuits intégrés que ce soit au niveau du boitier ou de la puce électronique elle-même. D'autres techniques rendant plus difficile la rétro-ingénierie ou visant à identifier les circuits intégrés recyclés appartiennent également à cette catégorie. Ainsi, ces mécanismes impliquent un effort supplémentaire lors de la conception des circuits mais ne redonnent pas la main au concepteur une fois les circuits fabriqués.

1.4.1 Marquage du boîtier du circuit intégré

Dans le but d'identifier un circuit intégré, il est possible d'inclure une marque spécifique sur le boîtier du circuit. Parmi les techniques utilisées, on trouve des exemples comme le marquage ADN, les *QR-codes*, les revêtements non-clonables ou les *nanorods*.

1.4.1.1 Marquage ADN

Le marquage ADN est un moyen de rendre les inscriptions visibles sur le boîtier des circuits intégrés uniques [162]. Dans la plupart des cas, de l'ADN (acide désoxyribonucléique) de plante est extrait. Puis la séquence génétique est mélangée et ré-assemblée afin de créer un identifiant unique. Enfin, cette séquence est mélangée à l'encre qui est appliquée sur le boîtier du circuit, pendant l'étape d'inscription des informations du boîtier. La Figure 1.9 présente ces différentes étapes du marquage ADN [7].

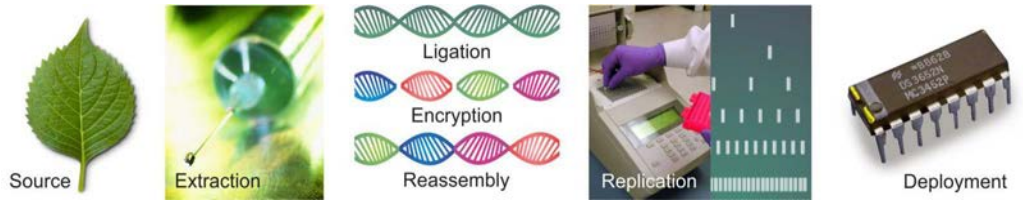

FIGURE 1.9 – Création et déploiement du marquage ADN. [7]

Cette technique de marquage du boîtier des circuits intégrés est très intéressante pour lutter contre plusieurs menaces existantes dans le cycle de vie des circuits comme le re-marquage ou le recyclage de circuits. En effet, lors du re-marquage d'un circuit, la première étape consiste à supprimer le marquage existant. Ce faisant, la structure ADN présente dans le marquage d'origine est détruite. Donc même si l'encre supprimée est mélangée à l'encre servant au nouveau marquage, la structure d'origine ne sera pas présente dans les nouvelles inscriptions, et il sera possible de détecter qu'il s'agit d'une contrefaçon. La Figure 1.10 illustre ce principe.

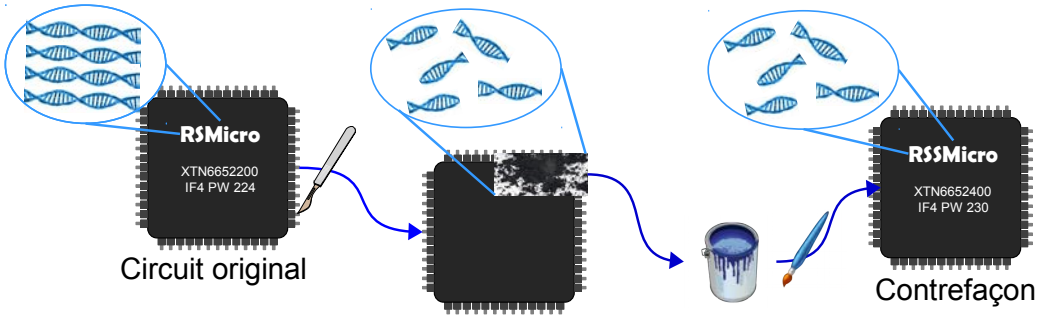

FIGURE 1.10 – Protection apportée par le marquage ADN contre le re-marquage.

1.4.1.2 *QR codes*

L'utilisation des codes QR (code barre en deux dimensions) [1] est une autre méthode permettant d'identifier un circuit intégré grâce au marquage du boîtier [101].

7. http://www.adnas.com/products/signaturedna

L'avantage de ces codes QR est qu'il est possible d'identifier très rapidement les circuits à l'aide d'un simple téléphone mobile [128].

Dans le cadre de la protection des circuits intégrés contre la contrefaçon et le vol, il est possible de chiffrer un code QR, de le compresser puis de l'apposer sur le boîtier d'un circuit. Dans [112], une technique de chiffrement des codes basée sur le comptage de photons est détaillée. Le marquage du boîtier est ensuite converti en code QR optique. La lecture de l'identifiant (Figure 1.11) passe par la reconstitution du code QR. Cette reconstruction est basée sur le déchiffrement du relief (Figure 1.11.a) de la zone du boîtier du circuit intégré contenant le code QR (Figure 1.11.b).

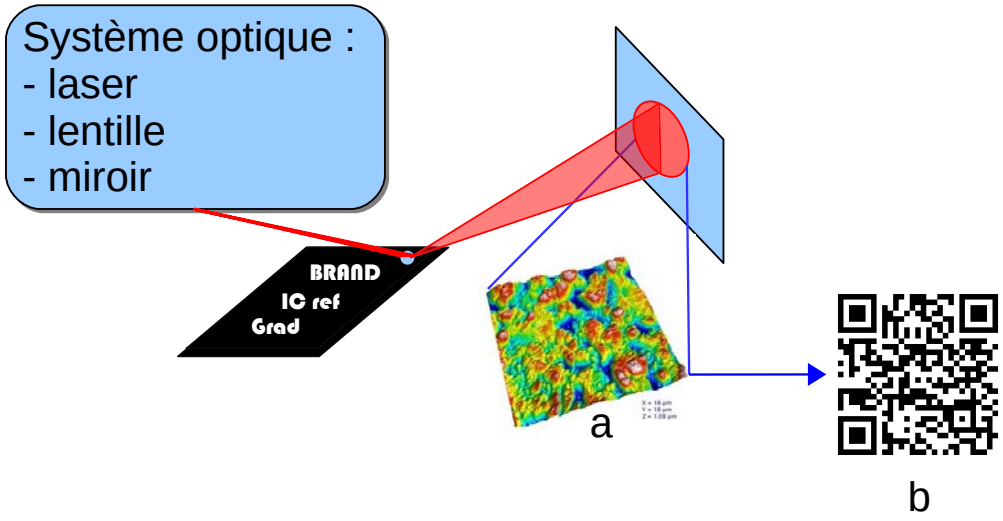

FIGURE 1.11 – Lecture d'un code QR optique inscrit sur le boîtier d'un circuit intégré [112].

1.4.1.3 Revêtement non-clonable (*coating PUF*) et nanosphère

Les revêtements non-clonables (appelés *coating PUF*) consistent à appliquer un revêtement contenant un mélange d'éléments, dont les constantes diélectriques sont différentes, sur le boîtier du circuit. Cette méthode a été introduite dans [174]. Comme pour les fonctions physiques non-clonables implantées dans les circuits (voir Section 1.4.2.4), le *coating PUF* fonctionne en deux phases : une phase d'enrôlement et une phase d'identification. Dans le cas du *coating PUF*, un challenge correspond à un signal périodique ayant une certaine amplitude et une certaine fréquence. Ce signal est alors appliqué aux deux extrémités du revêtement et la capacité est enregistrée en différents points. La réponse de la PUF correspond à la valeur de ces capacités. Une autre méthode, aussi basée sur la mesure d'une capacité, a été proposée dans [62].

Les nanosphères (*nanorods*) permettent de créer un autre type de revêtement

non-clonable. Elles ont été introduites par des chercheurs de la société IBM [95]. L'idée est de faire grandir des nanosphères de carbone incluant des particules d'or. Ensuite, un motif particulier est inscrit sur le boîtier des circuits intégrés. Même si le motif est identique pour deux circuits, les caractéristiques de ce marquage sont différentes pour chaque circuit car chaque nanosphère de carbone est unique. Une autre technique consiste à créer un tableau de sphères fluorescentes dont la couleur (rouge, vert ou bleu) n'est pas prévisible [8].

1.4.2 Marquage de la puce

Le boîtier n'est pas le seul élément sur lequel il est possible d'agir pour identifier un composant. En effet, il est également possible d'apposer une marque directement sur la puce, d'y programmer un identifiant secret, d'inclure une *watermark* dans le circuit ou encore d'extraire un identifiant basé sur les propriétés intrinsèques de la puce.

1.4.2.1 Marquage sur puce

Le marquage de la puce elle-même est l'une des premières idées qu'il est possible d'avoir pour protéger son circuit. Sur la Figure 1.3, on voit clairement le marquage de chacune des trois puces, ce qui permet de détecter une contrefaçon. La Figure 1.12 montre d'autres exemples de marquage présents sur les puces électroniques [9] [10] [11].

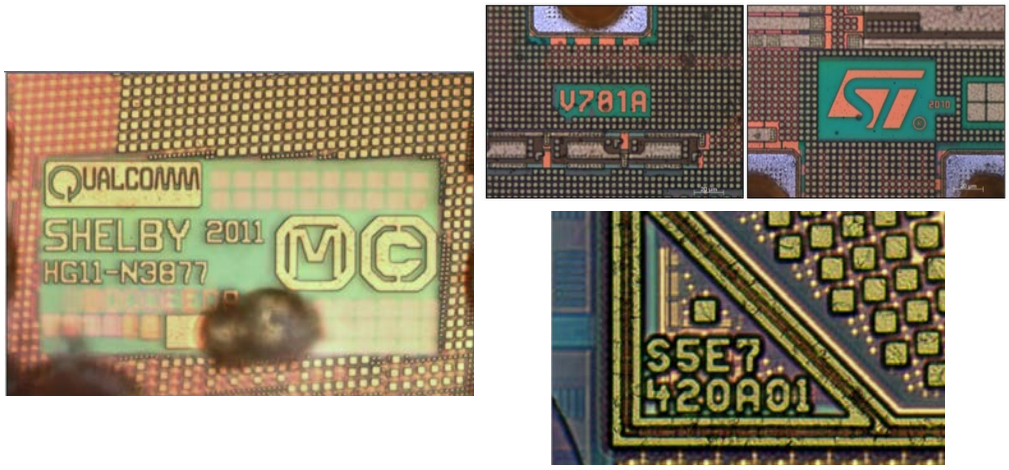

FIGURE 1.12 – Marquage sur puce et détection de contrefaçon.

8. http://www.research.ibm.com/articles/nano-counterfeit.shtml
9. http://tectalks.blogspot.fr/2012/10/iphone-5-torn-apart-and-on-display-pics.html
10. http://www.slideshare.net/Yole_Developpement/stmicroelectronics-lsm9ds0-9axis-mems-imu
11. https://www.chipworks.com/about-chipworks/overview/blog/inside-the-samsung-galaxy-s6

Le type de marquages sur puce le plus utilisé est le marquage laser [121, 151, 178]. L'avantage de ce type de marquages est que le concept (de marquer un produit) est très ancien [146], et que l'industrie est habituée à effectuer ce genre d'opérations. Sa mise en œuvre est donc relativement simple et les outils de marquage et de vérification des marques apposées sont bien établis.

1.4.2.2 Identifiant électronique programmé

La deuxième solution qui est souvent employée pour identifier un circuit intégré afin de le protéger contre la contrefaçon et le vol est d'y inclure un identifiant secret, directement programmé à l'intérieur. Généralement, cet identifiant est inscrit dans une mémoire non volatile comme les mémoires programmables une seule fois (OTP) ou les mémoires ROM. Il est également possible d'utiliser des procédés de post-fabrication tels que les fusibles lasers [7] ou électriques [91, 139]. Ces derniers, introduits par la société IBM [165], requièrent moins de surface et sont plus flexibles que les fusibles lasers.

Malheureusement, les identifiants programmés dans les circuits intégrés sont relativement simples à lire [122, 147] et donc à reproduire.

1.4.2.3 *Watermarking*

Pour inclure des informations permettant d'identifier la propriété intellectuelle d'un circuit, il est également possible d'utiliser le *watermarking*. Ce concept appartient au domaine de la stéganographie dont les premiers exemples remontent à l'Antiquité. Elle était utilisée pour cacher des messages sensibles à cette époque. Par exemple, durant la seconde guerre médique, Hérodote (historien grec) explique comment les Grecs ont pu être au courant de l'invasion Perse bien avant qu'elle ne se produise grâce à un message caché dans une tablette de cire. La cire de cette dernière a été enlevée, puis le message a été gravé à même le bois avant de remettre de la cire sur la tablette. Ainsi, la tablette de cire paraissait comme vierge alors qu'un message y était caché [12]. Cet exemple n'est pas le seul et la stéganographie est encore très utilisée aujourd'hui dans divers domaines. [131] présente une classification des différentes techniques permettant de cacher de l'information dans un système. En particulier, le *watermarking* est très répandu dans le domaine de l'image. Il est possible d'utiliser les bits de poids faible du code couleur de chaque pixel d'une image afin d'y cacher toutes sortes d'informations (texte, image, *etc.*), sans altérer la qualité de l'image (vue par un humain) [99, 182].

D'une manière générale, le *watermarking* est divisé en deux phases : l'insertion et la vérification. La Figure 1.13 représente ces deux étapes d'une façon analogue au modèle présenté dans [131]. La partie haute de cette illustration présente l'insertion d'une *watermark*. La partie basse en montre la vérification.

12. `http://www.bibmath.net/crypto/index.php?action=affiche\&quoi=stegano/histstegano`

En électronique, le *watermarking* est un mécanisme de détection qui permet de lutter contre la contrefaçon et le vol de circuits intégrés. Il peut être utilisé dans deux situations différentes. La Figure 1.14 présente les deux cas de figure dans lesquels le *watermarking* peut être utilisé.

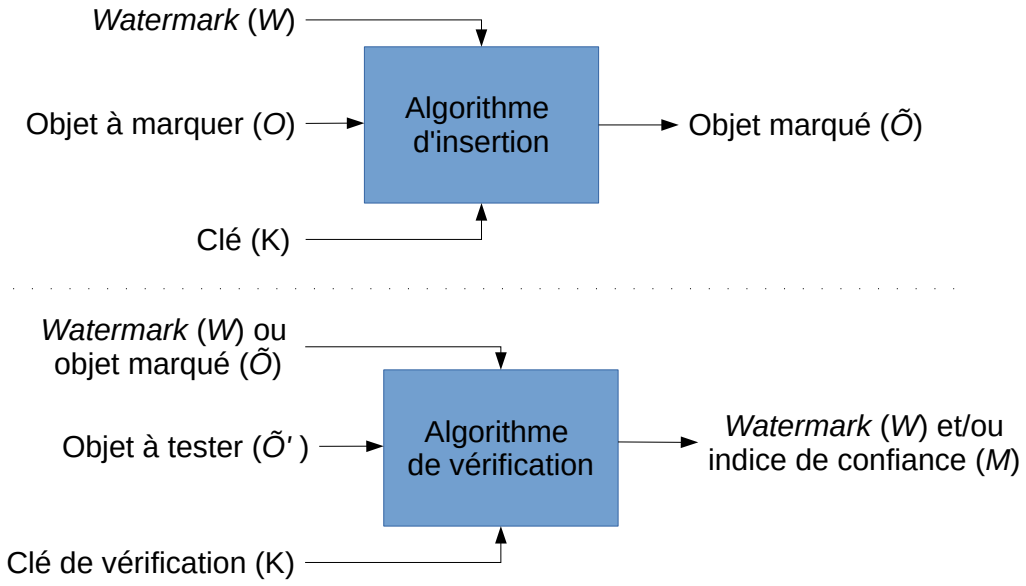

FIGURE 1.13 – Concept général du *watermarking*.

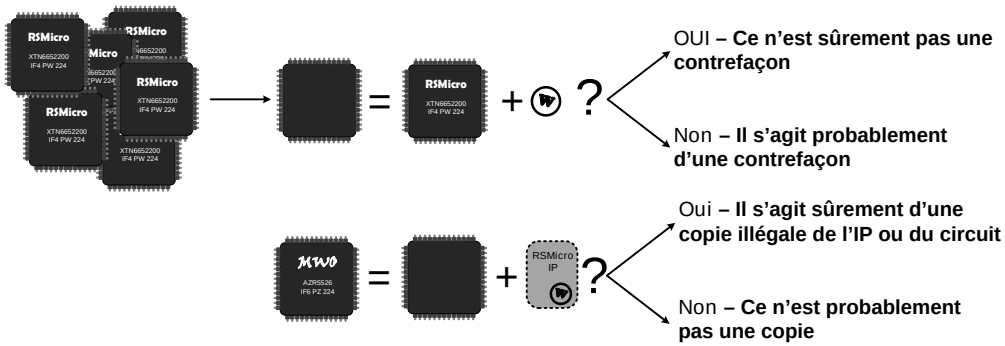

FIGURE 1.14 – Cas d'utilisation du *watermarking* dans la lutte contre le vol et la contrefaçon.

Dans le premier cas (partie supérieure de la Figure 1.14), l'intégrateur de systèmes achète un lot de circuits identiques. Lorsqu'il les reçoit, il vérifie si la marque insérée par le concepteur est bien présente dans chaque circuit. Si ce n'est pas le cas, les circuits en question sont probablement des contrefaçons. Pour le second cas (partie inférieure de la Figure 1.14), deux scénarios peuvent être envisagés :

Scénario 1 : Le concepteur de la puce achète un produit vendu par l'un de ses concurrents puis tente de vérifier si la *watermark* qui est présente dans ses propres produits se trouve dans le produit du concurrent. S'il retrouve sa marque, cela signifie que ce concurrent a probablement copié son circuit ;

Scénario 2 : Le concepteur de composants virtuels a vendu une IP marquée pour un produit en particulier. Il achète alors un autre produit, du même client (intégrateur de systèmes) ou d'une autre entité, dans lequel ce composant virtuel n'est pas censé se trouver et vérifie si la *watermark* est présente. S'il la retrouve, cela signifie que son client a probablement utilisé ou distribué l'IP sans autorisation.

Pour insérer efficacement une *watermark*, la méthode d'insertion ainsi que l'emplacement où la marque est insérée sont importants. En plus de donner une vue d'ensemble des différentes techniques d'insertion de *watermark* dans les composants virtuels dans [2], les auteurs présentent quelles sont les bonnes propriétés d'une *watermark*, qui sont au nombre de sept :

1. Comme tous les systèmes cherchant à apporter une sécurité, le *watermarking* doit respecter les principes de Kerckhoffs [89]. Ces principes indiquent que tout système de sécurité ne doit pas reposer sur le secret de l'algorithme utilisé. Ainsi, ce n'est pas la méthode d'insertion ou de vérification qui est importante mais sur quelles bases celle-ci reposent. Dans l'idéal, l'insertion d'une *watermark* doit se servir des propriétés intrinsèques du circuit ;
2. L'insertion d'une *watermark* ne doit en aucun cas affecter les fonctionnalités du circuit ;
3. La *watermark* qui est insérée doit être fiable d'un point de vue de sa détection. La probabilité de détecter une marque quand il n'y en a pas (faux positif), ainsi que la probabilité de ne pas la détecter s'il y en a une (faux négatif), doivent être faibles ;
4. Il est nécessaire que la *watermark* soit robuste. Du point de vue d'un attaquant, la suppression de la marque et la création d'une nouvelle marque valide doivent être difficiles ;
5. Une *watermark* doit contenir suffisamment d'information pour prouver la propriété intellectuelle du circuit ;
6. Le surcoût lié à l'insertion d'une *watermark* doit être faible. En effet, il s'agit d'un système de sécurité qui est ajouté à un système et ses coûts en surface, puissance et temps doivent être considérés lors de son insertion ;
7. La *watermark* insérée doit pouvoir être vérifiée. En effet, l'insertion d'une marque ne constitue que la moitié du concept et insérer une *watermark* sans être capable de la vérifier est complètement inutile.

L'insertion d'une *watermark*

Il est possible d'insérer une *watermark* à différents niveaux dans le flot de conception d'un circuit ou d'un composant virtuel : physique et structurel, comportemental ou algorithmique [123].

Au niveau physique, il est possible d'ajouter des contraintes pour le placement et le routage du circuit. Dans [86], les auteurs proposent d'utiliser des outils de résolution de problèmes pour aboutir à un circuit marqué qui respecte les contraintes choisies. Dans [124], des *obstacles* sont placés à des endroits clés du circuit, puis l'étape de routage du circuit est relancée.

Au niveau comportemental, c'est la machine d'état des circuits qui est visée la plupart du temps, car elle constitue le cœur du système. Ces techniques sont particulièrement intéressantes car la modification de la machine d'état d'un système complexe se traduit très souvent par la destruction des fonctionnalités du système. Traditionnellement, les techniques d'insertion de *watermarks* dans une machine d'état sont basées sur les transitions inutilisées pour le fonctionnement normal du circuit [129, 167]. Le problème à résoudre pour trouver les transitions à rajouter correspond à un problème de partitionnement [179] ou de coloriage de graphe [132]. Plus récemment, les auteurs de [84] proposent d'extraire une propriété intrinsèque d'une machine d'état en repensant son architecture pour créer une *watermark*. Avec cette dernière méthode, il est impossible de supprimer la marque du circuit puisque il n'y a ni ajout de transition, ni ajout d'état.

Pour les insertions au niveau algorithmique, on trouve des exemples d'insertion de marques dans des fonctions de traitement de signaux numériques (DSP) [37]. Ici, les auteurs jouent avec les paramètres des filtres pour créer une *watermark*.

D'autres techniques d'insertion de *watermarks* visent les chaines de test qui sont utilisées pour vérifier qu'un circuit fonctionne correctement [49], les ports d'entrées/sorties du système [55] ou encore le réseau d'horloge du circuit [96].

La vérification d'une *watermark*

En ce qui concerne la vérification des *watermarks*, certaines méthodes d'insertion requièrent une inspection visuelle du composant virtuel ou du circuit [86, 124], ce qui rend la vérification peu évidente. Il est également possible d'envoyer une suite d'entrées spécifique pour laquelle la réponse du circuit indique ou non la présence de la marque [55].

Dans le cadre du *watermarking* de machine d'état, la vérification nécessite souvent un accès à un registre d'état ou la possibilité de lire une séquence de transitions. C'est notamment le cas pour les techniques qui ajoutent des états et des transitions au circuit [129, 167]. Cela pose un sérieux problème de sécurité puisque il est alors possible de copier la *watermark* et donc de l'insérer dans un autre circuit.

Dans [189], la signature est générée grâce à une *look up table* convertie en un registre à décalage. La consommation de puissance dépend alors du contenu de ce registre. Cette technique consiste donc à créer volontairement un profil particulier sur un canal auxiliaire de la consommation de puissance. Dans [25], c'est le canal

électromagnétique qui est utilisé pour identifier le circuit. L'avantage de la méthode proposée est la rapidité avec laquelle il est possible d'effectuer l'identification : 53 Mbps.

Malheureusement, toutes les techniques de vérification proposées s'appliquent à un schéma d'insertion de *watermarks* spécifique. Il est intéressant de se demander s'il n'est pas possible d'utiliser les canaux auxiliaires afin de construire une méthode de vérification moins dépendante de l'algorithme d'insertion, pour vérifier les *watermarks* insérées dans les machines à états finies des systèmes numériques synchrones.

1.4.2.4 *Fingerprinting*

Pour identifier un circuit intégré, il est également possible de créer une empreinte digitale du circuit (semblable à celle de l'Homme). La création d'une telle empreinte dans le cadre des circuits intégrés s'appuie sur leurs propriétés physiques et intrinsèques. Parmi les méthodes de *fingerprinting*, on retrouve notamment les fonctions physiques non-clonables ainsi que l'extraction de paramètres physiques tels que les délais de propagation.

Extraction de paramètres physiques

La première solution pour créer un identifiant unique d'un circuit intégré consiste à extraire une propriété physique. Il est par exemple possible de mesurer directement les délais de propagation des signaux dans le circuit [119, 120], de créer un profil du circuit basé sur son émission électromagnétique [77] ou sur sa consommation de puissance [176, 189]. La création d'une signature basée sur les délais de propagation a donné naissance au concept de fonctions de tri [80], qui ont pour principal but de séparer les circuits usagés des circuits récents.

Le concept de *soft physical hash* a été introduit dans [88] et propose de reconnaître un circuit grâce à sa signature sur le canal de la consommation de puissance. Les fonctions qui sont utilisées sont des algorithmes de chiffrement symétrique et la non-linéarité de l'étape de substitution les rend très différents les uns des autres, ce qui facilite leur distinction.

Enfin, il est possible de construire un réseau de capteurs de type oscillateurs en anneau à différents endroits du circuit et de créer une signature basée sur les différentes fréquences d'oscillation de ces capteurs [98].

Fonctions physiques non-clonables

Les fonctions physiques non-clonables visent à extraire de l'entropie des variations qui interviennent lors de la fabrication des circuits intégrés afin de construire un identifiant unique pour chacun d'eux. Elles font donc partie des mécanismes de *fingerprinting*. Ces fonctions représentent une alternative aux mémoires non volatiles. Les deux utilisations les plus répandues des fonctions physiques non-clonables sont la génération de clés cryptographiques [102] et la création d'identifiants uniques [78] comparables à l'ADN ou à l'empreinte digitale de l'Homme. En effet, les variations

qui interviennent lors de la fabrication du circuit sont imprévisibles et chaque circuit est donc unique.

La plupart des propositions de fonctions physiques non-clonables s'appuient sur un protocole de challenges/réponses qui permet de générer une réponse différente pour chaque challenge. Malheureusement, la stabilité d'une réponse à un challenge particulier n'est pas parfaite, ce qui rend la génération de clés cryptographiques très compliquée à mettre en place dans la pratique. En effet, cela nécessite une correction d'erreurs [22, 51] qui est très coûteuse en surface. La Figure 1.15 résume le concept de fonction physique non-clonable.

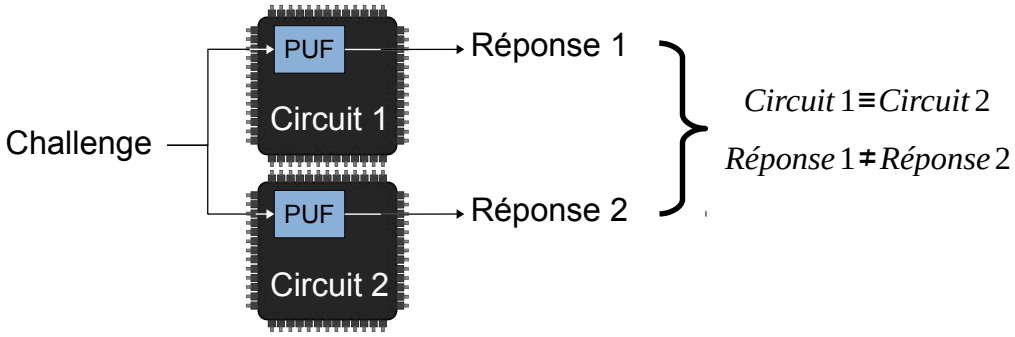

FIGURE 1.15 – Le concept d'une fonction physique non-clonable.

Les différents types de fonctions physiques non-clonables

Cette sous-section décrit rapidement les principes les plus répandus pour construire des fonctions physiques non-clonables. La liste n'est pas exhaustive mais donne un rapide aperçu des fonctions les plus connues et étudiées [21, 105].

La SRAM-PUF est une fonction physique non-clonable qui s'appuie sur des cellules de mémoires volatiles non initialisées [61, 72, 181]. Une cellule SRAM est constituée de deux inverseurs formant une boucle, comme le montre la Figure 1.16. Au démarrage du circuit, chaque cellule prend une valeur aléatoire qui dépend des variations apparues lors de leur fabrication. Ainsi, pour générer un identifiant de N bits, il faut au moins N cellules SRAM. Certaints bits peuvent être très instables et donc inutilisables dans la réponse de la PUF.

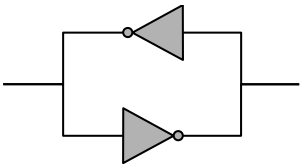

FIGURE 1.16 – Schéma d'une cellule SRAM.

La *Butterfly*-PUF s'appuie sur le même principe que la SRAM-PUF mais les cellules mémoires sont remplacées par des cellules métastables [97]. La *SR-latch*-PUF [67] est une variante de cette PUF dans laquelle ce n'est pas l'état final de la cellule métastable qui est pris en compte mais son nombre d'oscillations. C'est également dans cette catégorie de fonctions que peuvent être classées les *D-flip-flop*-PUF [104, 153, 171].

L'*arbiter*-PUF est une fonction physique non-clonable basée sur le délai de propagation d'un signal dans deux chemins théoriquement identiques. Mais à cause des variations de fabrication, les deux lignes de délais sont légèrement différentes. La détection du chemin le plus court permet de générer une réponse. Malheureusement, ce type de fonctions physiques non-clonables souffre d'une grande faiblesse face aux attaques par modélisation [75, 144]. La Figure 1.17 montre la structure des cellules composant cette fonction physique non-clonable.

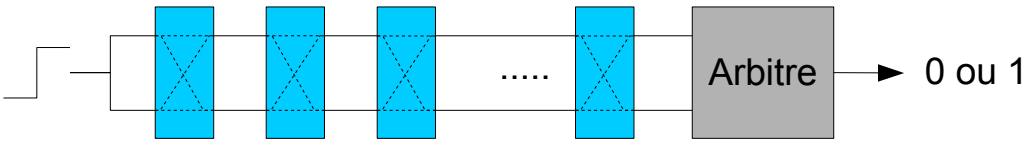

FIGURE 1.17 – Schéma d'une *arbiter*-PUF.

La RO-PUF s'appuie sur la différence entre les fréquences d'oscillateurs en anneau théoriquement identiques. Elle est souvent considérée comme l'une des meilleures candidates pour une utilisation sur FPGA. Malheureusement, il a été prouvé dans [15] qu'il est possible de localiser et de caractériser (c'est-à-dire de connaître la fréquence d'oscillation) des oscillateurs en anneau dans un circuit intégré, à partir d'une analyse EM. La Figure 1.18 montre une structure possible pour cette fonction physique non-clonable. Dans la partie haute de la Figure 1.18, un oscillateur en anneau est présenté et la partie basse montre le système complet.

Dans la plupart des publications concernant les RO-PUF, un seul bloc d'oscillateurs est utilisé (Figure 1.18.b). Cela a pour conséquence de rendre possible la modélisation de cette PUF [144].

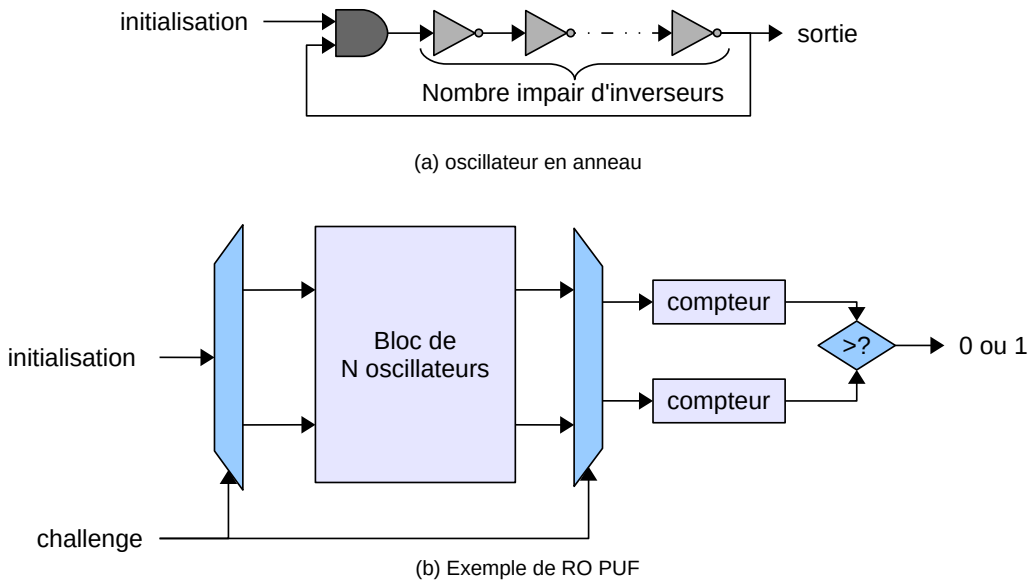

FIGURE 1.18 – Schéma d'une RO-PUF.

La *loop*-PUF est composée d'un seul oscillateur en anneau et d'une chaine de délai configurable. Cette fonction s'appuie sur la comparaison de la fréquence entre plusieurs configurations de la chaine de délais pour générer une réponse [38]. La Figure 1.19 montre la structure de cette fonction physique non-clonable.

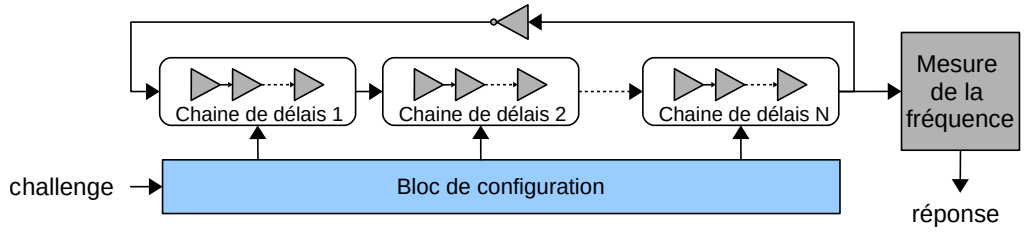

FIGURE 1.19 – Schéma d'une *loop*-PUF.

La TERO-PUF a été proposée pour contrecarrer l'analyse EM des RO-PUF [28]. La structure TERO avait déjà été étudiée pour la création d'un générateur de nombres vraiment aléatoires [172] où c'est l'état final de la cellule qui est considéré. Cette fonction physique non-clonable utilise également des oscillateurs en anneau dont les oscillations sont temporaires, contrairement à ceux utilisés pour les *RO-PUF*. Dans [28], le principe de cette PUF s'appuie sur la différence entre les nombres d'oscillations de deux oscillateurs et non sur l'état final de la cellule, ce qui permet d'obtenir plus d'informations que dans la RO-PUF où seule la fréquence est récupérée. Cela a par la suite inspiré d'autres travaux comme [67]. C'est cette fonction physique non-clonable qui est implantée et caractérisée dans le Chapitre 3.

La caractérisation des fonctions physiques non-clonables

Dans le but de prouver qu'une fonction physique non-clonable peut être utilisée comme identifiant d'un circuit, les propriétés statistiques de ses réponses sont généralement analysées grâce à trois métriques : la stabilité, l'unicité et l'imprévisibilité. De plus, le comportement de la fonction est traditionnellement étudié sous différentes conditions de température et de tension.

La stabilité représente la capacité d'une fonction physique non-clonable à répéter une réponse lorsqu'elle est soumise au même challenge. Son évaluation est faite par le calcul de la distance de *Hamming* (nombre de différences entre deux chaines binaires) entre une réponse de référence et les autres réponses du même circuit soumis au même stimuli.

L'unicité représente la capacité de la fonction physique non-clonable à différencier chaque circuit. Son évaluation est faite grâce au calcul de la distance de *Hamming* entre les réponses de référence de plusieurs circuits soumis au même challenge.

L'imprévisibilité représente l'incapacité à prévoir la réponse d'une PUF quel que soit le challenge qui lui est envoyé. Il n'existe pas réellement de test spécifique pour évaluer cette propriété dans le cadre des fonctions physiques non-clonables. Elle est généralement représentée par une simple mesure du biais des réponses de la fonction ou par le calcul de l'entropie de Shannon, ce qui est équivalent à une mesure du biais pour des chaines constituées d'une suite de blocs de 1 bit.

Plusieurs articles proposent la caractérisation de fonctions physiques non-clonables [108], principalement pour les fonctions basées sur des oscillateurs en anneau [107, 109, 175].

1.4.3 Détection de circuits usagés

Le marquage des boîtiers et des puces électroniques n'est pas la seule catégorie de mécanismes passifs de lutte contre la contrefaçon et le vol des circuits intégrés. En effet, il est également possible de détecter les circuits recyclés en utilisant des méthodes proches du *fingerprinting*, comme la mesure de paramètres physiques, ou d'inclure un système d'enregistrement du temps d'utilisation dans le circuit.

1.4.3.1 Mesures de paramètres physiques

Les techniques de *fingerprinting* basées sur l'extraction de paramètres physiques peuvent également être utilisées pour la détection des circuits recyclés. Par exemple, il est possible de déterminer l'âge d'un circuit en mesurant les délais de propagation [187]. Dans [187], il est établi que plus les temps de propagation dans un circuit sont longs, plus grande est la probabilité que le circuit soit âgé. Ainsi, une modélisation de l'évolution des temps de propagation couplée à quelques mesures de ces temps au cours de la vie des circuits permettent de détecter si un circuit est recyclé avec une probabilité de 90% [43]. La variation des délais de propagation avec l'âge des circuits intégrés est due à deux phénomènes appelés NBTI (*Negative-Bias Tem-*

perature Instability) et HCI (*Hot Carrier Injection*). Le NBTI augmente la tension de seuil des transistors PMOS, ce qui augmente le délai de propagation [106, 170]. L'effet HCI touche les transistors NMOS et cause une dégradation de la tension de seuil [71, 106]. Une caractérisation du délai de propagation des principales portes logiques ($NAND$, NOR et XOR) est présentée dans [187] et montre que le délai de propagation augmente avec l'âge du circuit. Une autre méthode consiste à construire un profil des circuits intégrés basé sur l'évolution du délai de propagation dans un chemin particulier en fonction de la fréquence d'utilisation du circuit. En fonction de l'âge du circuit, ce profil varie à cause de la variation du temps de propagation. Cette méthode à été introduite dans [169].

L'inconvénient de ces deux méthodes est la nécessité d'un circuit de référence (dont on est sûr qu'il est authentique, non modifié et non contrefait) pour relever les temps de propagation et construire le profil du circuit. De plus, elles ne sont pas facile à mettre en pratique pour les circuits analogiques ou à signaux mixtes par exemple.

Une autre approche a été proposée dans [79]. Elle est basée sur la mesure des paramètres électriques du circuit ainsi que sur l'utilisation d'une machine à vecteurs de support (SVM). Les SVM sont un ensemble de techniques d'apprentissage supervisé. Elles permettent donc de prévoir les dégradations qui peuvent intervenir à partir de mesures de certains paramètres du circuit, faites à différents moments de la vie de celui-ci. Malheureusement, cette méthode nécessite aussi un grand nombre de circuits de référence pour la construction de la SVM.

En revanche, ces techniques de détection ont l'avantage de ne rien rajouter au circuit et d'être applicables sur la plupart des circuits déjà utilisés. En effet, la caractérisation électrique ou des délais de propagation peut être effectuée sur un circuit neuf, même si le produit est déjà bien en place sur le marché et que d'autres exemplaires sont déjà utilisés.

1.4.3.2 Capteur à base d'oscillateurs en anneau

Une autre approche pour détecter les circuits usagés est d'ajouter un capteur d'utilisation. Pour cela, une première approche, proposée dans [186], consiste à comparer les fréquences d'oscillation de deux oscillateurs en anneau. Les deux oscillateurs sont supposés avoir exactement la même fréquence. Cependant, le premier oscillateur (dit de référence) est conçu pour avoir une faible sensibilité face au temps d'utilisation alors que le second (dit oscillateur stressé), est conçu de sorte que le temps d'utilisation du circuit ait un impact important sur sa fréquence. Ainsi, une grande différence entre les fréquences d'oscillation des deux oscillateurs montre que le circuit est usagé.

Pour que cette méthode soit efficace, il faut que l'impact de l'âge soit minimal sur l'oscillateur de référence et maximal sur l'oscillateur stressé. Malheureusement, cela est très difficile a obtenir à cause de la dégradation des transistors avec le temps d'utilisation. Plus de détails sur ces phénomènes de dégradation sont disponibles dans [76, 168]. Une solution à ce problème a été apportée dans [65], où l'oscillateur

stressé est spécialement conçu pour que chaque inverseur soit impacté exactement de la même manière par le temps d'utilisation. Pour cela, un transistor passant est introduit entre chaque inverseur de l'oscillateur stressé et l'entrée de ces inverseurs est connectée à la masse grâce à un réseau de transistors NMOS. La même structure est également ajoutée à l'oscillateur de référence pour être sûr de comparer deux oscillateurs théoriquement identiques [65, 160].

1.4.3.3 Enregistrement de la durée d'utilisation (*Anti-fuse*)

La dernière méthode permettant spécifiquement de lutter contre le recyclage de circuits intégrés consiste à enregistrer le temps d'utilisation des circuits. Pour cela, il est possible d'utiliser ce que l'on appelle les technologies anti-fusibles. Un anti-fusible est un circuit électronique qui passe d'un état non-conducteur (de haute résistance) à un état conducteur (de basse résistance) lorsqu'il est soumis à un stress électrique. La transition entre ces deux états est définitive et le lien créé est permanent. Les avantages des mémoires anti-fusibles face aux autres mémoires non-volatiles sont multiples [156] : (1) elles consomment moins d'énergie lors de leur programmation, (2) elles utilisent beaucoup moins de surface que les mémoires *efuse* (utilisées pour activer les circuits par IBM, voir Section 1.4.2.2), (3) elles ne nécessitent pas d'étapes supplémentaires lors de la fabrication des circuits.

Deux techniques ont été proposées pour enregistrer le temps d'utilisation grâce à cette technologie [185]. La première utilise l'horloge du système pour surveiller son âge et l'écrire dans la mémoire anti-fusible. Afin de mesurer l'âge du circuit, deux compteurs permettent de diviser la fréquence du système et donc de mesurer le temps d'utilisation avec une très grande précision. En revanche, le coût en surface de ce système peut être très important.

La seconde méthode est basée sur l'activité de certains signaux du circuit pour mesurer le temps d'utilisation. Contrairement à la première technique, celle-ci a l'avantage d'être légère en surface. Par contre, sa précision est moindre. La Figure 1.20 présente une implantation générique de ces techniques d'enregistrement du temps d'utilisation des circuits [160]. Pour adapter cette illustration aux deux méthodes présentées précédemment, il suffit de modifier le bloc de mesure temporelle (bloc rouge sur la Figure 1.20).

1.4.4 Camouflage du matériel

Le camouflage du matériel ou obscurcissement (plus connu sous le terme anglais *obfuscation*) est une technique de protection passive qui a pour but de rendre la rétro-ingénierie plus difficile d'un point de vue de l'attaquant. Il s'agit donc bien d'un mécanisme passif. En effet, une fois que le circuit a été conçu et fabriqué, seule la difficulté à retrouver son fonctionnement, ses différentes fonctionnalités et comment il a été conçu le protège de la copie, et donc du vol de la propriété intellectuelle du concepteur. Le camouflage de la partie matérielle d'un système peut être effectué à plusieurs niveaux pendant la conception du circuit.

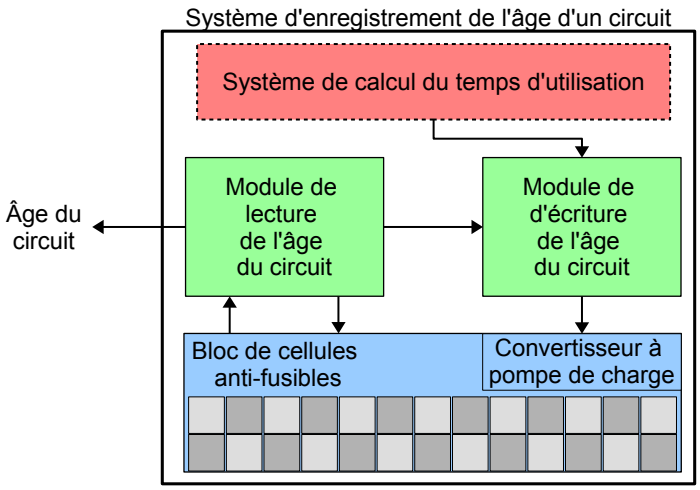

FIGURE 1.20 – Schéma générique des systèmes d'enregistrement de l'âge d'un circuit basés sur des cellules anti-fusibles [160].

La première possibilité est de camoufler la logique contenue dans le circuit. Pour cela, [31] propose de camoufler les opérations généralement utilisées dans les circuits numériques (XOR, AND, *etc*). L'application de cette méthode est directement effectuée et testée sur des algorithmes de chiffrement [31]. Cette méthode permet de camoufler la logique d'un circuit au niveau de sa conception. Il est également possible de le faire au niveau des transistors eux-mêmes avec l'utilisation de cellules particulières, pendant la conception d'un ASIC par exemple [13, 40, 42, 111, 136].

Une autre solution est de camoufler le flot d'opérations du circuit en agissant par exemple sur la machine à états finie du circuit. Dans [188], le principe est de rendre difficile la reconnaissance d'une structure dans le flot d'opérations du circuit. Cette proposition s'appuie sur le changement d'adresse de l'exécution d'un programme pour chaque apparition d'une même opération dans le traitement de données.

Dans cette section, les différents mécanismes de protection passifs permettant de lutter contre le vol et la contrefaçon des circuits intégrés ont été présentés. La plupart d'entre eux vise l'identification des circuits et apporte donc l'un des éléments manquant aux mécanismes de protection actifs (Section 1.3), afin qu'ils puissent être mis en place. La description des différentes menaces et des différents mécanismes de protection présentés dans ce chapitre ne recouvre pas toutes les menaces. Par exemple, nous avons délibérément omis de décrire les chevaux de Troie matériels ainsi que les mécanismes de protection contre cette menace [141, 163]. En effet, les travaux présentés dans ce manuscrit de thèse se concentrent exclusivement sur la contrefaçon et le vol des circuits intégrés et ne traite pas des chevaux de Troie matériels. De même, nous nous concentrons ici sur la protection des circuits intégrés et les mécanismes protégeant spécifiquement les composants virtuels (IP) n'ont pas été abordés.

Avant de présenter le projet SALWARE et quels mécanismes de protection seront étudiés dans ce manuscrit de thèse, le Tableau 1.2 reprend chacun des mécanismes de protections et indique contre quelles menaces (Section 1.1) ils permettent de lutter.

Tableau 1.2 – Quels mécanismes de protection pour quelles menaces ?

		Contrefaçon			Vol et copies illégales		rétro-ingénierie
		ré-emballage	re-marquage	recyclage	surproduction	vol du circuit	
Actif	Blocage/Déblocage	X	X		X	X	
	Reconfiguration d'une partie du circuit				X	X	X
	Séparation des étapes de fabrication	X	X		X	X	
	Chiffrement du fichier de configuration (FPGA)					X	X
Passif	Marquage ADN	X	X	X			
	QR Code	X	X				
	Revêtement non-clonable	X	X				
	Marquage sur puce	X	X			X	
	Identifiant programmé	X	X			X	
	Watermarking	X	X			X	
	Fingerprinting	X	X	X			
	Camouflage du matériel						X
	Détection de l'âge des circuits			X			

1.5 Le projet SALWARE

Le projet SALWARE est un projet «jeune chercheur» financé par l'Agence Nationale de la Recherche (ANR JCJC 2013) et co-financé par la FRAE (Fondation de la Recherche pour l'Aéronautique et l'Espace), ainsi que par la région Rhône-Alpes-Auvergne [13].

Le projet *SALWARE* [27] a pour principal objectif d'étudier et de proposer de nouveaux mécanismes de protection pour lutter contre le vol et la contrefaçon de circuits intégrés ou de composants virtuels. Le terme *salware* qui se traduit par *salutary hardware* en anglais fait opposition au terme *malware* (pour *malicious hardware* en anglais). Un *salware* utilise les mêmes stratégies et moyens que les *malwares* pour apporter une protection supplémentaire à un composant virtuel ou à un circuit intégré. La Définition 1 présente la définition officielle de ce terme [23] :

Définition 1. *Un matériel salutaire (ou salware) est un petit système matériel qui est inséré dans un circuit intégré ou un composant virtuel. Il doit être difficilement détectable ou difficile à contourner (du point de vue de l'attaquant). Enfin, un salware a pour but de fournir des informations concernant la propriété ou permettre l'activation du circuit après sa fabrication ou pendant son utilisation.*

Grâce à cette définition, il est possible de trouver des exemples de *salwares* dans la littérature liée à la lutte contre la contrefaçon et le vol de circuits intégrés. En effet, il est possible de classer comme *salwares* des solutions telles que le *watermarking*, les fonctions physiques non-clonables ou le blocage et déblocage de circuits.

Contrairement aux protections proposées dans la littérature, le mécanisme que le projet propose regroupe plusieurs solutions pour n'en former qu'une seule. Ce mécanisme a pour but de permettre l'activation d'un circuit authentifié. Les différents mécanismes qui sont regroupés dans ce schéma d'activation sont : le *watermarking*, les fonctions physiques non-clonables, la cryptographie légère ainsi que le blocage et déblocage d'un composant virtuel. Il s'agit donc d'une combinaison de mécanismes actifs et passifs pour ne former qu'une seule solution qui se veut à la fois réaliste et applicable pour le plus grand nombre de circuits possible.

Afin de concevoir un tel mécanisme, plusieurs aspects sont à prendre en compte. Premièrement, il faut un moyen d'identifier de manière unique chaque circuit. Ensuite, il faut être capable de communiquer cet identifiant de manière sécurisée. Enfin, le circuit doit être activé. La Figure 1.21 présente une vue d'ensemble de ce mécanisme de protection.

13. `http://www.univ-st-etienne.fr/salware/`

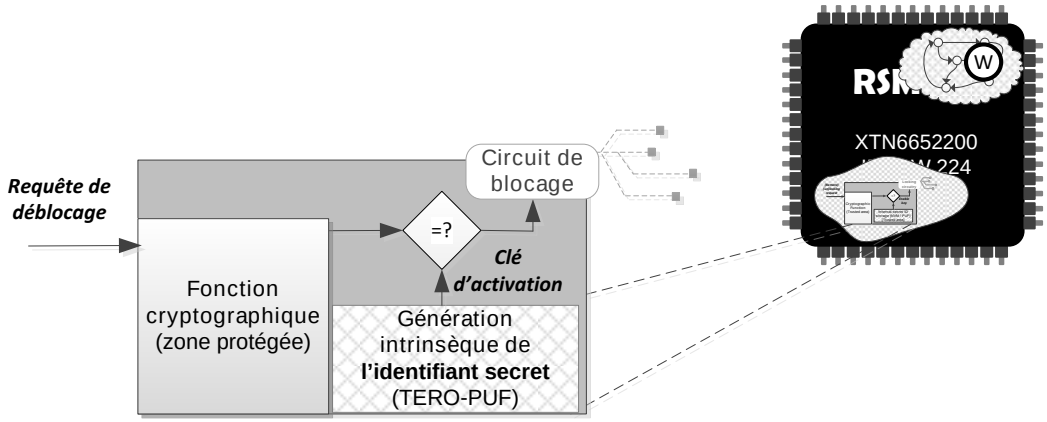

FIGURE 1.21 – Mécanisme de protection proposé par le projet SALWARE.

Dans cette thèse, trois mécanismes appartenant à ce schéma d'activation sont étudiés. Les deux premiers permettent d'identifier un circuit. Il s'agit du *watermarking* et des fonctions physiques non-clonables. Le dernier correspond à la partie cryptographique du mécanisme. Seuls des algorithmes (ultra-légers) de chiffrement par bloc sont étudiés dans ce manuscrit.

Comme l'activation d'un circuit ne se fait qu'une seule fois dans sa vie, il est impératif que l'intégralité du mécanisme soit léger en termes de surface. En effet, il est peu probable qu'un concepteur accepte de payer pour un mécanisme d'activation prenant beaucoup de surface puisqu'il ne sera utilisé qu'une seule fois. Il est donc nécessaire de réduire la surface de chaque partie du mécanisme d'activation. En revanche, le temps d'activation n'est pas un problème. Cette thèse ne traite pas du blocage des circuits intégrés, ni du protocole de communication, qui font partie du sujet de thèse d'un autre membre du projet SALWARE.

Dans la Section 1.4, une attention particulière a été apportée au *watermarking* et aux fonctions physiques non-clonables. Seule la partie concernant l'implantation légère d'algorithmes de chiffrement n'a pas été abordée. La prochaine section présente une vue d'ensemble rapide de l'implantation légère d'algorithmes de chiffrement par bloc.

1.6 Implantation légère d'algorithmes cryptographiques

La communication qui est établie entre le concepteur d'un composant virtuel et le système lors de son activation doit être sécurisée. Pour cela, un algorithme cryptographique doit être utilisé afin de chiffrer et déchiffrer les messages. Dans le cadre du projet SALWARE, la seule contrainte d'implantation est la surface. Le débit ainsi que la consommation de puissance ne sont pas pris en compte car le système d'activation n'est utilisé qu'une seule fois dans la vie d'un circuit.

Les algorithmes de chiffrement par bloc font partie des fonctions cryptographiques symétriques (ou à clé secrète). Ils transforment un message en un chiffré de

même taille grâce à une clé secrète. Il est également possible de déchiffrer un message en utilisant la même clé. D'une manière générale, les algorithmes de chiffrement par bloc sont composés de petites opérations organisées pour former une fonction qui est appliquée plusieurs fois sur le message. La clé est également transformée par une fonction de gestion de clé (*key-schedule* en anglais). Cette fonction permet de générer une nouvelle clé, dérivée de la clé principale, pour chaque itération de la fonction qui transforme le message. La Figure 1.22 présente un schéma général décrivant le chiffrement par bloc d'un message.

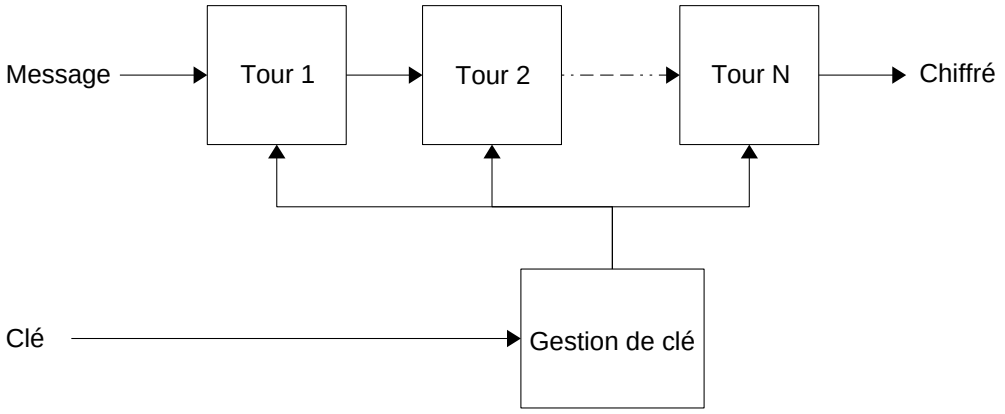

FIGURE 1.22 – Schéma générique d'un chiffrement par bloc.

L'algorithme de chiffrement par bloc le plus connu est l'*AES* [50], qui a remplacé l'algorithme *DES* [127]. Ces deux algorithmes s'appuient sur deux architectures complètement différentes qui sont reprises pour presque tous les autres algorithmes de chiffrement par bloc : un réseau de permutation et de substitution pour l'*AES*, et un réseau de Feistel pour *DES*.

Récemment, le nombre d'algorithmes spécialement développés pour les applications possédant de fortes contraintes, comme les puces de radio-identification (RFID) ou les réseaux de capteurs, a connu une rapide augmentation. L'un des algorithmes les plus connus et étudiés dans ce domaine de la cryptographie légère est PRESENT [20], qui sert aujourd'hui de référence pour comparer les nouveaux algorithmes et prouver qu'ils sont adaptés aux applications possédant de très fortes contraintes de consommation d'énergie, de puissance et de surface. En raison de la forte augmentation du nombre de ces algorithmes de chiffrement, plusieurs études essaient de les comparer [52, 116]. Dans la plus récente [116], les implantations matérielles et logicielles de 50 algorithmes sont comparées. De plus, pour chaque nouvelle proposition, une comparaison avec d'anciens algorithmes est effectuée comme dans l'article qui présente la famille d'algorithmes SIMON et SPECK [16] où ces deux familles d'algorithmes de chiffrement sont comparées à cinq autres : Twine [159], Present [20], Piccolo [150], Katan [34] et Klein [58]. Malheureusement, toutes les applications possédant de fortes contraintes, n'ont pas les mêmes besoins, et comparer deux algorithmes qui ne visent pas les mêmes applications est compliqué.

Lorsque le terme léger est mentionné pour un algorithme de chiffrement, cela s'accompagne généralement d'un bloc de données de taille réduite (32, 48 ou 64 bits) et d'une clé de taille réduite (64, 80 ou 96 bits) même si certains algorithmes conservent une clé de 128 bits. La taille la plus répandue est un message de 64 bits et une clé de 80 bits. De plus, les algorithmes de chiffrement légers sont traditionnellement construits grâce à des opérations qui transforment des petits mots (quartets ou bits) afin d'être implantés sur ASIC ou FPGA avec un minimum de ressources. De même, la fonction de gestion de clé est souvent réduite et seule une sélection de certains bits de la clé principale est effectuée pour certains algorithmes.

1.6.1 Les réseaux de substitution et de permutation

Cette structure est la plus répandue parmi les algorithmes de chiffrement par bloc et est celle utilisée dans l'algorithme *AES*. Un algorithme construit autour d'un réseau de substitution et de permutation comporte trois types d'opérations qui modifient le message :

- une opération d'ajout de la clé de ronde. Cette opération est généralement réalisée grâce à un *ou exclusif* appliqué entre la clé et le message ;
- une étape de substitution non linéaire et composée de tableaux de substitution (connus sous l'acronyme Sbox). Un tableau de substitution prend un mot de n bits en entrée et le remplace par un autre mot de m bits. La plupart du temps, la taille des mots en entrée et en sortie est la même. De plus, même si la taille la plus utilisée pour les entrées et les sorties de ces tableaux de substitution est 8 bits, les algorithmes de chiffrement légers optent parfois pour des tableaux transformant des mots de 4 bits ;
- une étape de diffusion qui est souvent réalisée grâce à une permutation et une fonction de mélange. L'opération de mélange est généralement réalisée par la multiplication du message avec une matrice bien choisie. La permutation change simplement l'ordre des mots du message.

Les algorithmes de chiffrement par bloc utilisant ce type d'architecture ont généralement besoin de moins de passages dans la fonction que les réseaux de Feistel et les autres types de construction. Ils sont relativement faciles à implanter dans le matériel et offrent un bon compromis entre la surface et le débit. Parmi les algorithmes légers utilisant cette structure, on peut par exemple citer KLEIN [58], LED [66], ou encore Midori [11].

1.6.2 Les réseaux de Feistel

Les réseaux de Feistel représentent le deuxième grand groupe architectural d'algorithmes de chiffrement par bloc. Un réseau de Feistel est également construit autour d'une fonction qui est appliquée plusieurs fois. Cependant, au lieu d'être appliquée sur le message entier, la fonction n'est appliquée que sur la moitié du message à chaque passage. Ensuite, une permutation échange les deux moitiés afin

que la fonction soit appliquée sur l'autre partie du message au prochain tour.

Cette structure est très populaire dans le domaine de la cryptographie légère car les réseaux de Feistel classiques utilisent exactement les mêmes fonctions pour le chiffrement et le déchiffrement des messages. Ainsi, seul le chiffrement peut être implanté dans le matériel, ce qui procure un avantage en termes de surface. Parmi les algorithmes de chiffrement légers qui utilisent cette structure, on peut par exemple citer LBlock [180], ITUbee [87] ou encore SEA [157].

Cependant, la permutation peut aussi être plus compliquée qu'un simple échange des deux moitiés du message, et également changer l'ordre des mots au sein de chaque moitié. Dans ce cas, on parle de réseaux de Feistel généralisés comme Lilliput [17], Hight [73] ou Clefia [152]. La permutation inverse doit être implantée pour ces algorithmes, ce qui amoindrit légèrement l'avantage concernant la surface d'implantation. La Figure 1.23 présente un schéma général qui décrit les différents types de réseaux de Feistel.

1.6.3 Les autres types de structure

Une autre famille d'algorithmes de chiffrement possède une structure très intéressante : les algorithmes KATAN et KTANTAN [34]. Ils sont basés sur des registres à décalage qui ne modifient qu'un seul bit du message par tour. Cette structure est très gourmande en nombre de passages dans chaque fonction mais permet une implantation très légère en surface. Il est possible de comparer cette famille d'algorithmes avec des algorithmes de chiffrement par flot comme Trivium [33] par exemple.

L'implantation d'algorithmes légers de chiffrement par bloc sera abordée plus en détails dans les Chapitres 4 et 5. Dans le chapitre 4, l'algorithme Lilliput ainsi que ces implantations parallèle et série sont présentées en détail. Le Chapitre 5 compare les implantations de Lilliput avec celles que trois autres algorithmes : Klein, LED et KTANTAN. Chacun de ces trois algorithmes y est présentés et leurs implantations parallèle et série sont détaillées.

1.7 Conclusion

Dans ce chapitre, nous avons présenté le contexte dans lequel les travaux présentés dans ce manuscrit ont été fait. Le cycle de vie d'un circuit intégré a été décrit ainsi que les menaces qui apparaissent. De plus, les différents types de solutions proposées ont été présentés. Enfin, le projet SALWARE a été décrit et une description détaillée des trois mécanismes de protection étudiés dans la suite de la thèse a été effectuée.

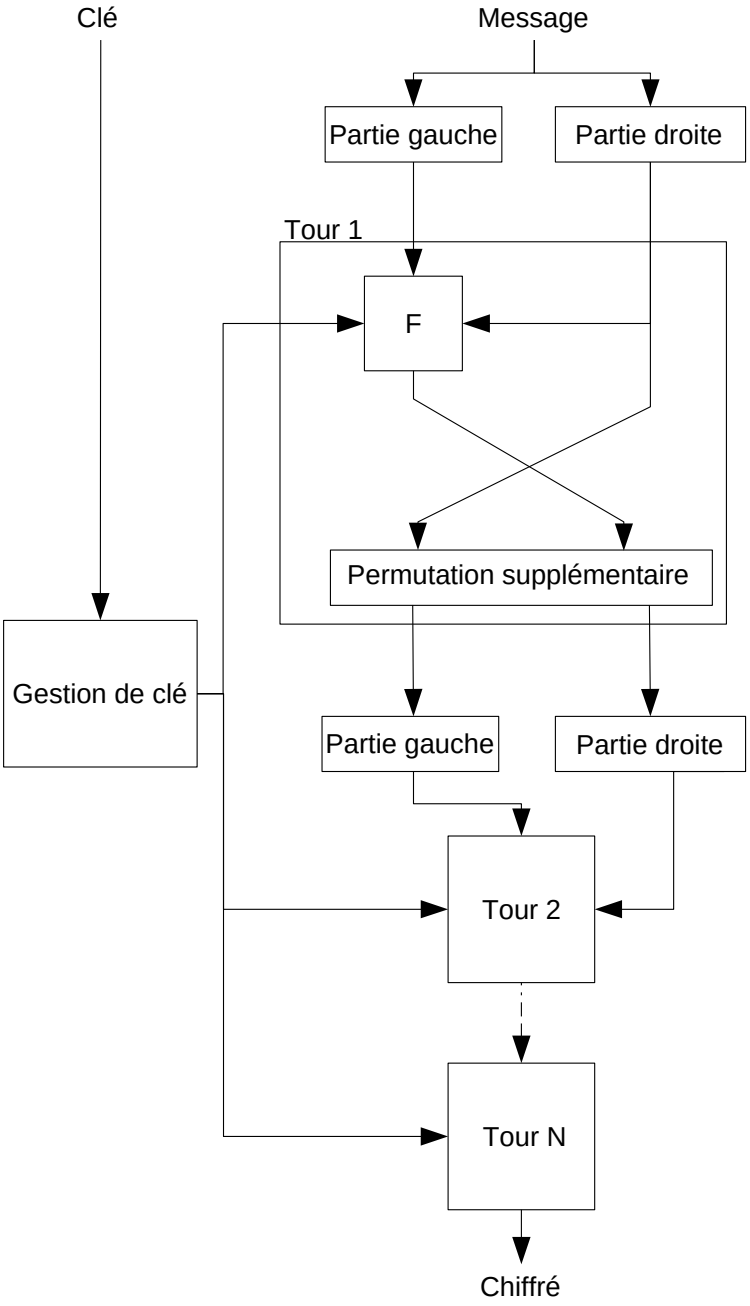

FIGURE 1.23 – Schéma générique d'un réseau de Feistel.

Chapitre 2

Insertion et vérification d'une *watermark* dans une machine à états finie

Ce chapitre s'intéresse aux problèmes d'insertion et de vérification d'une *watermark* présente dans une machine à états finie d'un système numérique synchrone. La Figure 2.1 montre le positionnement du *watermarking* (W entouré en rouge) dans le projet SALWARE. Comme il est possible de le remarquer, ces travaux ne font pas partie du mécanisme d'activation proposé par le projet SALWARE. Cependant, la vérification d'une *watermark* peut être utilisée à n'importe quel moment de la vie d'un circuit afin de lutter contre le vol et la contrefaçon.

La première section revient sur les différentes méthodes qui permettent d'insérer une marque, et en propose une qui a été élaborée grâce à une collaboration avec l'équipe du Professeur Edward Jung de la *Kennesaw State University* en Géorgie aux États-Unis. La méthode de vérification proposée est détaillée dans la Section 2.2. Cette méthode utilise le canal auxiliaire de la puissance consommée afin de retrouver la marque qui a été insérée dans une machine à états finie. La dernière section de ce chapitre présente deux expériences dont les résultats valident le fonctionnement de la méthode de vérification.

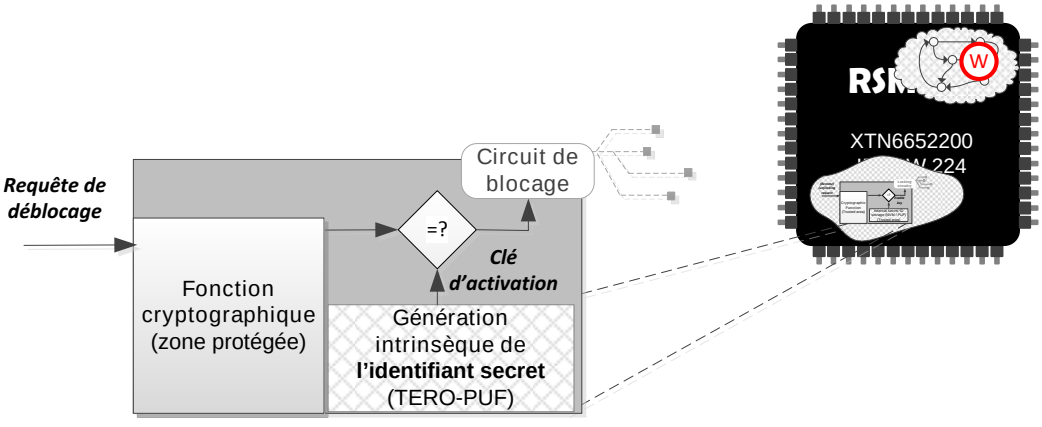

Figure 2.1 – Positionnement du *watermarking* dans le projet SALWARE.

2.1 Insertion algorithmique d'une *watermark*

Avant de décrire comment insérer algorithmiquement une *watermark* dans une machine à états finie, la prochaine sous-section reprend les techniques usuelles qui peuvent être utilisées pour marquer une machine à états finie.

2.1.1 Insertion d'une *watermark* dans une machine à états finie

Pour marquer efficacement un système numérique synchrone, il est nécessaire d'inclure suffisamment d'information afin d'identifier la propriété intellectuelle du système. Il faut également que la *watermark* soit difficile à supprimer ou à recréer. C'est pour cela que la machine à états finie d'un système est une cible intéressante pour l'insertion d'une *watermark*. Cette partie est le cœur de tous les systèmes numériques synchrones et modifier la machine à états finie d'un système complexe conduit souvent à l'endommager définitivement.

Plusieurs techniques, qui ont été brièvement évoquées dans le Chapitre 1, permettent d'inclure une *watermark* à l'intérieur d'une machine à états finie en y ajoutant des éléments (transitions ou états). La Figure 2.2 présente une machine à états finie simple qui sera utilisée dans ce chapitre comme exemple pour illustrer les différentes techniques d'insertion.

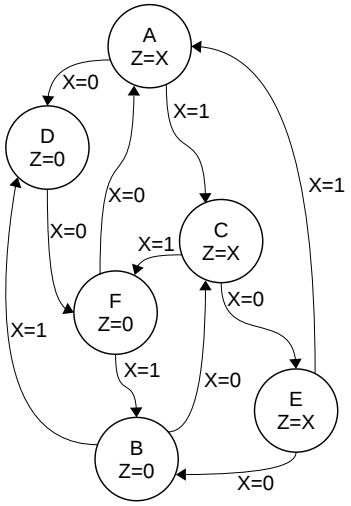

FIGURE 2.2 – Exemple de machine à états finie (M) à marquer.

Cet exemple simple de machine à états finie comporte six états (A, B, C, D, E et F). Pour chaque état, la sortie de la machine (Z) est définie, ainsi que les transitions qui dépendent de l'entrée de la machine (X).

2.1.1.1 Techniques d'insertion par ajout

Les techniques d'insertion de *watermark* abordées dans le Chapitre 1 peuvent être catégorisées comme techniques par ajout. Elles permettent d'inclure des informations supplémentaires à l'intérieur d'un système. Si l'utilisateur n'a aucune connaissance de ces informations, il lui est normalement très difficile de les trouver puisqu'elles ne sont pas divulguées lors du fonctionnement du circuit. De plus, l'ajout d'une *watermark* par l'utilisation de ces techniques ne modifie pas le fonctionnement de la machine à états finie.

Deux techniques d'insertion par ajout sont communément utilisées. La première consiste à rajouter des états ainsi que des transitions [129]. La seconde technique n'ajoute que des transitions [167], en puisant parmi celles qui ne sont pas utilisées lorsque le système fonctionne normalement. La Figure 2.3 présente simplement ces deux techniques.

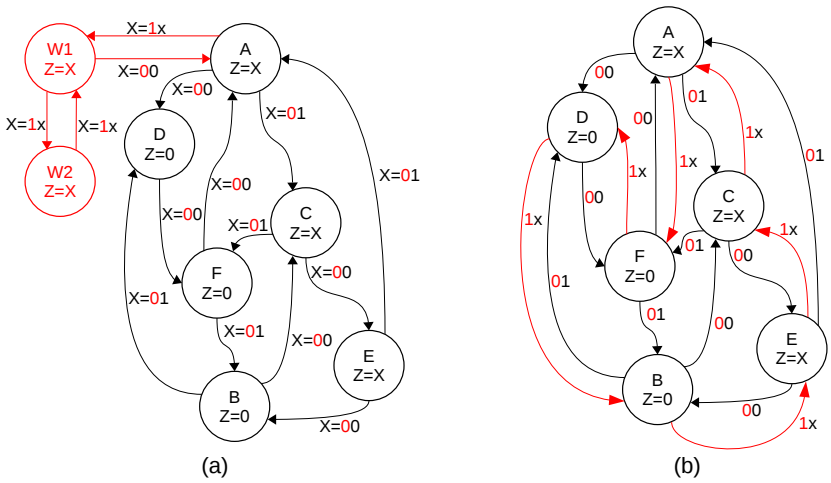

FIGURE 2.3 – Exemples de *watermarking* par ajout (éléments en rouge) dans la machine M.

Sur la partie gauche de la Figure 2.3, deux états ont été rajoutés. Les transitions entre les différents états ont également été modifiées afin de permettre à la *watermark* d'être vérifiée. Toutes ces modifications sont visibles en rouge sur la Figure 2.3. Sur la partie droite de cette figure, uniquement des transitions ont été rajoutées (en rouge). Elle ont été sélectionnées de sorte à créer un cycle de six transitions : peu importe depuis quel état la séquence de vérification de la *watermark* démarre (A, B, C, D, E ou F), un cycle de six transitions permet de revenir à cet état de départ. Dans les deux cas, la fonctionnalité de base de la machine à états finie est identique à celle présentée dans la Figure 2.2. Bien que ces deux techniques soient relativement simples à mettre en œuvre, elles possèdent deux inconvénients : (1) la vérification de la *watermark* nécessite un accès direct aux registres d'états et les articles de la littérature n'expliquent pas concrètement comment extraire la *watermark*, (2) elles ajoutent des ressources à la réalisation de la machine à états

finie dans la plupart des cas.

L'idéal serait donc (1) une méthode d'insertion de *watermark* qui n'ajoute ni transitions, ni états à la machine à états finie et (2) une méthode d'extraction du comportement de la machine à états finie non invasive.

2.1.1.2 Technique d'insertion algorithmique

La principale contribution de ce chapitre est de proposer à la fois une méthode d'insertion de *watermark* mais aussi et surtout une méthode de vérification (Section 2.2.2) qui ne nécessite aucun envoi d'information sur les sorties du système. La technique d'insertion de marque proposée [85] est complètement différente des deux précédentes. Elle repose sur l'extraction d'une propriété de la machine à états finie lors de sa conception et non sur l'ajout d'éléments. Cette méthode a été proposée conjointement avec l'équipe du Professeur Edward Jung de l'Université de la *Kennesaw State University* en Géorgie aux États-Unis. Cette technique d'insertion de *watermark* n'est pas une contribution strictement parlant de cette thèse, qui se concentre sur la méthode de vérification. Aussi, nous décrivons succinctement cette technique d'insertion dans cette section. Pour plus de précision, le lecteur intéressé peut se référer à [85].

Afin d'être capable d'extraire une propriété particulière d'une machine à états finie, son concepteur doit fournir un effort supplémentaire et trouver ce qui différencie la machine à implanter des autres. Par exemple, il peut s'agir de trouver un cycle d'état qui est indépendant des entrées, ou un enchaînement particulier d'états qui se retrouve pour chaque opération du système. Ensuite, la machine à états finie doit être repensée afin d'extraire cette propriété qui lui est intrinsèque. Pour des machines d'état complexes, chacune des propriétés est complètement spécifique et peut servir de *watermark*. Ainsi, aucun élément n'est ajouté au système et la marque est la machine à états finie elle-même. Cette méthode a un sérieux avantage face aux deux autres techniques : il est impossible d'enlever une *watermark* insérée avec cette méthode puisque cette marque est le cœur du système. De plus, si un concurrent parvient à copier le système complet, la *watermark* sera automatiquement incluse dans la copie et le véritable propriétaire de l'IP pourra prouver qu'elle lui a été volée.

Reprenons la machine exemple de la Figure 2.2. Il est possible de remarquer la chose suivante : peu importe l'état depuis lequel on démarre, une suite de trois transitions permet de retourner à cet état de départ ou de finir sur un seul autre état. Par exemple, pour l'état A, toutes les suites de trois transitions reviennent sur A ou se terminent sur l'état B. De même, toutes les suites de trois transitions démarrant de l'état D reviennent sur D ou arrivent sur l'état C et celles démarrant de E finissent soit sur E, soit sur F.

En regroupant A avec B, C avec D, et E avec F, il est possible d'extraire ce comportement. Ainsi, trois nouveaux états "composites" sont créés : $A' = \{A, B\}$, $B' = \{C, D\}$ et $C' = \{E, F\}$. Ces trois nouveaux états constituent la première partie (FSM_w) de la nouvelle forme de la machine à états finie. Les transitions de FSM_w

sont cycliques et indépendantes de l'entrée X de la machine M. C'est cette première partie de la nouvelle machine à états finie qui représente la *watermark* du système. Une seconde partie (FSM_r) permet de générer la sortie Z à partir l'entrée X et de la sortie de FSM_w. Elle est composée de deux états (S1 et S2). À partir de cette division en deux sous-machines à états finies, la machine exemple de la Figure 2.2 est entièrement décrite. Ainsi, la propriété de cyclicité de la machine à états finie M est extraite. La Figure 2.4 représente la nouvelle machine à états finie dont le fonctionnement est identique à celui de la Figure 2.2.

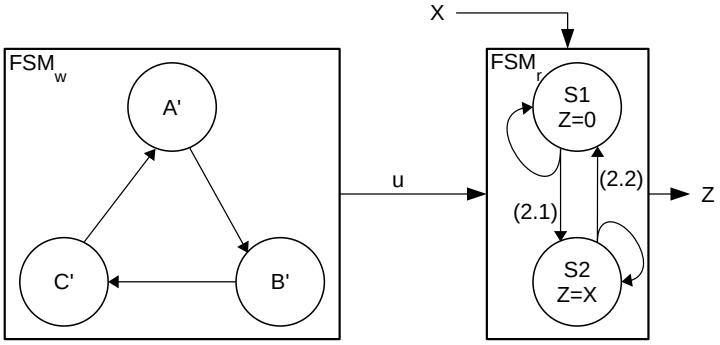

FIGURE 2.4 – Insertion algorithmique d'une *watermark* dans la machine à états finie présentée dans la Figure 2.2.

Le Tableau 2.1 montre les correspondances entre les états de la machine à états finie M et les états composites de la nouvelle machine. Les transitions entre les états de la machine FSM_r, visibles sur la Figure 2.4 (2.1 et 2.2) sont définies par les équations 2.1 et 2.2. Les transitions entre les états de la sous-machine à états finie FSM_w (A', B' et C') ne nécessitent aucune entrée et sont synchronisées avec l'horloge du système.

Tableau 2.1 – Correspondance entre les états de la machine à états finie exemple (Figure 2.2) et les états composites de la nouvelle machine à états finie (Figure 2.4).

État d'origine (Machine M)	FSM_w	FSM_r
A	A'	S2
B	A'	S1
C	B'	S2
D	B'	S1
E	C'	S2
F	C'	S1

$$S1 \rightarrow S2 \ : \ A' \cdot \overline{X} + C' \cdot \overline{X} \tag{2.1}$$

$$S2 \rightarrow S1 \ : \ A' \cdot \overline{X} + B' \cdot X + C' \cdot \overline{X} \tag{2.2}$$

Contrairement aux méthodes présentées précédemment (Sous-Section 2.1.1.1), cette méthode ne nécessite pas l'ajout de ressources supplémentaires et la vérification comportementale de la *watermark* doit passer par l'identification de la propriété de «cyclicité» la machine à états finie dans sa nouvelle forme.

La prochaine section porte un intérêt particulier à la vérification des *watermarks* qui est souvent oubliée ou laissée pour évidente. La méthode de vérification qui est proposée a pour principal objectif de distinguer des machines à états finies cycliques. Toutefois, s'il est possible de vérifier un comportement aussi simple que la cyclicité d'une FSM, il est théoriquement possible de vérifier les *watermarks* insérées grâce à d'autres techniques d'insertion (comme les techniques par ajout présentées dans la Section 2.1.1.1).

2.2 Conception d'une méthode de vérification basée sur les canaux auxiliaires

Dans cette thèse, la méthode de vérification de *watermark* proposée tire parti de l'analyse des canaux auxiliaires. Ces derniers sont bien connus dans le domaine des attaques contre les algorithmes cryptographiques et permettent de retrouver des informations sur ce qu'il se passe à l'intérieur d'un système. Dans le cadre des attaques par canaux auxiliaires, les informations permettent notamment de retrouver la clé secrète utilisée pour chiffrer ou déchiffrer un message. Ces mêmes informations peuvent être utilisées de façon salutaire (c'est à dire pour un *salware*) dans le cadre de la lutte contre la contrefaçon et le vol de circuits intégrés.

2.2.1 Utilisation des canaux auxiliaires

Les canaux auxiliaires constituent une palette d'outils très puissants pour extraire des informations concernant le fonctionnement d'un circuit sans le modifier. En effet, une simple mesure de la puissance consommée aux bornes du circuit [90], ou celle de son émission électromagnétique, ou photonique [56] permet d'obtenir beaucoup d'informations. Dans les travaux présentés dans ce chapitre, le canal auxiliaire utilisé est la consommation de puissance. Ce canal se mesure par la récupération de la tension sur les ports d'alimentation du circuit. Chaque opération effectuée dans le circuit provoque des variations de cette tension et leur analyse apporte des informations concernant le comportement du circuit, les données qu'il traite ou encore l'opération qui est effectuée au moment de la mesure.

Il est important de retenir que les informations recherchées doivent permettre de retrouver une marque insérée dans une machine à états finie. Nous allons montrer que la méthode de vérification de *watermark* proposée dans ces travaux permet de retrouver le comportement cyclique qui a été extrait de la machine M (Figure 2.2) ; ce qui illustrera la mise en pratique de la méthode d'insertion algorithmique. Cette méthode de vérification peut également être utilisée pour vérifier d'autres types de *watermarks* et ne se limite pas uniquement à la méthode d'insertion algorithmique

présentée dans la section 2.1.1.2. La Figure 2.5 présente de manière générale le concept de la vérification d'une *watermark*.

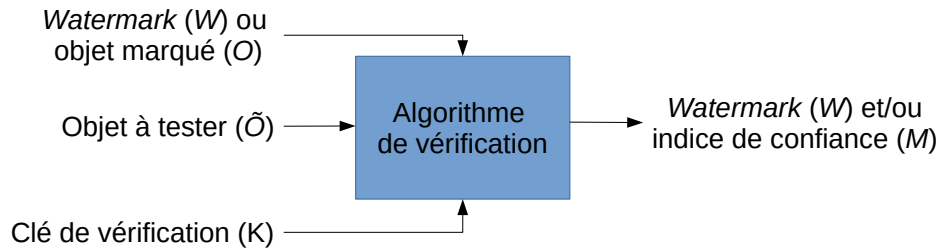

FIGURE 2.5 – Concept général de la vérification d'une *watermark*.

Considérons un circuit quelconque (Õ) dont la *watermark* doit être vérifiée. Il faut disposer de la marque à vérifier (W) ou d'un circuit de référence dont on est sûr qu'il contient la *watermark*. Enfin, il est possible d'avoir à utiliser une clé (K) pour activer la vérification et extraire la marque contenue dans Õ. Ensuite, un algorithme de vérification est utilisé pour retrouver la *watermark* mais il se peut aussi que le résultat de cette vérification soit seulement un indice de confiance sur la présence de la marque recherchée. C'est exactement ce que permet une méthode de vérification basée sur les canaux auxiliaires. La Figure 2.6 ajoute la partie concernant l'analyse des canaux auxiliaires au concept de vérification d'une *watermark*.

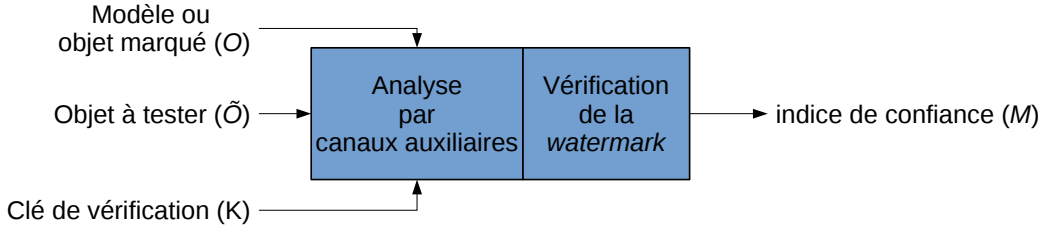

FIGURE 2.6 – Vérification d'une marque à partir des canaux auxiliaires.

Les objets nécessaires pour une vérification basée sur les canaux auxiliaires sont les mêmes que ceux nécessaires pour une vérification dans le cas général (Figure 2.5). Pour la référence, il peut s'agir d'un circuit dont on est sûr qu'il contient la marque mais il peut également s'agir d'un modèle liant la *watermark* au canal auxiliaire utilisé. Les types de modèles qui peuvent être utilisés, comme la distance ou le poids de *Hamming*, sont très largement utilisés dans le domaine des attaques par canaux auxiliaires. La prochaine section décrit en détail la méthode de vérification proposée.

2.2.2 Méthode de vérification d'une *watermark*

La méthode de vérification de *watermark* qui est décrite dans ce chapitre est constituée de trois grandes étapes : la mesure, le traitement et l'analyse. À la fin de

la vérification, un indice de confiance quant à la présence de la *watermark* cherchée est calculé.

Commençons par décrire chacune des trois étapes de la vérification de façon générale.

1. La première étape est l'acquisition des mesures du canal auxiliaire. Pour un circuit C, un nombre entier positif ($n \in \mathbb{N}^*$) de mesures de consommation de puissance (Pw) est effectué. Les n traces de consommation sont regroupées dans un ensemble (T_C) qui peut s'écrire :

$$T_C = Pw(C, n) \tag{2.3}$$

   Chacune des n traces de cet ensemble a une longueur, en nombre de points d'échantillonnage, qui sera notée l dans la suite de cette section.

2. La deuxième étape du processus de vérification consiste à sélectionner et à moyenner des traces de consommation afin d'en diminuer les bruits de mesures. Considérons donc un ensemble X de n traces. k éléments distincts ($\{e_1, e_2, \cdots, e_k\}$) sont sélectionnés uniformément dans X. Cette première fonction de la seconde étape est noté $U_X(k)$ et définie par :

$$\forall k \in [1; n],\ U_X(k) = \{e_1, ..., e_k\},\ \forall (i, j) \in [1; k],\ i \neq j \Leftrightarrow e_i \neq e_j \tag{2.4}$$

   Ensuite, les k éléments sélectionnés sont moyennés. La fonction moyenne est notée *mean*. Cette opération de sélection et de moyenne peut être répétée plusieurs fois pour former un nouvel ensemble noté A qui contient alors m traces moyennées. Ainsi pour un circuit C quelconque, si m moyennes sont effectuées à partir des sélections successives de k éléments de l'ensemble T_C, l'ensemble des traces moyennées est défini par :

$$A_{C,m} = \{mean(U_{T_C}(k))\}_m \tag{2.5}$$

   Cet ensemble de mesures moyennées est le résultat de la deuxième étape de la méthode de vérification. Par convention, cet ensemble sera noté A_C si $m = 1$.

3. La troisième et dernière étape du processus de vérification de *watermark* décrit dans cette section a pour objectif de calculer un ensemble de coefficients de corrélation entre un ensemble de mesures de test et une mesure de référence. Considérons donc un ensemble de mesures moyennées (A) ainsi qu'une trace de référence (A_{ref}). La corrélation entre chacun des éléments de A et la mesure de référence est effectuée grâce au coefficient de Pearson défini par :

$$\rho(x, y) = \frac{\sum_{i=1}^{l}(x_i - \overline{x}) \cdot (y_i - \overline{y})}{\sqrt{\sum_{i=1}^{l}(x_i - \overline{x})^2 \cdot \sum_{i=1}^{l}(y_i - \overline{y})^2}} \tag{2.6}$$

   Dans cette formule, x et y sont deux traces de longueur l et $\overline{x}$ représente la moyenne de x.
   Les valeurs ainsi calculées sont regroupées dans un nouvel ensemble noté $\mathfrak{C}$ et défini pour un circuit (C) et une référence (ref) par :

$$\mathfrak{C}_{ref,C,m,k} = \{\rho(A_{ref}, A_{C,m})\} \tag{2.7}$$

Enfin, c'est l'ensemble des coefficients de corrélation qui est analysé afin de savoir si la *watermark* est présente dans le circuit testé (appelé *DUT*). La Figure 2.7 présente ce calcul de corrélation dans le contexte de ce chapitre.

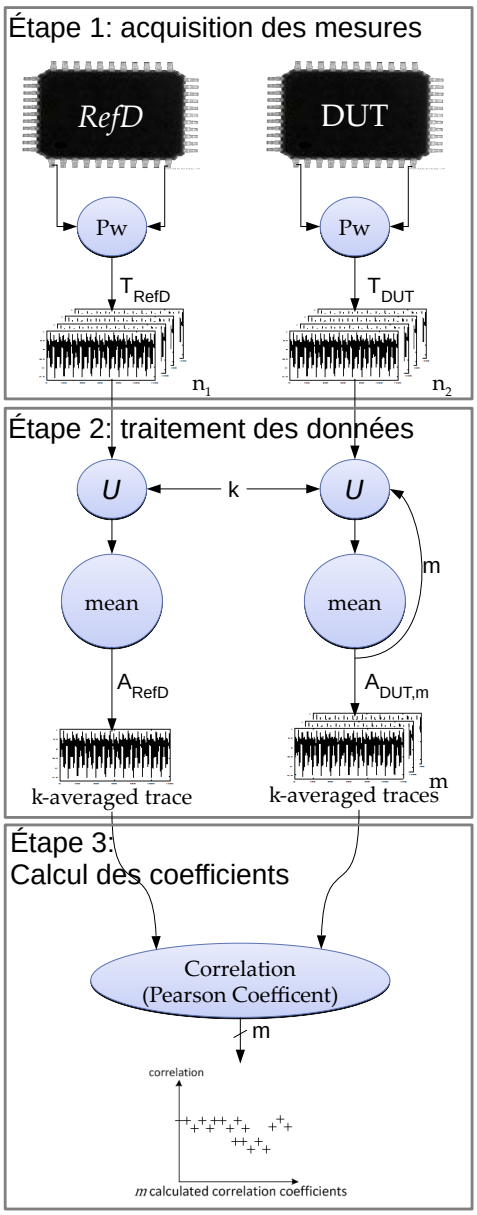

FIGURE 2.7 – Méthode de vérification appliquée aux *watermarks* insérées dans un circuit électronique.

Le schéma de la Figure 2.7 reprend les trois étapes du processus de vérification d'une *watermark* insérée dans un circuit électronique. Pour cela, un *DUT* et un circuit de référence ($RefD$) sont nécessaires. La première étape consiste à prendre

un nombre entier n_1 de mesures sur le circuit de référence ainsi qu'un nombre n_2 de mesures sur le circuit à tester. Ces deux groupes forment alors les deux ensembles T_{RefD} et T_{DUT} qui contiennent respectivement n_1 et n_2 éléments. La deuxième étape consiste alors à créer une seule courbe de référence à partir de T_{RefD} ainsi qu'un nombre m de courbes à partir de T_{DUT}, comme indiqué sur la Figure 2.7. Plusieurs mesures (k) sont donc sélectionnées aléatoirement et uniformément dans l'ensemble T_{RefD} puis moyennées. La même opération est répétée m fois pour l'ensemble T_{DUT}. Ainsi, deux nouveaux ensembles de traces moyennées sont créés : A_{RefD} et $A_{DUT,m}$.

Enfin, la troisième étape permet de calculer m coefficients de corrélation à partir de ces deux ensembles. Ce sont ces m valeurs qui seront analysées afin de savoir si la marque du circuit de référence se trouve dans le DUT. Cette analyse est détaillée dans la Section 2.4.4.

2.2.3 Choix des paramètres de la méthode de vérification

Afin de compléter cette méthode de vérification, certaines contraintes doivent être appliquées pour choisir correctement les paramètres de la méthode (n_1, n_2, k et m). La première concerne le nombre de traces acquises sur les circuits Ref_D et DUT. Il faut impérativement que le nombre de mesures effectuées sur $RefD$ (n_1) soit plus grand que le nombre k de mesures sélectionnées lors de la deuxième étape. De la même façon, le nombre de mesures effectuées sur DUT (n_2) doit être plus grand que m fois le nombre k de mesures sélectionnées :

$$n_1 \geq k \tag{2.8}$$

$$n_2 \geq k \times m \tag{2.9}$$

Cependant, il est possible de mieux choisir le nombre n_2 afin de limiter les chances qu'une trace soit sélectionnée deux fois lors de la vérification de la marque. Cette sélection aléatoire est importante pour diminuer les variations dues aux bruits qui peuvent survenir lors de l'acquisition des mesures. Par exemple, si la température change au cours de l'acquisition du canal auxiliaire, les mesures peuvent présenter de très fortes différences et conduire à de fausses interprétations. La sélection aléatoire des mesures permet de réduire cette influence en mélangeant des mesures prises à différents temps et donc différentes températures.

Toutefois, si une mesure est sélectionnée plusieurs fois, les mêmes informations se retrouveront sur plusieurs coefficients de corrélation. Pour éviter cela, le nombre de mesures sélectionnées (k) ainsi que le nombre de sélections doivent être pris en compte pour choisir le nombre de mesures (n_2) à effectuer sur le DUT.

Considérons que le nombre n_2 dépende des paramètres k et m. Il est alors possible de trouver un nombre réel α supérieur ou égal à 1 (d'après la condition 2.9) tel que :

$$n_2 = \alpha \times k \times m \tag{2.10}$$

Maintenant, considérons que les k éléments sont sélectionnés simultanément dans l'ensemble T_{DUT}. La probabilité ($P(e_i)$) qu'une mesure ($e_i,\ i \in [1, n_2]$) soit

présente dans un tirage de k mesures parmi les n_2 disponibles est :

$$\forall i \in [1, n_2], P(e_i, e_i \in T_{DUT}) = \frac{k}{n_2} \tag{2.11}$$

Et en remplaçant n_2 par son expression (2.10) dans la Formule 2.11, on obtient alors :

$$P(e_i) = \frac{1}{\alpha \times m} \tag{2.12}$$

Ainsi, nous connaissons la probabilité de sélectionner une mesure particulière lors d'un tirage. La première chose que l'on peut remarquer est que cette probabilité ne dépend pas du nombre d'éléments sélectionnés mais du nombre de tirages qui sont effectués. Il faut donc tenir compte du fait que nous faisons m tirages de k traces dans T_{DUT}. Considérons l'événement ζ suivant :

ζ : *l'élément e_i est sélectionné plus d'une fois lors de m sélections.*

Afin de calculer la probabilité de cet événement, l'événement contraire noté $\overline{\zeta}$ est utilisé. Ce dernier peut se traduire de la façon suivante :

$\overline{\zeta}$: *l'élément e_i est sélectionné au plus une fois lors de m sélections.*

La probabilité de *zeta* s'écrit :

$$P(\zeta) = 1 - P(\overline{\zeta}) \tag{2.13}$$

Afin de relier $P(\zeta)$ aux paramètres de la méthode de vérification, il faut donc exprimer $P(\overline{\zeta})$ en fonction de m, et α. De plus, pour $m \in \mathbb{N}^*$, seules deux possibilités permettent de valider l'événement $\overline{\zeta}$. Soit l'élément e_i n'a pas été sélectionné du tout, soit il ne l'a été qu'une seule fois. Or, si l'élément e_i a été sélectionné une seule fois lors des m tirages, cela signifie qu'il ne l'a pas été lors des $m-1$ autres sélections. Il y a donc exactement m possibilités pour sélectionner l'élément e_i une seule fois lors des m tirages. Ainsi, l'expression de $P(\overline{\zeta})$ peut s'écrire :

$$P(\overline{\zeta}) = (1 - P(e_i))^m + m[P(e_i)(1 - P(e_i))^{m-1}] \tag{2.14}$$

En remplaçant $P(e_i)$ par son expression (2.12) et en simplifiant $P(\overline{\zeta})$, on obtient alors :

$$P(\overline{\zeta}) = \left(1 + \frac{m-1}{\alpha \times m}\right)\left(1 - \frac{1}{\alpha \times m}\right)^{m-1} \tag{2.15}$$

Ainsi, l'expression de la probabilité de l'événement ζ devient :

$$P(\zeta) = 1 - \left(1 + \frac{m-1}{\alpha \times m}\right)\left(1 - \frac{1}{\alpha \times m}\right)^{m-1} \tag{2.16}$$

Le nombre de mesures à prendre sur le *DUT* peut donc être choisi grâce aux paramètres suivants :

- k, le nombre de mesures à sélectionner par moyenne pour l'étape 2 de la méthode de vérification ;
- α, un nombre réel supérieur ou égal à 1 ;
- la probabilité de l'événement ζ.

Ces trois paramètres vont nous permettre de choisir la valeur de m et ainsi la valeur de n_2. Les premières propriétés de l'expression 2.16 à remarquer sont ses limites à l'infini (2.17 et 2.18). L_1 correspond à la limite de l'expression 2.16 pour une valeur fixe de m et lorsque α tend vers l'infini. La limite L_2 est celle pour une valeur fixe de α lorsque m tend vers l'infini.

$$L_1 \ : \forall\ m \in \mathbb{N},\ \lim_{\alpha \to +\infty} P(\zeta) = 0 \tag{2.17}$$

$$L_2 \ : \forall\ \alpha \geq 1,\ \lim_{m \to +\infty} P(\zeta) = 1 - (\frac{\alpha + 1}{\alpha}) e^{\frac{-1}{\alpha}} \tag{2.18}$$

Fixons le paramètre α à la valeur qui sera utilisée dans la Section 2.4 : $\alpha = 10$. Cette valeur de α permet de choisir la valeur de m en fonction de la probabilité $P(\zeta)$ grâce à la limite L_2 (2.18). La Figure 2.8 représente l'évolution de $P(\zeta)$ en fonction de m pour $\alpha = 10$.

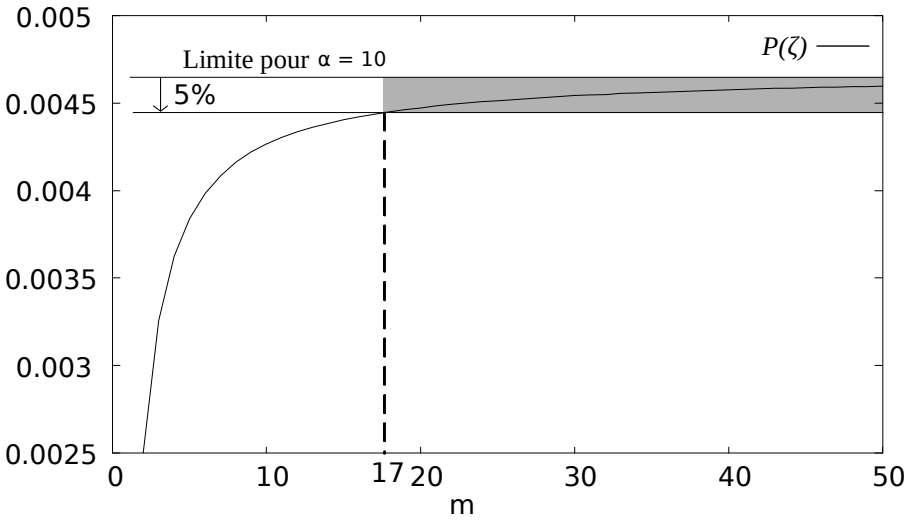

FIGURE 2.8 – Évolution de $P(\zeta)$ en fonction de m pour $\alpha = 10$.

Pour fixer la valeur de m, il est possible de s'imposer une contrainte concernant la limite L_2 de $P(\zeta)$, qui est représentée sur la Figure 2.8. Par exemple, fixons une contrainte de 5% sous cette limite, qui vaut $0,00468$ pour $\alpha = 10$. Cela signifie que m sera choisi de sorte à ce que la probabilité de sélectionner une trace plus d'une fois soit comprise entre 0.00444 et 0.00468. Ainsi, il faut choisir m supérieur ou égal à 17, comme le montre la Figure 2.8. m sera arbitrairement fixé à 20 dans le reste de ce chapitre.

Enfin, le nombre de traces à sélectionner à chaque tirage est arbitrairement fixé à $k = 50$. Ainsi, le nombre d'acquisitions à effectuer sur le DUT devient : $n_2 = 10 \times 20 \times 50 = 10000$. Le choix du nombre de mesures à prendre sur le circuit de référence est beaucoup plus simple et nécessite juste de satisfaire la condition 2.8, donc $n_1 = 50$ suffit.

Dans cette section, une méthode de vérification de *watermark* insérée dans un circuit numérique synchrone a été présentée. Cette méthode utilise le canal auxiliaire de la consommation de puissance. Plusieurs paramètres ont été introduits afin de mettre en œuvre cette méthode de vérification :

- n_1, le nombre de mesures à prendre sur le circuit de référence ;
- n_2, le nombre de mesures à prendre sur le circuit à tester ;
- k, le nombre de mesures à sélectionner et à moyenner ;
- m, le nombre de fois que la sélection est opérée sur le circuit à tester.

De plus, une méthode permettant de choisir le paramètre n_2 a été décrite. Cette méthode s'appuie sur la probabilité $P(\zeta)$ de sélectionner plusieurs fois une trace particulière de l'ensemble des traces mesurées sur le circuit de test (Section 2.2.2). La prochaine section présente la mise en place d'un banc de test qui a pour but de tester la méthode détaillée ci-dessus.

2.3 Mise en place d'un banc de test

Afin de vérifier que la méthode de vérification proposée peut être mise en pratique et fonctionne correctement, il est nécessaire de l'éprouver. Le banc de test qui a été mis en place est composé de trois parties :

- une partie matérielle ;
- un oscilloscope ;
- une partie logicielle.

La partie matérielle contient les circuits de référence et les circuits de test. Elle a été réalisée grâce au système Evariste II [53]. La Section 2.3.1 décrit succinctement ce système et comment il a été adapté pour tester la méthode de vérification de *watermark*. Plus d'informations concernant ce système matériel sont disponibles sur un site internet qui lui est dédié [1]. L'oscilloscope permet de prendre les mesures de consommation de puissance. Enfin, la partie logicielle permet d'automatiser les mesures, de calculer les coefficients de corrélation et de les analyser afin de retrouver les *watermarks*. La Figure 2.9 donne un aperçu complet du banc de test ainsi que des interactions entre ses différentes parties.

1. `https://labh-curien.univ-st-etienne.fr/wiki-evariste/index.php/Main_Page`

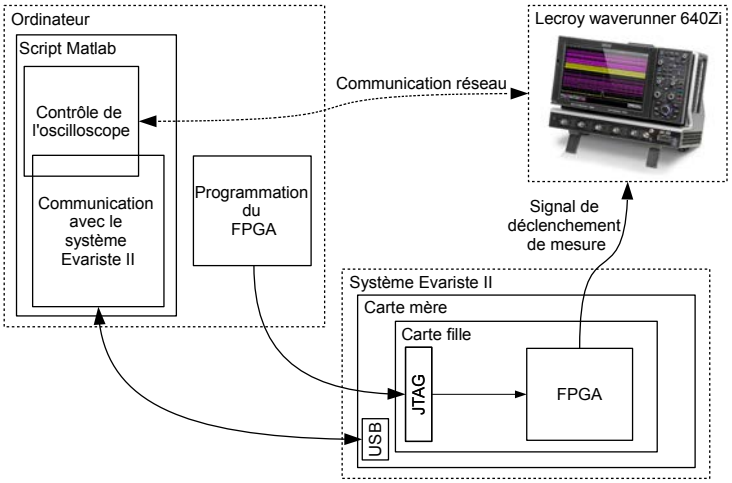

FIGURE 2.9 – Représentation schématique du banc de mesure complet.

2.3.1 Utilisation du système Evariste II

Evariste II est un système matériel modulaire [2] composé de trois grandes parties. La première partie est la carte mère du système. C'est sur cette carte que viennent se brancher des modules appelés "cartes filles". Celles-ci contiennent le circuit FPGA ou ASIC. La carte mère est également reliée à un ordinateur grâce à une connexion USB ; ce qui permet de communiquer avec le système complet. La Figure 2.10 présente la partie matérielle du système.

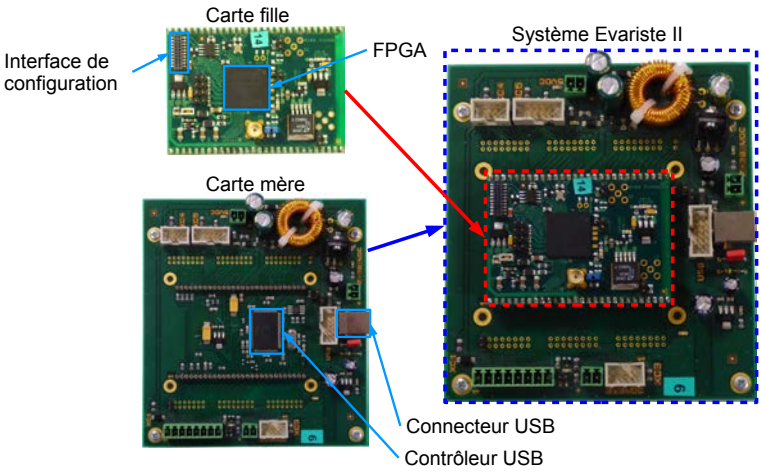

FIGURE 2.10 – Présentation du système Evariste II.

2. Il faut noter que le système Evariste II n'est pas une contribution propre de cette thèse. Il s'agit d'un système développé conjointement par de nombreux membres de l'équipe Systèmes Embarqués Sécurisés et Architectures Matérielles du Laboratoire Hubert Curien.

La carte fille est visible en haut à gauche de la Figure 2.10 et la carte mère en bas à gauche. Le système est complet lorsque les deux cartes sont assemblées, comme le montre la partie droite de la Figure 2.10. De plus, la connexion USB entre l'ordinateur et le système est assurée par un outil logiciel qui se présente sous la forme d'un exécutable auquel il faut fournir un script d'entrée ainsi que le nom du fichier de sortie. Cet outil envoie alors le script à la carte fille via la carte mère. Le fichier de sortie contient le résultat des opérations demandées à la carte fille. Les formats de ces fichiers ont été définis lors de la conception de ce système modulaire.

Pour implanter une application dans les FPGA, un projet dans lequel il est possible d'encapsuler n'importe quelle application est disponible avec l'environnement Evariste II. La Figure 2.11 montre le schéma bloc de ce projet.

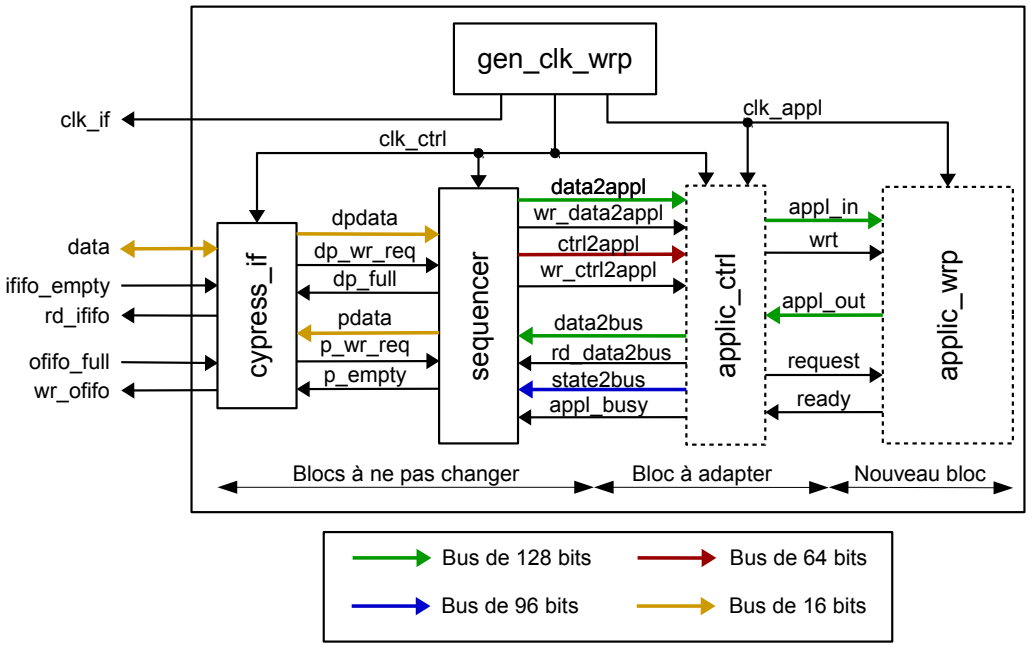

FIGURE 2.11 – Schéma bloc interne du FPGA pour le système Evariste II.

La partie de communication (*cypress_if* et *sequencer*) ne doit pas être modifiée pour intégrer une application dans ce système. Ce sont les deux blocs nommés *appl_ctrl* et *appl_wrp* qui permettent d'intégrer l'application dans ce système. Le bloc *appl_ctrl* permet de créer une interface entre l'application et la partie communication. C'est dans ce bloc que les commandes nécessaires au bon fonctionnement de l'application peuvent être insérées. Le bloc *appl_wrp* contient l'application elle-même.

2.3.2 Partie logicielle du banc de test

La dernière partie de ce banc de test est une partie logicielle développée sur Matlab qui permet d'automatiser l'enregistrement des mesures effectuées par l'os-

cilloscope ainsi que la communication avec la partie matérielle du banc. Le script Matlab assemble et gère l'exécution des éléments suivants :

- un oscilloscope Lecroy Waverunner 640Zi ;
- le logiciel de communication avec le système Evariste II ;
- le classement des mesures effectuées par l'oscilloscope.

L'oscilloscope est géré grâce à la libraire *ActiveDSO*, qui permet notamment de fixer le nombre de points que doivent contenir chaque mesure ainsi que le niveau de tension attendu pour le signal de déclenchement des mesures. La communication entre l'ordinateur et l'oscilloscope se fait par le réseau.

L'oscilloscope est ensuite mis en attente du signal de déclenchement qui lui sera envoyé directement par le FPGA. Le script d'entrée est alors transmis au système Evariste II. La mesure est classée en fonction du circuit testé ainsi que du numéro de la mesure effectuée. Toutes ces étapes sont répétées jusqu'à ce que le nombre de mesures enregistrées soient égal à celui demandé.

2.4 Résultats expérimentaux

Avant de présenter les résultats expérimentaux montrant que la méthode de vérification de *watermark* proposée fonctionne et peut être mise en pratique, la prochaine sous-section présente les machines à états finies qui ont été conçues pour l'expérience.

2.4.1 Conception des machines à états finies

2.4.1.1 Les machines à états finies

Pour des machines à états finies complexes, les transitions entre leurs états ne sont pas linéaires et chaque système aura alors des particularités qui apparaîtront directement sur les coefficients de corrélation. En revanche, l'utilisation de machines à états finies simples, cycliques, et très proches les unes des autres rendra la vérification d'une *watermark* plus difficile. C'est pourquoi nous avons choisi d'utiliser des compteurs 8 bits pour tester si la méthode de vérification de *watermark* proposée fonctionne correctement. Un compteur est cyclique et ses transitions sont parfaitement déterministes, ce qui en fait le pire cas pour la vérification par canaux auxiliaires. Il est toutefois possible d'aller encore plus loin en utilisant des compteurs dont les états sont codés à l'aide du code Gray. Pour un compteur Gray, la différence entre deux états consécutifs n'est que de un bit, ce qui diminue l'impact des transitions sur la consommation de puissance. Tout cela a pour conséquence d'augmenter les similitudes entre deux systèmes et donc de rendre la vérification par distingueur encore plus difficile.

Les deux premières machines à états finies conçues sont donc : un compteur binaire 8-bit (Ref_1) et un compteur Gray 8-bit (Ref_2). Afin de compléter les circuits de test, une *watermark* très simple est ajoutée à ces machines. Pour chacune

d'entre elles, une clé K est additionnée à l'état de sortie de la machine grâce à un *ou exclusif*. Cette nouvelle valeur est ensuite envoyée comme entrée dans un tableau de substitution (plus connu sous l'acronyme Sbox) qui est celui utilisé dans l'algorithme de chiffrement AES (*Advanced Encryption Standard*). L'ajout de cette Sbox permet d'amplifier les différences entre les transitions qui ont lieu en les rendant non linéaires. Ainsi, quatre nouveaux circuits (Ref_3, Ref_4, Ref_5 et Ref_6) sont conçus à partir des circuits de référence Ref_1 et Ref_2. Le Figure 2.12 présente les six circuits de référence conçus pour éprouver la méthode de vérification proposée.

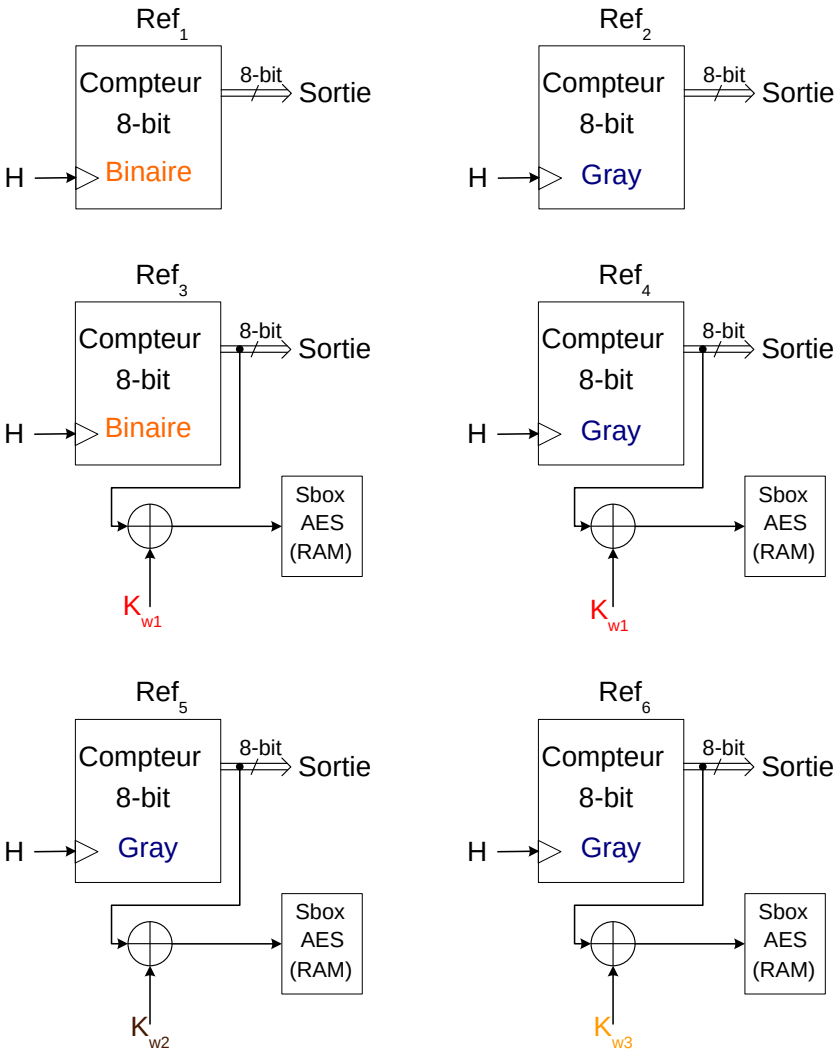

FIGURE 2.12 – Machines à états finies de référence pour la vérification de *watermark*.

Ces six compteurs sont implantés dans six FPGA Altera Cyclone III pour constituer les six circuits de référence (Ref_1, Ref_2, Ref_3, Ref_4, Ref_5 et Ref_6). Il sont également implantés dans six autres FPGA Altera Cyclone III pour créer les six circuits de test (DUT_1, DUT_2, DUT_3, DUT_4, DUT_5 et DUT_6). Le but de l'expérience

est d'identifier quel circuit de test contient quel circuit de référence (Figure 2.13).

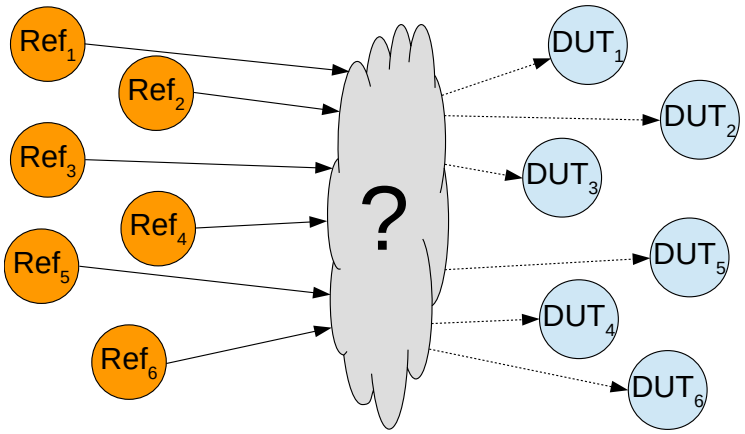

FIGURE 2.13 – Schéma du but de l'expérience de vérification de *watermark*.

Dans cette sous-section, les six machines à états finies, qui sont utilisées pour tester si la méthode de vérification de *watermark* proposée fonctionne, ont été présentée ainsi que leur intégration dans le système Evariste II. Le Tableau 2.2 résume les spécificités de chacun des six circuits de référence. Dans le Tableau 2.2, les différences entre les machines d'état sont mises en avant par la couleur des éléments qui caractérisent ces circuits. Par exemple, on peut remarquer que la seule différence entre les circuits Ref_1 et Ref_2 est le codage des états. Il en est de même pour les circuits Ref_3 et Ref_4. En revanche les circuits Ref_4, Ref_5 et Ref_6 ont pour seule différence la clé de marquage. L'identification de circuits différents est donc effectuée dans les pires conditions possibles d'un point de vue de la méthode de vérification par canaux auxiliaires (Section 2.2).

2.4.2 Choix des paramètres expérimentaux

Il est impératif de choisir les paramètres n_1, n_2, k et m de la méthode de vérification avant l'expérimentation. Le premier paramètre à choisir est k car il fixe la valeur minimale de n_1 et a une influence sur la valeur de n_2. Dans cette expérience, k est arbitrairement fixé à 50. Ainsi, la condition 2.8 impose de prendre n_1 supérieur à 50. La valeur choisie pour n_1 est fixée à 400.

Pour construire les courbes de référence, 50 traces parmi les 400 mesures effectuées sur les circuits de référence seront donc sélectionnées aléatoirement puis moyennées comme décrit dans la Section 2.2.2. Ainsi, six courbes de référence notées respectivement A_{Ref_1}, A_{Ref_2}, A_{Ref_3}, A_{Ref_4}, A_{Ref_5} et A_{Ref_6} représentent les circuits Ref_1, Ref_2, Ref_3, Ref_4, Ref_5 et Ref_6.

Il reste maintenant à fixer les paramètres n_2 et m de la méthode. Pour cela, une approche basée sur les probabilités est utilisée. Il faut donc choisir une valeur pour le paramètre α. Dans la Section 2.2, la Figure 2.8 montre que la probabilité de sélectionner plusieurs fois une trace particulière dans l'ensemble des mesures

Tableau 2.2 – Récapitulatif des circuits utilisés.

Nom	Description du circuit de référence
Ref_1	– Compteur binaire sur 8 bits
Ref_2	– Compteur Gray sur 8 bits
Ref_3	– Compteur binaire sur 8 bits – Clé de marquage K_{w1} – Sbox de l'AES
Ref_4	– Compteur Gray sur 8 bits – Clé de marquage K_{w1} – Sbox de l'AES
Ref_5	– Compteur Gray sur 8 bits – Clé de marquage K_{w2} – Sbox de l'AES
Ref_6	– Compteur Gray sur 8 bits – Clé de marquage K_{w3} – Sbox de l'AES

de test tend vers 0.00468 pour une valeur de α de 10. Cette limite est tout à fait satisfaisante pour tester la méthode de vérification. Par ailleurs, plus la valeur de m est grande, plus il faudra faire d'acquisitions. Cependant, si on regarde de nouveau la Figure 2.8, on s'aperçoit que à partir de $m = 17$, la probabilité de sélectionner plusieurs fois une courbe reste dans un intervalle de 5% en dessous de la limite de 0.00468. Cet intervalle représente une variation de seulement $0,000234$, ce qui est négligeable. La valeur de m est donc fixée à 20, ce qui permet d'avoir suffisamment de coefficients à analyser.

Les valeurs de k, de α et de m imposent alors une valeur de $\alpha \times k \times m = 10000$ pour n_2. Résumons les paramètres choisis pour l'expérience :

- $n_1 = 400$;
- $n_2 = 10000$;
- $k = 50$;
- $m = 20$.

En appliquant la méthode de vérification décrite dans la Section 2.2.2, on ob-

tient, pour chacun des six circuits de référence, quatre ensembles de 20 coefficients de corrélation correspondant aux six circuits de test. Pour le circuit Ref_1, les six ensembles de coefficients sont notés : $\mathfrak{C}_{Ref_1,DUT_1,50,20}$, $\mathfrak{C}_{Ref_1,DUT_2,50,20}$, $\mathfrak{C}_{Ref_1,DUT_3,50,20}$, $\mathfrak{C}_{Ref_1,DUT_4,50,20}$, $\mathfrak{C}_{Ref_1,DUT_5,50,20}$ et $\mathfrak{C}_{Ref_1,DUT_6,50,20}$. Dans la suite de cette section, pour un circuit de référence Ref_x avec $x \in [1,6]$, ces ensembles seront regroupés sous la notation :

$$\{\mathfrak{C}_{Ref_x,DUT_y,50,20}, y \in [1,6]\}$$

2.4.3 Résultats expérimentaux

Pour représenter les résultats expérimentaux, les coefficients de corrélation sont affichés pour chacun des six circuits de référence. En revanche, ces coefficients ne sont pas séparés pour les circuits de test. Ainsi, pour un circuit de référence Ref_x avec $x \in [1,6]$, 120 coefficients sont affichés, 20 pour chacun des circuits de test. La Figure 2.14 présente les résultats de cette expérience pour les circuits de référence Ref_1, Ref_2 et Ref_3, et la Figure 2.15 présente les résultats pour les circuits de références Ref_4, Ref_5 et Ref_6. Le nom du circuit de référence est indiqué en haut de chaque graphique. Les coefficients sont regroupés pour chaque DUT dont le nom est indiqué en bas du groupe de coefficients qui lui correspond.

Les Figures 2.14 et 2.15 montrent qu'il n'est pas si évident de retrouver quel circuit de test contient quel circuit de référence seulement en regardant graphiquement les résultats, même s'il est possible de le déterminer intuitivement. En effet, seuls les résultats concernant le circuit de référence Ref_1 (en haut de la Figure 2.14) et le circuit de référence Ref_6 (en bas de la Figure 2.15) indiquent clairement que le circuit de test DUT_1 contient le circuit de référence Ref_1 et que le circuit de test DUT_6 contient le circuit de référence Ref_6. Cependant, il est possible d'extraire d'autres informations des ensembles $\{\mathfrak{C}_{Ref_x,DUT_y,50,20}, y \in [1,6]\}$, $x \in [1,6]$ en s'intéressant par exemple aux variations des coefficients de corrélation dans chacun de ces six ensembles de coefficients.

2.4.4 Choix des métriques d'analyse

Afin d'extraire des informations permettant de dire quel circuit de test contient quel circuit de référence, deux métriques peuvent être utilisées. La première consiste à extraire ce qui est directement visible graphiquement : le niveau de corrélation entre un circuit de référence et un circuit de test. Pour cela, considérons un circuit de référence Ref_x avec $x \in [1,6]$. Le niveau global de corrélation avec le DUT_y ($y \in [1,6]$) est mesuré grâce à la moyenne de l'ensemble $\mathfrak{C}_{Ref_x,DUT_y,k,m}$. Cette moyenne sera notée $\overline{\mathfrak{C}_{Ref_x,DUT_y,k,m}}$.

La deuxième façon d'extraire une information de l'ensemble $\mathfrak{C}_{Ref_x,DUT_y,k,m}$ est de s'intéresser à ces variations. En effet, si le circuit testé possède bien la *watermark* attendue, les coefficients de corrélation ne doivent pas significativement varier. Cette information peut être extraite grâce à la variance des coefficients de corrélation. Cette variance sera notée $V(\mathfrak{C}_{Ref_x,DUT_y,k,m})$.

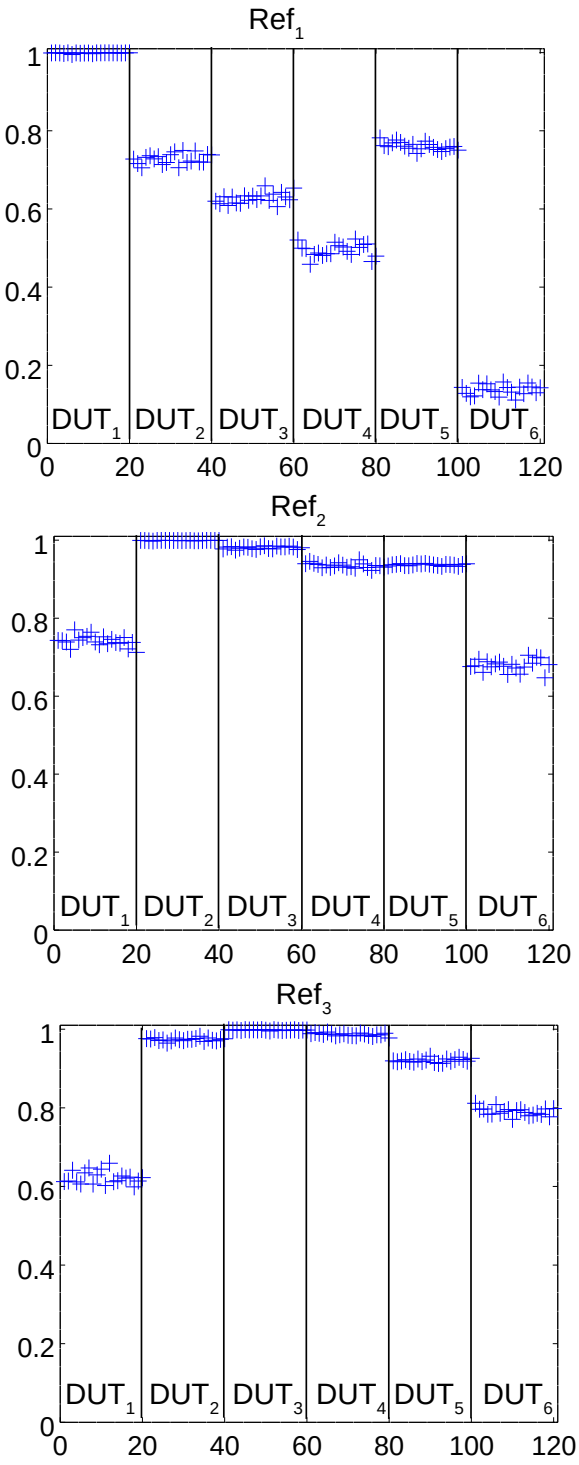

FIGURE 2.14 – Résultats de la méthode de vérification de *watermark* pour les circuits de référence Ref_1, Ref_2 et Ref_3.

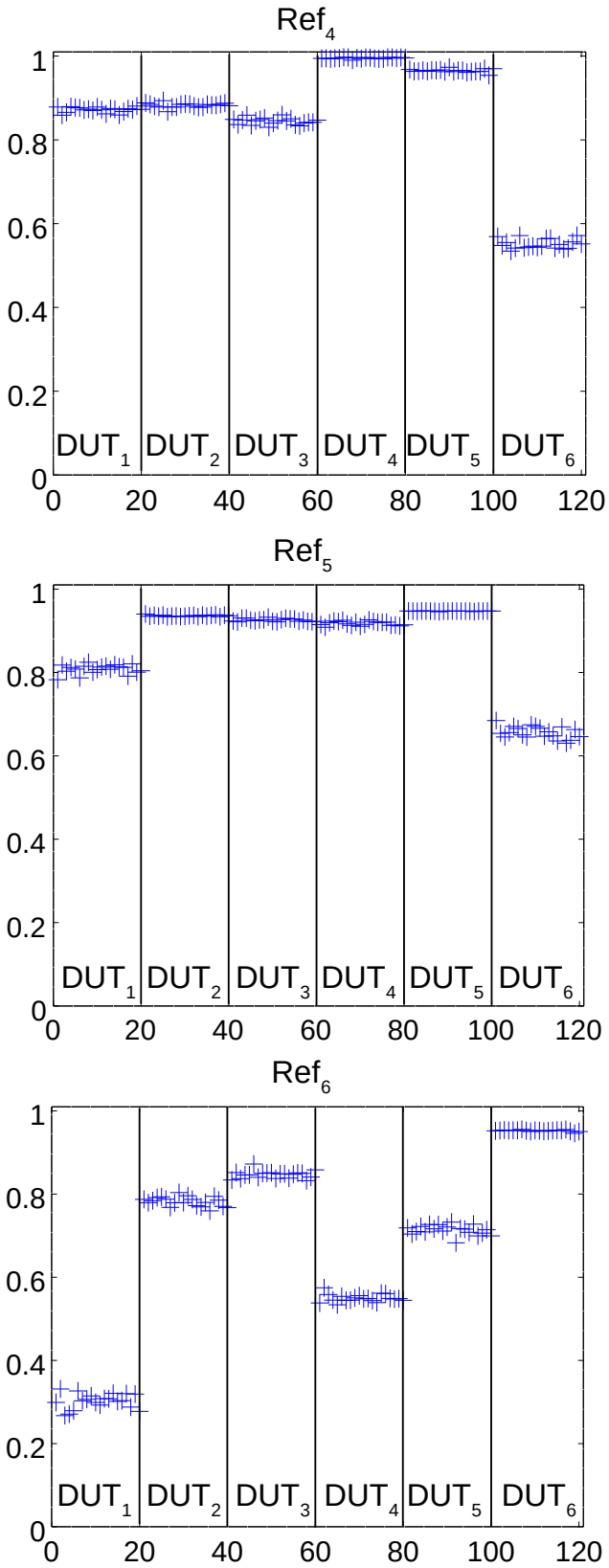

FIGURE 2.15 – Résultats de la méthode de vérification de *watermark* pour les circuits de référence Ref_4, Ref_5 et Ref_6.

2.4.4.1 Analyse de la moyenne des coefficients de corrélation

Afin d'évaluer l'efficacité de la moyenne des coefficients de corrélation, considérons l'ensemble des résultats pour un circuit de référence. La distance entre les deux meilleurs candidats indique l'efficacité de cette métrique. Cette distance représente donc un indice de confiance dans les résultats de l'expérience. Pour définir cette distance entre les moyennes des coefficients de corrélation, la fonction max_2, qui renvoie le deuxième plus grand élément d'un ensemble de nombres réels, est définie.

Considérons un circuit de référence Ref_x de notre expérience ($x \in [1, 6]$). L'indice de confiance de la moyenne des coefficients de corrélation est défini par :

$$\Delta_{mean} = 100 \times \left[1 - \frac{max_2(\{\overline{\mathfrak{C}_{Ref_x,DUT_y,k,m}}, y \in [1, 6]\})}{max(\{\overline{\mathfrak{C}_{Ref_x,DUT_y,k,m}}, y \in [1, 6]\})}\right]$$

Le Tableau 2.3 présente pour chacun des six circuits de référence les différentes moyennes obtenues ainsi que l'indice de confiance (Δ_{mean}).

Tableau 2.3 – Moyennes des coefficients de corrélation et indices de confiance.

	DUT_1	DUT_2	DUT_3	DUT_4	DUT_5	DUT_6	Δ_{mean}
Ref_1	**0.999**	0.728	0.628	0.494	0.760	0.137	23.92%
Ref_2	0.741	**0.999**	0.981	0.935	0.937	0.679	1.80%
Ref_3	0.622	0.974	**0.998**	0.986	0.920	0.790	1.20%
Ref_4	0.872	0.882	0.844	**0.995**	0.965	0.551	3.01%
Ref_5	0.808	0.936	0.926	0.918	**0.947**	0.656	1.16%
Ref_6	0.302	0.782	0.847	0.550	0.713	**0.952**	11.03%

Ce tableau montre bien que la moyenne des coefficients de corrélation ne fait pas clairement ressortir quel circuit de test contient quel circuit de référence même si le bon DUT est sélectionné à chaque fois. En effet, le meilleur indice est obtenu pour le circuit de référence Ref_1 et vaut 23.92% ; ce qui n'est pas significatif. Cela n'est pas surprenant car la moyenne des coefficients de corrélation n'extrait pas plus d'informations que l'inspection visuelle des Figures 2.14 et 2.15.

2.4.4.2 Analyse de la variance des coefficients de corrélation

De la même manière, un indice de confiance dans les résultats de l'expérience est obtenu en calculant la variance des coefficients de corrélation. Cette fois, une fonction notée min_2 renvoie le second plus petit élément d'un ensemble de nombres réels.

Considérons un circuit de référence Ref_x de notre expérience ($x \in [1, 6]$). L'indice de confiance de la variance des coefficients de corrélation est défini par :

$$\Delta_v = 100 \times \left[1 - \frac{min(\{v(\mathfrak{C}_{Ref_x,DUT_y,k,m}), y \in [1, 6]\})}{min_2(\{v(\mathfrak{C}_{Ref_x,DUT_y,k,m}), y \in [1, 6]\})}\right]$$

Le Tableau 2.4 présente pour chacun des six circuits de référence les différentes variances obtenues ainsi que l'indice de confiance (Δ_v).

Tableau 2.4 – Variances des coefficients de corrélation et indices de confiance.

	DUT_1	DUT_2	DUT_3	DUT_4	DUT_5	DUT_6	Δ_v
Ref_1	$\mathbf{1.11e^{-6}}$	$1.77e^{-4}$	$1.90e^{-4}$	$2.99e^{-4}$	$1.00e^{-4}$	$1.74e^{-4}$	98.89%
Ref_2	$1.95e^{-4}$	$\mathbf{2.39e^{-7}}$	$1.07e^{-5}$	$4.22e^{-5}$	$5.76e^{-6}$	$2.34e^{-4}$	95.85%
Ref_3	$2.63e^{-4}$	$1.54e^{-5}$	$\mathbf{8.52e^{-7}}$	$1.32e^{-5}$	$2.34e^{-5}$	$1.02e^{-4}$	93.55%
Ref_4	$3.98e^{-5}$	$2.91e^{-5}$	$6.31e^{-5}$	$\mathbf{2.75e^{-6}}$	$1.54e^{-5}$	$1.34e^{-4}$	82.14%
Ref_5	$1.24e^{-4}$	$2.92e^{-6}$	$1.92e^{-5}$	$2.56e^{-5}$	$\mathbf{2.49e^{-7}}$	$2.08e^{-4}$	91.47%
Ref_6	$3.31e^{-4}$	$1.30e^{-4}$	$8.30e^{-5}$	$8.72e^{-5}$	$1.39e^{-4}$	$\mathbf{4.09e^{-6}}$	95.07%

Cette fois, l'indice de confiance est très significatif pour chaque circuit de référence. En effet, le plus mauvais résultat donne une confiance de 82.14% dans la variance des coefficients de corrélation. Il est obtenu pour le circuit de référence Ref_4. La variance des coefficients de corrélation est donc plus efficace que la moyenne des coefficients de corrélation.

La faible efficacité de la moyenne des coefficients de corrélation peut s'expliquer par le fait que tous les circuits testés dans ce chapitre sont synchrones et qu'une forte corrélation avec l'horloge du système est donc présente. Chaque transition de chaque machine d'état se fait lors du front montant de l'horloge du système, ce qui induit une corrélation automatique entre les différents circuits. En revanche, la variance des coefficients fait ressortir les différences entre les transitions qui ont lieu entre les différents états des machines à états finies et permet donc de mieux les distinguer les unes des autres. On peut donc conclure que la moyenne des coefficients de corrélation fait ressortir le comportement synchrone des différents circuits alors que la variance met en valeur leurs différences. La variance des coefficients de corrélation peut donc être utilisée pour vérifier une *watermark* basée sur la cyclicité des machines à états finies, comme présenté dans la Section 2.1.1.2.

2.5 Conclusions et perspectives

Dans ce chapitre, une méthode de vérification de *watermark*, insérée dans les machines à états finies des systèmes numériques synchrones, a été présentée. Cette méthode utilise le canal auxiliaire de la consommation de puissance d'un circuit électronique afin d'extraire les informations indiquant la présence ou non d'une *watermark*. Le premier avantage de cette méthode est qu'elle est relativement indépendante de l'algorithme utilisé pour insérer la marque à l'intérieur du système. De plus, elle permet de distinguer des circuits très proches comme le montre l'expérience présentée dans la Section 2.4.

Un guide permettant de choisir les paramètres nécessaires à la mise en œuvre de la méthode de vérification est également présenté. Ce guide s'appuie sur la probabilité de sélectionner une mesure de consommation de puissance plus d'une fois lors du processus de calcul des coefficients de corrélation (Section 2.2.3). Enfin, une analyse détaillée des résultats de l'expérience montre que la variance des coefficients de corrélation contient plus d'information utile à la vérification de *watermarks* que le niveau moyen de corrélation.

Les travaux présentés dans ce chapitre laissent entrevoir un certain nombre de perspectives d'amélioration. En effet, il est indispensable de tester la méthode de vérification sur d'autres schémas d'insertion de *watermark*, dans des environnements plus bruités mais aussi sur d'autres cibles (FPGA ou ASIC). Par exemple, elle pourrait être testée pour vérifier des *watermarks* insérées grâce aux techniques par ajout. De plus, il faut également inclure les machines à états finies dans des environnements plus complets, avec d'autres périphériques fonctionnant en même temps et générant donc du bruit sur les canaux auxiliaires. Il est également possible de tester si la méthode de vérification fonctionne encore si le circuit de référence est différent des circuits de test (d'un point de vue du contenu implanté) : le circuit de référence peut-être constitué de l'IP marquée seule, alors que le circuit de test contiendrait un processeur ainsi que d'autres périphériques en plus de l'IP marquée par exemple.

La sensibilité de la méthode de vérification proposée doit aussi être évaluée face aux variations du processus de fabrication des circuits intégrés. De même, il serait intéressant de généraliser cette méthode en la testant sur des circuits complètement différents. Que se passe-t-il si le FPGA de référence appartient à une autre technologie que les FPGA de test ? Par exemple, il est possible d'utiliser un FPGA Xilinx comme référence et des FPGA Altera ou Microsemi pour les tests.

Chapitre 3

Conception et caractérisation d'une fonction physique non-clonable : la TERO-PUF

Dans ce chapitre, nous nous intéressons à un autre mécanisme participant à la protection contre la contrefaçon et le vol de circuits intégrés : une fonction physique non-clonable (PUF). La PUF étudiée dans ce chapitre est basée sur des oscillateurs en anneau dont les oscillations sont transitoires (TERO : *Transient Effect Ring Oscillator*). La TERO-PUF fait partie du schéma d'activation proposé dans le projet SALWARE comme le montre la Figure 3.1. Son principal objectif est l'identification intrinsèque d'un circuit avant qu'il puisse être activé. Pour cela, comme toutes les autres PUF, la TERO-PUF génère un identifiant unique pour chaque puce grâce à l'extraction d'entropie issue des variations qui apparaissent lors de la fabrication des circuits intégrés.

Avant de décrire comment la TERO-PUF a été conçue sur FPGA (Section 3.2), la prochaine section présente le concept de la cellule TERO. La Section 3.3 décrit le banc de caractérisation qui a été mis en place pour ces travaux et la Section 3.4 présente et analyse les résultats de la caractérisation de la TERO-PUF pour deux familles de FPGA. La Section 3.5 conclut ce chapitre et donne quelques perspectives d'améliorations.

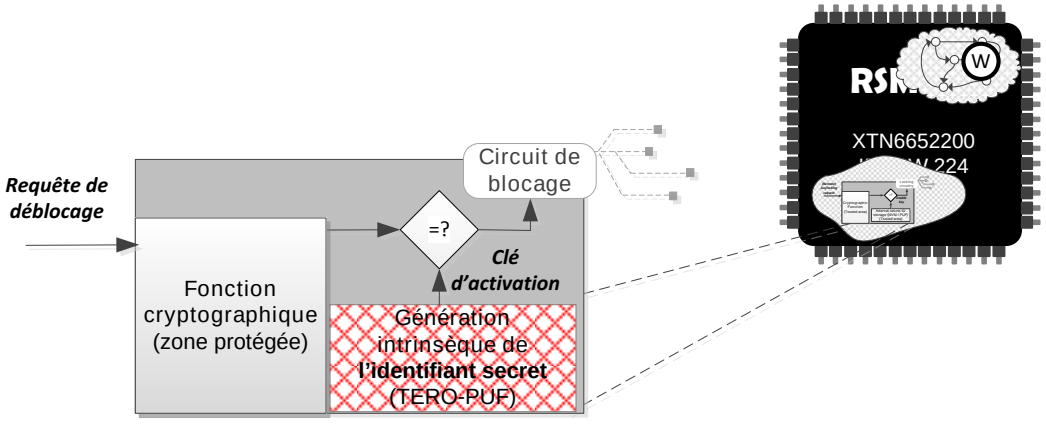

Figure 3.1 – Positionnement de la TERO-PUF (en rouge) dans le projet SALWARE.

3.1 La TERO-PUF

3.1.1 La cellule TERO

La cellule TERO est l'élément de base autour duquel nous allons construire la fonction physique non-clonable étudiée dans ce chapitre. Cette cellule est un oscillateur en anneau dont les oscillations sont temporaires. Elle a été présentée pour la première fois en 1990 [138] et a d'abord été utilisée comme générateur de nombres vraiment aléatoires (TRNG) [172]. Une étude récente [28] a montré que les caractéristiques de cette structure en font également un très bon candidat pour la conception d'une fonction physique non-clonable. La cellule TERO est constituée de deux branches connectées entre elles comme le montre la Figure 3.2.

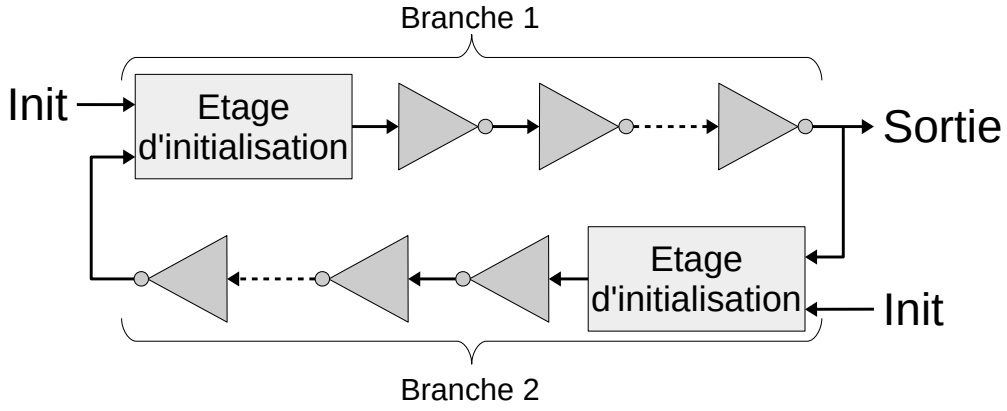

FIGURE 3.2 – Schéma d'une cellule TERO.

Chaque branche est composée d'un étage d'initialisation suivi d'une chaine de délais. Dans ce chapitre, l'initialisation sera effectuée par des portes *ET* suivies d'un nombre impair d'inverseurs.

La Figure 3.3 montre le comportement de la sortie d'une cellule TERO enregistrée grâce à un oscilloscope sur un FPGA Altera Cyclone II. Dans le cas de cette figure, la cellule commence à osciller lorsque le signal d'initialisation (*init*) passe à la valeur $'0'$. On peut distinguer deux états sur la Figure 3.3 :

1 un état oscillant situé après l'initialisation de la cellule ;
2 un état stable après que les oscillations se soient arrêtées. Cet état peut être un état logique haut ou bas en fonction des paramètres de variations.

Lorsque la cellule est initialisée, deux évènements électriques (fronts) commencent à se propager au sein de l'anneau. Ces deux évènements vont continuer à se propager dans l'anneau jusqu'à ce que l'un rattrape l'autre. Cette collision se traduit par la fin des oscillations de la cellule TERO.

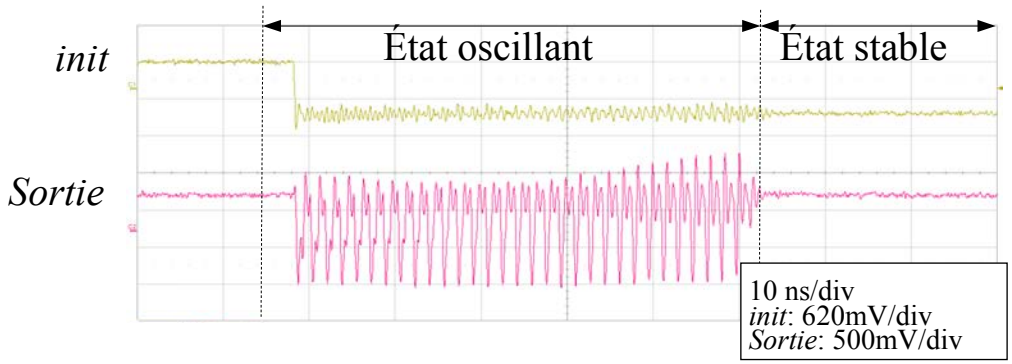

FIGURE 3.3 – Comportement de la sortie d'une cellule TERO.

Pour expliquer le comportement oscillatoire d'une cellule TERO, il faut s'intéresser aux délais de propagation des portes logiques composant cette cellule. Les premières observations montrent que la période d'oscillation des cellules TERO reste constante durant l'état oscillant. En revanche, la durée qui sépare les deux évènements décroit avec le temps jusqu'à atteindre 0, ce qui implique l'arrêt des oscillations et le début de l'état stable de la cellule. Ce phénomène a déjà été reporté par Winstanley *et al.* [177] et il est appelé *l'effet drafting*. Il peut être expliqué par la prise en compte des pentes dans le modèle de propagation d'un signal d'une porte inverseuse.

Avec les outils de simulation de circuits digitaux, le temps de propagation des portes logiques est généralement considéré constant car ces outils sont prévus pour la simulation de systèmes synchrones. Cette approche n'est pas satisfaisante pour la modélisation de signaux dont les oscillations sont très rapides. Dans la pratique, lorsque la sortie d'une porte logique change d'état très rapidement, le délai de propagation apparent peut être plus court qu'il ne l'est normalement, car la tension de sortie de la porte n'a pas complètement atteint l'état haut (ou bas) lorsque l'évènement suivant arrive à l'entrée de la porte. Dans le cas contraire, la tension de sortie converge asymptotiquement vers la valeur correspondant à l'état logique haut ou bas. Plus la pente du signal de sortie est raide et plus la durée entre deux évènements est courte, plus le temps de propagation de la porte est court. Dans le cas d'une cellule TERO, la pente de la sortie des portes logiques composant la cellule est directement liée aux capacités parasites présentes à chaque porte logique de la cellule. Dans [177], un modèle temporel est proposé pour expliquer cet effet dans des systèmes asynchrones. Ce modèle est basé sur la charge et la décharge exponentielle d'une capacité et peut être transposé pour des portes logiques.

Pour une porte logique *Non*, le modèle présenté dans [177] peut se traduire par l'Expression 3.1, où D_m représente le délai de propagation statique de la porte (quand $y \to \infty$). La partie exponentielle représente la charge et la décharge d'une capacité, dont l'amplitude est notée α et le temps caractéristique est noté τ. L'Expression 3.1 présente le délai de propagation ($D(y)$) d'une porte logique *Non*. Dans cette expression, y représente la durée séparant deux évènements à l'entrée de la

porte logique, comme le montre la Figure 3.4. Cette figure présente le chronogramme d'une porte inverseuse avec le temps de propagation d'un évènement à l'entrée de la porte $D(y)$ et la durée entre deux évènements y.

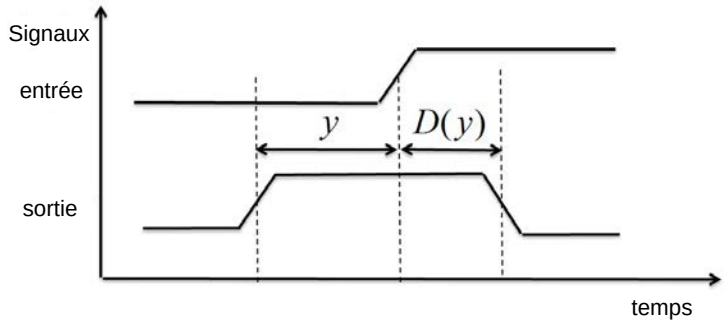

FIGURE 3.4 – Chronogramme du temps de propagation $D(y)$ d'une porte inverseuse.

$$D(y) = D_m - \alpha e^{-\frac{y}{\tau}} \tag{3.1}$$

On peut remarquer que dans ce modèle temporel, plus la durée séparant deux évènements est courte, plus le délai de propagation de la porte est court. Cela permet d'expliquer pourquoi un évènement rattrape l'autre dans le cas d'une cellule TERO. En effet, ce modèle indique que lorsque un évènement E_1 commence à rattraper l'autre évènement E_2 dans l'anneau, cet évènement aura tendance à se propager de plus en plus vite dans la cellule puisque le délai entre E_2 et E_1 sera de plus en plus court. En revanche, d'un point de vue de l'évènement E_2, la durée séparant E_1 de E_2 sera de plus en plus long, E_2 se propagera donc de moins en moins vite dans la cellule. Ce phénomène explique pourquoi les oscillations des cellules TERO sont transitoires.

3.1.2 La TERO-PUF complète

La TERO-PUF est composée de deux blocs de N cellules TERO, de deux sélecteurs permettant de choisir les deux cellules qui oscilleront en même temps (cette sélection correspond au challenge de la PUF), de deux compteurs synchrones de K bits ainsi que de deux multiplexeurs permettant de conduire le signal de sortie des cellules TERO choisies sur le signal d'horloge des compteurs. Une fois les nombres d'oscillations des deux cellules acquis, il faut en calculer la différence, puis extraire J bits utilisables pour construire la réponse de la PUF. La Figure 3.5 présente l'architecture de la TERO-PUF.

Comme la plupart des fonctions physiques non-clonables, la TERO-PUF mesure les différences entre deux cellules théoriquement identiques et génère une réponse en fonction du résultat de cette mesure. Un protocole de challenges/réponses est donc mis en place. Pour un challenge c, deux cellules TERO sont sélectionnées parmi $2 \times N$ cellules et la différence entre leurs nombres d'oscillations est calculée.

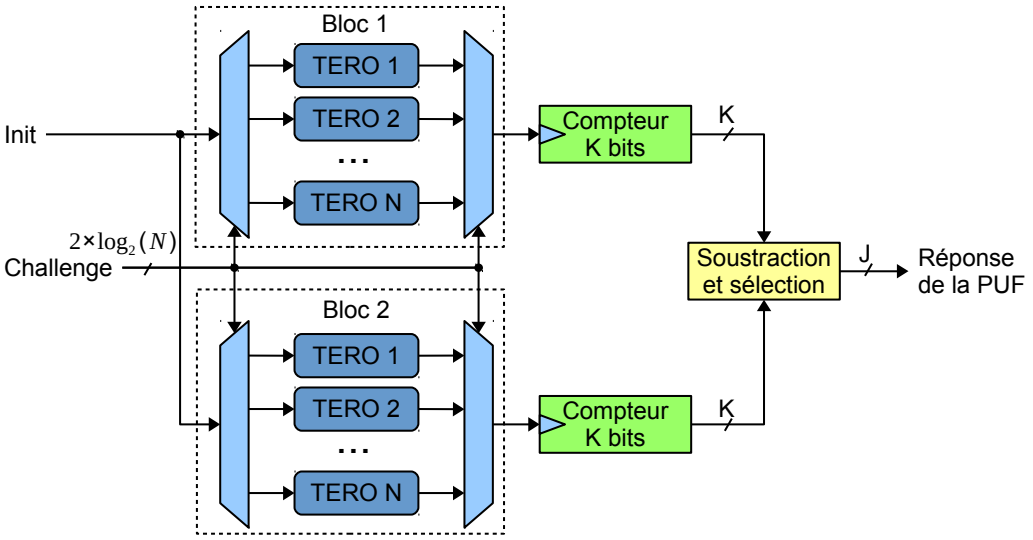

FIGURE 3.5 – Architecture de la TERO-PUF.

Les bits en sortie du soustracteur constituent la réponse r au challenge c. Ensuite, une sélection des bits utilisables de cette réponse est effectuée et un identifiant est généré grâce aux réponses de la PUF à plusieurs challenges. Le nombre de réponses à récupérer dépend à la fois de la taille de l'identifiant qui est généré et du nombre de bits sélectionnés pour chaque réponse de la PUF. Contrairement à des fonctions physiques non-clonables telles que les *arbiter*-PUF ou les RO-PUF, la TERO-PUF permet donc, théoriquement, de générer plusieurs bits de réponse par challenge.

La division en deux blocs des cellules TERO permet en particulier de partager moins d'information entre les différents challenges. En effet, une cellule d'un bloc n'est jamais comparée à une cellule du même bloc. De ce fait, il est plus difficile de trouver la réponse à un challenge particulier même après avoir obtenu les réponses à d'autres challenges. De plus, retrouver une telle information ne donnerait que le bit de signe de la différence entre les nombres d'oscillations des deux cellules TERO. Si plusieurs bits en sortie du soustracteur sont utilisés pour construire les réponses des circuits, cette information de signe ne permet pas de remonter jusqu'à ces réponses.

3.2 Conception d'une cellule TERO sur FPGA

Nous allons maintenant étudier et comparer les caractéristiques de la TERO-PUF pour deux familles de FPGA (Xilinx Spartan 6 et Aletra Cyclone V). Pour cela, le système complet doit être identique pour les deux familles. Malheureusement, l'implantation des cellules TERO elles-mêmes est dépendante du circuit utilisé. Elle ne peut pas être portée d'un FPGA à un autre facilement. Ainsi, seules les cellules TERO seront différentes entre les systèmes implantés sur les deux familles de FGPA. Toute la partie de communication et d'analyse des réponses de la fonction physique non-clonable est identique.

3.2.1 Généralité sur la conception d'une cellule TERO sur FPGA

La structure des FPGA Xilinx et Altera que nous utilisons est principalement basée sur des blocs élémentaires appelés *look up table* (LUT) et sur des registres. Une LUT est un tableau correspondant à la table de vérité de la fonction logique qu'elle doit réaliser. Ainsi, pour une LUT à n entrées et une sortie, il est possible d'effectuer n'importe quelle opération logique nécessitant au plus n entrées. Pour implanter une cellule TERO, seules les LUT sont utilisées. De plus, deux contraintes doivent être appliquées pour concevoir une cellule TERO :

- une LUT est utilisée pour implanter une et une seule porte logique ;
- les délais de routage entre deux portes doivent être identiques dans les deux branches de la cellule.

Pour satisfaire la première contrainte sur les FPGA Xilinx Spartan 6, la cellule TERO sera conçue par l'intégration du composant *LUT6*, directement disponible dans la librairie Xilinx : *UNISIM.vcomponents*. Chaque LUT est ensuite placée dans le fichier de contraintes (ucf). Cette étape de placement des LUT est très importante car elle permet de travailler sur la seconde contrainte. En effet, il n'est pas possible de forcer le routage du circuit. Mais en choisissant astucieusement le placement de chaque LUT de chaque branche de la cellule TERO, il est possible de répondre à la seconde contrainte.

Pour les FPGA Altera Cyclone V, ce sont les librairies *altera_mf* et *altera* qui permettent d'implanter la cellule TERO. Les composants utilisés pour créer une LUT sont *lut_input* et *lut_output*. Ces composants représentent les entrées et sorties de la LUT. Le composant *LCELL* est également utilisé afin de fixer le nombre de LUT à traverser dans chaque branche de la cellule. Chacun de ces éléments est ensuite placé dans le fichier de contraintes (qsf) afin de travailler sur la seconde contrainte. La partie la plus difficile de cette contrainte est de trouver un placement permettant également d'avoir les chemins reliant les branches de même longueur (en terme de délais).

La conception de cette cellule TERO peut se résumer aux étapes suivantes :

1. Écriture du code VHDL de la boucle TERO ;
2. Placement contraint de chacun des éléments de la cellule ;
3. Vérification des délais de routage de la cellule.

Enfin, pour implanter une TERO-PUF de N cellules, il faut dupliquer N fois la cellule qui a été conçue à travers tout le FPGA et donc placer à nouveau chaque LUT de chaque branche, et ce pour chaque cellule TERO. De plus, il faut s'assurer qu'aucun élément logique ne soit ajouté à l'intérieur des blocs utilisés par les cellules TERO, lors de la synthèse du circuit par les outils de conception fournis par les vendeurs de FPGA. Pour les FPGA Xilinx, l'utilisation des *hard macros* permet de dupliquer à volonté un bloc qui n'est pas modifié lors de la synthèse du circuit. De plus, aucune logique n'est ajoutée aux parties occupées par les *hard macros*. Pour les FPGA Altera, il n'existe pas d'équivalent direct des *hard macros* mais il est

possible de fixer une région du FPGA et d'interdire l'ajout d'éléments dans cette région lors de la synthèse du circuit. Ce type de région s'appelle *LogicLock*.

3.2.2 Choix du nombre d'inverseurs par branche de la cellule TERO

Il faut fixer la longueur des branches de la cellule TERO que nous allons implanter avant de commencer la caractérisation de la TERO-PUF. Cette première étape de l'étude de la TERO-PUF est réalisée sur les FPGA Xilinx Spartan 6 car l'utilisation des *hard macros* apporte une très grande flexibilité pour l'instanciation des cellules TERO.

Afin de choisir le nombre d'inverseurs des cellules TERO implantées dans le système de la TERO-PUF, le nombre moyen d'oscillations ainsi que leur écart-type sont analysés pour quatre tailles de cellule : un, trois, cinq et sept inverseurs par branche. Cette analyse est effectué sur quatre FPGA Xilinx Spartan 6 pour chaque taille de cellule TERO. Sur chacun de ces quatre FPGA, 256 cellules identiques sont implantées et le nombre d'oscillations de chaque cellule est mesuré 1000 fois. La Figure 3.6 (a) montre la moyenne du nombre d'oscillations des cellules TERO en fonction du nombre d'inverseurs par branche. La Figure 3.6 (b) présente l'écart-type moyen du nombre d'oscillations en fonction du nombre d'inverseurs par branche.

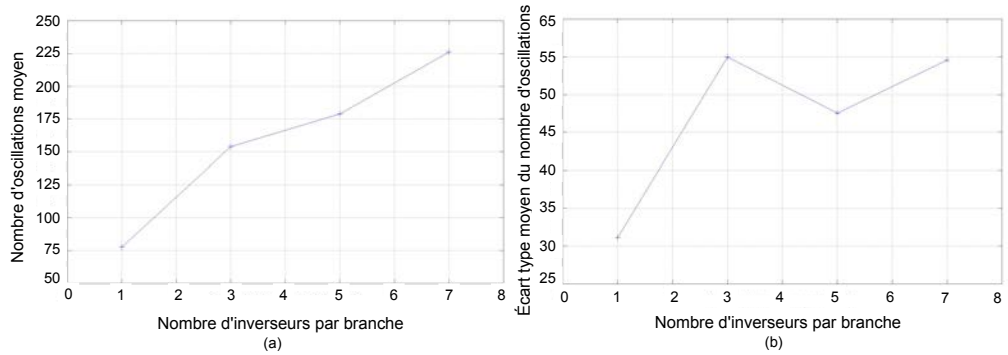

FIGURE 3.6 – Moyenne et écart-type du nombre d'oscillations des cellules TERO en fonction du nombre d'inverseurs par branche.

Sur la Figure 3.6.a, on remarque que la moyenne du nombre d'oscillations des cellules TERO évolue quasi linéairement en fonction du nombre d'inverseurs par branche. En revanche, il n'est pas possible de tirer de conclusion concernant l'écart-type à partir de la Figure 3.6.b. Augmenter le nombre d'inverseurs ne semble pas modifier significativement l'écart-type moyen du nombre d'oscillations des cellules TERO. L'utilisation de cellules n'ayant qu'un seul inverseur par branche a révélé une plus grande proportion de cellules qui n'oscillent pas du tout ou qui oscillent indéfiniment. Il est donc préférable d'utiliser des cellules comportant plus d'inverseurs par branche. De plus, des travaux préliminaires ont été effectués sur des ASIC avec des cellules TERO comportant sept inverseurs par branche [39]. L'utilisation

de cette taille permettra donc de comparer les résultats obtenus sur FPGA avec ceux obtenus sur ASIC. Ainsi, la caractérisation de la TERO-PUF est effectuée pour des cellules comportant sept inverseurs par branche.

La Figure 3.7 montre les schémas finaux de la cellule TERO implantée dans chaque FPGA. Pour les FPGA Xilinx Spartan 6 (Figure 3.7 a), cette cellule comporte une porte *ET* et sept inverseurs par branche. Elle est entièrement équilibrée et les délais de chaque branche sont égaux deux à deux, d'après l'estimation fournie par le logiciel *Xilinx FPGA Editor*. Pour les FPGA Altera Cyclone V (Figure 3.7 b), la cellule comporte une porte *ET*, un inverseur et huit *LCELL* par branche. Une différence de 0.035ns est toujours présente entre les délais de routage des deux branches, selon l'estimation fournie par l'outil *Quartus Timing Quest Analyzer*. Une description complète des étapes de conception de la cellule TERO est présentée pour chaque famille de FPGA dans l'Annexe A.

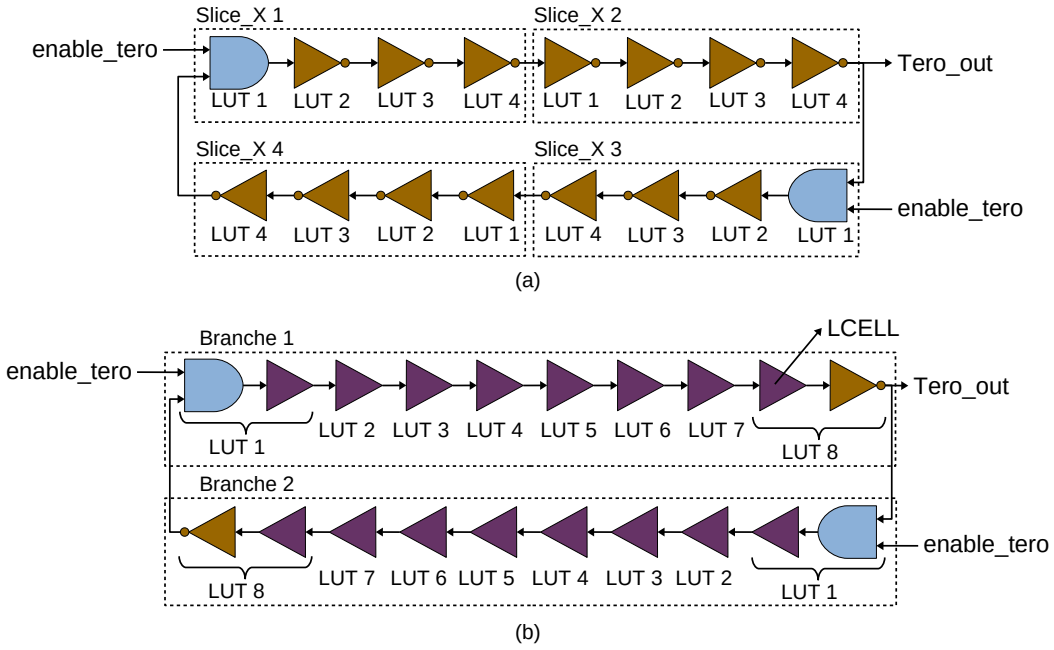

FIGURE 3.7 – Implantations finales de la cellule TERO pour les FPGA Xilinx Spartan 6 (a) et Altera Cyclone V (b).

3.2.3 Architecture de la TERO-PUF pour les expérimentations FPGA

Pour la caractérisation de la TERO-PUF présentée dans ce chapitre, le nombre de cellules TERO est fixé à $N = 128$ par bloc et les compteurs implantés dans les FPGA ont une taille de 16 bits. Les FPGA renvoient directement à l'ordinateur le nombre d'oscillations des cellules TERO choisies par le challenge. La différence entre les nombres d'oscillations, la sélection des bits utilisables ainsi que l'analyse statistique des réponses de la PUF sont effectuées par l'ordinateur. La Figure 3.8

présente le schéma utilisé pour la caractérisation de la TERO-PUF, avec les parties implantées sur FPGA et celles réalisées par ordinateur.

L'architecture présentée dans la Figure 3.8 est identique quelque soit la famille de FPGA utilisée (Xilinx Spartan 6 ou Altera Cyclone V). En effet, seules les cellules TERO sont différentes pour les différentes familles de circuits et il est possible de changer uniquement ces cellules pour implanter la TERO-PUF dans différentes familles de FPGA. Conserver la même architecture pour caractériser la PUF sur différentes cibles permet une comparaison équitable des résultats obtenus sur les différents circuits.

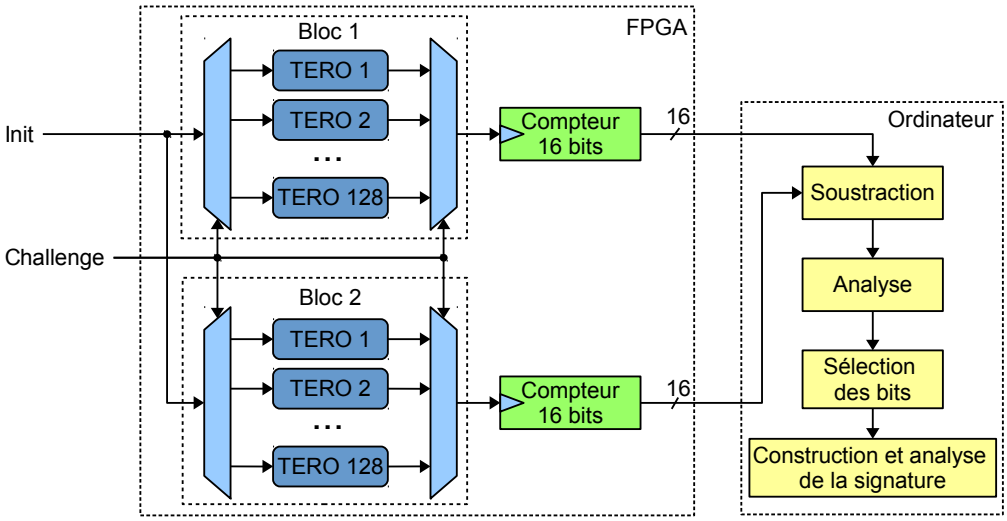

FIGURE 3.8 – Architecture matérielle/logicielle de la TERO-PUF pour la caractérisation expérimentale sur FPGA.

3.3 Méthodologie de caractérisation de la TERO-PUF

Pour caractériser efficacement la TERO-PUF sur FPGA, un banc d'acquisition et d'analyse a été spécialement mis en place. La prochaine sous-section présente les métriques utilisées. La Sous-section 3.3.2 décrit les paramètres pris en compte lors de la caractérisation de la TERO-PUF. Enfin, la Sous-section 3.3.3 présente le banc de caractérisation mis en place pour l'analyse de la TERO-PUF.

3.3.1 Analyse des fonctions physiques non-clonables

Pour analyser les propriétés statistiques de la TERO-PUF, trois métriques sont utilisées : la stabilité, l'unicité et l'imprévisibilité. Chacune de ces trois métriques permet de savoir si la fonction physique non-clonable est utilisable pour générer un identifiant unique pour chaque circuit, ou s'il est nécessaire d'y ajouter un système

de correction ou d'accumulation (moyennage) pour rendre la fonction utilisable. Ces trois métriques sont présentées dans la suite de cette section.

3.3.1.1 Stabilité

La stabilité représente la capacité de la fonction physique non-clonable à fournir la même réponse lorsqu'elle est soumise au même challenge. Cette métrique est évaluée par le calcul de la distance de *Hamming* (notée HD) entre la réponse de référence d'un circuit et les réponses de ce même circuit, soumis au challenge qui a généré la réponse de référence. La stabilité des réponses d'une fonction physique non-clonable dépend fortement des paramètres environnementaux auxquels est soumis le circuit testé. Pour un circuit i, elle sera notée : $IC_i(T, V)$; où T représente la température environnementale et V la tension d'alimentation du FPGA.

Considérons un circuit i auquel le même challenge est envoyé R fois. La réponse de référence ($r_{i,ref}$) est générée en calculant la moyenne des R réponses de taille n renvoyées par le circuit i lorsqu'il fonctionne dans des conditions normales : température ambiante (25° C) et tension d'alimentation nominale. Ainsi, la stabilité peut être définie par l'expression 3.2.

$$IC_i(T, V) = \frac{1}{R} \sum_{j=1}^{R} \frac{HD(r_{i,j}, r_{i,ref})}{n} \times 100\% \tag{3.2}$$

Cette expression indique le pourcentage moyen de bits différents entre les différentes réponses fournies par un circuit lorsqu'il est soumis au même challenge. Cette métrique mesure donc l'instabilité des réponses. Comme une bonne stabilité correspond à une faible instabilité, la valeur idéale de cette métrique est 0%.

Dans le but de donner un aspect pratique à cette métrique très importante, un seuil de stabilité est fixé à 10%. Ce seuil correspond généralement à une limite raisonnable du nombre d'erreurs qu'il est possible de corriger grâce à l'utilisation de blocs spécialisés. De plus, il permet de donner des intervalles de confiance pour lesquels la stabilité des réponses de notre fonction physique non-clonable est en-dessous de ce seuil. C'est une approche pratique qui permet de garantir la qualité statistique des réponses de la PUF.

3.3.1.2 Unicité

L'unicité représente la capacité de la fonction physique non-clonable à différencier chaque circuit. Cette métrique est évaluée grâce à un calcul de distance de *Hamming* entre les réponses de plusieurs circuits soumis au même challenge. Considérons M circuits contenant la même fonction physique non-clonable. Pour l'un des circuits k, la distance de *Hamming* entre la réponse de référence $\overline{r}_k$ (moyenne des réponses du circuit k) et toutes les réponses fournies par tous les autres circuits est calculée. Ce processus est répété jusqu'à ce que les M réponses de référence aient été comparées aux $(M-1) \times R$ réponses fournies par les autres circuits. Enfin, la

moyenne des distances calculées est effectuée et elle est transformée en pourcentage. Dans l'expression 3.3, n représente la taille des réponses de la PUF.

$$EC = \frac{2}{M(M-1)N} \sum_{i=1}^{M} \sum_{k=i+1}^{M} \sum_{j=1}^{N} \frac{HD(r_{i,j}, \bar{r_k})}{n} \times 100\% \tag{3.3}$$

Cette expression indique le nombre moyen de différences entre les réponses de référence et les réponses de plusieurs circuits soumis au même challenge. Sa valeur idéale est 50%.

3.3.1.3 Imprévisibilité

Contrairement aux deux premières métriques, il n'existe pas de mesure standard permettant d'évaluer l'imprévisibilité des réponses d'une fonction physique non-clonable. La plupart du temps, c'est une mesure du biais des réponses de la fonction qui est calculée. Malheureusement, cette métrique ne permet pas d'évaluer s'il existe des corrélations entre les différents bits d'une réponse et ne constitue en aucun cas une réelle mesure de l'imprévisibilité de la réponse d'une fonction physique non-clonable. Supposons que les réponses à un challenge soient récupérées sur M circuits. Si toutes les réponses comportent autant de zéro que de un, une simple mesure du biais ne soulèvera aucune anomalie même si ces réponses possèdent un motif qui se répète. Par exemple, si l'identifiant généré à partir des réponses de la PUF est composé d'une alternance parfaite de zéro et de un (10101010...), il ne comporte aucun biais mais n'est absolument pas imprévisible. Il est donc nécessaire de trouver une autre manière d'évaluer l'imprévisibilité des fonctions physiques non-clonables.

Dans le but d'aller plus loin dans l'évaluation de l'imprévisibilité des réponses des fonctions physiques non-clonables, nous proposons l'utilisation de six tests de la suite de tests statistiques NIST SP 800-22 [145]. Ces six tests peuvent être lancés avec un nombre réduit de données. Cependant, ils sont adaptés aux générateurs de nombres aléatoires et leur utilisation avec peu de données réduit la confiance qu'il est possible de leur accorder. Ainsi, si les réponses d'une fonction physique non-clonable passe avec succès ces six tests, cela ne signifie pas qu'elles sont aléatoires (imprévisibles). En revanche, un échec à ces tests permet de réfuter l'utilisation de la configuration qui a servi à construire les identifiants, prouvant qu'elles possèdent un défaut statistique trop important pour une utilisation comme identifiant du circuit, ou comme clé cryptographique. Afin de mieux comprendre l'utilité de ces tests dans le cadre des fonctions physiques non-clonables, en voici une description :

T1. Le premier test est intitulé «test de fréquence mono-bit». Il a pour but d'estimer le biais d'une suite binaire,

T2. Le deuxième test effectue la même opération mais en considérant le biais appliqué à des blocs de taille m (paramètre qu'il est possible de fixer avant de lancer le test),

T3. Le troisième test mesure le distance à zéro de la chaine de bits. Pour cela, un entier est fixé à 0 au début du test et le parcours de la chaine de bits provoque une addition (ou une soustraction) pour chaque 1 (0) rencontré. Si

la somme cumulée dépasse un certain seuil (fixé par le test), cela se traduit par un échec. Ce test appelé *Cumulative Sums*,

T4. Le quatrième test évalue la longueur des séquences de bits identiques au sein de la chaine. Il correspond au *run test* de la suite de tests NIST,

T5. Le cinquième test a pour objectif de tester l'irrégularité de la série de '1' la plus grande dans les blocs successifs (de taille B) de la chaine de bits. Il correspond au test intitulé *longest run of ones in a block*,

T6. Le sixième et dernier test correspond à celui appelé *approximate entropy*. Il a pour but d'estimer l'entropie de la chaine de bits testée.

Grâce à ces six tests, il est possible de tirer une analyse plus fine de la qualité statistique de la réponse d'une fonction physique non-clonable, même s'ils ne constituent pas une réelle preuve de leur imprévisibilité.

À l'aide de ces trois métriques, les identifiants construits avec une sélection particulière des bits de réponse de la TERO-PUF seront donc analysées. Un bon résultat sur la stabilité, l'unicité ou l'imprévisibilité dans les condition de fonctionnement normale n'est pas suffisant pour valider la configuration. En revanche, un défaut sur l'une de ces trois métriques implique le rejet de la configuration concernée. Afin de valider une configuration permettant de générer un identifiant unique, il faut également évaluer la robustesse de l'identifiant vis-à-vis des variations de l'environnement (tension d'alimentation et température) dans lequel le circuit fonctionne.

3.3.2 Paramètres de la caractérisation

L'étude et la caractérisation d'une fonction physique non-clonable ne se limite pas à l'analyse pure et simple de sa stabilité ou de son unicité. En effet, il est impératif de tester le comportement de la fonction vis-à-vis des variations de température et de tension.

3.3.2.1 Variations de température

Pour la caractérisation de la TERO-PUF vis-à-vis des variations de température, huit points ont été sélectionnés autour d'une référence fixée à 25° C. Cette température de référence correspond à une température de fonctionnement normal. Le pas de variation qui a été choisi est de 10° C. Ainsi, les réponses de la TERO-PUF sont analysées pour les températures suivantes : −15° C, −5° C, 5° C, 15° C, 25° C, 35° C, 45° C, 55° C et 65° C.

Pendant la caractérisation en température, la tension d'alimentation des FPGA est fixée à la tension nominale du circuit : 1.20 V pour les FPGA Xilinx Spartan 6 et 1.10 V pour les FPGA Altera Cyclone V. Cette tension d'alimentation est assurée par la connexion USB reliant les cartes à l'ordinateur ainsi que par un circuit de régulation de tension présent directement sur les cartes contenant les FPGA. Les variations de température sont assurées par une chambre climatique, ce qui apporte

un environnement contrôlé dans lequel la température est précisément mesurée. De plus, un temps de stabilisation est fixé à 30 minutes pour chaque température : une fois que la chambre climatique a atteint le point de température souhaité, le système d'acquisition attend une demi-heure avant de démarrer les mesures.

3.3.2.2 Variations de tension

En ce qui concerne la caractérisation en tension de la TERO-PUF, le même principe est utilisé : huit points de tension sont sélectionnés autour de la tension nominale des FPGA. Toutefois, les points sont ici choisis de sorte à couvrir l'intégralité des spécifications des FPGA et deux points sont pris en dehors de ces spécifications. Cette contrainte dans le choix des points de tension impose le pas de variation utilisé pour la caractérisation. Dans le cas des FPGA Xilinx Spartan 6, la tension nominale est de 1.2 V et l'intervalle de tension recommandé est entre 1.14 V et 1.26 V. Les tensions testées pour cette technologie sont donc fixées à : 1.10 V, 1.14 V, 1.16 V, 1.18 V, 1.20 V, 1.22 V, 1.24 V, 1.26 V et 1.30 V.

Dans le cas des FPGA Altera Cyclone V, la tension nominale est de 1.10 V et les spécifications du circuit recommandent de ne pas dépasser un intervalle de variation de plus de 5% autour de cette tension. Cela implique un intervalle de tensions allant de 1.07 V à 1.13 V. Les tensions choisies pour la caractérisation de la TERO-PUF sur cette cible sont sélectionnées de la même manière que pour les FPGA Xilinx Spartan 6. Les neuf points de tension sont : 1.05 V, 1.07 V, 1.08 V, 1.09 V, 1.10 V, 1.11 V, 1.12 V, 1.13 V et 1.15 V.

Pendant la caractérisation en tension, la température est fixée à la celle de référence utilisée lors de la caractérisation : 25° C. Lors de cette analyse, les cartes FPGA sont placées dans la chambre climatique dont la température est réglée sur 25° C, ce qui assure la stabilité de l'environnement de mesure. Pour les variations de la tension, le circuit d'alimentation par USB est déconnecté et une alimentation stabilisée programmable est utilisée pour alimenter directement les FPGA. Ceux-ci sont d'abord configurés à leur tension nominale, puis le balayage en tension commence par la tension la plus basse. Contrairement à la caractérisation en température, il n'est pas nécessaire d'attendre un certain temps avant de lancer l'acquisition des réponses de la TERO-PUF.

3.3.2.3 Codage et durée d'acquisition des réponses

Codage des réponses de la TERO-PUF

Une fois que les réponses de la TERO-PUF ont été récupérées, il est nécessaire de les extraire des fichiers afin de construire les réponses, avant de les analyser. De plus, les réponses de la TERO-PUF correspondent à la différence entre les nombres d'oscillations de deux cellules TERO, ce qui rend potentiellement possible l'extraction de plusieurs bits par challenge. Pour cela, une analyse fine des identifiants générés avec plusieurs configurations d'extraction de bits doit être effectuée, la seule sélection des bits de poids fort n'étant pas suffisante.

En effet, l'utilisation de compteurs rend l'extraction des bits de réponse plus compliquée qu'en apparence, car le compteur peut très bien être mal dimensionné. L'utilisation du code binaire naturel pour coder les différences entre les nombres d'oscillations des cellules TERO provoque une propagation des informations contenues dans le bit de signe. Il est donc nécessaire de découper la valeur de la différence du nombre d'oscillations en sortie du soustracteur en trois parties :

1 le bit signe ;
2 les bits inutiles (qui ne représentent pas d'information) ;
3 les bits représentant la valeur de la différence.

Malheureusement, le code binaire naturel signé ne permet pas ce découpage. En effet, les bits inutiles prennent automatiquement la même valeur et les mêmes propriétés statistiques que le bit de signe, ce qui rend ce codage très peu efficace pour sélectionner les bits utilisables de la réponse de la PUF. De plus, les variations entre deux valeurs successives sur ce codage peuvent être très importantes, notamment autour des puissances de deux. Cela se traduit par une distance de *Hamming* entre les valeurs successives qui varie de façon non linéaire sur l'ensemble de la plage de codage, comme cela est illustré sur la Figure 3.9. Ainsi, l'analyse de la réelle stabilité est très compliquée pour un couple de cellules TERO dont la différence moyenne entre leurs nombres d'oscillations se trouve autour d'une puissance de deux.

Ces deux problèmes majeurs sont résolus par l'utilisation du code Gray. En effet, le codage particulier du code Gray implique que deux valeurs successives ne diffèrent que d'un seul bit. Dans ce cas, la distance de *Hamming* entre deux valeurs successives est constante (égale à 1) sur l'ensemble de la plage de codage, comme cela est illustré dans la Figure 3.9. Cela aura pour conséquence d'augmenter la stabilité de la différence entre les nombres d'oscillations d'un couple de cellules TERO, quelque soit la valeur moyenne de cette différence. De plus, le code Gray, de part sa construction sépare la valeur d'un nombre de son signe, et tous les bits inutiles sont mis à 0. Ainsi, l'unicité de ces bits chute très fortement, ce qui permet de ne pas les sélectionner pour générer les identifiants des circuits. Enfin, le code Gray est une représentation particulière des nombres et ne constitue donc pas une amélioration artificielle de la qualité des réponses de la TERO-PUF.

Durée d'acquisition

Le dernier paramètre qu'il faut prendre en compte pour l'acquisition et l'extraction des réponses de la TERO-PUF est la durée durant laquelle les oscillations des cellules sont comptées. Cela peut être facilement réglé en jouant sur la durée pendant laquelle le signal *Init* est à l'état actif. Les premières observations montrent que certaines cellules oscillent indéfiniment (ou très longtemps) et cela implique donc un temps de réponse de la TERO-PUF très long. Dans l'idéal, il faut écarter chaque cellule TERO oscillant indéfiniment pour la génération des réponses. Cependant, l'évolution de la stabilité et de l'unicité des réponses de la TERO-PUF en fonction de cette durée d'acquisition (Figure 3.10) montre que l'unicité des réponses

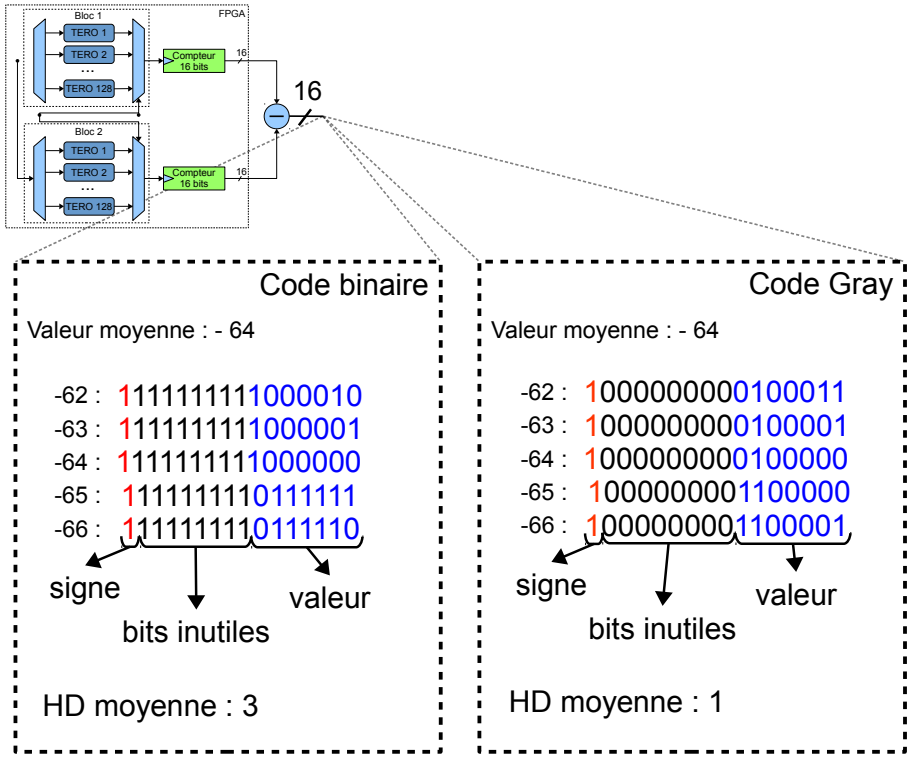

FIGURE 3.9 – Différence entre le code Gray et le code binaire naturelle.

arrive très vite autour de 50%, même si certaines cellules n'ont pas atteint leur état stable avant la fin de l'acquisition. De plus, la Figure 3.10 montre que plus la durée d'acquisition est courte, plus la stabilité des réponses est bonne (proche de 0%).

Toutefois, il est impératif de séparer les choix possibles de durée d'acquisition en trois catégories (visibles sur la Figure 3.10) :

Cas n°1 : la durée d'acquisition est très courte, ce qui implique qu'aucune cellule TERO n'a atteint son état stable avant la fin de l'acquisition. Dans ce cas, la stabilité des réponses est très bonne mais leur unicité est mauvaise. De plus, stopper l'acquisition du nombre d'oscillations des cellules TERO rapidement implique que la TERO-PUF ait alors un comportement identique à la RO-PUF : seule la fréquence d'oscillation des cellules est exploitable ;

Cas n°2 : la durée d'acquisition est suffisamment longue pour que les cellules TERO soient presque toutes à l'état stable à la fin de l'acquisition. Ce cas est très intéressant puisque il n'est pas possible de prévoir quelles sont les cellules à l'état stable et celles qui sont encore dans l'état oscillant à la fin de l'acquisition. L'étude de l'unicité et de la stabilité des différents bits du nombre d'oscillations en sortie des compteurs montre que certains bits peuvent être exploités pour générer les réponses.

Toutefois, l'idéal serait de ne pas considérer les cellules n'ayant pas atteint l'état stable lors de la génération des réponses, pour ne pas y inclure des

informations attaquables. En effet, si une cellule n'a pas atteint son état stable avant la fin de l'acquisition, le nombre d'oscillations enregistré permet de retrouver la fréquence d'oscillation de la cellule. Ainsi, il est possible de connaître à l'avance certains bits des réponses et donc de réduire la sécurité de la TERO-PUF ;

Cas n°3 : la durée d'acquisition est très longue et toutes les cellules atteignent leur état stable avant la fin de l'acquisition. C'est l'idéal si le nombre moyen d'oscillations des cellules TERO n'est pas trop élevé. Dans ce cas, l'unicité est très bonne et la stabilité est moindre. En revanche si le nombre d'oscillations est très grand, la stabilité des réponses n'est pas satisfaisante car elles sont plus sensibles aux bruits, comme le *jitter* qui s'accumule avec le temps. Dans ce cas, il est possible que seul le bit de signe de la différence des nombres d'oscillations des cellules TERO soit exploitable, comme pour la RO.

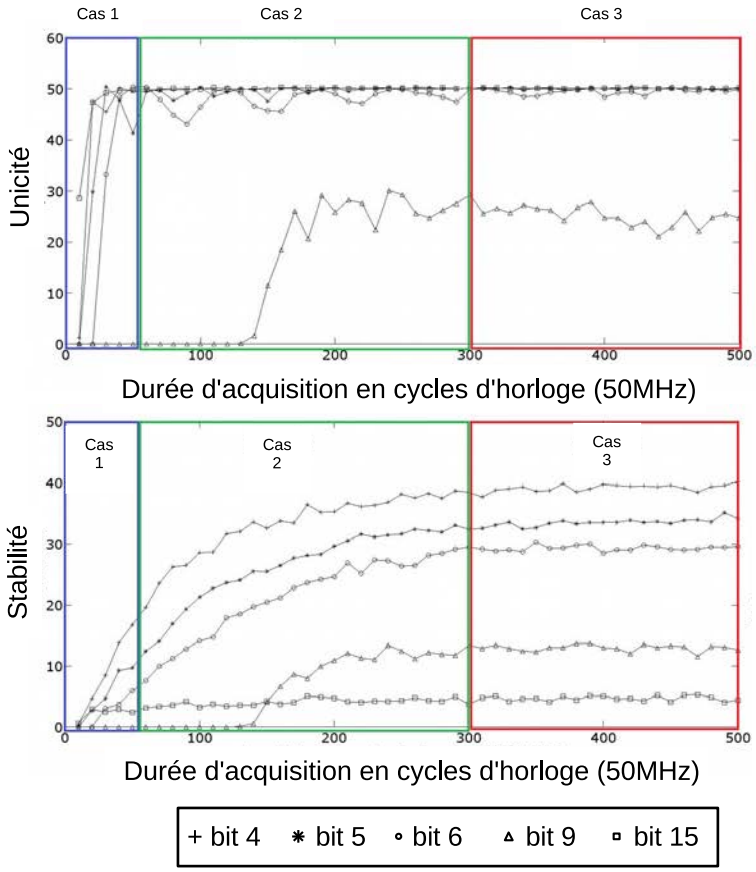

FIGURE 3.10 – Impact de la durée d'acquisition sur la stabilité et sur l'unicité du nombre d'oscillations des cellules TERO, enregistré sur des FPGA Xilinx Spartan 6.

Dans la suite de ce chapitre, les identifiants générés grâce aux réponses de la TERO-PUF sont analysés en code Gray. De plus, la durée d'acquisition utilisée

correspond à 30 cycles d'horloge à 50 MHz, ce qui situe l'analyse à la limite du cas n°1 et du cas n°2. Une deuxième durée d'acquisition très longue, correspondant au cas n°3 (2 000 cycles d'horloge), est également utilisée pour la caractérisation de la TERO-PUF sur les deux familles de FPGA pour montrer que seul le bit de signe est alors exploitable.

3.3.3 Développement d'un banc de caractérisation de PUF

Avant de présenter les résultats de la caractérisation de la TERO-PUF pour les deux cibles FPGA (Section 3.4), cette section décrit succinctement le banc de caractérisation qui a été mis en place. Ce banc est composé des éléments suivants :

- une chambre climatique ;
- une alimentation stabilisée ;
- une partie matérielle comportant les cartes FPGA ;
- une partie logicielle.

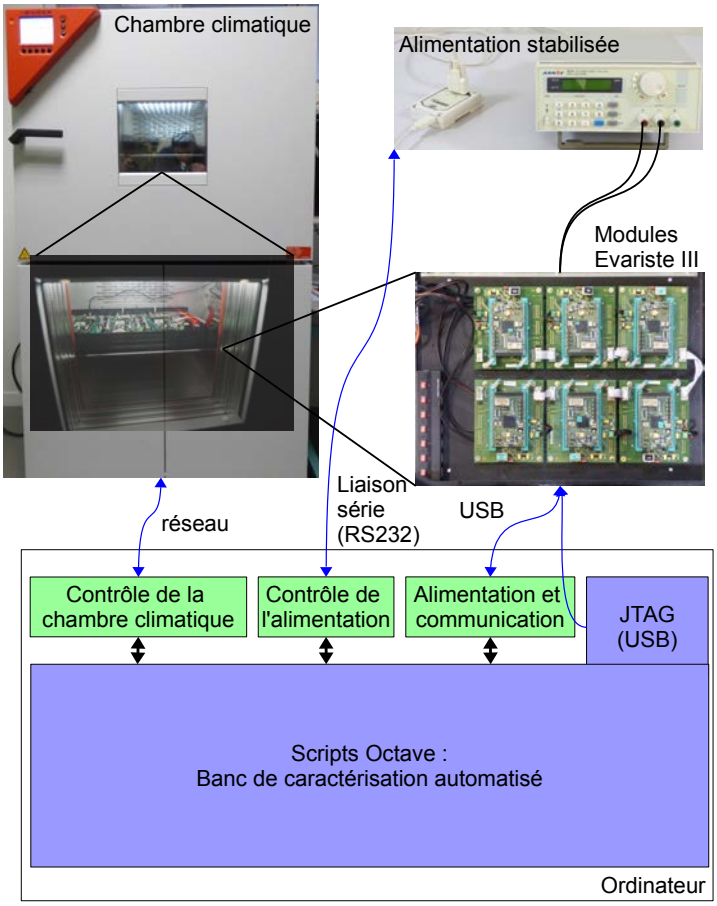

FIGURE 3.11 – Le banc de caractérisation complet de PUF.

La Figure 3.11 montre le banc de caractérisation complet avec les différentes interactions existantes entre chaque partie. La chambre climatique permet d'assurer le contrôle et la stabilité de la température environnementale à laquelle les FPGA fonctionnent. L'alimentation stabilisée remplace l'alimentation USB lors de la caractérisation de la TERO-PUF en tension. En revanche, elle n'est pas utilisée lors de la caractérisation en température. La partie matérielle du banc comporte le système Evariste III [19] ainsi qu'un boîtier permettant de brancher plusieurs cartes simultanément. La partie logicielle assure l'automatisation des mesures, la configuration des FPGA ainsi que le contrôle de la chambre climatique et de l'alimentation stabilisée.

3.3.3.1 Partie logicielle du banc de caractérisation

La partie logicielle de banc de caractérisation de PUF a été conçue grâce à des scripts, développés sur Octave/Matlab. Elle s'articule autour de quatre scripts permettant de configurer les FPGA, d'effectuer un balayage en température ou d'effectuer un balayage en tension. Le dernier script regroupe les différentes fonctionnalités de ce banc. Les différents paramètres sont modifiables grâce à un fichier de configuration.

Pour le contrôle des différentes parties du banc, deux outils ont été développés. Le premier permet de lire la température de la chambre climatique et d'écrire une nouvelle consigne de température. Le second contrôle l'alimentation stabilisée et permet de lire ou de changer la tension. Enfin, la configuration des FPGA passe également par une interaction entre le script Octave et les outils de configuration fournis par Xilinx et Altera.

3.3.3.2 Le système Evariste III [1]

La Figure 3.12 présente les différents composants de cette partie matérielle du banc de caractérisation de PUF. Elle a été conçue à partir de l'évolution du système Evariste II (présenté dans la Section 2.3.1, dans le Chapitre 2). Comme la précédente version, Evariste III est composé d'une carte mère et d'une carte fille [19].

De plus, les cartes mères possèdent une chaine JTAG dans Evariste III, ce qui permet de connecter plusieurs cartes, de les configurer en série, et d'effectuer la caractérisation de PUF de plusieurs cartes simultanément. Pour faciliter la connection de plusieurs cartes, un boîtier dédié contenant six cartes mère est utilisé.

Le projet à adapter pour inclure une application dans Evariste III est exactement le même que celui pour Evariste II (Section 2.3.1). Plus de détails sont disponibles sur un site internet qui lui est dédié [2].

Dans le but d'intégrer le système complet de la TERO-PUF dans Evariste III, un protocole de communication a été mis en place. Ce protocole permet de choisir le

1. Le système Evariste III n'est pas une contribution stricte de cette thèse. Il a été conçu en collaboration avec plusieurs membres de l'équipe SESAM du laboratoire Hubert Curien, dont l'auteur de cette thèse.

2. `https://labh-curien.univ-st-etienne.fr/wiki-evariste/index.php/Main\_Page`

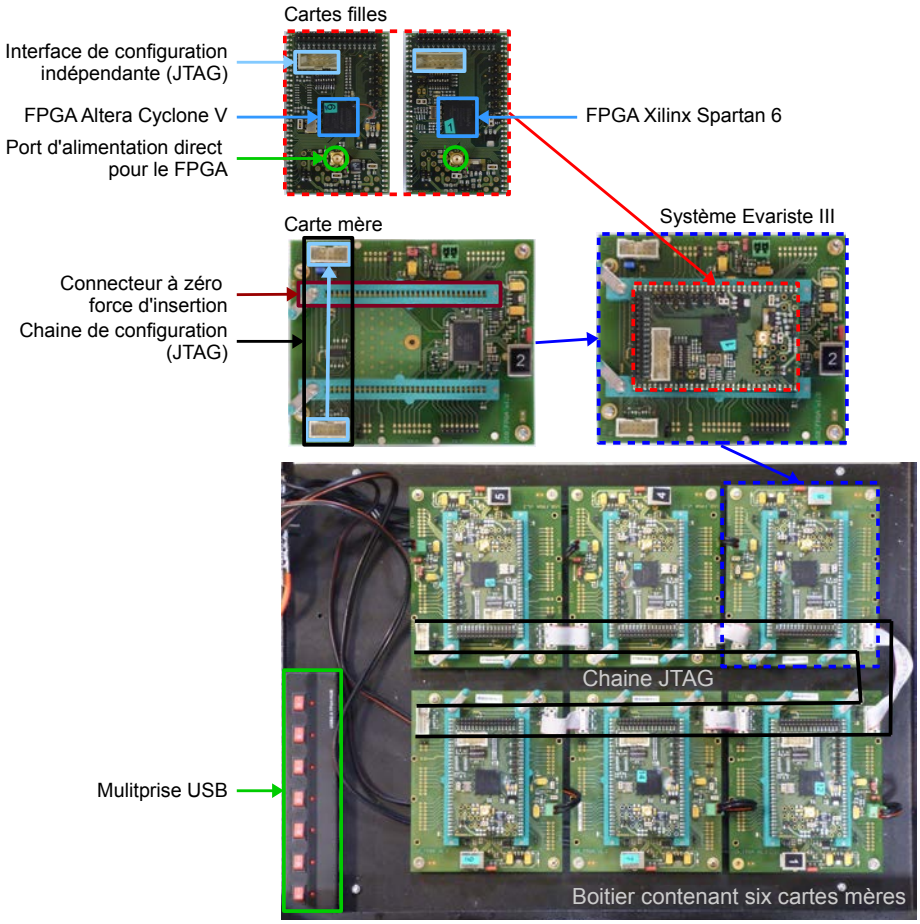

FIGURE 3.12 – Le système Evariste III [19].

challenge (couple de cellules TERO qui oscillent en même temps), de régler la durée d'acquisition et de mesurer autant de fois que nécessaire les nombres d'oscillations des deux cellules sélectionnées. Cette partie matérielle du banc est spécifique à la TERO-PUF. L'implantation matérielle diffère bien entendu pour une RO-PUF ou une autre PUF.

3.4 Résultats de la caractérisation de la TERO-PUF

Cette section présente les résultats de la caractérisation de la TERO-PUF pour les cibles FPGA Altera Cyclone V et Xilinx Spartan 6. Les paramètres expérimentaux utilisés ont été décrits dans les sections précédentes et sont exactement les mêmes pour les deux familles, sauf pour la tension d'alimentation (voir Section 3.3.2.2). Toutefois, les différents points de tension ont été sélectionnés de la même manière et assurent donc que les mêmes analyses sont effectuées.

Pour chaque résultat présenté dans cette section, le nombre d'acquisitions par challenge est fixé à 936 [1]. De plus, la suite de challenges utilisée est toujours la même et correspond à la suivante : notons les cellules TERO de chaque bloc de 0 à 127. Chaque cellule $i \in [0, 127]$ du premier bloc oscille en même temps que la cellule portant le même indice dans le deuxième bloc. Concrètement, cela signifie que la cellule d'indice 0 du premier bloc oscille avec la cellule d'indice 0 du second bloc, que celles d'indice 1 oscillent ensemble, *etc.*

Il serait envisageable de choisir d'autres challenges, notamment pour éviter des corrélations liées physiquement à l'implantation matérielle de la TERO-PUF et de comparer les résultats avec un tirage aléatoire des challenges. Mais le temps nécessaire à la caractérisation complète sur 30 cartes (Xilinx Spartan 6) ne nous a pas permis d'explorer toutes ces possibilités.

3.4.1 Xilinx Spartan 6

Pour les FPGA Xilinx Spartan 6, la caractérisation de la TERO-PUF est effectuée sur 30 FPGA, en température et en tension, avec deux durées d'acquisitions. La première correspond à 30 cycles d'horloge (50MHz), ce qui signifie que quelques cellules seront stoppées avant d'avoir atteint l'état stable. La seconde durée d'acquisition est beaucoup plus longue et correspond à 2 000 cycles d'horloge. Dans ce cas, toutes les cellules atteignent leur état stable avant la fin de l'acquisition.

3.4.1.1 Caractérisation de la TERO-PUF pour une durée d'acquisition de 30 cycles d'horloge

La première étape de la caractérisation consiste à sélectionner les bits qui seront utilisés pour construire les identifiants, dont la stabilité vis-à-vis des variations de température et de tension sera analysée. Pour cela, un premier jeu de données est récupéré à température et tension nominales (25° C et 1.2 V). Le Tableau 3.1 présente les résultats de cette évaluation rapide des réponses de la TERO-PUF sur FPGA Xilinx Spartan 6.

1. Ce nombre d'acquisitions a été fixé pour répondre aux besoins des scripts d'analyses de PUF qui étaient déjà en place. Ces scripts ont été développés par un autre membre de l'équipe.

Tableau 3.1 – Stabilité et unicité moyenne des différents bits en sortie du compteur donnant le nombre d'oscillations des cellules TERO pour une durée d'acquisition de 30 cycles d'horloge sur 30 FPGA Xilinx Spartan 6.

Bit	Unicité (%)	Stabilité (%)
0	47.65	28.41
1	42.37	23.02
2	41.92	15.86
3	42.33	10.09
4	40.04	5.04
5	45.56	2.16
6	27.03	1.40
7	0.04	0.04
8	0	0
...	...	...
15	48.46	2.63

Comme on peut le voir dans le Tableau 3.1, le bit de signe (bit 15) et au moins deux autres bits (les bits 4 et 5) ont des caractéristiques suffisamment bonnes pour la construction d'identifiants sur 128 bits. En revanche, au-delà du bit 5, on remarque que l'unicité chute brusquement jusqu'à tomber à 0 à partir du bit 8. Ce comportement indique que les bits 6 à 14 ne sont peu ou pas utilisés dans la valeur du nombre d'oscillations des cellules TERO. Ils ne peuvent donc pas servir pour générer des identifiants. Les identifiants sur 128 bits analysés seront donc générés en utilisant 1, 2 ou 3 bits par challenge (couple de TERO). Les configurations retenues dans un premier temps sont :

1 bit 4 ;
2 bit 5 ;
3 bit 15 ;
4 bit 5 et 15 ;
5 bits 4, 5 et 15.

Seul le bit 15 (correspondant au bit de signe de la différence entre les nombres d'oscillations des cellules TERO) est utilisé dans toutes les réponses. L'analyse de la robustesse en température et en tension est donc effectuée pour cinq configurations différentes avec 1, 2 ou 3 bits par challenge. La Figure 3.13 présente les résultats de stabilité de l'analyse en température et la Figure 3.14 présente les résultats de la caractérisation en tension.

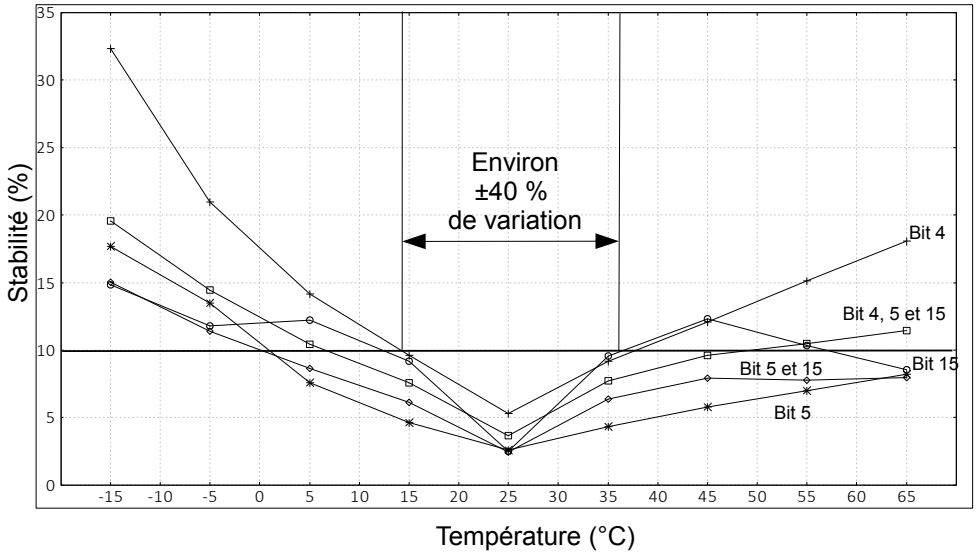

FIGURE 3.13 – Étude de la stabilité moyenne des identifiants de 128 bits générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de température, pour une durée d'acquisition de 30 cycles d'horloge à 50 MHz sur 30 FPGA Xilinx Spartan 6.

Comme on peut le voir sur les Figures 3.13, les identifiants dont la stabilité est la moins bonne sont obtenues pour ceux générés avec le bit 4. Pour tous les identifiants, la stabilité reste en dessous de 10% pour des températures allant de 15° C à 35° C, ce qui représente une variation de 40% autour de la température nominale de 25° C. On peut donc dire que la TERO-PUF montre une assez faible sensibilité vis-à-vis des variations de température dans les gammes «normales» de fonctionnement du circuit.

La Figure 3.14 montre que les identifiants de 128 bits générés avec le bit 15 présentent une forte sensibilité vis-à-vis des variations de tension autour de la tension nominale. Elle est en dessous des 10% uniquement sur une plage de 1.18 V à 1.22 V, ce qui représente une variation de seulement 1.5% autour de la tension nominale de 1.2 V. La TERO-PUF est donc plus sensible aux variations de la tension qu'aux variations de la température. Cela conduit à recommander au concepteur d'apporter un soin très important dans l'alimentation des circuits et sa régulation.

Maintenant que nous avons analysé le comportement des réponses de la TERO-PUF vis-à-vis des variations environnementales, il faut s'intéresser à l'imprévisibilité des réponses. Cette dernière étape est effectuée pour les identifiants de 128 bits générés à partir des réponses obtenues à température et tension nominale (25° C et 1.2 V). Les six tests de la suite de tests NIST, présentés dans la Section 3.3.1.3, sont utilisés. Le Tableau 3.2 présente les résultats de ces tests, ainsi que les indices de stabilité et d'unicité. Dans le Tableau 3.2, une case contient «V» si le test passe et la case est vide s'il ne passe pas. Si la case contient «na», cela indique que le

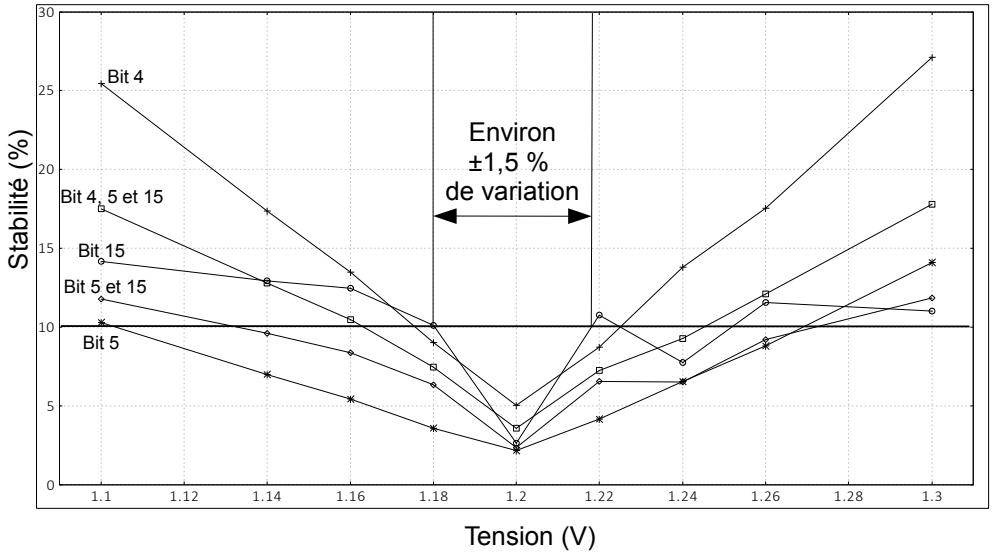

FIGURE 3.14 – Étude de la stabilité moyenne des identifiants de 128 bits générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de tension, pour une durée d'acquisition de 30 cycles d'horloge à 50 MHz sur 30 FPGA Xilinx Spartan 6.

test n'a pas été effectué. En particulier, il n'est pas utile de faire le test T2 avec les configurations utilisant un seul bit des réponses de la TERO-PUF pour générer les identifiants.

Les tests NIST sont appliqués à 10 séquences de 384 bits. Ces séquences sont obtenues en concaténant 3 identifiants de 128 bits construits à partir des réponses de la TERO-PUF. La taille des blocs utilisée dans les tests de fréquence et d'entropie correspondent au nombre de bits utilisés pour construire les réponses. Ainsi, pour celles construites à partir d'un seul bit de la différence entre les nombres d'oscillations des cellules TERO, la taille du bloc utilisée est 1. Pour les configurations utilisant deux bits, ce paramètre est fixé à 2 et pour celle utilisant trois bits, il est fixé à 3.

Tableau 3.2 – Résultats statistiques des identifiants de 128 bits générés avec les configurations utilisant de 1 à 3 bits sur FPGA Xilinx Spartan 6.

Configuration	Stabilité %	Unicité %	Tests statistiques					
			T1	T2	T3	T4	T5	T6
Bit 4	5.31	41.72		na				
Bit 5	2.60	46.49		na				
Bit 15	2.50	48.48		na			V	
Bits 5 et 15	2.46	47.83		V		V	V	V
Bits 4, 5 et 15	3.67	46.60		V		V	V	

On peut remarquer qu'aucune des configurations utilisées ne réussit l'intégralité des six tests, ce qui signifie que la qualité statistique des réponses n'est pas suffisante pour générer un identifiant totalement imprévisible. La meilleure configuration correspond à celle qui utilise les bits 5 et 15, ce qui signifie qu'il est possible d'extraire deux bits de la différence entre les nombres d'oscillations des cellules TERO avec une durée d'acquisition courte. Toutefois, l'analyse de l'imprévisibilité des identifiants indique que cette durée de 30 cycles d'horloge à 50 MHz semble être trop courte pour être utilisée.

3.4.1.2 Caractérisation de la TERO-PUF pour une durée d'acquisition de 2 000 cycles d'horloge

Afin d'être plus complet sur la caractérisation de la TERO-PUF sur FPGA Xilinx Spartan 6, il est nécessaire d'analyser les identifiants générés avec une autre durée d'acquisition, pour laquelle toutes les cellules TERO atteignent leur état stable avant la fin de l'acquisition. Pour cela, la durée d'acquisition est fixée à 2 000 cycles d'horloge et la même étude que celle effectuée pour 30 cycles d'horloge est réalisée. Dans un premier temps, l'évaluation rapide de la qualité statistique (stabilité et unicité) des bits de la différence des nombres d'oscillations des cellules TERO en sortie du soustracteur est effectuée. Le Tableau 3.3 présente le résultat de cette évaluation.

Comme on peut le voir sur le Tableau 3.3, seul le bit de signe (bit 15) possède une unicité et une stabilité suffisante pour servir de réponse de la TERO-PUF. Toutefois, afin de comprendre l'impact de la durée d'acquisition sur la qualité statistique des réponses de la TERO-PUF, la robustesse des réponses est analysée pour les mêmes configurations que précédemment. La Figure 3.15 montre le résultat de la caractérisation en température et la Figure 3.16 présente ceux de la caractérisation en tension.

Pour la température, les identifiants sur 128 bits générés avec le bit 15 de la différence entre les nombres d'oscillations des cellules TERO ont une stabilité en dessous de 10% entre 3° C et 34° C. Cela représente une variation de 36% au dessus de 25° C et de 88% en dessous pour la température, ce qui confirme une sensibilité assez faible vis-à-vis des variations de température.

Pour la tension, ces identifiants sont en dessous de 10% de stabilité entre 1.18 V et 1.21 V, ce qui représente une variation de 1% au dessus de 1.2 V et une variation de 1.6% en dessous. Cela confirme la forte sensibilité des réponses de la TERO-PUF vis-à-vis des variations de tension. Toutes les autres configurations montrent une stabilité supérieure à 20% quelle que soit la température ou la tension. Elles ne sont donc pas utilisables pour générer des identifiants de 128 bits, si la durée d'acquisition est de 2 000 cycles d'horloge (50 MHz).

Tableau 3.3 – Stabilité et unicité moyenne des différents bits de la différence du nombre d'oscillations des cellules TERO pour une durée d'acquisition de 2 000 cycles d'horloge sur 30 FPGA Xilinx Spartan 6.

Bit	Unicité (%)	Stabilité (%)
0	50.00	48.66
1	49.99	48.28
2	49.99	47.29
3	50.00	45.99
4	50.04	43.85
5	49.95	41.08
6	50.01	36.46
7	48.90	28.74
8	49.02	17.88
9	26.52	14.10
10	3.46	3.32
11	0.39	0.39
12	2.13	0.39
13	0.00	0.00
14	0.00	0.00
15	48.44	7.05

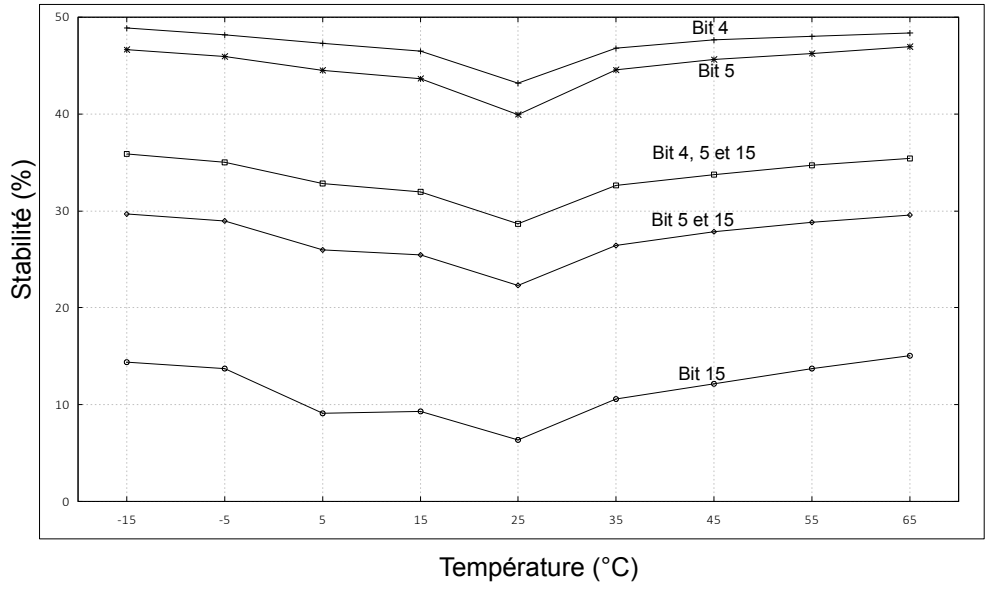

FIGURE 3.15 – Étude de la stabilité moyenne des identifiants de 128 bits générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de température, pour une durée d'acquisition de 2 000 cycles d'horloge à 50 MHz sur 30 FPGA Xilinx Spartan 6.

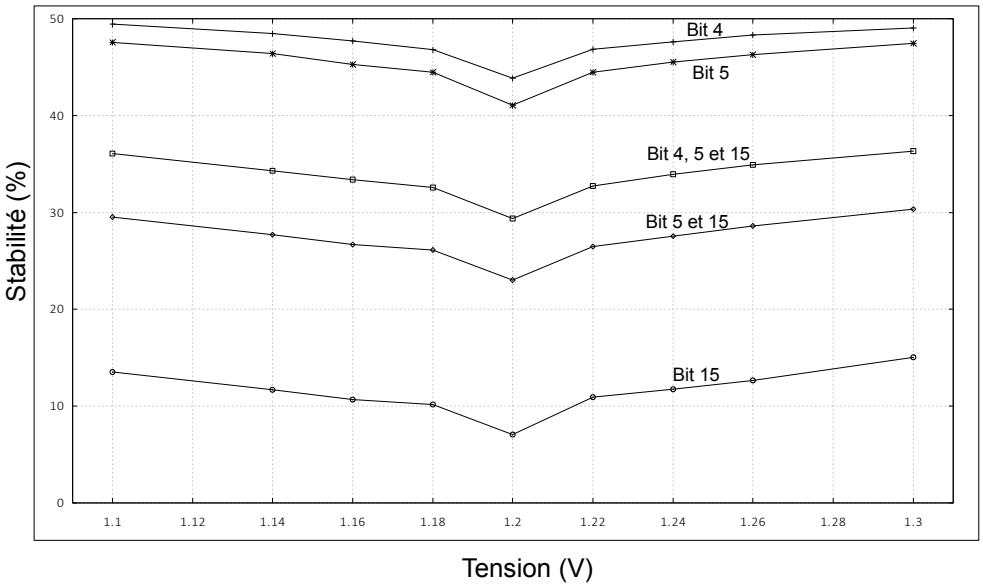

FIGURE 3.16 – Étude de la stabilité moyenne des identifiants de 128 bits générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de tension, pour une durée d'acquisition de 2 000 cycles d'horloge à 50 MHz sur 30 FPGA Xilinx Spartan 6.

Enfin, il faut également s'intéresser à l'imprévisibilité des réponses afin de compléter la caractérisation de la TERO-PUF pour cette durée d'acquisition. Pour cela, la même procédure est appliquée que pour la caractérisation à 30 cycles d'horloge : les identifiants sur 128 bits des 30 circuits sont regroupés et concaténés, ce qui crée une chaine binaire de 3 840 bits. Les tests de la suite NIST sont ensuite effectués sur dix chaines de 384 bits. Pour les configurations utilisant seulement un bit de la différence entre les nombres d'oscillations des cellules TERO, la taille du bloc utilisée pour les tests T2 et T6 est fixée à 1. Pour les identifiants utilisant deux bits, cette taille est fixée à 2 et pour pour ceux utilisant trois bits, elle est fixée à 3.

Le Tableau 3.4 présente les résultats des six tests statistiques sur les identifiants générés avec ces configurations ainsi que leur stabilité et leur unicité pour des conditions de fonctionnement normales (25° C et 1.2 V).

Dans le Tableau 3.4, on remarque que toutes les configurations passent avec succès les six tests. En revanche, la stabilité n'est satisfaisante que pour la configuration utilisant uniquement le bit 15.

La caractérisation de la TERO-PUF avec ces deux durées d'acquisition (30 et 2 000 cycles d'horloge à 50 MHz) a montré qu'il est possible d'extraire plusieurs bits de la différence entre le nombre d'oscillations des cellules TERO. En revanche, si la durée d'acquisition est trop courte, les réponses ne présentent pas une qualité statistique suffisante pour être utilisées. De même, si la durée est très longue, seul le bit de signe est utilisable mais la qualité statistique des réponses est suffisante.

Tableau 3.4 – Résultats des tests statistiques des identifiants de 128 bits générés avec les configurations utilisant de 1 à 3 bits sur FPGA Xilinx Spartan 6.

Configuration	Stabilité %	Unicité %	Tests statistiques					
			T1	T2	T3	T4	T5	T6
Bit 4	43.22	49.96	V	na	V	V	V	V
Bit 5	39.96	50.03	V	na	V	V	V	V
Bit 15	6.38	48.85	V	na	V	V	V	V
Bits 4 et 15	24.05	49.09	V	V	V	V	V	V
Bits 5 et 15	22.33	49.09	V	V	V	V	V	V
Bits 4, 5 et 15	28.64	49.46	V	V	V	V	V	V

On peut alors se poser la question de savoir s'il est possible de trouver une durée d'acquisition intermédiaire, qui permettrait d'extraire plusieurs bits par challenge, et dont au moins une configuration d'extraction permettrait de générer des réponses dont la qualité statistique serait suffisante (aucun échec sur la suite de tests NIST).

Afin de tester cela, une analyse rapide des identifiants de 128 bits générés avec différentes durées d'acquisition a été réalisée à une température de 25° C et pour la tension d'alimentation nominale des FPGA (1.2 V). Les durées d'acquisition utilisées sont les suivantes : 60, 100 et 500 cycles d'horloge à 50 MHz. Le Tableau 3.5 présente les résultats de cette caractérisation rapide. En se basant sur une analyse rapide de la stabilité et de l'unicité des différents bits de la différence entre le nombre d'oscillations des cellules TERO, les bits dont la stabilité et l'unicité sont suffisantes ont été sélectionnés. Pour une durée d'acquisition de 60 cycles d'horloge, les bits intéressants (d'un point de vue de la stabilité et de l'unicité) sont les bits 5, 6 et 15. Pour 100 cycles, ces bits sont les bits 6, 7 et 15. Enfin, les bits intéressants pour une durée de 500 cycles d'horloge sont les bits 7, 8 et 15. Seules les configurations utilisant ces bits sont présentées dans le Tableau 3.5.

Pour une durée d'acquisition de 60 cycles d'horloge, on voit dans le Tableau 3.5 que toutes les configurations sélectionnées permettent de générer des identifiants de 128 bits passant avec succès l'ensemble des six tests NIST. Seuls les identifiants construits uniquement à partir du bit 5 de la différence entre le nombre d'oscillations des cellules TERO ont une stabilité au dessus de 10%. Il est donc possible de générer des identifiants avec toutes les autres configurations proposées : bit 6, bit 15, bits 5 et 15, bits 6 et 15, et bits 5, 6 et 15. Toutefois, la configuration la plus intéressante correspond à la construction des identifiants à partir des bits 6 et 15. En effet, en plus d'avoir une excellente stabilité et une unicité très proche de 50%, elle utilise deux bits. La configuration utilisant les bits 5, 6 et 15 est également très intéressante.

Pour une durée d'acquisition de 100 cycles d'horloge, toutes les configurations ne passent pas tous les tests statistiques et certaines sont donc à éliminer (Tableau 3.5). En revanche, il est possible d'extraire deux ou trois bits des valeurs des différences entre le nombre d'oscillations des cellules TERO. En effet, les identifiants générés avec les bits 7 et 15 ou les bits 6, 7 et 15 présentent une stabilité suffisamment basse

Tableau 3.5 – Analyse statistique des identifiants de 128 bits générés avec les réponses de la TERO-PUF, pour différentes configurations d'extraction de bits et pour différentes durées d'acquisition sur FPGA Xilinx Spartan 6.

Durée (cycles d'horloge)	Configuration	Stabilité	Unicité	Tests statistiques					
				T1	T2	T3	T4	T5	T6
60	Bit 5	10.17	49.88	V	na	V	V	V	V
	Bit 6	5.09	49.88	V	na	V	V	V	V
	Bit 15	2.38	49.37	V	na	V	V	V	V
	Bits 5 et 15	6.08	49.51	V	V	V	V	V	V
	Bits 6 et 15	3.68	49.65	V	V	V	V	V	V
	Bits 5, 6 et 15	5.89	49.49	V	V	V	V	V	V
100	Bit 6	13.26	48.51		na				
	Bit 7	5.12	49.59	V	na	V	V	V	V
	Bit 15	3.83	48.48	V	na	V	V	V	V
	Bits 6 et 15	8.13	48.48		V		V		
	Bits 7 et 15	4.18	48.76	V	V	V	V	V	V
	Bits 6, 7 et 15	7.13	48.73	V	V	V	V	V	V
500	Bit 7	26.28	48.11		na				
	Bit 8	16.96	48.56		na		V		
	Bit 15	5.64	48.22	V	na	V	V	V	V
	Bits 7 et 15	14.81	47.58		V		V		
	Bits 8 et 15	10.39	47.56		V		V	V	
	Bits 7, 8 et 15	14.55	47.35	V		V		V	

et une unicité proche de 50%, tout en passant l'ensemble des six tests. Enfin, pour une durée d'acquisition de 500 cycles d'horloge, le même constat que celui de la caractérisation à 2 000 cycles d'horloge (Tableau 3.4) peut être établi à partir du Tableau 3.5. En effet, seuls les identifiants utilisant uniquement le bit 15 peuvent être utilisés.

Ainsi, les trois cas de figure concernant la durée d'acquisition des réponses de la TERO-PUF ont été analysés et montrent qu'une durée d'acquisition située entre 30 cycles d'horloge à 50 MHz et 500 cycles permet d'extraire plusieurs bits de la différence entre le nombre d'oscillations des cellules TERO, tout en conservant de très bonnes propriétés statistiques.

3.4.2 Altera Cyclone V

Pour la caractérisation de la TERO-PUF sur les FPGA Altera Cyclone V, nous avons rencontré plus de difficultés en raison d'un problème d'alimentation des cartes. Malheureusement, ce problème ayant été résolu trop tard, seule la caractérisation en tension de 16 cartes est présentée dans ce manuscrit. Toutefois, les deux durées d'acquisition (30 et 2 000 cycles d'horloge) ont été étudiées exactement de la même manière que pour les FPGA Xilinx Spartan 6.

3.4.2.1 Caractérisation de la TERO-PUF pour une durée d'acquisition de 30 cycles d'horloge

Pour caractériser les réponses de la TERO-PUF sur les FPGA Altera Cyclone V, la même procédure que celle appliquée pour les FPGA Xilinx Spartan 6 est suivie. Dans un premier temps, une évaluation rapide de la qualité de chacun des bits de la différence entre le nombre d'oscillations des cellules TERO-PUF est effectuée. Le Tableau 3.6 présente les résultats de cette première étape.

Tableau 3.6 – Stabilité et unicité moyenne des différents bits de la différence du nombre d'oscillations des cellules TERO pour une durée d'acquisition de 30 cycles d'horloge sur 16 FPGA Altera Cyclone V.

bit	Unicité (%)	Stabilité (%)
0	49.98	42.07
1	49.83	36.55
2	49.88	28.99
3	49.82	21.27
4	49.52	12.49
5	45.88	6.42
6	50.09	3.21
7	26.19	2.04
8	0	0
...	...	...
15	47.62	1.80

Dans le Tableau 3.6, on remarque que les bits 5, 6 et 15 possèdent une stabilité et une unicité suffisante pour permettre la génération d'identifiants. La robustesse de ceux-ci est donc évaluée vis-à-vis des variations de tension d'alimentation. Les configurations utilisées pour la construction des identifiants de 128 bits sont les suivantes : bit 5, bit 6, bit 15, bits 5 et 15, bits 6 et 15, et bits 5, 6 et 15.

La tension d'alimentation nominale des FPGA Altera Cyclone V étant différente de celle des FPGA Xilinx Spartan 6, les points de tension étudiés sont différents, comme nous l'avons présenté dans la Section 3.3.2. La Figure 3.17 présente les résultats de l'analyse de la robustesse des identifiants de 128 bits générés à partir des réponses de la TERO-PUF pour ces différentes tensions.

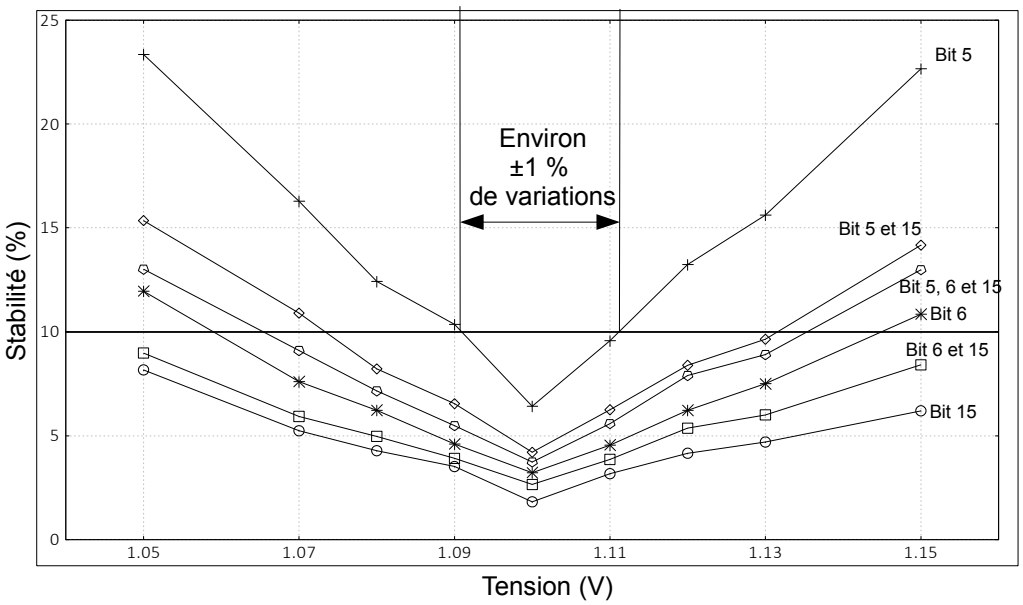

FIGURE 3.17 – Étude de la stabilité moyenne des identifiants de 128 bits générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de tension, pour une durée d'acquisition de 30 cycles d'horloge à 50 MHz sur 16 FPGA Altera Cyclone V.

Sur la Figure 3.17, on peut voir que les identifiants qui varient le plus vis-à-vis des variations de tension correspondent à ceux générés à partir du bit 5 de la différence du nombre d'oscillations des cellules TERO. Pour cette configuration, les identifiants ont une stabilité en dessous de 10% sur une plage de tension allant de 1.09 V à 1.11 V, ce qui représente une variation de 1% autour de la tension nominale (1.1 V). En revanche, toutes les autres configurations présentent une stabilité qui reste en dessous de 10% pour tout l'intervalle de tension recommandé par Altera : entre 1.07 V et 1.13 V. Il est donc possible d'extraire plus d'un bit par challenge avec une durée d'acquisition de 30 cycles d'horloge à 50 MHz. On notera une plus grande sensibilité aux variations de tension que pour les FPGA Xilinx Spartan 6 (voir Figure 3.14).

La dernière étape de l'analyse des réponses de la TERO-PUF pour cette durée d'acquisition correspond à l'étude de leur imprévisibilité. Pour cela, les identifiants de 128 bits des 16 FPGA Altera Cyclone V sont générés pour une tension d'alimentation de 1.1 V et pour une température de 25° C. Ils sont ensuite concaténés afin de créer une chaine binaire de 2048 bits (16×128) puis recoupés en dix chaines de taille égale. La taille de la chaine binaire n'étant pas un multiple de dix, les tests statistique sont appliqués à dix chaines de 204 bits et les 8 bits restants ne sont pas pris en compte. Le Tableau 3.7 présente la stabilité, l'unicité ainsi que l'imprévisibilité des identifiants de 128 bits générés avec les six configurations utilisées pour analyser la robustesse des réponses de la TERO-PUF.

Tableau 3.7 – Résultats des tests statistiques des identifiants de 128 bits générés avec les configurations possibles utilisant de 1 à 3 bits sur FPGA Altera Cyclone V et pour une durée d'acquisition de 30 cycles d'horloge.

Configuration	Stabilité %	Unicité %	Tests statistiques					
			T1	T2	T3	T4	T5	T6
Bit 5	6.42	45.88		na				
Bit 6	3.21	50.09	V	na	V	V	V	V
Bit 15	1.80	47.62	V	na	V	V	V	V
Bits 5 et 15	4.19	46.19		V	V	V		V
Bits 6 et 15	2.66	45.58	V	V	V	V	V	V
Bits 5, 6 et 15	3.73	47.39	V	V	V	V	V	V

D'après les résultats présentés dans le Tableau 3.7, seules deux configurations ne passent pas l'ensemble des six tests NIST : bit 5, et bits 5 et 15. Toutes les autres configurations passent avec succès l'ensemble des tests et peuvent donc être utilisées pour la construction d'identifiants. On peut également remarquer qu'une configuration utilisant deux bits par challenge ainsi que celle utilisant trois bits sont retenues. Cela prouve qu'il est possible d'extraire un, deux ou trois bits de la différence du nombre d'oscillations des cellules TERO pour construire des identifiants sur les FPGA Altera Cyclone V. De plus, la durée d'acquisition de 30 cycles d'horloge ne semble pas être trop courte comme c'était le cas pour les FPGA Xilinx Spartan 6.

3.4.2.2 Caractérisation de la TERO-PUF pour une durée d'acquisition de 2 000 cycles d'horloge

Pour la durée d'acquisition de 2 000 cycles d'horloge à 50 MHz, la même procédure est une nouvelle fois appliquée. Le Tableau 3.8 présente les résultats de l'évaluation rapide des différents bits de la différence du nombre d'oscillations des cellules TERO.

La Tableau 3.8 montre que seul le bit de signe (bit 15) possède une unicité et une stabilité suffisamment bonnes pour permettre la construction d'identifiants. Ce constat est exactement le même que pour les FPGA Xilinx Spartan 6 (Section 3.4.1.2). Toutefois, dans un souci de comparaison, la robustesse des identifiants de 128 bits générés avec les mêmes configurations de bits que précédemment est étudiée. La Figure 3.18 présente le résultat de la caractérisation des identifiants de 128 bits générés à partir des réponses de la TERO-PUF vis-à-vis des variations de tension, pour une durée d'acquisition de 2 000 cycles d'horloge.

Tableau 3.8 – Stabilité et unicité moyenne des différents bits de la différence du nombre d'oscillations des cellules TERO pour une durée d'acquisition de 2 000 cycles d'horloge sur 16 FPGA Altera Cyclone V.

bit	Unicité (%)	Stabilité (%)
0	49.99	47.98
1	50.05	46.03
2	50.02	41.96
3	49.67	36.91
4	49.84	30.89
5	49.85	26.36
6	48.93	21.89
7	45.44	18.71
8	40.99	15.46
9	36.96	13.15
10	37.11	11.01
11	28.91	8.86
12	33.27	7.99
13	27.49	7.07
14	0.00	0.00
15	47.29	2.68

Comme on peut le voir sur la Figure 3.18, seuls les identifiants générés à partir du bit 15 sont exploitables et ont une stabilité qui reste en dessous de 10%, pour toute les tensions étudiées. Toutes les autres configurations génèrent des identifiants dont la stabilité est au-dessus des 10% quelle que soit la tension et ne sont donc pas utilisables. La dernière étape de l'analyse, comme pour une durée d'acquisition de 30 cycles d'horloge, est l'analyse de l'imprévisibilité des réponses. Pour cela, les tests NIST sont appliqués de la même manière que pour la durée d'acquisition précédente et le Tableau 3.9 présente les résultats de ces tests ainsi que la stabilité et l'unicité des identifiants de 128 bits, pour une tension de 1.1 V et une température de 25° C.

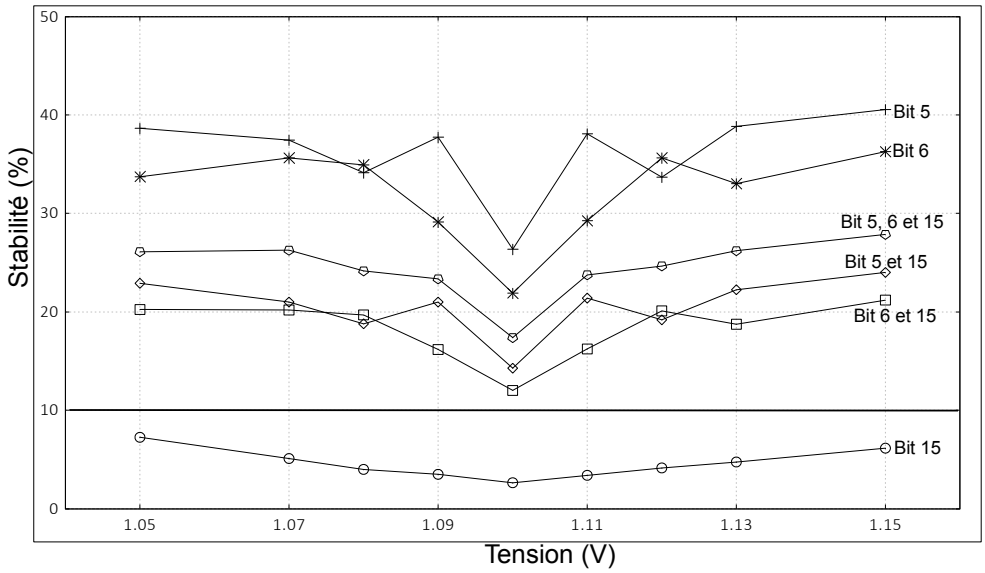

FIGURE 3.18 – Étude de la stabilité moyenne des identifiants de 128 générés grâce aux réponses de la TERO-PUF en fonction des bits de sortie sélectionnés et vis-à-vis des variations de tension, pour une durée d'acquisition de 2 000 cycles d'horloge à 50 MHz sur 16 FPGA Altera Cyclone V.

Tableau 3.9 – Résultats des tests statistiques des identifiants de 128 bits générés avec les configurations possibles utilisant de 1 à 3 bits sélectionnés sur FPGA Altera Cyclone V pour une durée d'acquisition de 2 000 cycles d'horloge.

Configuration	Stabilité %	Unicité %	Tests statistiques					
			T1	T2	T3	T4	T5	T6
Bit 5	26.36	49.85	V	na	V	V	V	V
Bit 6	21.89	48.93		na		V	V	V
Bit 15	2.68	47.29	V	na	V	V	V	V
Bits 5 et 15	14.28	48.22	V	V	V	V	V	V
Bits 6 et 15	12.04	47.80	V	V	V	V	V	V
Bits 5, 6 et 15	17.39	48.89	V	V	V	V	V	V

Pour cette durée d'acquisition, le Tableau 3.9 montre que seul le bit de signe (bit 15) permet de construire des identifiants dont la qualité statistique est suffisante. Malgré la réussite de la plupart des tests NIST, toutes les autres configurations présentent une stabilité et une unicité bien trop mauvaises.

Dans cette section, nous avons caractérisé la TERO-PUF en analysant les identifiants générés à partir de ses réponses aux challenges envoyés. Cette caractérisation a été effectuée pour deux familles de FPGA : Xilinx Spartan 6 et Altera Cyclone V. Les conclusions quant à la construction des identifiants sont les mêmes pour les

deux familles de circuits. De plus, l'impact de la durée d'acquisition sur la qualité des réponses a clairement été démontré.

3.4.3 Comparaison

Afin de compléter la caractérisation de la TERO-PUF qui vient d'être présentée pour deux familles de FPGA (Xilinx Spartan 6 et Altera Cyclone V), il est indispensable de comparer leurs résultats. Pour cela, les meilleures configurations utilisant un, deux ou trois bits par challenge sont sélectionnées pour les deux durées d'acquisition analysées (30 et 2 000 cycles d'horloge à 50 MHz) et reportées dans les Tableaux 3.10, 3.11 et 3.12.

Tableau 3.10 – Performances de la TERO-PUF pour les FPGA Xilinx Spartan 6 pour une durée d'acquisition de 30 cycles d'horloge.

Configuration de la TERO-PUF	Nombre de bits par challenge	Bits utilisés	Unicité	Stabilité	Stabilité maximale pour les intervalles de stabilité	Intervalle de température	Intervalle de tension
C1	1	15 (MSB)	48.48%	2.50%	10%	12° C à 36° C	1.18 V à 1.22 V
C2	**2**	**5 et 15**	**47.83%**	**2.46%**	**10%**	**2°C à 65°C**	**1.13V à 1.27V**
C3	3	4, 5 et 15	46.60%	3.67%	10%	7° C à 48° C	1.16 V à 1.25 V

Tableau 3.11 – Performances de la TERO-PUF pour les FPGA Altera Cyclone V pour une durée d'acquisition de 30 cycles d'horloge.

Configuration de la TERO-PUF	Nombre de bits par challenge	Bits utilisés	Unicité	Stabilité	Stabilité maximale pour les intervalles de stabilité	Intervalle de température	Intervalle de tension
C1	1	15 (MSB)	47.62%	1.80%	10%	-	1.05 V à 1.15 V
C2	**2**	**6 et 15**	**45.58%**	**2.66%**	**10%**	-	**1.05V à 1.15V**
C3	3	5, 6 et 15	47.39%	3.73%	10%	-	1.07 V à 1.13 V

À partir des Tableaux 3.10 et 3.11, on remarque que les résultats de la caractérisation de la TERO-PUF sont similaires. La configuration la plus intéressante est, dans les deux cas, celle qui utilise deux bits par challenge. En effet, elles présentent un intervalle de stabilité (en dessous de 10%) qui englobe l'intégralité des spécifications des FPGA. De plus, leur unicité et leur stabilité à température et tension nominales sont tout à fait satisfaisantes.

Pour une durée d'acquisition de 2 000 cycles d'horloge, les résultats sont moins proches pour les deux technologies, comme le montre le Tableau 3.12. En effet, les identifiants générés avec les FPGA Xilinx Spartan 6 montrent une stabilité moins bonne que ceux générés avec les FPGA Altera Cyclone V. Pour ces derniers, les identifiants présentent un intervalle de stabilité qui englobe les spécifications de tension du FPGA.

Tableau 3.12 – Performances de la TERO-PUF pour les FPGA Xilinx Spartan 6 et Altera Cyclone V pour une durée d'acquisition de 2 000 cycles d'horloge.

Technologie FPGA	Nombre de bits par challenge	Bits utilisés	Unicité	Stabilité	Stabilité maximale pour les intervalles de stabilité	Intervalle de température	Intervalle de tension
Xilinx Spartan 6 (CMOS 45 nm)	1	15 (MSB)	48.85%	6.38%	10%	3° C à 33° C	1.18 V à 1.22 V
Altera Cyclone V (CMOS 28 nm)	1	15 (MSB)	47.29%	2.68%	10%	-	1.05 V à 1.15 V

Pour compléter l'analyse, le Tableau 3.13 présente les résultats de la caractérisation de la TERO-PUF réalisée sur des circuits ASIC réalisés en technologie AMS CMOS $0.35\mu m$ [2]. Cette étude a été réalisée par un autre membre de l'équipe [39] mais il est intéressant de comparer les résultats obtenus sur ASIC et ceux obtenus sur FPGA. De plus, les cellules TERO réalisées sur ASIC possèdent exactement le même nombre d'inverseurs par branche que les cellules que nous avons implantées sur FPGA, ce qui rend possible une comparaison des résultats. L'étude complète de la TERO-PUF sur ces ASIC est présentée dans [39]. Dans le Tableau 3.13, seuls les résultats de l'analyse des identifiants de 128 bits générés à partir des bits les plus intéressants de la différence entre les nombres d'oscillations des cellules TERO sont présentés. Les identifiants générés avec les configurations présentes dans le Tableau 3.13 passent l'ensemble des six tests NIST présentés dans ce chapitre.

Tableau 3.13 – Performances de la TERO-PUF sur ASIC réalisés en technologie AMS CMOS $0.35\mu m$.

Configuration de la TERO-PUF	Nombre de bits par challenge	Bits utilisés	Unicité	Stabilité	Stabilité maximale pour les intervalles de stabilité	Intervalle de température	Intervalle de tension
C1	1	8 (MSB)	49.72%	0.61%	7%	-20° C à 70° C	3.0 V à 3.6 V
C2	2	4 et 8	48.26%	1.04%	10%	-15° C à 45° C	3.1 V à 3.5 V
C3	3	3, 4 et 8	49.51%	1.73%	10%	15° C à 40° C	3.2 V à 3.4 V

Comme on peut le voir dans le Tableau 3.13, l'unicité et la stabilité des réponses de la TERO-PUF sont nettement meilleures sur ASIC que sur FPGA. En effet, même l'utilisation de trois bits pour construire les identifiants donnent une stabilité en dessous de 2% pour des conditions de fonctionnement normales (température et tension d'alimentation nominales). Cela s'explique par le fait que le concepteur d'un

2. `http://ams.com/eng/Products/Full-Service-Foundry/Process-Technology/CMOS`

circuit ASIC possède une plus grande maîtrise des délais de routage de son circuit. La précision dans les délais de routage des cellules TERO permet de maximiser l'extraction d'entropie issue des variations liées au processus de fabrication des circuits, avec des délais beaucoup plus courts que pour les circuits FPGA.

En revanche, le même constat est à faire quant à la forte sensibilité des réponses de la TERO-PUF vis-à-vis des variations de tension. En effet, pour les identifiants générés avec trois bits de la différence entre le nombre d'oscillations des cellules TERO, la stabilité est en dessous de 10% seulement dans un intervalle allant de 3.2 V à 3.4 V, ce qui représente une variation de 3% autour de la tension d'alimentation nominale (3.3 V). Pour la stabilité vis-à-vis des variation de température, les résultats sont similaires avec une stabilité qui reste en dessous de 10% pour un intervalle allant de 15° C à 45° C, ce qui représente une variation de 40% autour de la température nominale de 25° C.

Pour compléter cette section, le Tableau 3.14 compare les résultats de caractérisation obtenus pour d'autres PUF de la littérature basées sur des cellules oscillantes avec ceux présentés dans ce chapitre. Pour les travaux concernant les TERO-PUF, seuls les résultats des identifiants générés avec le bit de signe des différences entre le nombre d'oscillations sont présentés.

Tableau 3.14 – Comparaison de la TERO-PUF avec d'autres PUF.

Métrique	RO-PUF [107]	RO-PUF [103]	TERO-PUF [28]	TERO-PUF [39]	TERO-PUF (présentée dans ce chapitre)	
Technologie utilisée	Xilinx Spartan-3E (90 nm)	ASIC (65 nm)	Altera Cyclone-II (90 nm)	ASIC (350 nm)	Xilinx Spartan-6 (45 nm)	Altera Cyclone-V (28 nm)
Unicité	47.3%	49.5%	48.0%	49.7%	48.5%	47.6%
Stabilité	0.9%	2.8%	1.7%	0.6%	2.6%	1.8%
Stabilité (variations de tension et de température)	15%	3.9%	Non présentée	6.2%	10%	10%
Cellules de base	1 NAND et 4 inverseurs	1 NAND et 40 inverseurs	2 NAND et 14 inverseurs	2 NAND et 14 inverseurs	2 AND et 14 inverseurs	2 AND et 2 inverseurs et 12 LCELL

On peut remarquer dans le Tableau 3.14 que plus le nombre d'inverseurs est important, plus la PUF semble robuste vis-à-vis des variations de température et de tension, comme le montrent les résultats de la RO-PUF sur ASIC [103] par rapport à ceux de la TERO-PUF sur ASIC [39]. En revanche, la RO-PUF sur ASIC utilise plus du double d'inverseurs que la TERO-PUF sur ASIC et présente une stabilité moins bonne dans les conditions de fonctionnement normales. L'implantation de la RO-PUF sur FPGA qui est la plus légère en termes de surface [107] présente quant à elle une très forte sensibilité vis-à-vis des variations de température et de tension.

Pour ces deux fonctions physiques non-clonables (RO-PUF et TERO-PUF), il

est possible d'augmenter l'unicité des réponses ainsi que leur robustesse vis-à-vis des variations de températures et de tension en augmentant le nombre d'inverseurs des cellules. Toutefois, cela réduit la stabilité de la PUF. Le dimensionnement des cellules des fonctions physiques non-clonables pour une technologie donnée est donc un compromis entre l'unicité et la stabilité des réponses. Enfin, on peut remarquer dans le Tableau 3.14 que la TERO-PUF possède une plus grande unicité que la RO-PUF, même si les cellules possèdent moins d'inverseurs. Cela implique que la TERO-PUF permet d'extraire plus d'entropie issue des variations du processus de fabrication des circuits que la RO-PUF.

3.5 Conclusion et Perspectives

Dans ce chapitre, nous avons présenté la conception et la caractérisation d'une fonction physique non-clonable : la TERO-PUF. L'étape de conception est toujours très importante pour ce type de fonctions car elle doit être recommencée pour chaque nouvelle technologie. En effet, les fonctions physiques non-clonables sont basées sur les éléments les plus basiques de chaque technologie et souffrent d'un défaut de portabilité.

Le banc de caractérisation qui a été spécialement mis en place pour la caractérisation des PUF a également été présenté. L'avantage de ce banc se situe dans sa modularité et dans sa facilité d'utilisation. En effet, il est possible de ne changer que les blocs de base constituant la fonction physique non-clonable et de se re-servir du banc sans autres modifications. De même, il est tout à fait possible de modifier le banc pour lui rajouter des fonctionnalités comme la caractérisation croisée en température et tension.

Enfin, la caractérisation de la TERO-PUF a été présentée pour deux familles de FPGA : Xilinx Spartan 6 et Altera Cyclone V. Les résultats ont été précisément décrits et montrent qu'il est possible d'extraire plus d'un bit par challenge grâce à la TERO-PUF. Cela constitue clairement un avantage en termes de surface d'implantation puisque deux fois moins de cellules sont nécessaires, par rapport à la RO-PUF. Lors de la caractérisation, l'évaluation de l'imprévisibilité des réponses a été effectuée grâce à certains tests de la suite de tests statistiques NIST. Cette approche permet une analyse plus fine qu'une simple mesure du biais. En revanche, ces tests ne sont pas adaptés aux fonctions physiques non-clonables et il est nécessaire d'aller plus loin dans cette partie de l'analyse et d'étudier une modélisation stochastique de l'extraction d'entropie.

Les travaux présentés dans ce chapitre ont besoin d'être à la fois complétés et améliorés. En effet, la caractérisation en température de la TERO-PUF sur les FPGA Altera Cyclone V est manquante et doit être faite. De plus, le paramètre de la durée d'acquisition introduit dans ce chapitre doit être étudié plus en détail. Il faudrait par exemple trouver un moyen d'écarter les cellules dont l'état oscillant est très long (voir permanent) de la génération des réponses de la PUF.

Enfin, il est également nécessaire de trouver et de tester des attaques sur la TERO-PUF. En effet, la sécurité de la TERO-PUF n'a encore jamais été éprouvée. Il est certainement possible de récupérer la moyenne du nombre d'oscillations des cellules TERO par l'utilisation d'un canal auxiliaire comme le rayonnement électromagnétique ou la consommation de puissance. Une analyse spectrale de la consommation de puissance pourrait par exemple être effectuée sur une fenêtre glissante. De cette manière, une raie devrait apparaître à la fréquence d'oscillation d'une cellule lorsque celle-ci oscille. La raie disparaîtra lorsque la cellule se sera arrêtée. Ainsi, en enregistrant le temps écoulé entre l'apparition et la disparition des raies, il est possible de remonter jusqu'à un nombre d'oscillations. De plus, l'amplitude de la raie est linéairement liée avec le nombre d'oscillations de la cellule. Il est donc possible de retrouver ce nombre.

Chapitre 4

Étude et implantation de LILLIPUT

La dernière partie de ce manuscrit de thèse est composée de deux chapitres dans lesquels nous nous intéressons au dernier mécanisme de protection nécessaire dans le projet SALWARE. Ce mécanisme correspond à la mise en place d'un algorithme léger de chiffrement permettant une communication sécurisée lors de l'activation du circuit. Comme cette activation ne se fait qu'une fois dans la vie du circuit intégré, l'algorithme de chiffrement choisi doit occuper le moins de surface possible. En revanche, le temps d'exécution de ce dernier n'est pas une contrainte puisqu'il ne sera utilisé que lors de l'activation. Ainsi, la seule et unique contrainte d'implantation pour le projet SALWARE est la surface. Dans les deux chapitres qui suivent, le terme "léger" sera donc relatif à cette contrainte. La Figure 4.1 montre le positionnement de ce mécanisme dans le projet SALWARE.

Les travaux présentés dans ce chapitre ainsi que dans le Chapitre 5 ont été réalisés dans le cadre d'une collaboration et d'un échange avec l'équipe du Professeur Kris Gaj de l'Université George Mason à Fairfax aux États-Unis. De plus, les fichiers sources de toutes les implantations matérielles présentées dans ce chapitre sont disponibles sur le site du projet SALWARE [1].

Dans ce chapitre, un algorithme léger de chiffrement par bloc, récent (2015), est étudié : LILLIPUT. Nous présentons ici les premières implantations matérielles de cet algorithme sur FPGA. De plus, des résultats de synthèse pour la technologie ASIC AMS $0.35\mu m$ [2] sont également présentés pour les implantations matérielles décrites dans ce chapitre. Dans la première section de ce chapitre, l'algorithme de chiffrement est présenté. La Section 4.2 présente les stratégies d'implantation ainsi que la méthode utilisée pour analyser leur surface. Ensuite, les Sections 4.3 et 4.4 détaillent les implantations matérielles parallèle et série de LILLIPUT. Enfin, la Section 4.5 présente les résultats de ces implantations pour FPGA et ASIC. La Section 4.6 conclue ce chapitre et donne quelques perspectives d'améliorations.

1. http://www.univ-st-etienne.fr/salware/lightweight_block_cipher.htm
2. http://ams.com/eng/Products/Full-Service-Foundry/Process-Technology/CMOS

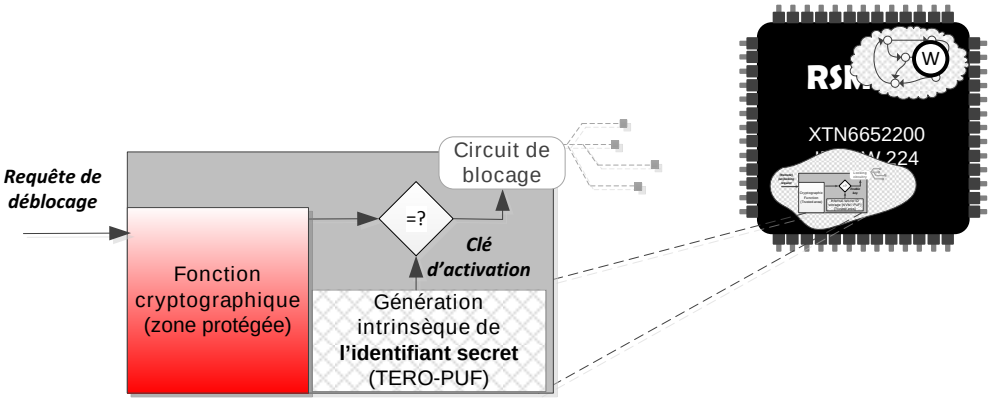

FIGURE 4.1 – Positionnement de l'implantation d'algorithme léger de chiffrement dans le projet SALWARE.

4.1 Description de l'algorithme LILLIPUT

LILLIPUT est un algorithme léger de chiffrement par bloc utilisant une architecture basée sur un réseau de Feistel généralisé. Les spécifications de cet algorithme ainsi qu'une analyse complète de sa sécurité ont été publiées en 2015 dans [17]. Dans cette section, une description de chaque fonction de LILLIPUT est présentée dans le but de faciliter la compréhension de son implantation.

LILLIPUT prend un bloc de 64 bits ainsi qu'une clé de 80 bits en entrée et produit un message de 64 bits en sortie. Dans le but de fournir une description claire, le bloc de données X à chiffrer (ou à déchiffrer) est vu comme une suite de quartets (mots de 4 bits) : $X = X_{15}X_{14} \cdots X_0$.

LILLIPUT est composé d'une fonction de ronde qui est appliquée trente fois afin de chiffrer ou de déchiffrer un message. Cette fonction de ronde s'articule autour de trois opérations nommées : *NonLinearLayer*, *LinearLayer* et *PermutationLayer*. Pour chaque tour, un clé de ronde est générée grâce à une fonction de gestion ainsi qu'une fonction d'extraction de clé. De plus, seule la moitié du bloc de données correspondant aux bits de poids fort du message ($X_{15}X_{14} \cdots X_8$) est modifiée lors de chaque passage dans la fonction de ronde.

4.1.1 La fonction de ronde

La fonction de ronde commence par une étape non linéaire (*NonLinearLayer*). Cette fonction est définie par l'expression suivante :

$$X_{8+i} = X_{8+i} \oplus S(X_{7-i} \oplus RK_i^j), \ i \in [0, 7]$$

où la fonction S représente le tableau de substitution utilisé dans LILLIPUT (*Sbox*) et RK_i^j représente le quartet i de la clé de ronde correspondante au tour j. Le tableau de substitution est présenté en notations hexadécimales dans le Tableau 4.1. Dans ce Tableau 4.1, $x_{(4)}$ représente un mot de 4 bits.

Tableau 4.1 – Tableau de substitution utilisé dans l'algorithme LILLIPUT.

$x_{(4)}$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x_{(4)})$	4	8	7	1	9	3	2	E	0	B	6	F	A	5	D	C

Ensuite, la deuxième fonction (*LinearLayer*) peut être appliquée. Elle consiste en une série de *ou exclusif* entre les quartets de poids faible du bloc de données $(X_7X_6 \cdots X_0)$ et le résultat de la première fonction du tour. Plus précisément, cette fonction est définie par :

$$\begin{aligned} X_{15} &= X_1 \oplus X_2 \oplus X_3 \oplus X_4 \oplus X_5 \oplus X_6 \oplus X_{15}; \\ X_i &= X_7 \oplus X_i,\ i \in [9, 14]; \\ X_8 &= X_8 \end{aligned}$$

Enfin, la dernière fonction permute les deux moitiés du bloc de données mais effectue également des permutations à l'intérieur de chaque moitié. Cette fonction est appelée *PermutationLayer*. Elle est définie dans le Tableau 4.2. Dans ce Tableau 4.2, la permutation inverse utilisée pour le déchiffrement d'un message est également présentée.

Tableau 4.2 – Permutation du bloc de données utilisé dans LILLIPUT et son inverse.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(i)$	13	9	14	8	10	11	12	15	4	5	3	1	2	6	0	7
$\pi^{-1}(i)$	14	11	12	10	8	9	13	15	3	1	4	5	6	0	2	7

4.1.2 La gestion de clé

La dernière partie de l'algorithme correspond à la gestion de la clé tout au long du chiffrement (ou du déchiffrement) d'un message. Cette fonction est divisée en deux parties. La première partie permet de générer une clé de 80 bits à partir de la clé générée lors du tour précédent. La deuxième partie a pour but d'extraire la clé de ronde de 32 bits servant dans la fonction non-linéaire du tour.

Pour générer les clés successives (de 80 bits), la fonction utilisée est basée sur quatre registres à décalage à rétroaction linéaire (LFSR pour *Linear Feedback Shift Register* en anglais). Pour décrire cette fonction, notons Y la clé qui entre dans la fonction. Cette clé est divisée en vingt quartets : $Y = Y_{19}Y_{18} \cdots Y_0$. De plus, un décalage de i bits vers la gauche est noté $<< i$, un décalage de i bits vers la droite est noté $>> i$ et les rotations de i bits vers la gauche et vers la droite sont notées $<<< i$ et $>>> i$. Les quatre registres à décalage sont notés $\mathcal{L}_0$, $\mathcal{L}_1$, $\mathcal{L}_2$ et $\mathcal{L}_3$ et définis sur la Figure 4.2. Dans cette figure, le décalage du registre est appelé *PermutationLFSR* et l'opération logique est nommée *MixingLFSR*.

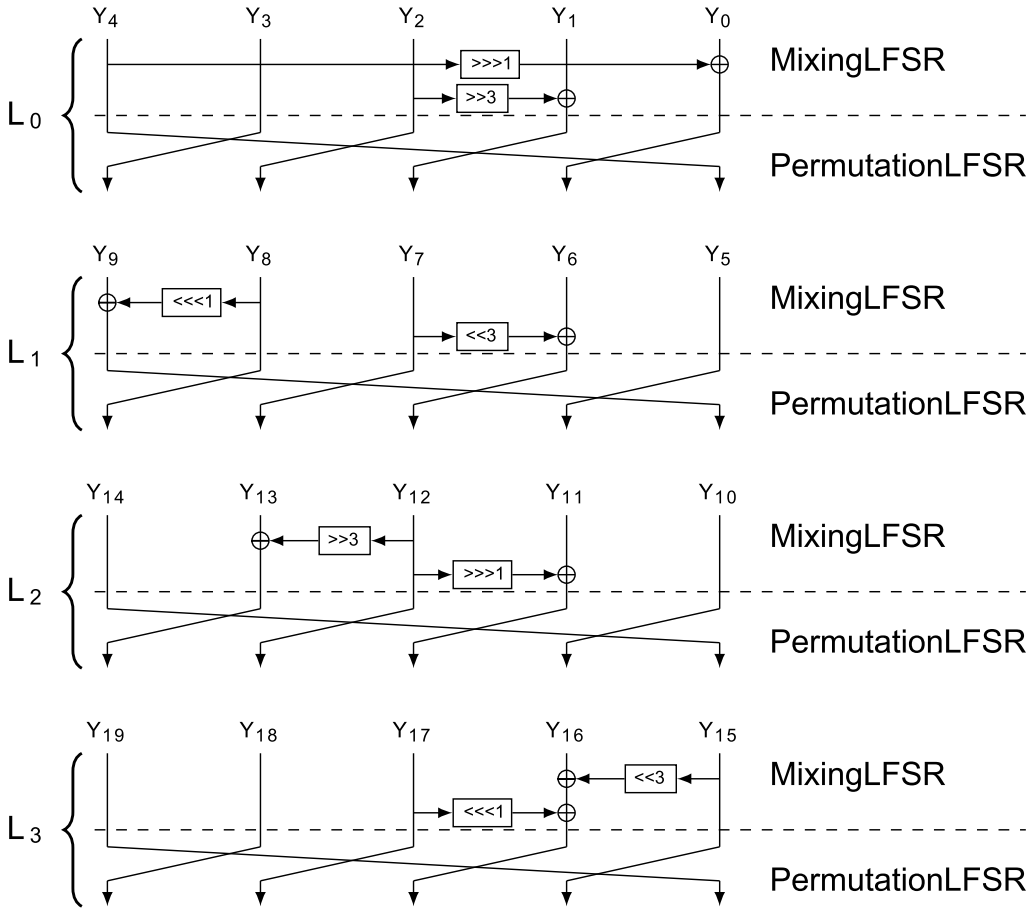

FIGURE 4.2 – Description de la fonction de gestion de clé de l'algorithme LILLIPUT.

Pour la fonction d'extraction de la clé de ronde, la clé générée lors du tour précédent est prise en compte. Pour cette fonction, la chaine de 32 bits $Z_{(32)}$, qui est composée de certains quartets de la clé Y, est définie par :

$$Z_{(32)} = Y_{18}||Y_{16}||Y_{13}||Y_{10}||Y_9||Y_6||Y_3||Y_1$$

Ensuite, chaque quartet RK_i^j de la clé de ronde du tour j (RK^j) est définie par :

$$RK_i^j = S(Z_i||Z_{i+8}||Z_{i+16}||Z_{i+24}),\ i \in [0,7]$$

où S représente la même fonction de substitution que celle utilisée dans la fonction de ronde (Tableau 4.1). Enfin, le numéro du tour $j \in [0,29]$ est ajouté à la clé de ronde pour donner $RK_{(32)}^j = RK^j \oplus j_{(5)}||0_{(27)}$.

4.1.3 Chiffrement et déchiffrement

Comme LILLIPUT est basé sur un réseau de Feistel, le processus de déchiffrement est très proche du chiffrement. En effet, la seule partie de la fonction de ronde à inverser est la permutation car elle ne se contente pas d'intervertir les deux moitiés du bloc de données mais elle permute également les quartets au sein de chaque moitié.

La gestion de clé est également très proche entre le chiffrement et le déchiffrement. Il faut simplement remonter les clés dans l'ordre inverse : de RK^{29} à RK^0. Pour cela, il faut donc au préalable recalculer la dernière. Pour calculer les clés de ronde dans l'ordre inverse, il faut effectuer la permutation inverse des registres à décalage avant de ré-appliquer les mêmes modifications sur les quartets de la clé (fonction *MixingLFSM* présentée dans la Figure 4.2).

Pour plus de détails sur cet algorithme léger de chiffrement par bloc, le lecteur intéressé peux se référer directement aux spécifications de l'algorithme [17].

4.2 Stratégies d'implantations et analyse de la surface d'implantation d'un algorithme

Afin d'implanter dans le matériel (FPGA ou ASIC) un algorithme de chiffrement par bloc, deux stratégies sont communément utilisées. La première stratégie est nommée parallèle et la seconde implantation série. D'autres stratégies peuvent être employées selon les objectifs de l'application pour laquelle les algorithmes de chiffrement sont implantés. Ces autres stratégies sont généralement appelées implantations séries partielles [16]. Dans cette section, les deux stratégies utilisées dans ce chapitre (parallèle et série) sont présentées. La méthode d'analyse de la surface des implantations proposées est expliquée dans la Section 4.2.3.

4.2.1 Stratégie parallèle

La stratégie d'implantation matérielle parallèle est probablement la plus simple qu'il est possible d'utiliser pour implanter un algorithme de chiffrement par bloc. Il s'agit d'une implantation séquentielle dont la granularité est la fonction de ronde : celle-ci est implantée une fois, puis elle est appliquée sur le bloc de données (à chiffrer ou à déchiffrer) plusieurs fois. Lors de chaque passage dans la fonction de ronde, l'intégralité du bloc de données est mis à jour en une seule fois (un seul cycle d'horloge). Si l'une des opérations utilisées dans la fonction de ronde s'applique sur un mot de taille réduite, elle est dupliquée jusqu'à ce que le nombre d'entrées totales dans la fonction soit égale à la taille du bloc de données. Exactement de la même manière, la clé est entièrement mise à jour en un seul cycle d'horloge dans une implantation parallèle et la fonction de gestion de clé est implantée une seule fois. Si besoin, une clé de ronde est extraite de la clé avant d'être envoyée à la fonction de ronde.

Comme cette stratégie est la plus simple à utiliser, elle est le point de départ de l'étude menée dans ce chapitre. En effet, l'utilisation de cette stratégie permet de bien comprendre les différentes fonctions de l'algorithme et donc de trouver où il est possible de réduire la surface d'implantation. La Figure 4.3 montre un schéma général décrivant l'implantation parallèle de n'importe quel algorithme de chiffrement par bloc, pour le chiffrement et le déchiffrement. Pour certains algorithmes, quelques changements sont à apporter à ce schéma. Par exemple, il est possible que l'algorithme partage certaines opérations entre le chiffrement et le déchiffrement. C'est notamment le cas de LILLIPUT, qui est présenté dans ce chapitre. Pour d'autres algorithmes, il y a une fonction de gestion de clé très simple qui est utilisée à la fois pour le chiffrement et pour le déchiffrement des messages. Dans ce cas, la Figure 4.3 peut très facilement être adaptée pour décrire l'implantation matérielle parallèle de ces algorithmes.

Dans la Figure 4.3, le bloc de données à traiter a une taille de p bits, la clé une taille de k bits et la clé de ronde une taille de sk bits. Lorsque l'opération demandée (chiffrement ou déchiffrement) est terminée, la sortie du système est directement extraite du registre de données.

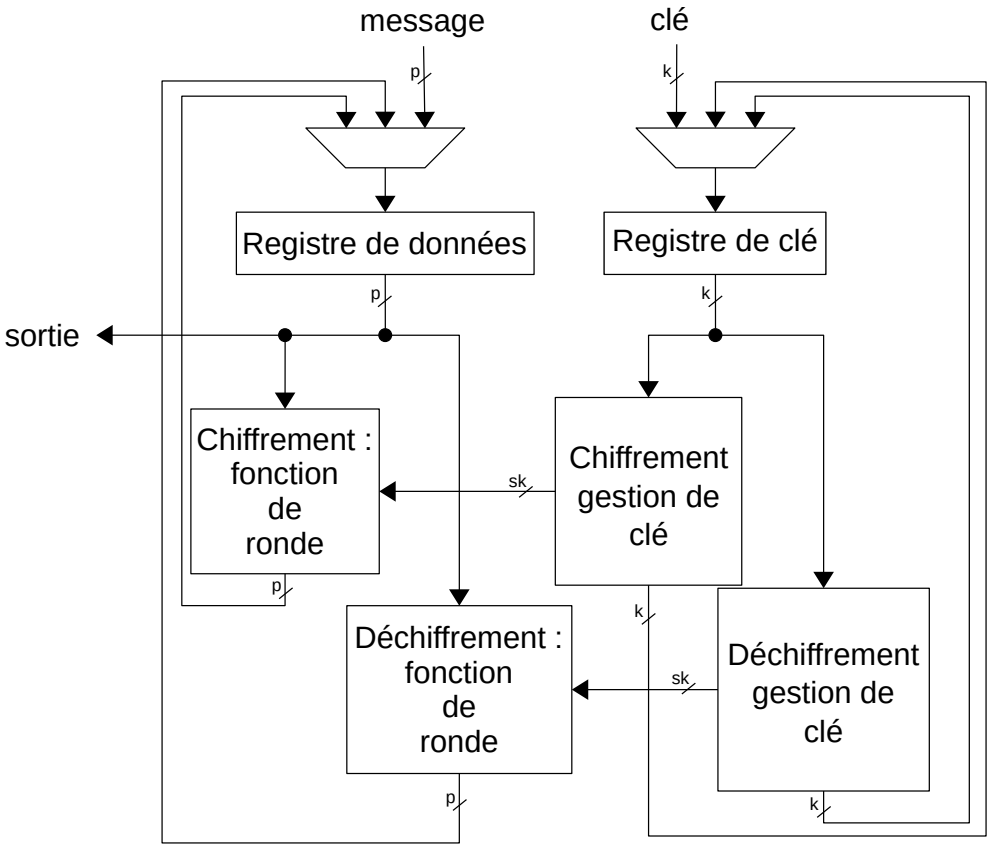

FIGURE 4.3 – Schéma général d'une implantation matérielle parallèle d'un algorithme de chiffrement par bloc effectuant le chiffrement et le déchiffrement.

4.2.2 Stratégie série

La stratégie d'implantation série est généralement utilisée pour les implantations matérielles des algorithmes légers de chiffrement par bloc. Le but de cette stratégie est d'utiliser le minimum de ressources pour implanter l'algorithme. En particulier, ce sont très souvent les ressources logiques qui sont prises en compte. Le chemin de données doit être le plus petit possible pour une implantation série. Il faut donc analyser chaque opération utilisée dans l'algorithme pour trouver la taille de ce chemin de données. L'utilisation du plus petit chemin de données se fait dans l'espoir de réduire la surface d'implantation au détriment de la vitesse d'exécution de l'algorithme. Par exemple, supposons que l'algorithme à implanter utilise un message de 64 bits et possède un tableau de substitution transformant des mots de 4 bits. Contrairement à la stratégie parallèle où 16 tableaux de substitution seraient implantés pour mettre à jour les 64 bits du message en une fois, un seul tableau sera implanté avec la stratégie d'implantation série, ce qui implique que ce tableau sera utilisé 16 fois pour mettre à jour tout le message. L'implantation série est donc l'implantation séquentielle possédant la plus petite granularité fonctionnelle de l'algorithme implanté.

Pour parvenir à implanter un algorithme de chiffrement par bloc avec cette stratégie, les registres de données et de clé sont généralement transformés en registres à décalage, ce qui permet d'appliquer la fonction de ronde sur une petite partie du message. Malheureusement, la fonction de gestion de clé est très souvent compliquée (voir impossible) à implanter pour cette stratégie. Dans ce cas, la clé est mise à jour en un seul cycle d'horloge, puis elle est conservée durant le temps nécessaire à la mise à jour complète du message.

Une implantation série n'est pas toujours la meilleure stratégie pour réduire la surface d'implantation d'un algorithme de chiffrement par bloc. En effet, le séquencement de la fonction de ronde est très intéressant pour réduire la partie logique de l'implantation, mais elle implique souvent une complexification du contrôleur du système ainsi que l'ajout de registres intermédiaires ou de multiplexeurs permettant d'envoyer la bonne partie du message (ou de la clé) à chaque opération. Toutefois, c'est la stratégie d'implantation série qui est utilisée pour prouver qu'un algorithme de chiffrement par bloc est adapté aux applications nécessitant un algorithme léger.

4.2.3 Analyse de la surface d'implantation d'un algorithme de chiffrement par bloc

Dans ce chapitre, seule la surface d'implantation est analysée. De plus, les résultats d'implantation sont générés pour des FPGA Xilinx Spartan 6, Xilinx Spartan 3 et pour la technologie ASIC AMS $0.35\mu m$.

En ce qui concerne les résultats sur FPGA Xilinx, la métrique la plus utilisée est le *Slice*. Un *Slice* est un bloc élémentaire contenu dans les FPGA Xilinx. Malheureusement, un *Slice* des FPGA Xilinx Spartan 6 est complètement différent de celui des FPGA Xilinx Spartan 3. En effet, un *Slice* dans les FPGA Xilinx Spar-

tan 6 contient quatre *look up tables* (LUT) avec six entrées et deux sorties (LUT6) ainsi que huit registres alors qu'un *Slice* dans les FPGA Xilinx Spartan 3 contient deux LUT avec quatre entrées (LUT4) et une sortie ainsi que deux registres. Il est donc impossible de comparer deux implantations réalisées sur ces deux familles de FPGA en utilisant le *Slice* comme métrique. De plus cette métrique ne peut pas être utilisée pour comparer des implantations réalisées sur des FPGA de constructeurs différents comme Altera ou Microsemi puisque l'élément *Slice* est spécifique aux FPGA Xilinx. Il ne faut donc pas utiliser le *Slice* comme métrique de comparaison de la surface d'implantation d'un système, quelqu'il soit.

Toutefois, il existe des éléments communs à presque tous les FPGA, ce sont les LUT et les registres. Dans ce chapitre, nous avons choisis d'utiliser ces deux éléments comme métrique d'analyse de la surface d'implantation de LILLIPUT. Même s'il n'est pas possible de comparer des implantations effectuées sur des FPGA dont les LUT ont une structure différente, cette métrique permet une analyse plus fine de la surface que le *Slice*. En effet, un *Slice* n'est pas toujours rempli à 100% et même si un seul registre du *Slice* est utilisé, ce dernier sera décompté dans la surface utilisée d'implantation, ce qui conduit à une sur-estimation de la surface. En revanche, pour une LUT ou un registre, soit la ressource est utilisée, soit elle ne l'est pas et il n'est pas possible de diviser ces ressources. La surface finale sera donc plus précise.

Dans le cas des implantations ASIC, la surface de l'implantation peut-être exprimée soit en μm^2, soit en porte équivalente (GE). L'utilisation de la surface en μm^2 est très dépendante de la technologie alors qu'une porte équivalente correspond à la surface d'une porte *Non Et* à deux entrée et une sortie (NAND2). Cette métrique est également dépendante de la technologie utilisée pour implanter l'algorithme puisque chaque porte n'utilise pas la même surface (comparée aux autres portes) dans toutes les technologies. Cependant, cette métrique est plus générique que la surface en μm^2.

4.3 Implantation matérielle parallèle de LILLIPUT

Comme cela a été présenté dans la section précédente, cette stratégie consiste à implanter la fonction de ronde une fois pour que la totalité du bloc de données soit mise à jour en un cycle d'horloge. Il en est de même pour la clé dont les 80 bits sont mis à jour en un seul cycle d'horloge. Le chemin de données a donc une largeur de 64 bits pour le message et de 80 bits pour la clé.

Dans LILLIPUT, la fonction de ronde doit être appliquée 30 fois sur le message pour le chiffrer ou le déchiffrer. Il faut donc 30 cycles d'horloge pour effectuer un chiffrement. En revanche, il est nécessaire de recalculer la dernière clé avant de commencer tout déchiffrement. Ensuite, le bloc de données est entièrement mis à jour à chaque cycle d'horloge et la clé de ronde correspondante est reproduite à l'aide de la fonction de gestion de clé inverse. En effet, pour arriver à une implantation légère en surface, aucune clé n'est stockée en mémoire et chaque clé doit être recalculée. Cela a pour conséquence d'imposer la mise en attente du message avant le début du

déchiffrement ainsi qu'une latence supplémentaire de 30 cycles d'horloge. Ainsi, un total de 60 cycles d'horloge est nécessaire pour déchiffrer un message. La Figure 4.4 présente l'implantation de LILLIPUT pour cette stratégie. Le chiffrement ainsi que le déchiffrement y sont présentés.

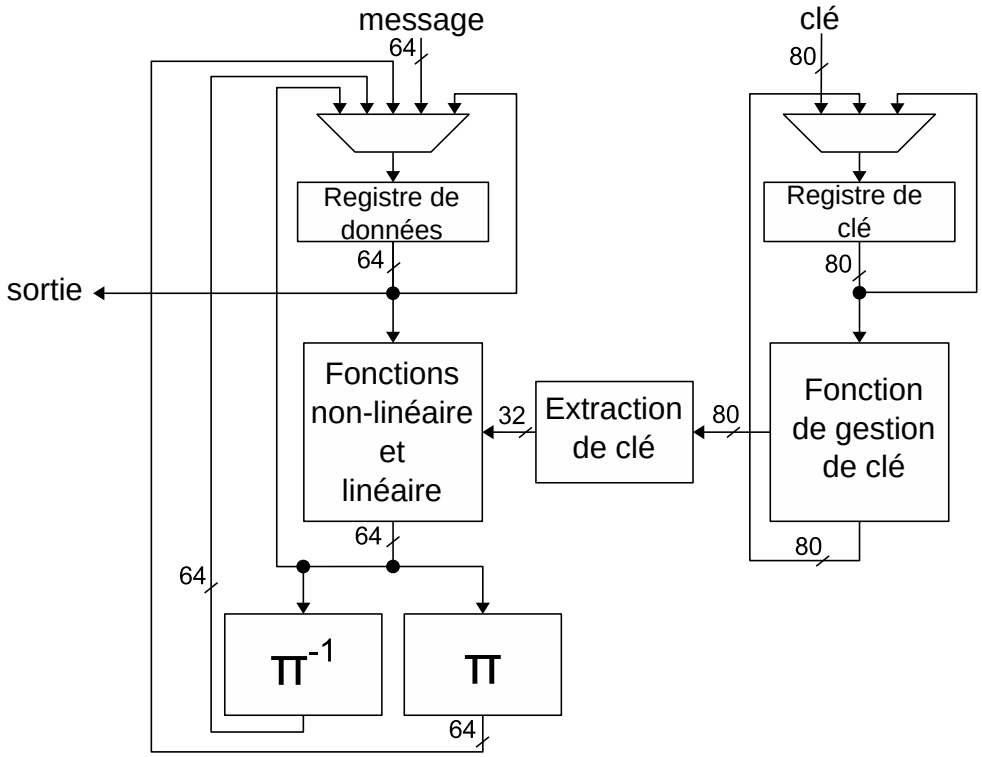

FIGURE 4.4 – Implantation matérielle parallèle de LILLIPUT.

4.3.1 La fonction de ronde

En raison de la structure en réseau de Feistel utilisée dans LILLIPUT, la fonction de ronde est très proche entre le chiffrement et le déchiffrement. En effet, deux des trois opérations de chaque tour peuvent être partagées entre les deux modes de fonctionnement. Seule la permutation doit absolument être changée et la permutation inverse est utilisée pour le chiffrement. La Figure 4.5 présente l'implantation matérielle de cette fonction de ronde pour la stratégie parallèle.

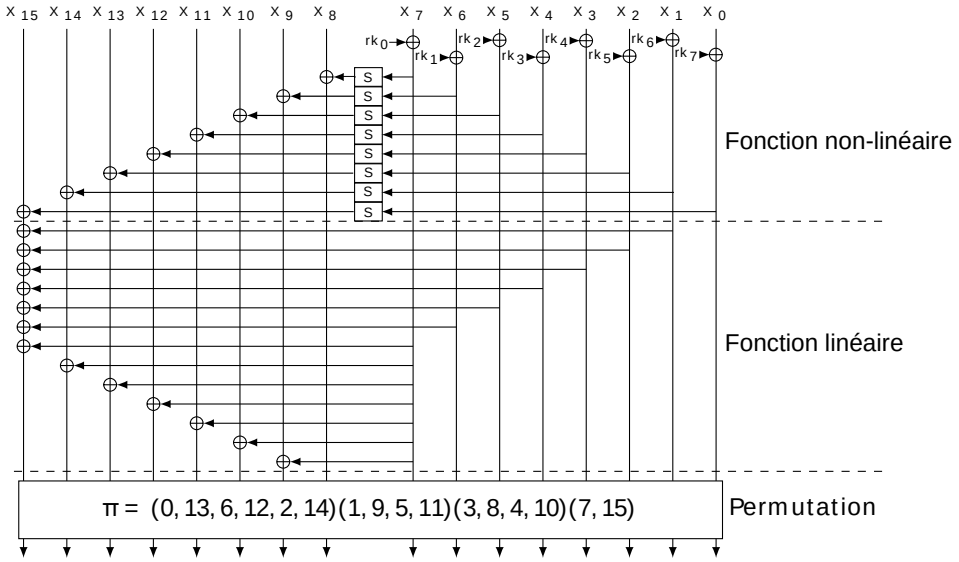

FIGURE 4.5 – Implantation matérielle parallèle de la fonction de ronde de LILLIPUT pour le chiffrement.

Il est possible d'estimer la quantité de ressources utilisées par cette implantation grâce à la structure interne des FPGA Xilinx Spartan 6. Cette estimation des ressources est faite en nombre de LUT6 car il s'agit du bloc de plus bas niveau disponible dans ce type de FPGA. Pour la première partie de la fonction de ronde (*NonLinearLayer*), la moitié du bloc de données (les bits de poids faible : $X_7 X_6 \cdots X_0$) est ajoutée à la clé de ronde avec un *ou exclusif* (XOR). Le résultat de cette addition est envoyé dans le tableau de substitution. Enfin, la sortie est ajoutée à l'autre moitié du bloc de données (les bits de poids fort). Du fait de la taille du tableau de la *Sbox*, qui prend 4 bits en entrée et produit 4 bits en sortie, il est donc possible d'implanter une *Sbox* avec 4 LUT6. Cela laisse deux entrées libres dans chaque LUT, ce qui permet d'y ajouter la dernière addition du *NonLinearLayer*. En revanche, il n'est pas possible d'inclure la première addition de cette fonction dans les même LUT6 car il n'y a pas assez d'entrées. Il faut donc en utiliser une deuxième. Ainsi, 2 LUT6 par bit sont nécessaires pour implanter cette première étape de la fonction de ronde, soit un total de 64 LUT6 pour mettre à jour les 32 bits de poids fort du bloc de données.

La fonction *LinearLayer* est composée de 24 portes XOR sur 2 bits et de une porte XOR sur 8 bits. L'ensemble des portes XOR sur 2 bits peut être inclus dans les LUT6 du *NonLinearLayer* car il reste une entrée disponible sur les LUT6 contenant les *Sbox*. Pour la porte XOR sur 8 bits, une LUT6 supplémentaire est nécessaire. Ainsi, un total de 68 LUT6 est requis pour implanter la fonction de ronde de LILLIPUT pour la stratégie parallèle. Les résultats expérimentaux montrent que cette analyse du nombre de LUT6 utilisé conduit à une légère sous-estimation de la surface. L'outil de synthèse de Xilinx donne un résultat de 71 LUT6 pour l'implantation de la fonction de ronde. Même si l'estimation fournie par l'analyse du nombre de

LUT6 nécessaires est en dessous de la réalité, on peut constater qu'elle en est très proche (seulement 3 LUT6 de différence). De plus, l'estimation des ressources nécessaires pour implanter une fonction est une approche très intéressante pour savoir où il est possible de réduire la surface d'implantation mais aussi pour comprendre pourquoi une implantation est plus gourmande en surface qu'une autre.

La dernière partie de la fonction de ronde correspond à l'étape de permutation. Elle n'ajoute pas de LUT6 à l'implantation mais augmente le nombre d'entrées des multiplexeurs (MUX) se trouvant avant le registre de données (voir Figure 4.4). Comme la permutation est différente pour le chiffrement et pour le déchiffrement, deux entrées sont ajoutées à ce multiplexeur. De plus, il n'y a pas de permutation lors du dernier passage du bloc de données dans la fonction de ronde, ce qui ajoute encore une entrée au multiplexeur.

4.3.2 La fonction de gestion de la clé

La fonction de gestion de clé est divisée en deux parties bien distinctes. L'une de ces parties extrait la clé de ronde de 32 bits à partir de la clé de 80 bits présente au début du tour. Pour cela, certains bits de la clé de 80 bits sont sélectionnés, réarrangés puis l'ensemble passe au travers d'une *Sbox*. Enfin, les 5 bits de poids fort du résultat sont ajoutés avec le numéro du tour en cours. Il faut donc 8 *Sbox*, soit un total de 32 LUT6 : une par bit à générer. Les cinq portes XOR sur 2 bits nécessaires peuvent être inclus dans les LUT6 contenant les *Sbox*. La Figure 4.6 présente le schéma de cette implantation de la fonction d'extraction de la clé de ronde.

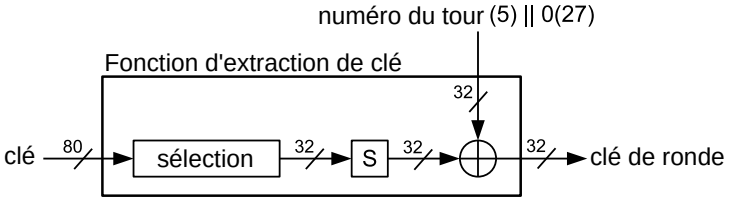

FIGURE 4.6 – Implantation parallèle de la fonction d'extraction de clé de ronde de LILLIPUT.

L'autre partie du bloc de gestion de clé correspond à la mise à jour de la clé de 80 bits. Elle est composée de 4 registres à décalage à rétroaction linéaire (LFSR) qui mettent à jour 20 bits de la clé chacun. Chaque LFSR a besoin du même nombre de portes logiques pour être implanté : cinq portes XOR sur 2 bits. Ils sont tous composés d'une fonction nommée *MixingLFSR* (notée M dans la Figure 4.2) ainsi que d'une permutation (P). Pour le déchiffrement, il faut utiliser la permutation inverse qui sera notée P^{-1}. Décrivons l'implantation de l'un des quatre LFSR : $\mathcal{L}_0$ (représenté en haut de la Figure 4.2). Grâce à la structure du LFSR, il est possible de partager la fonction M entre le chiffrement et le déchiffrement. La Figure 4.7 présente quatre implantations de cette fonction de gestion de clé. Sur la Figure 4.7,

chaque rectangle en pointillé délimite l'implantation (a, b, c ou d) de la fonction et le reste représente le multiplexeur présent avant le registre de clé ainsi que ce registre.

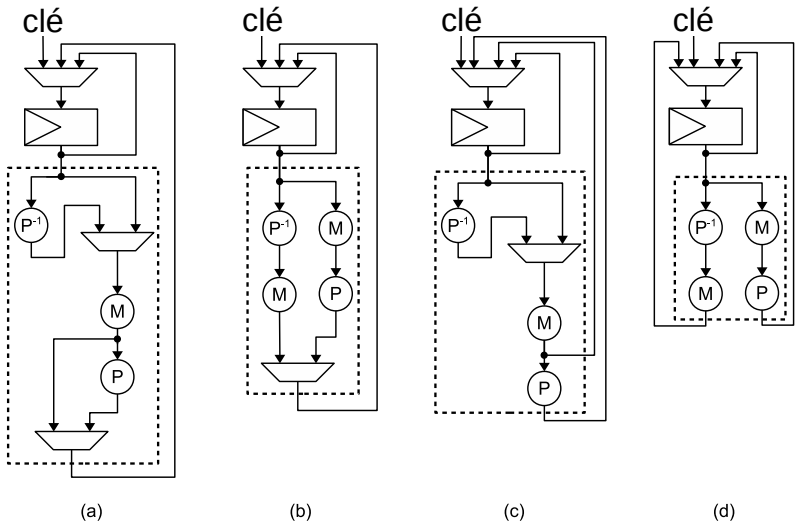

FIGURE 4.7 – Quatre implantations parallèles différentes de la fonction de gestion de clé de LILLIPUT.

Dans le but de comparer ces différentes implantations de la fonction de gestion de clé, le Tableau 4.3 présente le nombre des différents éléments nécessaires pour les effectuer.

Tableau 4.3 – Comparaison des ressources nécessaires pour les quatre implantations de la fonction de gestion de clé.

Implantation	XOR	MUX 4to1	MUX 3to1	MUX 2to1	Nombre de LUT6	GE (AMS 0.35μm)
Fig. 4.7 (a)	5	0	20	40	40	165.05
Fig. 4.7 (b)	10	0	20	20	40	136.7
Fig. 4.7 (c)	5	20	0	20	45	138.25
Fig. 4.7 (d)	10	20	0	0	30	109.9

Les nombres présentés peuvent être surestimés et il est possible que l'outil de synthèse de Xilinx effectue des optimisations réduisant le nombre de ressources utilisées. Dans le Tableau 4.3, la notation $MUXxtoy$ représente un multiplexeur avec x entrées et y sorties. Les résultats sont très proches sur FPGA, ce qui peut s'expliquer par la structure interne des FPGA, qui permet de partager beaucoup de ressources. La dernière colonne du Tableau 4.3 indique le nombre de portes équivalentes nécessaires, qui a été calculé à partir de la technologie ASIC AMS 0.35μm. Il est indispensable de noter que les conversions directes du nombre d'éléments logiques en portes équivalentes ne sont pas précises. En effet, elles ne prennent pas en compte le routage du circuit qui peut occuper un partie importante de la surface

utilisée. Toutefois, on constate que l'implantation (d) de la fonction de gestion de clé occupe 60 GE de moins que l'implantation (a). Cela indique que l'implantation (d) est potentiellement plus petite en termes de surface que l'implantation (a).

De plus, les résultats présentés ne concernent que le LFSR $\mathcal{L}_0$ de la fonction de gestion de clé alors qu'elle est composée de quatre LFSR. Chacun des trois autres LFSR est également composé de cinq portes XOR sur 2 bits et peut être implanté de la même manière que $\mathcal{L}_0$. Ainsi, même si l'estimation sur FPGA indique des résultats de surface similaire, ce n'est pas le cas pour ASIC avec une réduction d'environ 222.4 GE entre les implantations (d) et (a) : $4 \times (165.5 - 109.9)$.

4.3.3 L'implantation complète

L'implantation complète de LILLIPUT permettant le chiffrement et le déchiffrement de messages est présentée dans la Figure 4.4. Seule une partie de cette implantation n'a pas encore été analysée en termes de nombre de LUT6 nécessaires : les multiplexeurs présents devant les registres de données et de clés. En raison du nombre d'entrées des LUT dans les FPGA Xilinx Spartan 6, il est possible d'implanter un multiplexeur à quatre entrées et une sortie dans une seule LUT6. Toutefois, si le nombre d'entrées requises par le multiplexeur dépasse quatre, le coût en surface des multiplexeurs devient alors un problème pour une implantation légère en surface. En effet, un multiplexeur à cinq entrées utilisera deux ou trois LUT6, selon les optimisations faites par l'outil de synthèse Xilinx. Ainsi, la taille des multiplexeurs est le tendon d'Achille de toutes les implantations légères en surface d'algorithmes de chiffrement par bloc et cela est particulièrement vrai pour LILLIPUT. Sur la Figure 4.4, on peut remarquer que le nombre d'entrées du multiplexeur devant le registre de données est déjà de cinq, cela signifie qu'il faut au moins deux LUT6 par bit pour l'implanter. Pour le multiplexeur présent devant le registre de clé, le nombre d'entrées dépend de l'implantation de la fonction de gestion de clé (Figure 4.7). Toutefois, il y a au plus quatre entrées, ce qui implique qu'une LUT6 par bit suffit pour ce multiplexeur.

Le problème lié à la taille des multiplexeurs est encore plus vrai si la taille des LUT est réduite. Par exemple, les FPGA Xilinx Spartan 3 possèdent des LUT avec seulement quatre entrées, ce qui fait augmenter plus rapidement la taille des multiplexeurs en termes de nombre de LUT4 utilisées. L'utilisation de ce type de FPGA permet de mettre en avant la réduction de surface apportée par l'implantation (d) de la fonction de gestion de clé. Le Tableau 4.4 présente les résultats d'implantation de LILLIPUT pour ces deux familles de FPGA Xilinx : Spartan 6 et Spartan 3. Dans ce Tableau 4.4, quatre implantations sont présentées, elles sont nommées LILLIPUT$_a$, LILLIPUT$_b$, LILLIPUT$_c$ et LILLIPUT$_d$. La lettre correspond à l'implantation de la fonction de gestion de clé (visible sur la Figure 4.7). Les résultats présentés dans le Tableau 4.4 contiennent l'algorithme, l'interface de communication et le contrôleur du système. La plateforme qui a été développée pour les implantations légères en surface est disponible sur le site du projet SALWARE [3].

3. http://www.univ-st-etienne.fr/salware/lightweightblockcipher.htm

Comme on peut le voir dans le Tableau 4.4, les résultats des quatre implantations sont très proches sur les FPGA Xilinx Spartan 6, sauf pour LILLIPUT$_c$ qui occupe légèrement plus de surface. En revanche, les résultats des quatre mêmes implantations sont très différents sur les FPGA Xilinx Spartan 3, ce qui s'explique par la taille réduite des LUT de ces FPGA. On remarque ainsi un gain de 121 LUT4 en passant de l'implantations LILLIPUT$_a$ à l'implantation LILLIPUT$_d$.

Tableau 4.4 – Résultats de l'implantation parallèle de LILLIPUT pour les FPGA Xilinx Spartant 6 (XC6SLX16) et Xilinx Spartan 3 (XC3S50).

FPGA	Version d'implantation	LUT6	Registres	LUT6 + registres	Fréquence max (MHz)	Chiffrement débit (Mbps)	Déchiffrement débit (Mbps)
Spartan 6 XC6SLX16	LILLIPUT$_a$	287	157	444	172.8	197.4	128.6
	LILLIPUT$_b$	287	157	444	162.2	185.4	120.7
	LILLIPUT$_c$	368	157	525	165.0	188.6	122.8
	LILLIPUT$_d$	288	157	445	163.4	186.8	121.6
Spartan 3 XC3S50	LILLIPUT$_a$	633	162	795	120.0	137.1	89.3
	LILLIPUT$_b$	597	162	759	126.6	144.7	94.2
	LILLIPUT$_c$	596	162	758	119.7	136.8	89.1
	LILLIPUT$_d$	512	162	674	111.5	127.4	83.0

4.4 Implantation matérielle série de LILLIPUT

Dans cette section, l'implantation série de LILLIPUT est décrite. Pour cette stratégie, le chemin de données est réduit à sa taille minimale, soit 4 bits pour LILLIPUT. Ce faisant, on espère généralement une réduction de la surface d'implantation de l'algorithme en sacrifiant son temps d'exécution. Pour LILLIPUT, seules les fonctions de ronde et d'extraction des clés de ronde peuvent être implantées en série. En effet, il n'y a aucun motif qui se répète dans la fonction de gestion de clé, il n'est donc pas possible d'implanter cette partie en série.

4.4.1 La fonction de ronde

Grâce à la structure très régulière de la fonction de ronde de LILLIPUT, il est possible d'implanter les deux premières opérations (*NonLinearLayer* et *LinearLayer*) avec un chemin de données de 4 bits. La permutation nécessite l'intégralité du bloc de données et ne peut pas fonctionner avec moins de 64 bits. La Figure 4.8 présente l'implantation matérielle série des deux premières opérations de la fonction de ronde avec un chemin de données de 4 bits.

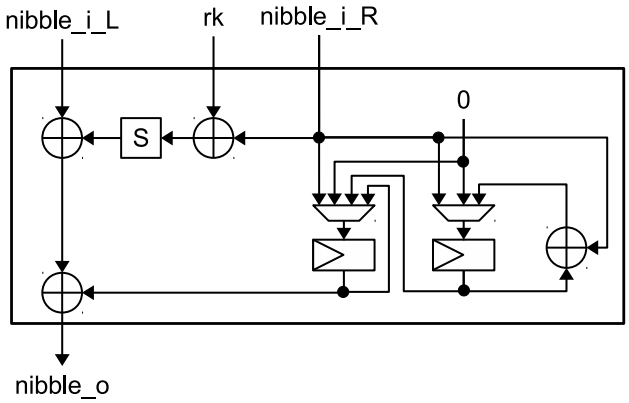

FIGURE 4.8 – Implantation matérielle série des deux premières opérations de la fonction de ronde de LILLIPUT.

Dans la Figure 4.8, l'entrée notée *nibble_i_L* représente un quartet de la moitié de poids fort du bloc de données, celle notée *nibble_i_R*, un quartet de la moitié de poids faible et *rk* représente un quartet de la clé de ronde. Le mot de 4 bits produit en sortie de cette implantation met à jour le quartet ayant le poids le plus fort du registre de données. Toutes les connections présentes dans cette Figure 4.8 ont une largeur de 4 bits.

L'implantation série des deux premières opérations de la fonction de ronde utilise donc quatre portes XOR sur 4 bits, une *Sbox*, deux registres de 4 bits ainsi qu'un MUX4to1 et un MUX3to1. Il est possible d'inclure ce dernier MUX avec l'une des portes XOR et la *Sbox* avec les deux portes XOR qui la suivent. Ainsi, cette implantation matérielle série utilise 12 LUT6 et 8 registres. En comparant cela avec

l'implantation matérielle parallèle des mêmes opérations, cela représente un gain de 56 LUT6 et un ajout de 8 registres.

4.4.2 L'extraction des clés de ronde

L'implantation de la fonction d'extraction des clés de ronde avec la stratégie série est assez évidente. Les 32 bits sélectionnés à partir de la clé de 80 bits sont stockés dans un registre et seuls quatre d'entre eux vont dans la *Sbox*. Au coup d'horloge suivant, le registre est décalé de quatre bits. La Figure 4.9 présente l'implantation matérielle série de cette fonction.

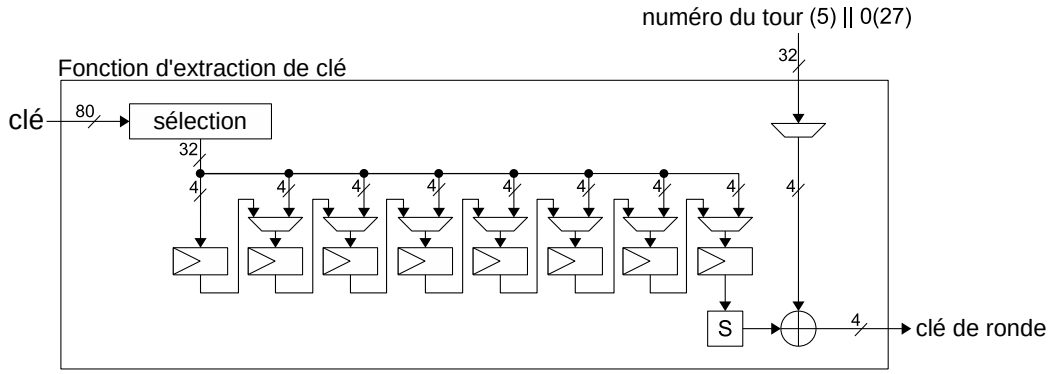

FIGURE 4.9 – Implantation matérielle série de la fonction d'extraction de clé de LILLIPUT.

Pour cette fonction d'extraction, un total de 48 LUT6 et de 32 registres sont nécessaires, ce qui représentent une augmentation de 16 LUT6 et un ajout de 32 registres par rapport à l'implantation parallèle de la même fonction. Cela illustre parfaitement que l'implantation série d'un algorithme ou d'une fonction n'est pas obligatoirement la plus légère en surface. Ce comportement est clairement dû à la différence entre la taille des entrées et des sorties des différentes parties de la fonction. En effet, l'ajout du registre est une conséquence de la réduction du nombre de *Sbox* : seuls 4 bits peuvent être traités à la fois, il faut donc stocker les autres et permettre le décalage des bits dans le registre, d'où l'ajout du multiplexeur.

4.4.3 L'implantation complète

Dans le but de fournir les bons quartets au bon moment durant les 30 tours du chiffrement (ou du déchiffrement), le registre de données est transformé en deux registres à décalage. Pour la moitié de poids fort du message, chaque quartet subit une rotation de 4 bits vers les bits de poids faible à chaque cycle d'horloge. Pour l'autre moitié, la rotation se fait dans l'autre sens. L'implantation série proposée nécessite 8 cycles d'horloge pour effectuer un tour de la fonction de ronde de LILLIPUT. Ainsi, il faut 240 cycles d'horloge pour chiffrer un message (30×8) et 270 cycles d'horloge pour un déchiffrement à cause de la latence supplémentaire induite

par le calcul de la dernière clé $(30 \times 8 + 30)$. La Figure 4.10 présente l'implantation matérielle série proposée pour LILLIPUT.

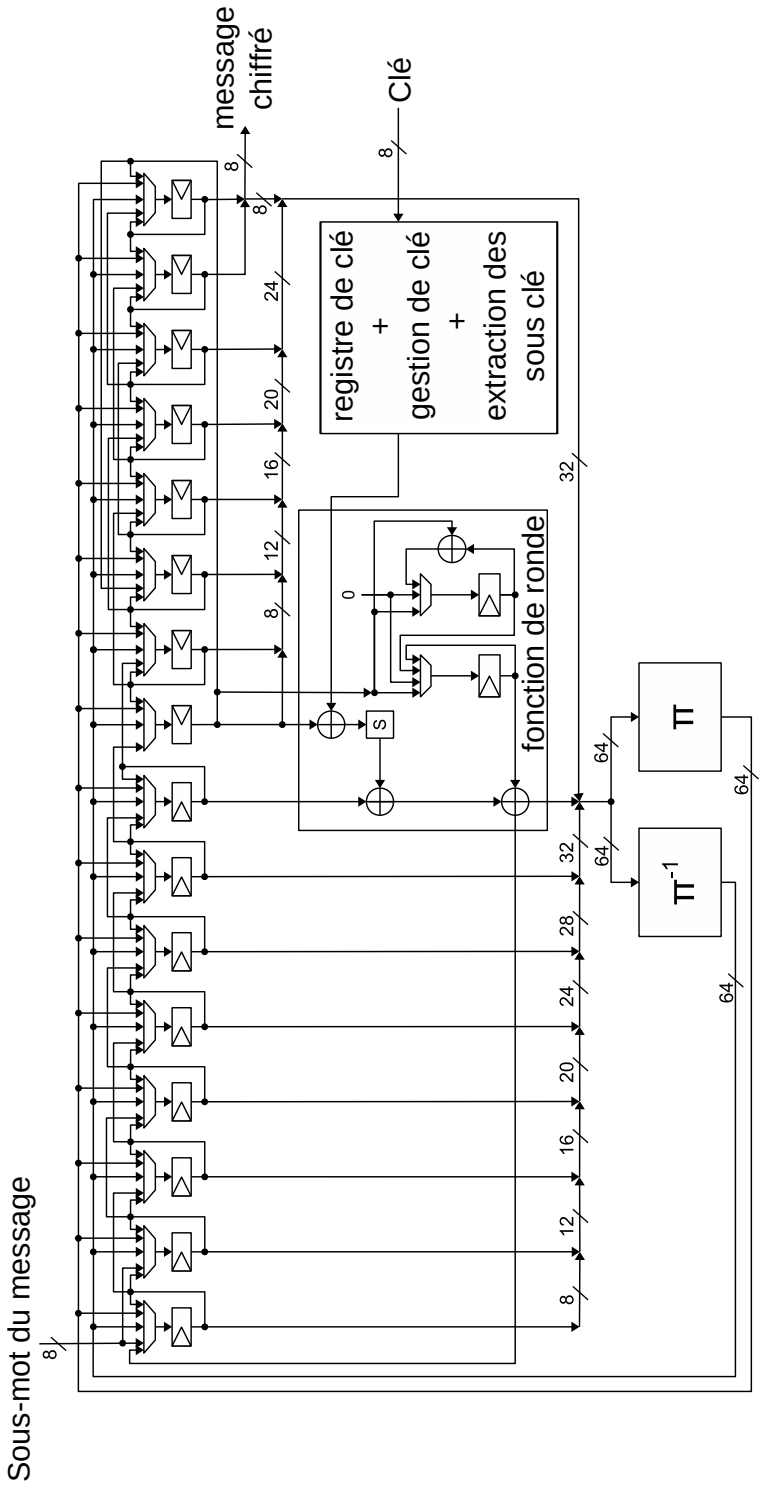

FIGURE 4.10 – Implantation série de LILLIPUT.

À partir de l'analyse du nombre de LUT6 nécessaires à cette implantation, on peut se rendre compte que la surface attendue sur FPGA n'est pas plus petite que celle obtenue pour l'implantation parallèle. En effet, la fonction de gestion de clé reste identique entre les deux stratégies. La fonction d'extraction voit une augmentation très nette de sa surface. Même si les deux premières opérations de la fonction de ronde peuvent être réduites, cela laisse penser que l'implantation matérielle série de LILLIPUT sur les FPGA Xilinx Spartan 6 sera plus gourmande en surface que l'implantation matérielle parallèle. Cela est confirmé par les résultats d'implantation présentés dans le Tableau 4.5.

Tableau 4.5 – Résultats des implantations matérielles séries de LILLIPUT sur les FPGA Xilinx Spartan 6 (XC6SLX16) et Xilinx Spartan 3 (XC3S50).

FPGA	version d'implantation	LUT6	Flip flop	LUT6 + Flip flop	Fmax (MHz)	Débit lors d'un Chiffrement(Mbps)	Débit lors d'un Déchiffrement (Mbps)
XC6SLX16	LILLIPUT$_a$	302	200	502	171.9	41.4	37.2
	LILLIPUT$_b$	303	200	503	197.2	45.0	40.5
	LILLIPUT$_c$	382	200	582	157.3	37.8	34.0
	LILLIPUT$_d$	304	200	504	172.9	41.6	37.4
XC3S50	LILLIPUT$_a$	637	205	842	127.5	30.7	27.6
	LILLIPUT$_b$	637	205	842	129.6	31.2	28.0
	LILLIPUT$_c$	637	205	842	128.5	30.9	27.8
	LILLIPUT$_d$	558	205	763	116.7	28.1	25.2

Les résultats présentés dans le Tableau 4.5 sont ceux des quatre implantations correspondantes aux différentes versions de la fonction de gestion de clé (Figure 4.7).

4.5 Comparaison des résultats d'implantation

Dans cette section, la comparaison des résultats d'implantation matérielle de LILLIPUT pour les deux stratégies d'implantation (parallèle et série) est effectuée. De plus, les résultats de synthèse de ces implantations obtenues pour la technologie ASIC AMS $0.35\mu m$ sont présentés. Ces résultats permettront d'estimer plus précisément la surface occupée par LILLIPUT puisque le routage sera pris en compte.

4.5.1 Comparaison des résultats sur FPGA Xilinx

Comme attendu, les Tableaux 4.4 et 4.5 montrent que la surface d'implantation augmente avec le passage à l'implantation série de LILLIPUT. Cela montre clairement que partager les ressources n'est pas forcément la bonne stratégie sur FPGA. En effet, selon les différences de tailles entre les entrées et les sorties des différentes opérations, des pénalités de surface sont à prendre en compte. Elles incluent généralement des registres, des multiplexeurs et parfois une complexification du contrôleur du système. Ce phénomène a été mis en avant avec la fonction d'extraction de clé (Section 4.4.2).

Les résultats sur les FPGA Xilinx Spartan 3 pour les deux stratégies confirment que l'implantation LILLIPUT$_d$, contenant l'implantation de la fonction de gestion de clé visible dans la Figure 4.7 (d), est la plus petite en surface. Cela confirme encore qu'il est parfois préférable de ne pas partager les ressources sur FPGA.

Enfin, la fréquence maximale utilisable est légèrement supérieure pour les implantations séries. Cependant, comme il n'y aucune réduction de surface, le rapport débit/surface est très mauvais. Finalement, le meilleur résultat d'implantation est obtenu pour l'implantation matérielle parallèle LILLIPUT$_a$ sur les FPGA Xilinx Spartan 6, avec 287 LUT6 et 157 registres pour un débit de chiffrement de 197 Mbps. En ce qui concerne les FPGA Xilinx Spartan 3, le meilleur résultat est obtenu pour l'implantation matérielle parallèle LILLIPUT$_d$, avec 512 LUT4 et 162 registres pour un débit de chiffrement de 127.4 Mbps.

4.5.2 Comparaison des résultats sur ASIC

Pour compléter l'analyse de la surface d'implantation occupée par LILLIPUT, les fichiers sources des huit implantations matérielles ont été utilisés pour générer des résultats sur ASIC en technologie AMS $0.35\mu m$. Nous avons conscience que cette technologie est maintenant dépassée, mais nous avions à disposition les outils et les compétence pour obtenir rapidement des résultats de synthèse dont le but est de mettre en évidence une implantation particulière pour son potentiel de réduction en surface de silicium. Pour cela, nous avons utilisé le logiciel RTL compiler qui permet de transformer un projet FPGA en projet ASIC pour en effectuer la synthèse dans

une technologie donnée. Le Tableau 4.6 présente les résultats obtenus pour les huit implantations de LILLIPUT.

Tableau 4.6 – Surface occupée par les implantations matérielles de LILLIPUT sur ASIC en technologie AMS $0.35\mu m$.

Stratégie	Implantation	Nombre de cellules	Surface des cellules (GE)	Surface des interconnections (GE)	Pourcentage d'interconnection	Surface totale (GE)	Réduction (%)
Parallèle	LILLIPUT$_a$	1309	2516.7	543.3	17.8	3060.0	
	LILLIPUT$_b$	1329	2514.0	525.0	17.3	3039.0	
	LILLIPUT$_c$	1309	2516.7	543.3	17.8	3060.0	
	LILLIPUT$_d$	1244	2408.3	509.3	17.5	2917.7	
Série	LILLIPUT$_a$	1222	2401.7	484.8	16.8	2886.4	5.7
	LILLIPUT$_b$	1135	2361.0	459.7	16.3	2820.7	7.2
	LILLIPUT$_c$	1138	2399.6	491.4	17.0	2891.0	5.5
	LILLIPUT$_d$	1071	2290.0	457.1	16.6	2747.1	5.8

Dans le Tableau 4.6, tous les résultats sont donnés en portes équivalentes. Il est indispensable de noter qu'il s'agit d'une estimation de la surface occupée après synthèse du circuit et non des résultats de réelles implantations ASIC. De plus, il faut être très prudent avec l'utilisation des portes équivalentes. Très souvent, seule la logique est prise en compte et le routage du circuit est considéré comme «gratuit». Cette approche est fausse et il est impératif de prendre en compte la logique, les mémoires et le routage utilisés dans le circuit. Si tel n'est pas le cas, il faut expliquer clairement pourquoi et ce que cela implique sur le résultat présenté. Dans la Section 4.5.2, les résultats seront calculés de la façon suivante :

$$nombre\ de\ GE\ = \frac{Surface\ en\ \mu m^2\ totale}{Surface\ en\ \mu m^2\ d'une\ porte\ NAND2}$$

Ainsi, tout le circuit est pris en compte et pas seulement la logique utilisée par le circuit. La première chose qu'il faut remarquer est que la surface occupée par les implantations matérielles séries est plus petite que la surface occupée par les implantations matérielles parallèles, contrairement au cas des implantations sur FPGA. Une réduction moyenne de 6% est observée pour les deux implantations ayant la même version de la fonction de gestion de clé.

Le Tableau 4.6 montre également que les interconnections du circuit occupent environ 20% de la surface totale de l'implantation. Cela montre qu'une permutation n'est pas gratuite en surface et qu'il est donc nécessaire de prendre en compte ces interconnections lorsqu'on parle de surface d'implantation sur ASIC.

Enfin, on remarque dans le Tableau 4.6 que les implantations LILLIPUT$_d$, qui utilisent la version d de la fonction de gestion de clé (voir Figure 4.7.d), sont toujours les plus petites, quelque soit la stratégie utilisée. Cela confirme que le partage d'opérations logiques n'est pas nécessairement la meilleure stratégie pour réduire la surface d'implantation matérielle d'un algorithme. En effet, si le gain apporté par le partage de quelques portes logiques est inférieur à la surface qu'il faut rajouter pour inclure les multiplexeurs et registres nécessaires à ce partage, la surface totale augmente au lieu de diminuer.

4.6 Conclusions et perspectives

Dans ce chapitre, l'algorithme léger de chiffrement par bloc LILLIPUT a été étudié. Une description complète de cet algorithme a été effectuée (Section 4.1) et les premières implantations matérielles de LILLIPUT sur FPGA ont été présentées. L'analyse approfondie de LILLIPUT a permis de mettre en avant que le partage de ressources logiques n'est pas nécessairement la meilleure stratégie pour obtenir une implantation matérielle légère en surface sur FPGA. En effet, l'implantation série de LILLIPUT sur FPGA est plus volumineuse que son implantation parallèle. Ce constat a été effectué à partir des résultats d'implantations matérielles générés pour deux familles de FPGA : Xilinx Spartan 6 et Xilinx Spartan 3.

En revanche, les résultats sur ASIC montrent que les implantations séries de LILLIPUT sont plus légères en surface que les implantations parallèles. Cela vient

du fait que toutes les opérations logiques ayant le même nombre d'entrées possèdent le même coût en surface sur FPGA, alors que sur ASIC, chaque transistor compte. La surface d'implantation de LILLIPUT sur ASIC montre également que cet algorithme est bien un algorithme léger puisque la plupart des implantations proposées occupent une surface plus petite que le seuil de 3 000 GE, qui est généralement utilisée comme limite pour séparer les algorithmes légers des autres. Toutefois, ces résultats sont très loin de ceux présentés dans [17]. Cela s'explique par le fait que nous n'avons pas réalisé de vraies implantations ASIC de LILLIPUT. En effet, nous avons synthétisé des projets conçus pour FPGA sur une technologie ASIC.

Afin de compléter l'étude de LILLIPUT, il faut tester la robustesse des implantations réalisées vis-à-vis d'attaques, comme celles utilisant les canaux auxiliaires par exemple. De plus, il est indispensable de comparer les implantations de cet algorithme avec celles d'autres algorithmes mais aussi avec les implantations de LILLIPUT réalisées par d'autres équipes.

Il pourrait également être intéressant d'aller plus loin sur l'implantation de LILLIPUT sur ASIC. En effet, seuls les projets réalisés sur FPGA ont été synthétisés sur ASIC. Suivre le flot de conception ASIC et trouver des optimisations spécifiques à ce type de circuit pourrait permettre de réduire la surface d'implantation de LILLIPUT.

Chapitre 5

Étude comparative d'implantations matérielles d'algorithmes légers de chiffrement par bloc

Afin de choisir le meilleur algorithme léger de chiffrement pour le projet SALWARE, il faut comparer les implantations de LILLIPUT présentées dans le chapitre précédent avec d'autres algorithmes. Cette comparaison doit tenir compte des contraintes du projet et seule la surface d'implantation est donc considérée.

Dans ce dernier chapitre, les implantations matérielles de LILLIPUT sont comparées avec les implantations matérielles de trois autres algorithmes légers de chiffrement par bloc : KLEIN, LED et KTANTAN. Afin que la comparaison des différentes implantations soit équitable, une plateforme d'implantation commune a été conçue. Les trois algorithmes implantés ont ensuite dus être choisis et la Section 5.1 présente comment nous avons effectué cette sélection. Enfin, les trois algorithmes et leurs implantations matérielles sont décrits dans la Section 5.2. Les résultats d'implantation et la comparaison avec ceux de LILLIPUT sont présentés dans la Section 5.3. Enfin, la Section 5.4 conclue ce chapitre et donne quelques perspectives d'améliorations.

Les fichiers sources de toutes les implantations matérielles présentées dans ce chapitre sont disponibles sur le site du projet SALWARE [1].

1. http://www.univ-st-etienne.fr/salware/lightweight_block_cipher.htm

5.1 Choix des algorithmes pour le projet SALWARE

Le nombre d'algorithmes de chiffrement par bloc disponibles dans la littérature est important et il n'est pas facile de sélectionner quels sont ceux qui peuvent être candidats pour le projet SALWARE. L'émergence des applications à fortes contraintes, comme les tags d'identification par fréquence radio (plus connus sous l'acronyme RFID pour *Radio Frequency Idenfication* en anglais) a provoqué une croissance rapide du nombre d'algorithmes légers de chiffrement. Parmi eux, ceux permettant uniquement de chiffrer ou de déchiffrer un message de taille fixe sont les plus répandus, il sont regroupés sous le nom d'algorithmes légers de chiffrement par bloc.

5.1.1 Présélection des algorithmes

Dans le cadre du projet SALWARE, il est nécessaire d'implanter un algorithme léger de chiffrement pour établir une communication sécurisée lors de l'activation d'un circuit. Pour cela, huit algorithmes légers de chiffrement par bloc ont été présélectionnés. Cette première sélection d'algorithmes a été faite à l'aide du site CryptoLux [2] qui ne recense pas moins de vingt-quatre algorithmes légers de chiffrement par bloc. Parmi tous ces algorithmes, nous en avons choisi huit parmi les plus récents : aucun algorithme sélectionné n'a été publié avant 2006. Travailler sur des algorithmes récents permet de comparer les résultats d'implantation matérielle obtenus pour des algorithmes plus anciens, qui ont déjà été très étudiés et pour lesquels il est extrêmement difficile d'apporter des nouveautés. Ensuite, en se basant sur les spécifications des algorithmes restants, les huit qui ont été sélectionnés sont : HIGHT, LBlock, ITUBee, LILLIPUT, PRESENT, LED, KLEIN et KTANTAN.

5.1.2 Choix des algorithmes

Les huit algorithmes ont été ensuite implantés sur Octave/Matlab, ce qui permet de générer très facilement des nouveaux vecteurs de tests. De plus, ce type d'implantation logicielle apporte un autre avantage : il est très facile d'afficher les valeurs intermédiaires pendant le chiffrement ou le déchiffrement d'un message. Cette particularité rend ces implantations très utiles pour valider les implantations matérielles, ce qui n'est pas le cas avec une implantation des algorithmes en langage C, avec laquelle il est moins facile d'accéder aux valeurs intermédiaires.

Enfin, seuls quatre algorithmes ont été retenus pour le projet SALWARE. Il ont été choisis grâce à la taille des messages et des clés utilisés. Tous les algorithmes sélectionnés permettent le chiffrement d'un message de 64 bits. De plus, d'après la littérature relative aux algorithmes légers de chiffrement, une taille de clé de 80 bits semble être un choix raisonnable. Nous avons donc choisi de garder les algorithmes KLEIN, LILLIPUT et KTANTAN. L'algorithme LED a également été sélectionné même s'il utilise une taille de clé de 128 bits. En effet, la propriété très

2. `https://www.cryptolux.org/index.php/Lightweight_Block_Ciphers`

intéressante de sa fonction nommée *MixColumnsSerial* en fait un très bon candidat pour le projet SALWARE et nous voulions comparer l'implantation matérielle de cet algorithme avec les autres. Parmi les quatre algorithmes légers de chiffrement sélectionnés, KLEIN et LED sont des réseaux de substitutions et de permutations (SPN), LILLIPUT est un réseau de Feistel généralisé (GFN) et KTANTAN possède une structure particulière proche de celle utilisée par les algorithmes de chiffrement par flot. Le Tableau 5.1 donne les informations de base de ces quatre algorithmes : la date de publication, la structure et les tailles des messages et de la clé.

Tableau 5.1 – Présentation des algorithmes légers de chiffrement par bloc implantés dans ce chapitre.

Algorithme	**Année de publication**	**Structure**	**Taille de la clé**	**Taille du message**
KTANTAN [34]	2009	Bloc/Stream	80	64
LED [66]	2011	SPN	128	64
KLEIN [58]	2012	SPN	80	64
LILLIPUT [17]	2015	GFN	80	64

Dans la prochaine section, une description succincte des algorithmes KLEIN, LED et KTANTAN est présentée ainsi que leurs implantations matérielles parallèle et série. Les implantations de LILLIPUT ont déjà été présentées dans le Chapitre 4.

5.2 Description et implantation des algorithmes de chiffrement

5.2.1 KLEIN

KLEIN est un algorithme de chiffrement par bloc basé sur un réseau de substitution et de permutation. Il permet le chiffrement de messages de 64 bits grâce à une clé qui peut avoir une taille de 64, 80, ou 96 bits. La construction de la fonction de gestion de clé est proche d'un réseau de Feistel. L'algorithme a été publié en 2012 [58].

5.2.1.1 Description de l'algorithme

Afin de chiffrer ou de déchiffrer un message, le nombre de tours à effectuer dans la fonction de ronde dépend de la taille de la clé. Le nombre de tours est respectivement 12, 16, ou 20 pour une taille de clé de 64, 80 ou 96 bits. Chaque tour de la fonction de ronde est composé des quatre types d'opérations classiques, que l'on retrouve habituellement dans les réseaux de substitution et de permutation.

La fonction ***AddRoundKey*** additionne dans GF(2) la clé de ronde avec le message grâce à un *ou exclusif* appliqué bit à bit. Notons A cette fonction, $m =$

$m_0m_1m_2...m_{64}$ le message de 64 bits et $K = k_0k_1k_2...k_{64}$ la clé qui doit être ajoutée au message. L'opération *AddRoundKey* peut donc être définie par l'expression suivante :

$$A(M, K) = M \oplus K = (m_0 \oplus k_0)(m_1 \oplus k_1)...(m_{64} \oplus k_{64})$$

Le fonction ***SubNibbles*** correspond à l'étape de substitution. Elle est constituée d'un tableau de substitution qui prend un mot de quatre bits en entrée et le change en un autre mot de quatre bits. L'avantage du tableau de substitution de l'algorithme KLEIN (Tableau 5.2) est qu'il est involutif. Ainsi, le même tableau est utilisé pour le chiffrement et le déchiffrement. Notons S cette fonction de substitution et x un mot de quatre bits quelconque. La propriété d'involution s'écrit comme suit :

$$S(S(x)) = x$$

Tableau 5.2 – Tableau de substitution utilisé dans l'algorithme KLEIN.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

Cette propriété est extrêmement intéressante pour réduire la surface occupée par l'implantation matérielle de KLEIN car elle permet le partage d'une opération entre le chiffrement et le déchiffrement. Pour appliquer cette opération à l'ensemble du message m, ce dernier est divisé en quartets ($m = m_0m_1...m_{15}$) et chaque quartet est transformé grâce au tableau de substitution (Tableau 5.2). On peut donc définir l'opération *SubNibbles* par l'expression suivante :

$$S(M) = S(m_0)S(m_1)...S(m_{15})$$

La fonction ***RotateNibbles*** consiste en une rotation vers la gauche du message de deux octets par tour. Notons R cette opération et m le message. Ce dernier est divisé en seize quartets : $m = m_0m_1...m_{15}$. La fonction R est définie par l'expression suivante :

$$R(M) = m_4m_5...m_{15}m_0m_1m_2m_3$$

La fonction ***MixNibbles*** est exactement la même que l'opération de *MixColumn* de l'algorithme AES. Notons mix cette opération. Le message m est divisé en deux parties comprenant chacune 32 bits : $P_1 = m_0m_1...m_7$ et $P_2 = m_8m_9...m_{15}$. L'opération *MixNibbles* consiste en la multiplication de P_1 et de P_2 avec la matrice C dans $GF(2^4)$. Ainsi, l'opération *MixNibbles* de l'algorithme KLEIN est définie par :

$$mix(M) = (P_1 \times C)(P_2 \times C) \; avec$$

$$P_1 \times C = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} m_0m_1 \\ m_2m_3 \\ m_4m_5 \\ m_6m_7 \end{pmatrix} et \; P_2 \times C = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} m_8m_9 \\ m_{10}m_{11} \\ m_{12}m_{13} \\ m_{14}m_{15} \end{pmatrix}$$

Avec ces quatre opérations, il est possible de définir la fonction de ronde F par l'expression suivante :

$$F(M, K) = Mix(R(S(A(M, K))))$$

La fonction de gestion de clé permet de générer une clé de ronde pour chaque tour de la fonction transformant le message. Cette opération est basée sur une variante d'un réseau de Feistel. La première clé de ronde correspond aux 64 bits de poids fort de la clé principale. Cette clé est ensuite transformée en suivant les étapes décrites dans la Figure 5.1. Dans cette figure, la clé de ronde du tour i est notée rk^i. Cette clé de ronde est divisée en octets notés rk^i_j où $j \in [0; t]$ avec $t = 7, 9$ ou 11 pour les tailles de clé de 64, 80 ou 96 bits.

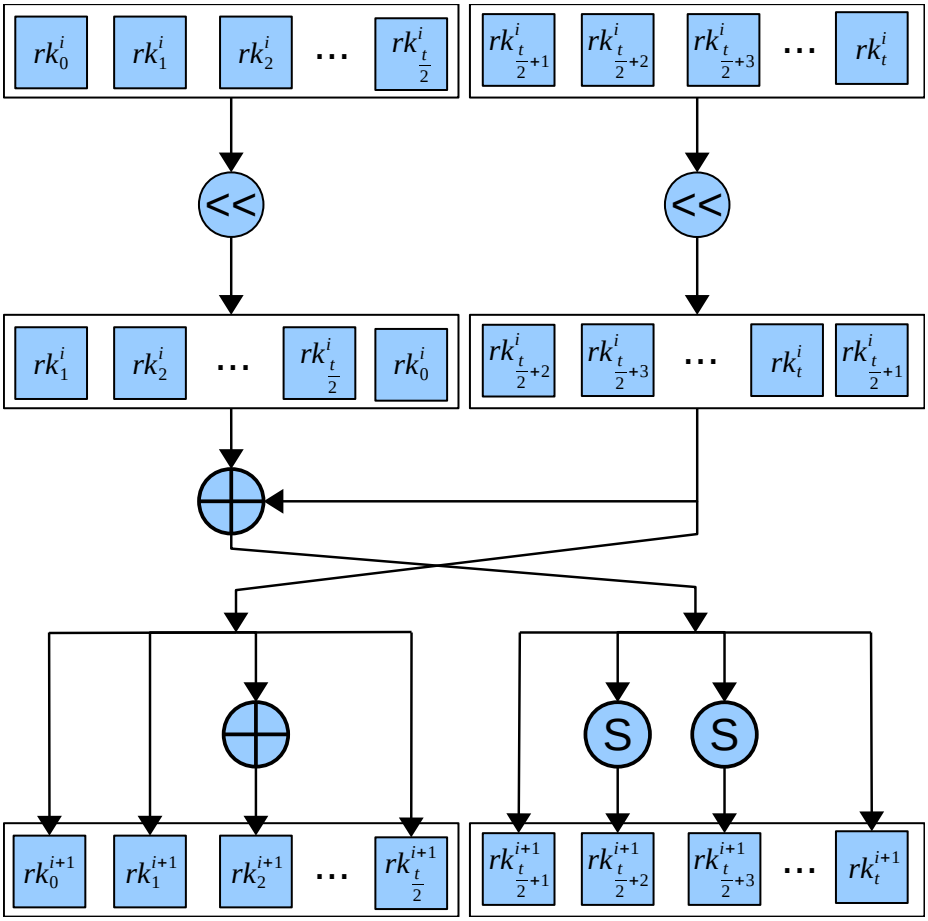

FIGURE 5.1 – Description générale de la fonction de gestion de clé de l'algorithme KLEIN.

Pour le déchiffrement d'un message, les opérations inverses des fonctions *RotateNibbles*, *MixNibbles* ainsi que l'inverse de la fonction de gestion de clé doivent être implantées. Plus de précisions sur toutes ces fonctions sont disponibles dans les spécifications de l'algorithme KLEIN [58].

À partir de la description de KLEIN qui vient d'être faite, il est possible de trouver trois inconvénients à cet algorithme. Le premier est l'utilisation de l'opération de *MixColumns* de l'AES car son inverse est très coûteux en termes de surface d'implantation, à moins d'utiliser des mémoires ainsi que des implantations de type *T-boxes* [41, 68]. Or, pour le projet SALWARE, l'utilisation de mémoire n'est pas envisagée à cause de leur coût important en surface de silicium. Le deuxième inconvénient de KLEIN réside dans sa fonction de gestion de clé. En effet, comme il ne s'agit pas d'un réseau de Feistel classique, il est impératif d'implanter la fonction inverse pour le déchiffrement. De plus, il faut également pré-calculer la dernière clé de ronde avant de commencer le déchiffrement d'un message. Enfin, on s'aperçoit que seule l'opération *SubNibbles* est conçue pour transformer des quartets et non des octets. L'algorithme de chiffrement KLEIN n'est donc pas optimisé pour un chemin de données sur 4 bits mais pour un chemin de données sur 8 bits.

Dans le but de comparer les implantations d'algorithme fonctionnant sur des taille de bloc et de clé similaire, seule la version de KLEIN utilisant une clé de 80 bits est implantée en suivant les deux stratégies (parallèle et série).

5.2.1.2 Implantation matérielle parallèle de KLEIN

Afin d'implanter l'algorithme KLEIN, l'opération *SubNibbles* peut être partagée entre le chiffrement et le déchiffrement grâce à sa propriété d'involution. Pour partager cette opération entre les deux modes de fonctionnement, la fonction de ronde est divisée en deux parties. Selon le mode de fonctionnement (chiffrement ou déchiffrement) la sortie de l'opération *AddRoundKey* ou de l'opération *RotateNibbles* est sélectionnée en entrée de l'étape de substitution. Sa sortie est envoyée à la deuxième partie de la fonction de ronde de chaque mode de fonctionnement.

Le registre contenant le message en cours de traitement est mis à jour en sélectionnant la sortie de la fonction de ronde correspondant au mode de fonctionnement utilisé (chiffrement ou déchiffrement). Pour le chiffrement de message, la première partie de la fonction de ronde contient l'opération *AddRoundKey* et la deuxième partie contient les opérations *RotateNibbles* et *MixNibbles*. Pour le déchiffrement, la première partie de la fonction de ronde contient les opérations *RotateNibbles* et *MixNibbles* alors que la deuxième partie ne contient que l'opération d'ajout de la clé.

Pour la fonction de gestion de clé, le schéma présenté sur la Figure 5.1 est directement implanté dans le matériel ainsi que son inverse pour une taille de clé de 80 bits. La Figure 5.2 montre le schéma bloc de cette implantation parallèle de l'algorithme de chiffrement KLEIN. Les deux modes de fonctionnement sont présentés sur cette figure.

5.2.1.3 Implantation matérielle série de KLEIN

Pour réussir à faire une implantation matérielle série de l'algorithme KLEIN, un chemin de données de 8 bits est choisi. Le registre du message est transformé en

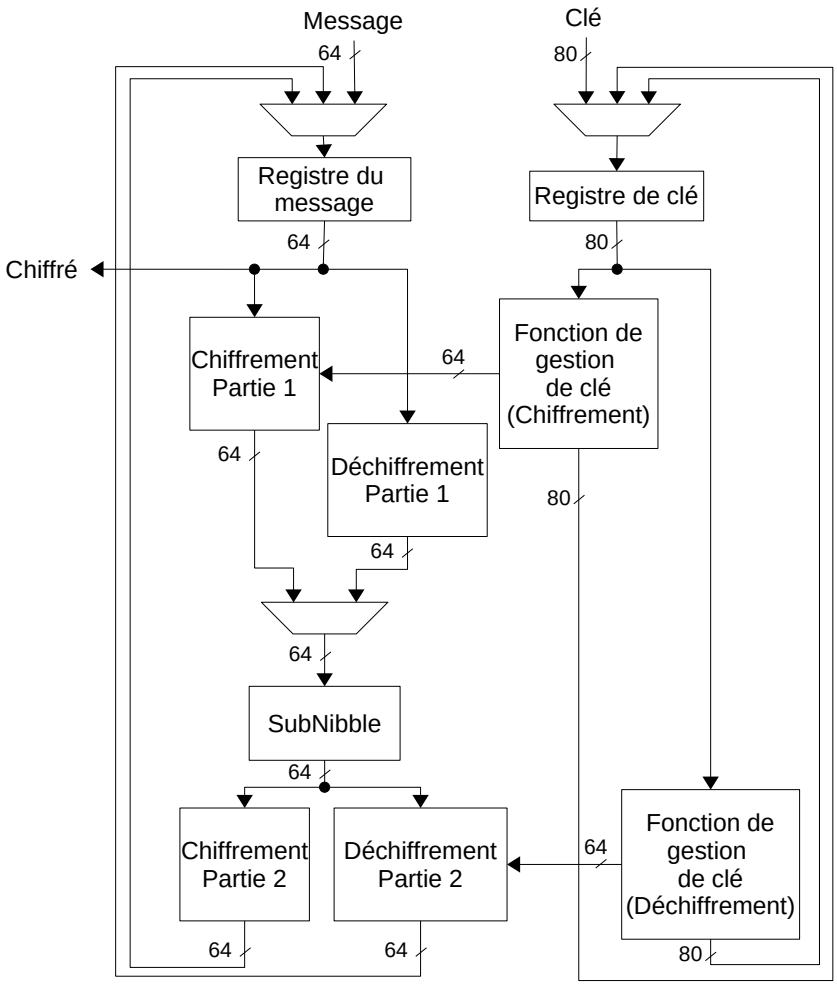

FIGURE 5.2 – Schéma de l'implantation parallèle de l'algorithme KLEIN.

un registre à décalage avec plusieurs choix de chargement. Pour le chiffrement, seul un octet du message est traité par les deux premières opérations de la fonction de ronde. L'opération *RotateNibbles* est inclue dans les choix de chargement du registre contenant le message. L'opération *MixNibbles* nécessite 32 bits afin de produire un octet en sortie. Il faut donc stocker temporairement les octets produits en sortie du *SubNibbles*. Une fois les 32 bits stockés, il est possible de générer 32 bits de sortie en appliquant une rotation d'un octet sur le registre temporaire et en utilisant l'opération *MixNibbles* quatre fois. Pour cela, l'opération *MixNibbles* est réduite à une seule ligne de la matrice C présentée dans la Sous-section 5.2.1. Ainsi, seize cycles d'horloge sont nécessaires pour achever un tour de la fonction de ronde contre un seul pour l'implantation parallèle. Les quatre premiers sont utilisés pour stocker le résultat des trois premières opérations pour les 32 premiers bits du message. Ensuite, l'opération *MixNibbles* est appliquée quatre fois afin de générer le résultat de la fonction de ronde des 32 premiers bits du message. Ces deux étapes sont

répétées deux fois afin de traiter l'ensemble des 64 bits du message.

Pour le déchiffrement, le même principe est appliqué et seize cycles d'horloge sont également nécessaires. Les quatre premiers cycles sont dédiés à l'opération *MixNibbles* inverse et les quatre suivants au reste de la fonction de ronde. Cet enchainement est appliqué deux fois pour finir un tour de la fonction de ronde sur l'ensemble des 64 bits du message.

La fonction de gestion de clé ne peut pas réellement être réduite en raison de sa structure et le même schéma d'implantation est donc utilisé dans l'implantation série de KLEIN. En revanche, il est nécessaire de conserver la clé de ronde d'un tour pendant les 16 cycles d'horloge que dure la fonction de ronde. Afin de fournir le bon octet de la clé de ronde à l'opération *AddRoundKey*, le registre de clé est également transformé en un registre à décalage et la clé est décalée d'un octet à chaque cycle d'horloge. Lors du dernier cycle de chaque tour, la clé de ronde du tour suivant est générée et remplace la clé précédente.

La Figure 5.3 montre le schéma d'implantation matérielle série de l'algorithme KLEIN. Dans le but de rester lisible, seul le chiffrement est présenté sur cette figure. De plus, toutes les connections ont une largeur de 8 bits sauf si une autre largeur est indiquée.

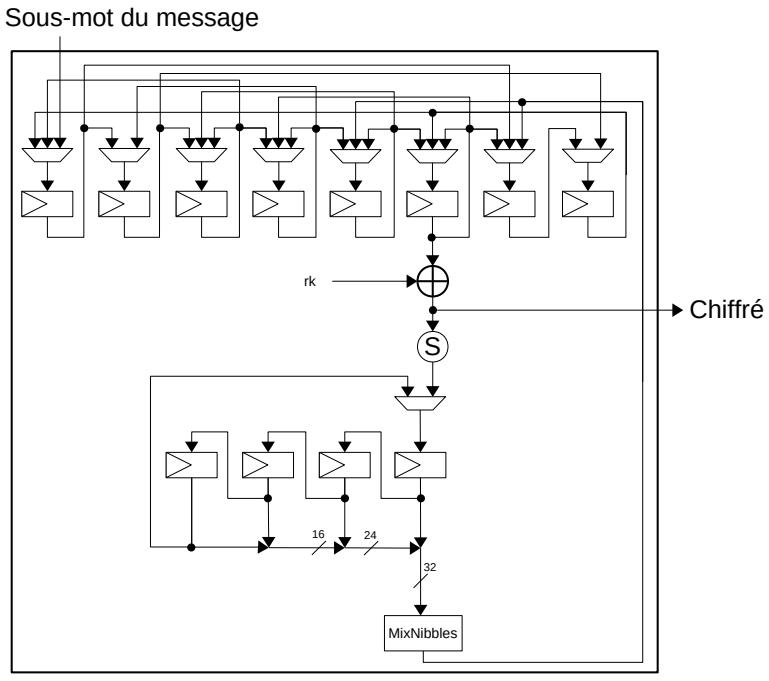

FIGURE 5.3 – Schéma du chiffrement de l'implantation matérielle *série* de l'algorithme KLEIN.

5.2.2 LED

LED est un algorithme léger de chiffrement par bloc dont l'architecture est basée sur un réseau de permutation et de substitution. La taille du message est de 64 bits et deux tailles de clé sont possibles : 64 et 128 bits. Dans ce chapitre, seule la version utilisant une clé de 128 bits est étudiée. Les spécifications de cet algorithme ont été publiées en 2011 [66].

5.2.2.1 Description de l'algorithme

Contrairement à d'autres réseaux de permutation et de substitution tels que AES ou KLEIN, LED est organisé en étapes, chacune comprenant quatre passages dans la fonction de ronde. La clé de ronde est ajoutée au bloc de données entre chaque étape et non entre chaque passage dans la fonction de ronde. Cette légère différence implique une augmentation du nombre total de passages à effectuer dans cette fonction. En effet, pour chiffrer (ou déchiffrer) un message avec une clé de 128 bits, il faut 12 étapes, ce qui correspond à 48 passages dans la fonction de ronde. La Figure 5.4 présente le chiffrement d'un message avec cet algorithme.

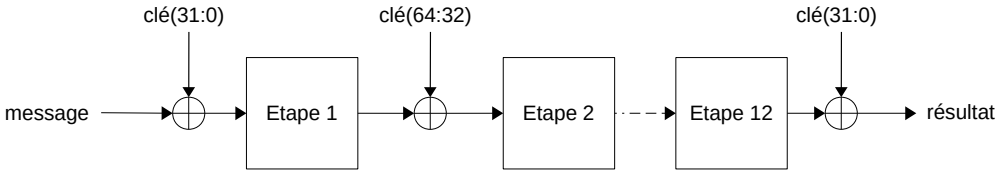

FIGURE 5.4 – Schéma du chiffrement d'un message avec l'algorithme LED.

Comme on peut le voir sur la Figure 5.4, la gestion des clés de ronde est très simple dans LED. En effet, un simple rotation de 64 bits est effectuée sur la clé envoyée au système. Ainsi, pour les étapes impaires, la moitié de poids faible de la clé est ajoutée au bloc de données et pour les étapes paires, c'est la moitié de poids fort de la clé qui est ajoutée. De plus, on peut également remarquer que l'ordre dans lequel les deux parties de la clé sont utilisées est le même pour le chiffrement et pour le déchiffrement d'un message.

La fonction de ronde utilisée dans les étapes est composée de quatre fonctions nommées : *AddConstants*, *SubCells*, *ShiftRows* et *MixColumnsSerial*. Pour représenter ces différentes fonctions correctement, le bloc de données est divisé en 16 quartets arrangés dans une matrice 4×4. Ainsi, le message X est représenté par la forme suivante :

$$X = X_0 X_1 \cdots X_{15} = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix}$$

AddConstants est la première opération de la fonction de ronde. Elle consiste à ajouter une constante au bloc de données. Celle-ci est dépendante du tour en cours. Cet ajout est effectué grâce à un *ou exclusif*. Premièrement, notons ks la taille de la clé utilisée par l'algorithme, codé sur 8 bits : $ks = ks_7 ks_6 \cdots ks_0$. Ensuite, une autre constante est générée grâce à un registre à décalage de 6 bits, présenté dans la Figure 5.5.

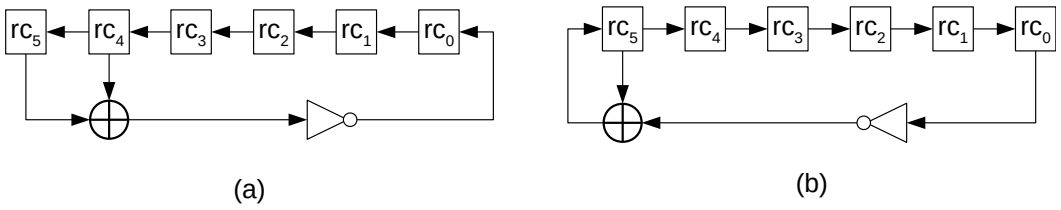

FIGURE 5.5 – Calcul des constantes utilisées dans la fonction *AddConstants* de l'algorithme LED, pour le chiffrement (a) et pour le déchiffrement (b).

La Figure 5.5.a présente l'opération à effectuer pour calculer les constantes utilisées pour le chiffrement d'un message. La valeur de ce registre est initialisée à 0 pour le chiffrement et la première constante est calculée avant d'effectuer le premier passage dans la fonction de ronde. Pour le déchiffrement, il faut remonter les valeurs des constantes. Cela est possible en décalant le registre dans l'autre direction et en appliquant la fonction logique présentée dans la Figure 5.5.b. Dans ce cas, le registre est initialisé à la valeur hexadécimale 09.

La fonction *AddConstants* consiste alors à effectuer l'opération suivante sur le message :

$$AddConstants(X) = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_4 & X_5 & X_6 & X_7 \\ X_8 & X_9 & X_{10} & X_{11} \\ X_{12} & X_{13} & X_{14} & X_{15} \end{pmatrix} \oplus \begin{pmatrix} ks_7 ks_6 ks_5 ks_4 & rc_5 rc_4 rc_3 & 0 & 0 \\ 1 \oplus ks_7 ks_6 ks_5 ks_4 & rc_2 rc_1 rc_0 & 0 & 0 \\ 2 \oplus ks_3 ks_2 ks_1 ks_0 & rc_5 rc_4 rc_3 & 0 & 0 \\ 3 \oplus ks_3 ks_2 ks_1 ks_0 & rc_2 rc_1 rc_0 & 0 & 0 \end{pmatrix}$$

SubCells consiste à appliquer un tableau de substitution sur le message. Il est présenté dans le Tableau 5.3, il prend en entrée un quartet et renvoie un autre quartet en sortie. Ainsi, il faut appliquer le tableau sur chaque quartet du message : $SubCells(X) = S(X_0)S(X_1)\cdots S(X_{15})$

Tableau 5.3 – Tableau de substitution utilisé dans l'algorithme KLEIN.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2
$S^{-1}(x)$	5	E	F	8	C	1	2	D	B	4	6	3	0	7	9	A

Pour le déchiffrement d'un message, le tableau de substitution inverse (présenté dans le Tableau 5.3) doit être utilisé.

ShiftRows est la troisième opération de la fonction de ronde. Elle consiste à effectuer une rotation sur chaque ligne de la matrice du message. La rotation se fait de i quartets vers les bits de poids fort où i représente le numéro de la ligne dans la matrice m. On peut donc définir cette fonction comme suit :

$$ShiftRows(X) = \begin{pmatrix} X_0 & X_1 & X_2 & X_3 \\ X_5 & X_6 & X_7 & X_4 \\ X_{10} & X_{11} & X_8 & X_9 \\ X_{15} & X_{12} & X_{13} & X_{14} \end{pmatrix}$$

MixColumnsSerial est la dernière partie de la fonction de ronde. Cette fonction constitue la réelle force de l'algorithme LED pour une implantation légère en surface. En effet, elle consiste à multiplier le message X par une matrice (notée m) qui possède une forme réduite (notée A). Pour appliquer cette fonction sur X, il est possible d'appliquer la matrice réduite A quatre fois. L'avantage est que la matrice A nécessite très peu d'opérations logiques pour être implantée. On peut donc définir la fonction *MixColumnsSerial* de la manière suivante :

$$MixColumnsSerial(X) = M \times X = (A^4) \times X, \text{ avec}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 4 & 1 & 2 & 2 \end{pmatrix}, \text{ et } M = \begin{pmatrix} 4 & 1 & 2 & 2 \\ 8 & 6 & 5 & 6 \\ B & E & A & 9 \\ 2 & 2 & F & B \end{pmatrix}$$

De plus, pour le déchiffrement, il est possible d'inverser la matrice A et de l'appliquer quatre fois pour obtenir la matrice inverse de m.

Ainsi, la fonction de ronde de l'algorithme LED consiste à appliquer les quatre fonctions présentées précédemment sur le message X. Elle peut donc être définie par une fonction *Round* de la manière suivante :

$$Round(X) = MixColumnsSerial(ShiftRows(SubCells(AddConstants(X))))$$

Pour le déchiffrement, il faut inverser l'ensemble des fonctions et les appliquer dans l'ordre inverse. Pour plus de détails concernant LED, le lecteur intéressé est invité à se référer aux spécifications de l'algorithme [66].

5.2.2.2 Implantation matérielle parallèle de LED

L'implantation matérielle parallèle de LED est relativement simple à mettre en œuvre du fait des opérations utilisées. La fonction de gestion de clé est partagée entre le chiffrement et le déchiffrement des messages ainsi que l'ajout de clé, qui a lieu entre deux étapes. Toutes les autres fonctions doivent être implantées séparément pour le chiffrement et le déchiffrement.

Le tableau de substitution est implanté seize fois en parallèle pour être capable de mettre à jour l'intégralité du message en un seul cycle d'horloge. C'est également pour cette raison que la matrice réduite A est implantée quatre fois pour l'implantation parallèle de LED. Dans la définition de la matrice A, on remarque que seules deux multiplications sont nécessaires : la multiplication par 2 et la multiplication par 4. Ces deux opérations sont très simples à implanter et nécessitent très peu de ressources logiques, comme le montre la Figure 5.6.

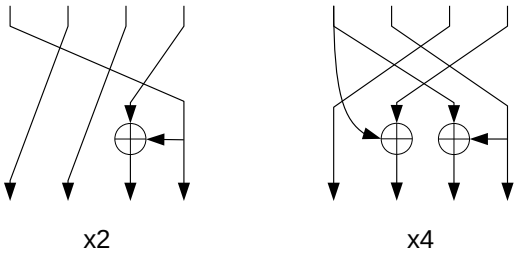

FIGURE 5.6 – Implantation matérielle des multiplications par 2 et par 4 nécessaires dans l'algorithme LED.

Pour le déchiffrement, la matrice réduite inverse A^{-1} est implantée quatre fois et le tableau de substitution inverse seize fois, toujours dans le but de mettre à jour tout le message en un cycle d'horloge. La Figure 5.7 présente l'implantation parallèle de l'algorithme de chiffrement LED, pour le chiffrement et le déchiffrement.

5.2.2.3 Implantation matérielle série de LED

Pour l'implantation série de LED, les tableaux de substitution sont implantés une seule fois, ce qui permet de prendre en entrée un seul quartet. De plus, il est également possible d'appliquer la fonction *MixColumnsSerial* pour produire 4 bits en sortie, même si elle nécessite 16 bits en entrée dans tous les cas. Pour la fonction *ShiftRows*, aucune logique n'est ajoutée et cette fonction est incluse dans les choix des multiplexeurs présents devant le registre de données. Ainsi, il est possible d'utiliser un chemin de données de 4 bits pour l'implantation série de LED.

La seule fonction légèrement plus difficile à adapter est la fonction *AddConstants*. En effet, il faut s'assurer que chaque quartet du message est mis à jour avec la bonne constante, ce qui entraîne une complexification de la logique de contrôle pour cette fonction.

Avec l'implantation série de LED, chaque passage dans la fonction de ronde requière 32 cycles d'horloge. Dans le cas du chiffrement, les 16 premiers sont utilisés pour appliquer les trois premières fonctions (*AddConstants*, *SubCells* et *ShiftRows*) et les 16 derniers cycles d'horloge servent à appliquer la fonction *MixColumnsSerial* sur le message. La clé est ajoutée uniquement une fois que quatre passages dans la fonction de ronde sont effectués.

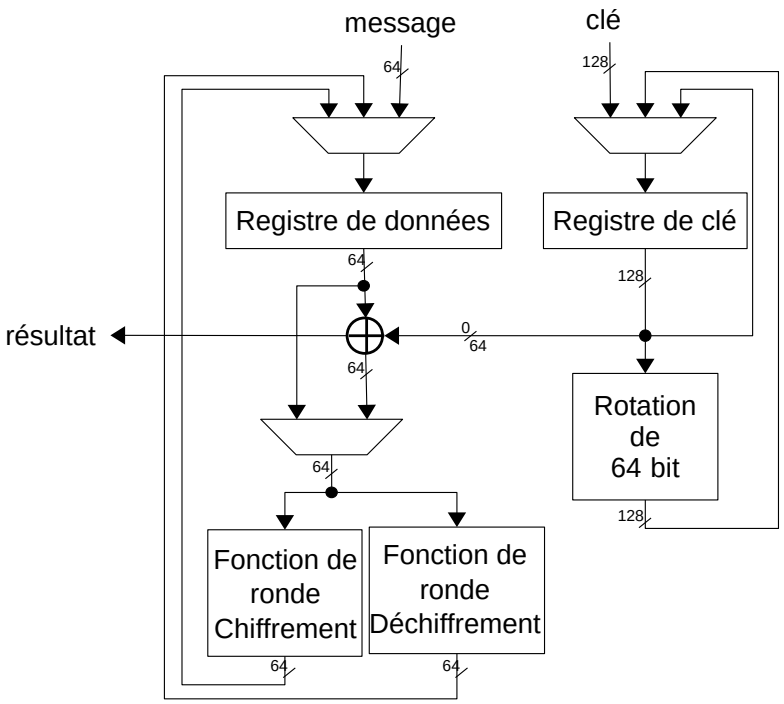

FIGURE 5.7 – Implantation matérielle parallèle de LED.

Pour le déchiffrement, les 16 premiers cycles d'horloge sont utilisés pour la fonction *MixColumnsSerial* inverse et les 16 derniers cycles pour les autres fonctions. La Figure 5.8 présente l'implantation matérielle série de LED. Dans un souci de lisibilité, seul le chiffrement y est présenté. De plus, toutes les connections de la Figure 5.8 ont une largeur de 4 bits, sauf si une autre taille est indiquée.

Finalement, toutes les parties de la fonction de ronde utilisent moins de ressources dans l'implantation série de LED que l'implantation parallèle. Cela est dû au choix fait dans les différentes opérations utilisées dans LED. Toutefois, comme on peut le voir sur la Figure 5.8, le nombre d'entrées des multiplexeurs présents devant le registre de données a augmenté. Cela pourrait impliquer une augmentation de la surface utilisée comme pour l'algorithme LILLIPUT.

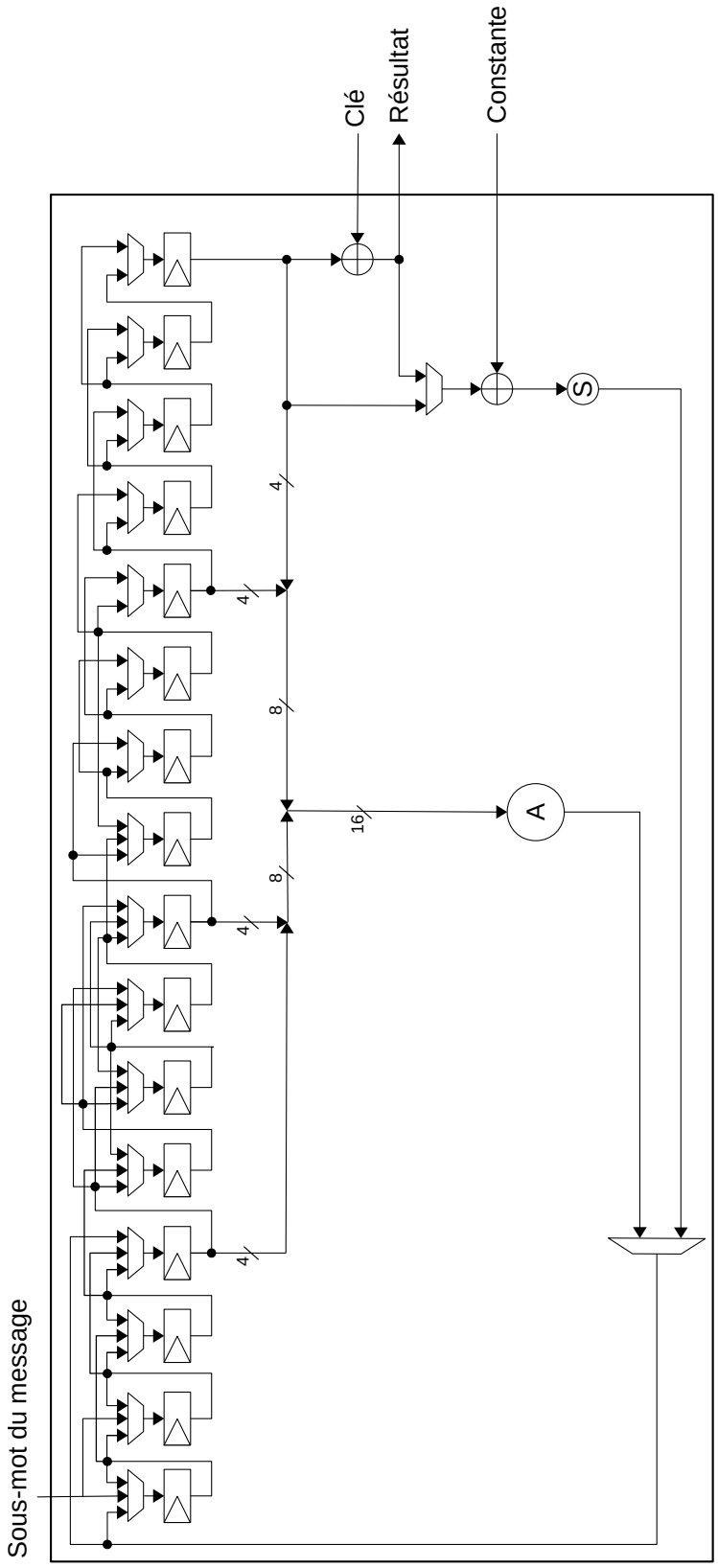

FIGURE 5.8 – Implantation matérielle série de LED.

5.2.3 KTANTAN

Le dernier algorithme retenu comme candidat potentiel pour le projet SALWARE est KTANTAN. Cet algorithme léger de chiffrement par bloc est très différent des autres. En effet, sa structure est très proche de celle utilisée dans les algorithmes de chiffrement par flot. Cependant, il ne peut chiffrer ou déchiffrer qu'un message de taille fixe, c'est donc bien un algorithme de chiffrement par bloc. La clé utilisée dans KTANTAN est de taille fixe : 80 bits. Pour le message, trois tailles sont possibles : 32, 48 ou 64 bits. Dans ce chapitre, seule la version de KTANTAN utilisant une taille de 64 bits est décrite. Les spécifications de cet algorithme ont été publiées en 2009 [34].

5.2.3.1 Description de l'algorithme

KTANTAN est construit autour de registres à décalage et ne possède que deux fonctions logiques très simples notées f_a et f_b. Cette construction particulière nécessite un grand nombre de cycles d'horloge pour chiffrer un message. En effet, 254 passages dans les fonctions f_a et f_b sont nécessaires, ce qui constitue un inconvénient pour cet algorithme. En revanche, la simplicité des fonctions font de KTANTAN un très bon candidat pour le projet SALWARE puisque seule la surface est prise en compte.

Le message de 64 bits $m = m_{63}m_{62}\cdots m_0$ est séparé dans deux registres à décalage ($\mathcal{L}_1$ et $\mathcal{L}_2$) de la manière suivante :

$$\mathcal{L}_1 = m_{63}m_{62}\cdots m_{39} \text{ et } \mathcal{L}_2 = m_{38}m_{37}\cdots m_0$$

Un tour de la fonction de ronde consiste alors à appliquer la fonction f_a sur le registre $\mathcal{L}_1$, de sorte à mettre à jour trois bits du registre $\mathcal{L}_2$. De même, lors d'un tour, la fonction f_b est appliquée sur $\mathcal{L}_2$ pour mettre à jour trois bits du registre $\mathcal{L}_1$. Comme les deux fonctions f_a et f_b ont été conçues pour renvoyer un seul bit, il faut donc les appliquer trois fois lors de chaque tour. Le fonction f_a est définie par l'expression 5.1 et la fonction f_b par l'expression 5.2.

$$f_a(\mathcal{L}_1) = \mathcal{L}_1(24) \oplus \mathcal{L}_1(15) \oplus (\mathcal{L}_1(20) \cdot \mathcal{L}_1(11)) \oplus (\mathcal{L}_1(9) \cdot T(7)) \oplus k_a \quad (5.1)$$

$$f_b(\mathcal{L}_2) = \mathcal{L}_2(38) \oplus \mathcal{L}_2(25) \oplus (\mathcal{L}_2(33) \cdot \mathcal{L}_2(21)) \oplus (\mathcal{L}_2(14) \cdot \mathcal{L}_2(9)) \oplus k_b \quad (5.2)$$

Dans les deux expressions 5.1 et 5.2, on peut voir trois éléments que nous n'avons pas encore défini : $T(7)$, k_a et k_b. Les deux derniers représentent la clé de ronde et $T(7)$ représente le bit de poids fort d'un LFSR (T), utilisé pour compter le nombre de tours effectués. Ce registre à décalage a la particularité d'avoir un cycle de 255 tours. Cela signifie qu'après 255 cycles d'horloge, la valeur contenue dans le registre est identique à la valeur de départ. Pour le déchiffrement, il faut inverser la fonction logique effectuée dans la rétroaction du registre et le décaler dans le sens opposé. La Figure 5.9.a présente ce LFSR pour le chiffrement et la Figure 5.9.b pour le déchiffrement.

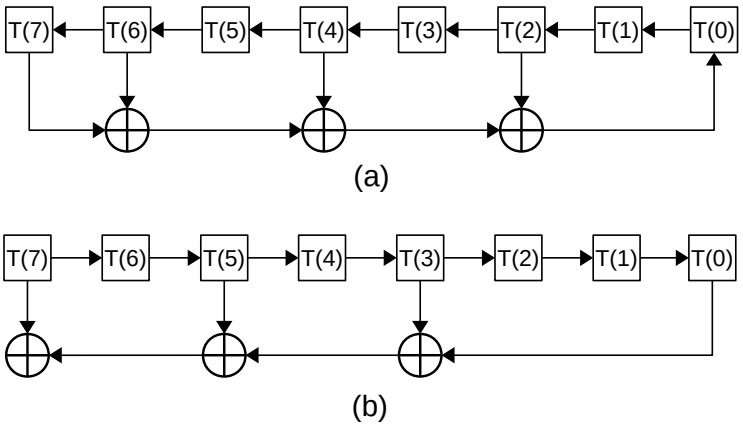

FIGURE 5.9 – LFSR T utilisé pour le décompte du nombre de tours effectués dans l'algorithme KTANTAN.

Pour générer les clés de ronde, la clé entrée avec le message est fixée et seuls certains bits de cette clé sont sélectionnés, puis une équation logique est appliquée entre ces bits et le registre T. Notons la clé de chiffrement $K = w_4w_3w_2w_1w_0$ où les $w_i,\ i \in [0,4]$ représentent un mot de 16 bits de la clé. De plus, notons $a_i,\ i \in [0,4]$ les bits définis par l'Expression 5.3.

$$a_i = MUX16to1(w_i, T(7)T(6)T(5)T(4)) \tag{5.3}$$

Dans l'Expression 5.3, $MUX16to1$ représente un multiplexeur sélectionnant 1 bit parmi 16. Avec ces notations, il est possible de définir les bits k_a et k_b représentant les clés de ronde utilisées dans les fonctions f_a et f_b :

$$k_a = \overline{T(3)} \cdot \overline{T(2)} \cdot a_0 \oplus (T(3) + T(2)) \cdot MUX4to1(a_4a_3a_2a_1, T(1)T(0)) \tag{5.4}$$

$$k_b = \overline{T(3)} \cdot T(2) \cdot a_4 \oplus (T(3) + \overline{T(2)}) \cdot MUX4to1(a_3a_2a_1a_0, \overline{T(1)T(0)}) \tag{5.5}$$

Dans les Expressions 5.4 et 5.5, les opérations logiques *et, ou, non* et *xor* sont représentées respectivement par les symboles $\cdot$, $+$, $\overline{x}$ et $\oplus$. De plus, l'expression $MUX4to1$ représente un multiplexeur sélectionnant 1 bit parmi 4.

Pour le déchiffrement d'un message, il n'est pas nécessaire d'implanter la fonction de gestion de clé inverse. En effet, si les bits utilisés de la clé sont sélectionnés dans l'ordre inverse tout au long du déchiffrement, les clés de ronde seront générées dans le bon ordre. Cela est possible simplement par l'inversion du registre à décalage T. En revanche, il est indispensable d'implanter les fonctions de ronde inverses f_a^{-1} et f_b^{-1}. Plus de détails concernant cet algorithme léger de chiffrement par bloc sont disponibles dans [34].

5.2.3.2 Implantation matérielle parallèle de KTANTAN

L'implantation matérielle parallèle de KTANTAN ne suit pas le schéma classique de l'implantation d'un algorithme de chiffrement par bloc (Figure 4.3 dans

le Chapitre 4). En effet, le message est séparé dans deux registres à décalage de tailles différentes et seuls trois bits de ces registres sont mis à jour à chaque passage dans les fonctions de ronde. Comme ces fonctions (f_a et f_b) revoient un seul bit en sortie, elles sont simplement implantées trois fois en parallèle pour l'implantation parallèle de KTANTAN. La Figure 5.10 présente le chemin de données du chiffrement de l'implantation matérielle parallèle de KTANTAN. La fonction de gestion de clé n'est pas représentée dans cette figure puisqu'il ne s'agit que d'une sélection de certains bits de la clé.

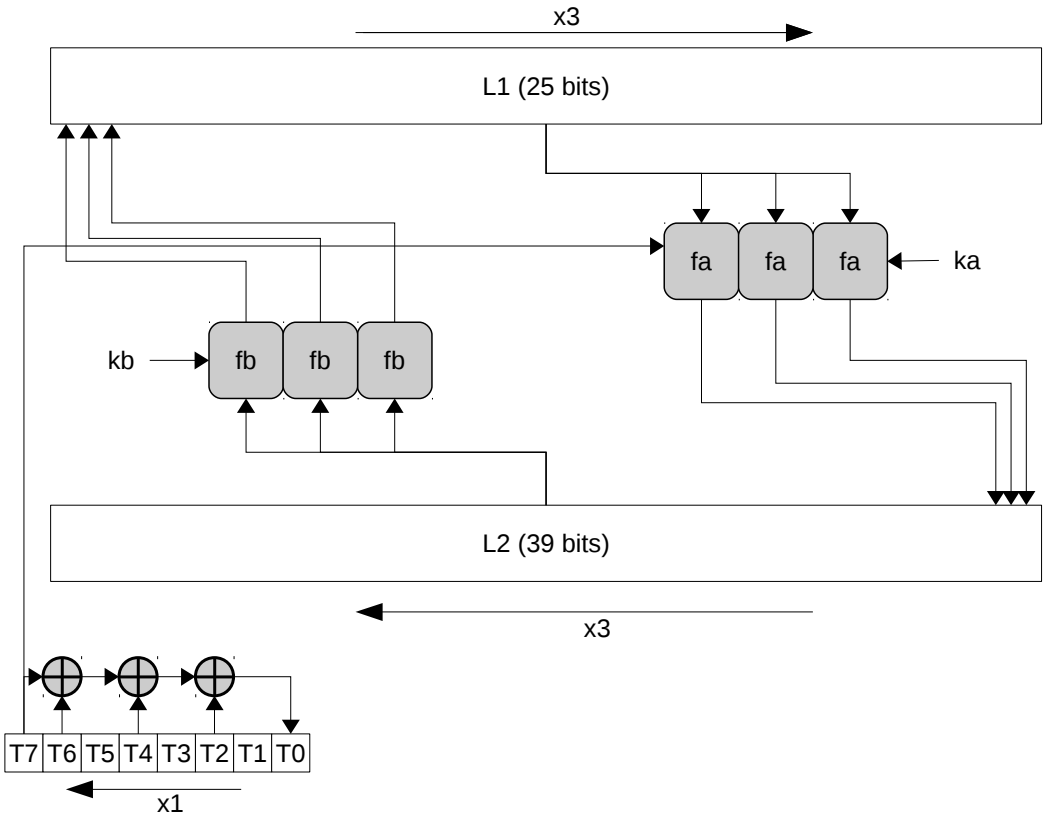

FIGURE 5.10 – Implantation matérielle parallèle de KTANTAN.

Comme les fonctions f_a et f_b sont extrêmement simples, elles ne peuvent pas être réduites (en termes de surface). De plus, il est nécessaire d'implanter leurs inverses pour effectuer le déchiffrement d'un message.

5.2.3.3 Implantation matérielle série de KTANTAN

L'implantation série de KTANTAN est très simple à effectuer une fois l'implantation parallèle réalisée. En effet, la simplicité de cet algorithme fait que l'implantation série consiste à ne mettre à jour qu'un seul bit par cycle d'horloge. Pour cela,

les fonctions f_a et f_b sont implantées une seule fois et les registres à décalage de données sont décalés d'un seul bit par cycle. Il faut cependant conserver la valeur du registre T pendant trois cycles d'horloge car les trois bits mis à jour par passage dans les fonctions de ronde doivent utiliser la même valeur de T. Pour cela, un compteur synchrone de deux bits est rajouté ainsi que quelques multiplexeurs devant ce registre T.

Les ajouts nécessaires pour l'implantation série de KTANTAN vont diminuer le gain en surface de cette implantation mais elle devrait quand même être légèrement plus petite que l'implantation parallèle.

5.3 Résultats et analyse des implantations matérielles

Dans le but de comparer les différentes implantations matérielles d'algorithmes légers de chiffrement par bloc qui ont été sélectionnés, toutes les implantations ont été effectuées dans une plateforme d'implantation matérielle commune. Cette plateforme d'implantation a été spécialement développée pour permettre l'implantation de n'importe quel algorithme de chiffrement avec le minimum de changement possible.

Pour cela, l'unité de contrôle et le chemin de données sont complètement séparés dans le matériel et seuls quelques signaux de contrôle sont transmis entre les deux blocs. Cette séparation permet notamment d'utiliser le même flot d'opération pour chiffrer ou déchiffrer un message, quelque soit l'algorithme implanté. La Figure 5.11 présente le schéma de cette plateforme générique d'implantation d'algorithme de chiffrement.

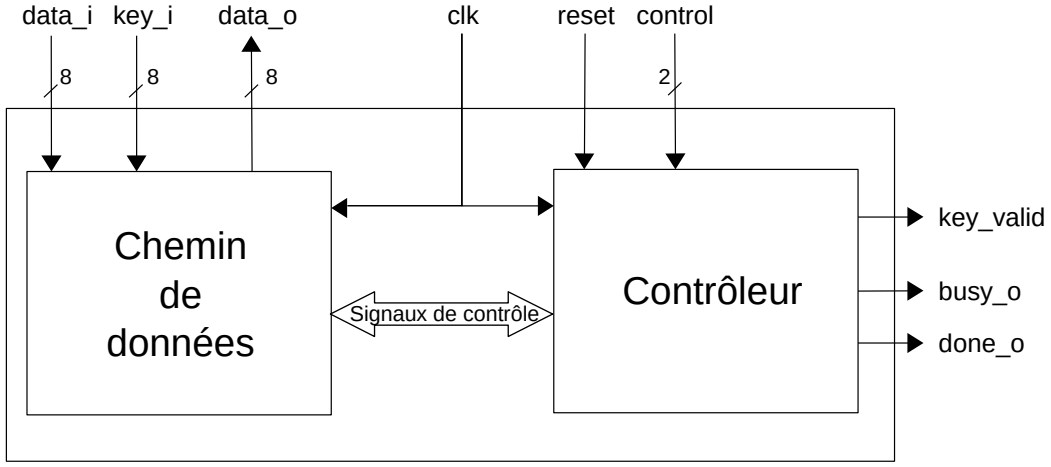

FIGURE 5.11 – Schéma de la plateforme commune utilisée pour l'implantation des algorithmes de chiffrement.

Pour le bloc *Chemin de données*, le message et la clé sont envoyés séparément grâce à deux bus de 8 bits différents (*data_i* et *key_i* sur la Figure 5.11). Le résultat de l'opération demandée est retourné sur le bus de 8 bits nommé *data_o*.

Pour le contrôleur, une entrée de 2 bits nommée *control* permet de choisir l'opération voulue (charger une clé, effectuer un chiffrement ou effectuer un déchiffrement). De plus, trois signaux de sortie permettent de connaître l'état du système.

Le choix d'utiliser des bus de 8 bits en entrée et en sortie de cette plateforme vient de la taille du chemin de données la plus grande parmi les implantations séries des quatre algorithmes de chiffrement. Cette taille est donc celle du chemin de données de KLEIN qui est de 8 bits. Il faut noter que réduire cette interface de communication à 4 bits pour les algorithmes utilisant cette taille pour leurs implantations séries (LILLIPUT et LED) pourrait diminuer légèrement la surface des implantations, mais cela rendrait les comparaisons non équitables.

5.3.1 Résultats des implantations matérielles parallèles

Comme dans le Chapitre 4, les résultats de la surface d'implantation des algorithmes sont exprimés en LUT et en registres. Tous les résultats présentés dans cette section ont été générés sur un FPGA Xilinx Spartan 6. De plus, toutes les implantations ont été testées par des simulations faites après l'étape de placement et de routage de l'outil de synthèse Xilinx, il s'agit donc des résultats après implantation et non d'estimations de la surface utilisée. Enfin, toutes les implantations permettent à la fois de chiffrer et de déchiffrer des messages.

Le Tableau 5.4 présente les résultats des trois algorithmes présentés dans ce chapitre. Pour LILLIPUT, la meilleure implantation a été sélectionnée à partir du Chapitre 4. Dans chaque cas, les tailles du chemin de données, du contrôleur et de l'implantation complète sont présentées dans le Tableau 5.4.

Tableau 5.4 – Résultats des implantations matérielles parallèles pour les algorithmes de chiffrement KLEIN, LED, LILLIPUT et KTANTAN sur FPGA Xilinx Spartan 6.

		LUT6	Registres	LUT6 + Registres
KLEIN	Contrôleur	24	13	37
	Chemin de données	604	144	748
	Implantation complète	**632**	**157**	**789**
LED	Contrôleur	19	13	32
	Chemin de données	515	198	713
	Implantation complète	**549**	**211**	**760**
LILLIPUT	Contrôleur	24	13	37
	Chemin de données	229	144	373
	Implantation complète	**287**	**157**	**444**
KTANTAN	Contrôleur	17	7	24
	Chemin de données	143	144	287
	Implantation complète	**150**	**151**	**301**

Comme on peut le voir dans le Tableau 5.4, KTANTAN est l'algorithme dont

l'implantation matérielle parallèle occupe le moins de surface avec seulement 301 LUT6 plus registres. Cela est clairement due à la simplicité de sa structure et des fonctions utilisées dans cet algorithme. LILLIPUT est le deuxième algorithme dont la surface d'implantation est la plus petite avec 444 LUT6 plus registres. Cela peut être expliqué par le partage de deux parties parmi les trois de la fonction de ronde entre le chiffrement et le déchiffrement.

Enfin, les deux autres algorithmes, KLEIN et LED, sont légèrement plus volumineux en termes de surface. Toutefois, on peut remarquer que LED nécessite plus de registres à cause de la taille de la clé. Cela indique également que la logique utilisée dans LED occupe une surface plus petite que celle utilisée dans KLEIN.

5.3.2 Résultats des implantations matérielles séries

Les résultats des implantations séries pour les quatre algorithmes sont présentés dans le Tableau 5.5 exactement de la même manière que les résultats présentés dans le Tableau 5.4.

Tableau 5.5 – Résultats des implantations matérielles séries pour les algorithmes de chiffrement KLEIN, LED, LILLIPUT et KTANTAN sur FPGA Xilinx Spartan 6.

		LUT6	Registres	LUT6 + Registres
KLEIN	Contrôleur	35	18	53
	Chemin de données	304	176	480
	Implantation complète	**330**	**194**	**524**
LED	Contrôleur	32	19	51
	Chemin de données	319	198	517
	Implantation complète	**373**	**217**	**590**
LILLIPUT	Contrôleur	35	16	51
	Chemin de données	235	180	415
	Implantation complète	**302**	**200**	**502**
KTANTAN	Contrôleur	17	9	26
	Chemin de données	137	144	281
	Implantation complète	**137**	**153**	**290**

Dans le Tableau 5.5, on peut remarquer que KTANTAN reste l'algorithme qui occupe le moins de surface, viennent ensuite LILLIPUT, KLEIN et LED. Alors que le nombre de registres utilisés par chaque algorithme était très proche pour les implantations parallèles, il est complètement différent pour les implantations séries. Cela s'explique assez facilement par les choix effectués pour mettre en place les implantations séries. En effet, chaque algorithme a été profondément étudié dans le but de réduire la surface d'implantation de chaque partie de ces algorithmes.

On peut également remarquer que le contrôleur du système occupe plus de surface pour les implantations séries. Cela est dû aux compteurs ajoutés pour s'assurer que les bonnes valeurs du message et de la clé arrivent au bon moment à l'entrée de

chaque fonction des algorithmes. Enfin, seul LILLIPUT voit sa surface d'implantation augmenter entre l'implantation parallèle et l'implantation série et tous les autres algorithmes possèdent une implantation série plus petite en surface que leurs implantations parallèles.

5.3.3 Comparaison des résultats d'implantations matérielles parallèles et séries

Pour effectuer la comparaison des implantations matérielles parallèles et séries obtenus pour les quatre algorithmes, le Tableau 5.6 reprend les résultats présentés dans les sections précédentes, uniquement pour le système complet (comprenant le contrôleur et le chemin de données).

Tableau 5.6 – Comparaison des implantations séries et parallèles sur FPGA Xilinx Spartan 6.

		LUT6	Registres	LUT6 + Registres
KLEIN	fullwidth	632	157	789
	serial	330	194	**524**
LED	fullwidth	549	211	760
	serial	373	217	**590**
LILLIPUT	fullwidth	287	157	**444**
	serial	302	200	502
KTANTAN	fullwidth	150	151	301
	serial	137	153	**290**

L'algorithme dont la surface a été la plus réduite par l'implantation série est KLEIN avec une réduction de 33% de la surface occupée. En revanche, la surface d'implantation de LILLIPUT a augmenté d'environ 13%, ce qui s'explique par l'ajout de registres et de multiplexeurs entre l'implantation parallèle et l'implantation série. Enfin, même si la réduction observée pour KTANTAN est la plus petite, seulement 4%, cet algorithme reste le plus léger en termes de surface.

Dans le but de compléter l'étude comparative présentée dans ce chapitre, les résultats d'implantation des quatre algorithmes ont été générés pour un FPGA Xilinx Spartan 3. Cela permet de comparer les implantations effectuées avec celles présentées dans d'autres travaux. Malheureusement, très peu d'études donnent les résultats d'implantation d'algorithmes légers de chiffrement par bloc sur FPGA et seuls [10] et [6] proposent des implantations de l'algorithme LED sur FPGA. Le Tableau 5.7 présente les résultats des implantations décrites dans ce chapitre sur FPGA Xilinx Spartan 3 et les compare avec les résultats présentés dans [6, 10].

Tableau 5.7 – Comparaison des implantations séries et parallèles sur FPGA Xilinx Spartan 3 et avec les travaux présentés dans [6, 10].

Algorithme	Taille de la clé	Mode	Stratégie d'implantation	LUT4	Registres	Mémoire BRAM	LUT4 + Registres
KLEIN	80	Chiffrement/ Déchiffrement	Parallèle	1097	162	0	1259
			Série	633	194	0	**827**
LED	128	Chiffrement/ Déchiffrement	Parallèle	970	211	0	1181
			Série	555	218	0	**773**
LED [10]	128	Chiffrement/ Déchiffrement	Parallèle	319	194	**16**	513 [1]
LED [6]	128	**Chiffrement**	Série	302	216	0	518 [2]
LILLIPUT	80	Chiffrement/ Déchiffrement	Parallèle	512	162	0	**674**
			Série	558	205	0	763
KTANTAN	80	Chiffrement/ Déchiffrement	Parallèle	256	151	0	407
			Série	222	153	0	**375**

1. L'impact des mémoires BRAM n'a pas été pris en compte dans le calcul de la surface d'implantation.

2. Seul le chiffrement de messages est possible avec cette implantation.

La première chose que l'on peut remarquer dans le Tableau 5.7 est que la comparaison des résultats d'implantation présentés dans ce manuscrit de thèse avec ceux présentés dans [6, 10] n'est pas aisée. En effet, dans [10], l'implantation proposée de LED fonctionne pour le chiffrement et le déchiffrement mais utilise des ressources spécifiques au FPGA utilisé : les mémoires BRAM. De plus, ces éléments n'ont pas été pris en compte dans le calcul de la surface occupée par l'algorithme et c'est pour cela que l'implantation proposée dans [10] occupe une surface apparente plus petite que l'implantation que nous proposons. Dans [6], l'implantation qui est décrite permet uniquement le chiffrement de messages. De plus, il est clairement stipulé dans ces deux articles que les implantations proposées ont pour objectif de réduire la surface mais aussi de garder une vitesse de chiffrement rapide. Les objectifs et les moyens utilisés pour ces implantations sont donc complètement différents de ceux pour l'implantation de LED proposée dans ce chapitre.

Toutefois, il est possible de voir dans le Tableau 5.7 que KTANTAN reste le plus petit algorithme en termes de surface d'implantation. De plus, l'implantation de KTANTAN que nous proposons dans ce chapitre utilise uniquement des éléments génériques des FPGA et fonctionne à la fois pour le chiffrement et le déchiffrement. Enfin, la répétabilité des implantations proposées dans [10] et dans [6] n'est pas possible puisque les auteurs de ces articles ne donnent pas accès aux fichiers sources de leur implantations, contrairement à celles présentées dans ce chapitre et dans le Chapitre 4.

5.4 Conclusion et perspectives

Dans ce chapitre, les implantations matérielles parallèles et séries de trois algorithmes légers de chiffrement par bloc ont été présenté : KLEIN, LED et KTANTAN. Pour chacun de ces algorithmes, une description succincte a été effectuée et permet de mieux comprendre les implantations qui ont été réalisées.

Les résultats d'implantation ont été présentés pour deux familles de FPGA : Xilinx Spartan 6 et Xilinx Spartan 3. De plus, tous les algorithmes de chiffrement ont été implantés à l'aide d'une plateforme commune, ce qui rend la comparaison des résultats équitable. Enfin, s'il fallait choisir le meilleur candidat pour le projet SALWARE, l'algorithme retenu serait sûrement KTANTAN puisqu'il présente une implantation série très légère en termes de surface (290 LUT6 + Registres). Cependant, même si la contrainte en surface est celle qui est privilégiée pour le projet SALWARE, la lenteur du chiffrement de KTANTAN pourrait conduire à sélectionner, en pratique, un meilleur compromis entre la surface et le débit, tel que proposé par LILLIPUT.

Il est nécessaire d'explorer d'autres pistes et d'implanter d'autres algorithmes de chiffrement. Par exemple, il serait intéressant d'essayer d'implanter des algorithmes de chiffrement par flot ou d'autres algorithmes de chiffrement par bloc comme SIMON [16] qui semble être le plus léger des algorithmes de chiffrement par bloc.

De même, il est indispensable d'éprouver les implantations qui ont été effectuées et de les attaquer. Enfin, l'exploration de la réduction en surface des algorithmes de chiffrement par bloc a permis de montrer que le partage de ressources n'est pas toujours la meilleure stratégie. Il serait intéressant de pouvoir prévoir si le partage d'une fonction permet une réduction en surface, ou si les ajouts nécessaires pour ce partage vont augmenter la surface plus que la réduction apportée par le partage de cette fonction, comme c'est le cas pour LILLIPUT sur FPGA.

Chapitre 6

Conclusion et perspectives

Tout au long de ce manuscrit de thèse, nous avons étudié trois mécanismes de protection participant à la lutte contre la contrefaçon et le vol de circuits intégrés. L'importance du problème de la contrefaçon ne cesse ne grandir et les menaces pesant sur la propriété intellectuelle du concepteur de circuits intégrés sont très nombreuses. Ces menaces ont été abordées dans le Chapitre 1 et montrent qu'il est indispensable de trouver des solutions pratiques permettant au concepteur de circuits de protéger son travail. Dans le Chapitre 1, nous avons également présenté les différents mécanismes de protection permettant de lutter contre la contrefaçon et le vol de circuits intégrés. C'est dans ce premier chapitre que nous avons décrit les objectifs du projet SALWARE ainsi que le mécanisme de protection qu'il propose. La Figure 6.1 rappelle ce mécanisme et met en avant les trois parties (en rouge) qui ont été étudiées dans cette thèse.

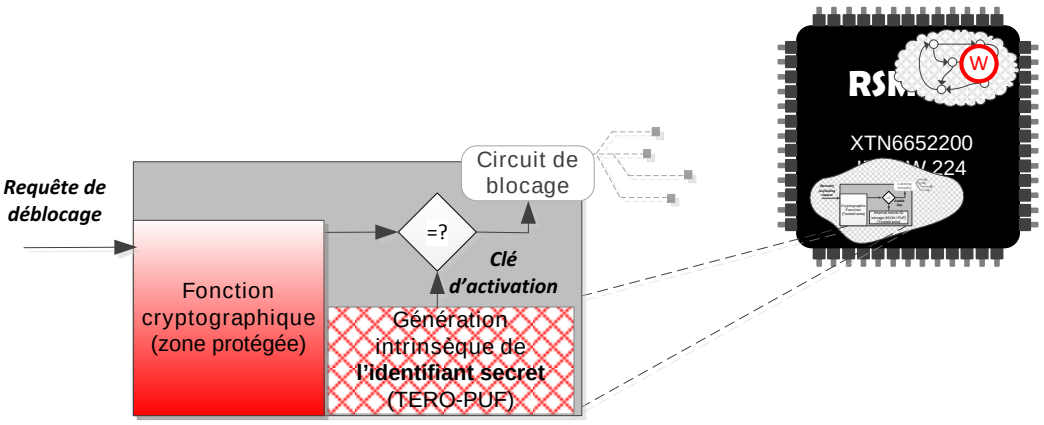

Figure 6.1 – Mécanismes de protection proposés dans le projet SALWARE et parties étudiées dans ce manuscrit (en rouge)

Résumé des contributions

Dans le Chapitre 2, nous avons étudié l'insertion et la vérification de *watermarks* insérées dans la machine à états finie d'un système numérique synchrone. Une méthode de vérification de *watermark* a été présentée. Elle utilise le canal auxiliaire de la consommation de puissance d'un circuit électronique afin d'extraire les informations indiquant la présence ou non d'une *watermark*. Le premier avantage de cette méthode est qu'elle est relativement indépendante de l'algorithme utilisé pour insérer la marque à l'intérieur du système. De plus, un guide permettant de choisir les paramètres nécessaires à la mise en œuvre de la méthode de vérification est également présenté. Enfin, une analyse détaillée des résultats de l'expérience montre que la variance des coefficients de corrélation contient plus d'informations utiles à la vérification de *watermarks* que le niveau moyen de corrélation.

Dans le Chapitre 3, nous avons présenté la conception et la caractérisation d'une fonction physique non-clonable : la TERO-PUF. De plus, un banc de caractérisation spécialement mis en place pour la caractérisation des PUF a été présenté. Enfin, la caractérisation de la TERO-PUF a été réalisée pour deux familles de FPGA : Xilinx Spartan 6 et Altera Cyclone V. Les résultats ont été précisément décrits et montrent qu'il est possible d'extraire plus d'un bit par challenge grâce à la TERO-PUF. Cela constitue clairement un avantage en termes de surface d'implantation puisque deux fois moins de cellules sont nécessaires, par rapport à la RO-PUF. Lors de la caractérisation, l'évaluation de l'imprévisibilité des réponses a été effectuée grâce à certains tests de la suite de tests statistiques NIST. Cette approche permet une analyse plus fine qu'une simple mesure du biais. En revanche, ils ne sont pas adaptés aux fonctions physiques non-clonables et il est nécessaire d'aller plus loin dans cette partie de l'analyse et d'étudier une modélisation stochastique de l'extraction d'entropie.

Dans les Chapitres 4 et 5, nous avons étudié une troisième partie du mécanisme de protection proposé dans le projet SALWARE : l'implantation légère en surface d'algorithmes légers de chiffrement par bloc. Dans le Chapitre 4, la description et les implantations matérielles parallèles et séries de l'algorithme LILLIPUT ont été présentées. Il s'agit des premières implantations matérielles de LILLIPUT sur FPGA. Nous avons montré que le partage de ressources logiques n'est pas la meilleure stratégie d'implantation sur FPGA. En effet, l'implantation série de LILLIPUT occupe plus de surface que son implantation parallèle.

En revanche, les résultats sur ASIC montrent que les implantations séries de LILLIPUT sont plus légères en surface que les implantations parallèles. Cela vient du fait que toutes les opérations logiques ayant le même nombre d'entrées possèdent le même coût en surface sur FPGA alors que sur ASIC, chaque transistor compte. La surface d'implantation de LILLIPUT sur ASIC montre également que cet algorithme est bien un algorithme léger puisque la plupart des implantations proposées occupent une surface plus petite que le seuil empirique de 3 000 GE.

Dans le Chapitre 5, nous avons comparé les implantations matérielles de LILLIPUT avec celles de trois autres algorithmes légers de chiffrement par bloc : KLEIN,

LED et KTANTAN. Cette étude comparative a permis de mettre en avant l'algorithme KTANTAN qui occupe le moins de surface sur FPGA.

Pespectives d'améliorations

Les différents travaux qui ont été présentés dans ce manuscrit laissent entrevoir un certain nombre d'améliorations possibles. Premièrement, nous n'avons étudié que trois parties du mécanisme de protection proposé dans le projet SALWARE. Les autres parties de ce mécanisme étant étudiés par d'autres membres de l'équipe, il sera nécessaire de tous les regrouper au sein d'un même bloc. Le mécanisme de protection complet (Figure 6.1) devra également être testé par différents types d'attaques et chaque partie de ce mécanisme également. On peut par exemple utiliser les canaux auxiliaires pour attaquer la TERO-PUF ou l'implantation matérielle de l'algorithme de chiffrement.

En ce qui concerne les travaux présentés sur le *watermarking*, il est indispensable de tester la méthode de vérification sur d'autres schémas d'insertion de *watermark*, dans des environnements plus bruités mais aussi sur d'autres cibles (FPGA ou ASIC). Par exemple, elle pourrait être testée pour vérifier des *watermarks* insérées grâce aux techniques d'insertion par ajout. De plus, il faut inclure les machines à états finies dans des environnements plus complets, avec d'autres périphériques fonctionnant en même temps. Il est également possible de tester si la méthode de vérification fonctionne encore si le circuit de référence est différent des circuits de test (d'un point de vue du contenu implanté) : le circuit de référence peut être constitué de l'IP marquée seule, alors que le circuit de test contiendrait, par exemple, un processeur ainsi que d'autres périphériques en plus de l'IP marquée. La sensibilité de la méthode de vérification proposée doit aussi être évaluée face aux variations du processus de fabrication des circuits intégrés. De même, il serait intéressant de généraliser cette méthode en la testant sur des circuits complètement différents. Que se passe-t-il si le FPGA de référence appartient à une autre technologie que les FPGA de test ? Par exemple, il est possible d'utiliser un FPGA Xilinx comme référence et des FPGA Altera ou Microsemi pour les tests.

Les travaux présentés sur la TERO-PUF ont besoin d'être à la fois complétés et améliorés. En effet, la caractérisation en température de la TERO-PUF sur les FPGA Altera Cyclone V est manquante et doit être faite. De plus, le paramètre de la durée d'acquisition introduit dans ce chapitre doit être étudié plus en détail. Il faudrait par exemple trouver un moyen d'écarter les cellules dont l'état oscillant est très long (voir permanent) de la génération des réponses de la PUF. Enfin, il est également nécessaire de trouver et de tester des attaques sur la TERO-PUF. En effet, la sécurité de la TERO-PUF n'a encore jamais été éprouvée. Il est certainement possible de récupérer la moyenne du nombre d'oscillations des cellules TERO par l'utilisation d'un canal auxiliaire comme le rayonnement électromagnétique ou la consommation de puissance. Une analyse spectrale de la consommation de puissance pourrait par exemple être effectuée sur une fenêtre glissante. De cette manière, une

raie devrait apparaitre à la fréquence d'oscillations d'une cellule lorsque celle-ci oscille. La raie disparaîtra lorsque la cellule aura atteint son état stable. Ainsi, en enregistrant le temps écoulé entre l'apparition et la disparition des raies, il est possible de remonter jusqu'à un nombre d'oscillations. De plus, l'amplitude de la raie est linéairement liée avec le nombre d'oscillations de la cellule. Il est donc possible de retrouver ce nombre.

D'une manière plus générale, nous avons montré que l'étude de l'imprévisibilité des PUF est une partie critique de leur analyse. Nous pensons qu'il est nécessaire d'élaborer des tests spécifiques aux fonctions physiques non-clonables, permettant de tester leur imprévisibilité. Ces tests doivent être basés sur des modèles stochastiques des PUF, comme pour les générateurs de nombres vraiment aléatoires.

En ce qui concerne les implantations matérielles d'algorithmes légers de chiffrement, il est indispensable de tester la robustesse des implantations réalisées face aux attaques, comme celles par canal auxiliaire par exemple. En effet, la sécurité du mécanisme de protection proposé par le projet SALWARE peut être remise en cause si l'algorithme implanté est facilement attaquable. De plus, il serait intéressant d'explorer d'autres pistes et d'implanter d'autres algorithmes de chiffrement. Par exemple, il serait intéressant d'essayer d'implanter des algorithmes de chiffrement par flot ou d'autres algorithmes de chiffrement par bloc comme SIMON [16] qui semble être le plus léger des algorithmes de chiffrement par bloc. Enfin, l'exploration de la réduction en surface des algorithmes de chiffrement par bloc a permis de montrer que le partage de ressources n'est pas toujours la meilleure stratégie. Il serait intéressant de pouvoir prévoir si le partage d'une fonction permet une réduction en surface, ou si les ajouts nécessaires pour ce partage vont augmenter la surface occupée, plus que la réduction apportée par la partage, comme c'est le cas pour LILLIPUT sur FPGA.

Pour finir, on comprends bien que toutes ces contributions ne permettent pas d'éliminer la contrefaçon et le vol de circuits intégrés à elles seules. À long terme, il faudrait développer des méthodes de conception qui intègrent des mécanismes de protection dès les premières étapes de la création d'un circuit intégré. En effet, tous les mécanismes de protection que nous avons présenté dans ce manuscrit imposent au concepteur de circuits de fournir un effort supplémentaire pour protéger sa propriété intellectuelle. Dans l'idéal, une protection contre la contrefaçon ou le vol de circuits intégrés ne doit pas être rajoutée à un circuit mais intégrée dans le flot de développement de celui-ci. Ainsi, on peut imaginer la création d'outils de conception permettant l'insertion automatique de certains mécanismes de protection.

Communications : publications et présentations

Publications dans des journaux internationaux

1. Cherkaoui, A., Bossuet, L., Marchand, C., ***Design, Evaluation and Optimization of Physical Unclonable Functions based on Transient Effect Ring Oscillators***, IEEE Transactions on Information Forensics & Security, Juin 2016.
2. Jung, E., Bossuet, L., Choi, S., Marchand, C., ***Identification of IP control units by state encoding and side channel verification***, Microprocessors and Microsystems, February 2016.
3. Marchand, C., Bossuet, L., Gaj, K., ***Area-oriented comparison of lightweight crypto-hardware on FPGAs***, International journal of Circuit Theory and Applcations, Wiley, à paraître fin 2016 dans un numéro spécial : *Secure lightweight crypto-hardware*.

Publications dans des conférences internationales

5. Marchand, C., Bossuet, L. Jung, E., ***IP watermark verification based on power consumption analysis***, In proceedings of 27th International Systems-on-Chip Conference, SOCC 2014.
6. Jung, E., Marchand, C., Bossuet, L., ***Identification of Embedded Control Units by State Encoding and Power Consumption Analysis.***, In proceedings of the 30th ACM/SIGAPP Symposium on Applied Computing, SAC 2015.
7. Marchand C., Cherkaoui, A., Bossuet, L., ***Design and characterization of the TERO-PUF on SRAM FPGAs*** To appears in proceeding of the IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2016.
8. Marchand C., Cherkaoui, A., Bossuet, L., ***Enhanced TERO-PUF implementations and characterization on FPGAs***, In proceedings of the 24th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2016. (poster)

Chapitre de livre

9. Marchand, C., Bossuet, L., Gaj, K.,***Ultra-lightweight implementation in area of block ciphers***, Foundations of Hardware IP Protection, L. Bossuet, L. Torres (Eds), Springer, à paraitre fin 2016 (Chapter 9 : 27 pages).
10. Bossuet, L., Marchand, C., ***Side Channel Analysis, an efficient ally for IP protection***, Foundations of Hardware IP Protection, L. Bossuet, L. Torres (Eds), Springer, à paraitre fin 2016 (Chapter 5 : 21 pages).

Communications non actées

11. Marchand, C., Bossuet, L., ***Correlation analysis of the power consumption applied to IP watermark verification***, GdR SoC-SiP 2014.
12. Marchand, C., Bossuet, L., Jung, E., ***Verification of IP watermark using correlation analysis***, Cryptographic Architectures Embedded in Reconfigurable Device (Cryptarchi), 2014.
13. Marchand, C., Bossuet, L., Jung, E., ***Analyse de la consommation de puissance appliquée à la vérification du marquage d'IP***, Journée Sécurité Numérique, 2014.
14. Marchand, C., Cherkaoui, A., Bossuet, L., ***Enhanced TERO-PUF design and characterization with FPGA***, Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE) 2015.
15. Marchand, C., Bossuet, L., ***Counterfeits detection and IP Protection by using FSM watermarking and Side Channel Verification***, Workshop Phisic 2015.
16. Bochard, N., Marchand, C., Petura, O., Bossuet, L., Fischer, V., ***Evariste III : A new multi-FPGA system for fair benchmarking of hardware dependent cryptographic primitives.*** In Workshop on cryptographic hardware and Embedded Systems, CHES 2015. (poster)

Bibliographie

[1] ISO/IEC JTC 1. Information technology. Automatic identification and data capture techniques. QR Code 2005 bar code symbology specification, 2006.

[2] A. T. Abdel-Hamid, Sofiène T., and E. M. Aboulhamid. A survey on IP watermarking techniques. *Design Autom. for Emb. Sys.*, 9(3) :211–227, 2004.

[3] Y. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX Security Symposium*, 2007.

[4] Y. Alkabani and F. Koushanfar. Active control and digital rights management of integrated circuit IP cores. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 227–234, 2008.

[5] Y. Alkabani, F. Koushanfar, and M. Potkonjak. Remote activation of ics for piracy prevention and digital right management. In *International Conference on Computer-Aided Design*, pages 674–677, 2007.

[6] N. Nalla Anandakumar, T. Peyrin, and A. Poschmann. A very compact FPGA implementation of LED and PHOTON. In *International Conference on Cryptology in India*, pages 304–321, 2014.

[7] K. Arndt, C. Narayan, A. Brintzinger, W. Guthrie, D. Lachtrupp, J. Mauger, D. Glimmer, S. Lawn, B. Dinkel, and A. Mitwalsky. Reliability of laser activated metal fuses in DRAMs. In *Electronics Manufacturing Technology Symposium*, pages 389–394, 1999.

[8] B. Badrignans, J.L. Danger, V. Fischer, G. Gogniat, and L. Torres. *Security Trends for FPGAS : From Secured to Secure Reconfigurable Systems*. Springer Science & Business Media, 2011.

[9] Y. Bahout. Integrated circuit containing a protected memory and secured system using said integrated circuit. Patent, 1997.

[10] S. Bangari and E. Elavarasi. Fpga implementation of data encryption and decryption using optimized led algorithm. *International Journal on Advanced Computer Theory and Engineering*, 2014.

[11] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni. Midori : A block cipher for low energy. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 411–436, 2015.

[12] A. Basak, Y. Zheng, and S. Bhunia. Active defense against counterfeiting attacks through robust antifuse-based on-chip locks. In *IEEE VLSI Test Symposium*, pages 1–6, 2014.

[13] J.P. Baukus, W.M. Clark, L.W. Chow, and A.R. Kramer. Integrated circuit security system and method with implanted interconnections. Patent, 1999.

[14] A.C. Baumgarten. *Preventing integrated circuit piracy using reconfigurable logic barriers*. PhD thesis, Iowa State University, 2009.

[15] P. Bayon, L. Bossuet, A. Aubert, and V. Fischer. Electromagnetic analysis on ring oscillator-based true random number generators. In *IEEE International Symposium on Circuits and Systems*, pages 1954–1957, 2013.

[16] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers. The SIMON and SPECK families of lightweight block ciphers. *IACR Cryptology ePrint Archive*, 2013.

[17] T.P. Berger, J. Francq, M. Minier, and G. Thomas. Extended generalized feistel networks using matrix representation to propose a new lightweight block cipher : Lilliput. *IEEE Trans. Computers*, 65(7) :2074–2089, 2016.

[18] A. Bertrand. *Le droit d'auteur et les droits voisins*. Dalloz, 1999.

[19] N. Bochard, C. Marchand, O. Pet'ura, L. Bossuet, and V. Fischer. Evariste III : A new multi-FPGA system for fair benchmarking of hardware dependent cryptographic primitives. In *International Workshop on Cryptographic Hardware and Embedded Systems (Poster)*, 2015.

[20] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT : an ultra-lightweight block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 450–466, 2007.

[21] C. Böhm and M. Hofer. *Physical Unclonable Functions in Theory and Practice*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[22] C. Bösch, J.G., A. Sadeghi, J. Shokrollahi, and P. Tuyls. Efficient helper data key extractor on fpgas. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 181–197, 2008.

[23] L. Bossuet. Lutte contre le vol, la copie illégale, le reverse-engineering et la contrefaçon de circuits intégrés. *Jounrée sécurité numérique du GDR SoC-SiP*, 2012.

[24] L. Bossuet. Sustainable electronics : On the trail of reconfigurable computing. *Sustainable Computing : Informatics and Systems*, 4(3) :196–202, 2014.

[25] L. Bossuet, P. Bayon, and V. Fischer. An ultra-lightweight transmitter for contactless rapid identification of embedded IP in FPGA. *Embedded Systems Letters*, 7(4) :97–100, 2015.

[26] L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for SRAM FPGA bitstreams. *International Journal of Embbeded System*, 2(1/2) :73–85, 2006.

[27] L. Bossuet and D. Hely. SALWARE : Salutary Hardware to design Trusted IC. In *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices*, 2013.

[28] L. Bossuet, X.T. Ngo, Z. Cherif, and V. Fischer. A PUF based on a transient effect ring oscillator and insensitive to locking phenomenon. *IEEE Trans. Emerging Topics in Computing*, 2(1) :30–36, 2014.

[29] M.S. Branham. *Semiconductors and sustainability : energy and materials use in integrated circuit manufacturing.* PhD thesis, Massachusetts Institute of Technology, 2008.

[30] M.S. Branham and T.G. Gutowski. Deconstructing Energy Use in Microelectronics Manufacturing : An Experimental Case Study of a MEMS Fabrication Facility. *Environmental Science and Technology*, 2010.

[31] J. Bringer, H. Chabanne, and T. Icart. On physical obfuscation of cryptographic algorithms. In *International Conference on Cryptology in India*, pages 88–103, 2009.

[32] Douane Française. Les chiffres clés de la lutte contre la fraude, 2014.

[33] C. Cannière. Trivium : A stream cipher construction inspired by block cipher design principles. In *International Conference on Information Security*, pages 171–186, 2006.

[34] C. Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 272–288, 2009.

[35] J. Castillo, P. Huerta, V. López, and J.I. Martínez. A secure self-reconfiguring architecture based on open-source hardware. In *International Conference on Reconfigurable Computing and FPGAs*, 2005.

[36] J. Castillo, P. Huerta, and J.I. Martínez. Secure IP downloading for SRAM fpgas. *Microprocessors and Microsystems*, 31(2) :77–86, 2007.

[37] R. Chapman and T.S. Durrani. IP protection of DSP algorithms for system on chip implementation. *IEEE Trans. Signal Processing*, 48(3) :854–861, 2000.

[38] Z. Cherif, J.L. Danger, S. Guilley, and L. Bossuet. An easy-to-design PUF based on a single oscillator : The loop PUF. In *Euromicro Conference on Digital System Design*, pages 156–162, 2012.

[39] A. Cherkaoui, L. Bossuet, and C. Marchand. Design, evaluation, and optimization of physical unclonable functions based on transient effect ring oscillators. *IEEE Trans. Information Forensics and Security*, 11(6) :1291–1305, 2016.

[40] L.W. Chow, J.P. Baukus, B.J. Wang, and R.P. Cocchi. Camouflaging a standard cell based integrated circuit. Patent, 2012.

[41] S. Chow, P.A. Eisen, H. Johnson, and P.C. van Oorschot. White-box cryptography and an AES implementation. In *International Workshop on Selected Areas in Cryptography*, pages 250–270, 2002.

[42] R.P. Cocchi, J.P. Baukus, L.W. Chow, and B.J. Wang. Circuit camouflage integration for hardware IP protection. In *Design Automation Conference*, pages 153 :1–153 :5, 2014.

[43] B. Colombier and L. Bossuet. Survey of hardware protection of design data for integrated circuits and intellectual properties. *IET Computers & Digital Techniques*, 8(6) :274–287, 2014.

[44] B. Colombier and L. Bossuet. Functional locking modules for design protection of intellectual property cores. In *International Symposium on Field-Programmable Custom Computing Machines*, page 233, 2015.

[45] B. Colombier, L. Bossuet, and D. Hély. From secured logic to IP protection. *Microprocessors and Microsystems*, 2016.

[46] G.K. Contreras, M.T Rahman, and M. Tehranipoor. Secure split-test for preventing IC piracy by untrusted foundry and assembly. In *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 196–203, 2013.

[47] N. Couture and K.B. Kent. Periodic licensing of FPGA based intellectual property. In *International Conference on Field Programmable Technology*, pages 357–36034, 2006.

[48] M. Crawford, T. Telesco, C. Nelson, J. Bolton, K. Bagin, and B. Botwin. DEFENSE INDUSTRIAL BASE ASSESSMENT : COUNTERFEIT ELECTRONICS. Technical report, U.S. Department of commerce bureau of Industry and security office of technology evaluation, 2010.

[49] A. Cui and C.H. Chang. Intellectual property authentication by watermarking scan chain in design-for-testability flow. In *International Symposium on Circuits and Systems*, pages 2645–2648, 2008.

[50] J. Daemen and V. Rijmen. *The Design of Rijndael : AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[51] S. Drimer and M.G. Kuhn. A protocol for secure remote updates of FPGA configurations. In *International Workshop on Reconfigurable Computing : Architectures, Tools and Applications*, pages 50–61, 2009.

[52] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel. A survey of lightweight-cryptography implementations. *IEEE Design & Test of Computers*, 24(6) :522–533, 2007.

[53] V. Fischer and N. Bochard. Evariste II : Modular hardware system for fair TRNG benchmarking. In *International Workshops on Cryptographic Architectures Embedded in Reconfigurable Devices*, 2013.

[54] É Freyssinet. *La cybercriminalité en mouvement.* Hermès Science Publications, 2012.

[55] B. Le Gal and L. Bossuet. Automatic low-cost IP watermarking technique based on output mark insertions. *Design Autom. for Embedded System.*, 16(2) :71–92, 2012.

[56] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis : Concrete results. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 251–261, 2001.

[57] K.M. Goertzel and BA. Hamilton. Integrated circuit security threats and hardware assurance countermeasures. *Crosstalk-Real Time Information Assurance*, pages 33–38, 2013.

[58] Z. Gong, S. Nikova, and Y.W. Law. KLEIN : A new family of lightweight block ciphers. In *International Workshop on RFID. Security and Privacy*, pages 1–18, 2011.

[59] C. Gorman. Counterfeit chips on the rise. *IEEE Spectrum*, 49(6) :16–17, 2012.

[60] GSA. http ://www.gsaglobal.org/gsa-resources/publications/charting-a-new-course-for-semicoductors/, 2015.

[61] J. Guajardo, S. Kumar, G.J. Schrijen, and P. Tuyls. FPGA intrinsic pufs and their use for IP protection. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 63–80, 2007.

[62] J. Guajardo, B. Skoric, P. Tuyls, S. Kumar, T. Bel, A.H.M. Blom, and G.J. Schrijen. Anti-counterfeiting, key distribution, and key storage in an ambient world via physical unclonable functions. *Information Systems Frontiers*, 11(1) :19–41, 2009.

[63] N. Guibert. Les systèmes d'armes américains truffés de composants électroniques de contrefaçon. *Le monde*, 2012.

[64] U. Guin, D. DiMase, and M. Tehranipoor. Counterfeit integrated circuits : Detection, avoidance, and the challenges ahead. *Journal of Electronic Testing*, 30(1) :9–23, 2014.

[65] U. Guin, X. Zhang, D. Forte, and M. Tehranipoor. Low-cost on-chip structures for combating die and IC recycling. In *Design Automation Conference 2014*, pages 87 :1–87 :6, 2014.

[66] J. Guo, T. Peyrin, A. Poschmann, and M.J.B. Robshaw. The LED block cipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 326–341, 2011.

[67] B. Habib, J.P. Kaps, and K. Gaj. Efficient sr-latch PUF. In *International Symposium on Applied Reconfigurable Computing*, pages 205–216, 2015.

[68] P. Hämäläinen, T. Alho, M. Hännikäinen, and T.D.Hämäläinen. Design and implementation of low-area and low-power AES encryption hardware core. In *Euromicro Conference on Digital System Design : Architectures, Methods and Tools*, pages 577–583, 2006.

[69] S. Hamdioui, J.L. Danger, G. Di Natale, F. Smailbegovic, G. van Battum, and M. Tehranipoor. Hacking and protecting IC hardware. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–7, 2014.

[70] J.B. Heinemann. Hardware memory write lock circuit. Patent, 1983.

[71] P. Heremans, R. Bellens, G. Groeseneken, and H.E. Maes. Consistent model for the hot-carrier degradation in n-channel and p-channel MOSFETs. *IEEE Trans. Electron Devices*, 35(12) :2194–2209, 1988.

[72] D.E. Holcomb, W.P. Burleson, and K. Fu. Power-up SRAM state as an identifying fingerprint and source of true random numbers. *IEEE Trans. Computers*, 58(9) :1198–1210, 2009.

[73] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee. HIGHT : A new block cipher suitable for low-resource device. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 46–59, 2006.

[74] Y. Hori, A. Satoh, H. Sakane, and K. Toda. Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems. In *International Conference on Field Programmable Logic and Applications*, pages 23–28, 2008.

[75] G. Hospodar, R. Maes, and I. Verbauwhede. Machine learning attacks on 65nm arbiter pufs : Accurate modeling poses strict bounds on usability. In *International Workshop on Information Forensics and Security*, pages 37–42, 2012.

[76] C. Hu, S.C. Tam, F.C. Hsu, P.K. Ko, T.Y. Chan, and K.W. Terrill. Hot-electron-induced MOSFET degradation ; Model, monitor, and improvement. *IEEE Trans. Electron Devices*, 32(2) :375–385, 1985.

[77] H. Huang, A. Boyer, and S.B. Dhia. The detection of counterfeit integrated circuit by the use of electromagnetic fingerprint. In *International Symposium on Electromagnetic Compatibility*, pages 1118–1122, 2014.

[78] J. Huang and J. Lach. IC activation and user authentication for security-sensitive systems. In *IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 76–80, 2008.

[79] K. Huang, J.M. Carulli Jr., and Y. Makris. Parametric counterfeit IC detection via support vector machines. In *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 7–12, 2012.

[80] K. Huang, J.M. Carulli Jr., and Y. Makris. Counterfeit electronics : A rising threat in the semiconductor manufacturing industry. In *International Test Conference*, pages 1–4, 2013.

[81] F. Imeson, A. Emtenan, S. Garg, and M.V. Tripunitara. Securing computer hardware using 3d integrated circuit (IC) technology and split manufacturing for obfuscation. In *USENIX Security Symposium*, pages 495–510, 2013.

[82] Components Technology Institute. Counterfeit Examples, Electronic Components. http ://www.cti-us.com/pdf/CCAP-101InspectExamplesA6.pdf, 2013.

[83] R.W. Jarvis and M.G. McIntyre. Split manufacturing method for advanced semiconductor circuits. Patent, 2007.

[84] E. Jung, C.C. Hung, S. Choi, and M. Yang. An efficient locking and unlocking method of sequential systems. In *Research in Applied Computation Symposium*, pages 428–433, 2012.

[85] E. Jung, C. Marchand, and L. Bossuet. Identification of embedded control units by state encoding and power consumption analysis. In *Symposium on Applied Computing*, pages 1957–1959, 2015.

[86] A.B. Kahng, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe. Robust IP watermarking methodologies for physical design. In *Design Automation Conference*, pages 782–787, 1998.

[87] F. Karakoç, H. Demirci, and A.E. Harmanci. Itubee : A software oriented lightweight block cipher. In *International Workshop on Lightweight Cryptography for Security and Privacy*, pages 16–27, 2013.

[88] S. Kerckhof, F. Durvaux, F.X. Standaert, and B. Gérard. Intellectual property protection for FPGA designs with soft physical hash functions : First experimental results. In *International Symposium on Hardware-Oriented Security and Trust*, pages 7–12, 2013.

[89] A. Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, 9, 1883.

[90] P.C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *International Cryptology Conference on Advances in Cryptology*, pages 388–397, 1999.

[91] C. Kothandaraman, S.K. Iyer, and S.S. Iyer. Electrically programmable fuse (eFUSE) using electromigration in silicides. *IEEE Electron Device Letters*, 23(9) :523–525, 2002.

[92] F. Koushanfar. Hardware Metering : A Survey. In *Introduction to Hardware Security and Trust*, pages 103–122. Springer New York, 2012.

[93] F. Koushanfar, S. Fazzari, C. McCants, W. Bryson, M. Sale, P. Song, and M. Potkonjak. Can EDA combat the rise of electronic counterfeiting ? In *Design Automation Conference*, pages 133–138, 2012.

[94] F. Koushnafar and I. Markov. Designing Chips that Protect Themselves. *Design Automation Conference, Knowledge Center Article*, 2010.

[95] C. Kuemin, L. Nowack, L. Bozano, N.D. Spencer, and H. Wolf. Oriented Assembly of Gold Nanorods on the Single-Particle Level. *Advanced Functional Materials*, 22(4) :702–708, 2012.

[96] J. Kufel, P.R. Wilson, S. Hill, B.M. Al-Hashimi, P.N. Whatmough, and J. Myers. Clock-modulation based watermark for protection of embedded processors. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2014.

[97] S.S. Kumar, J. Guajardo, R. Maes, G.J. Schrijen, and P. Tuyls. The butterfly PUF : protecting IP on every FPGA. In *International Workshop on Hardware-Oriented Security and Trust*, pages 67–70, 2008.

[98] M. Lecomte, J.J. Fournier, and P. Maurine. On-chip fingerprinting of IC topology for integrity verification. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 133–138, 2016.

[99] H.K. Lee, H.J. Kim, K.R. Kwon, and J.K. Lee. ROI Medical Image Watermarking Using DWT and Bit-plane. In *Asia-Pacific Conference on Communications*, pages 512–515, 2005.

[100] B. Liu and B. Wang. Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2014.

[101] S. Liu. Anti-counterfeit System Based on Mobile Phone QR Code and Fingerprint. In *International Conference on Intelligent Human-Machine Systems and Cybernetics*, volume 2, pages 236–240, 2010.

[102] R. Maes, A. Van Herrewege, and I. Verbauwhede. PUFKY : A fully functional puf-based cryptographic key generator. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 302–319, 2012.

[103] R. Maes, D. Schellekens, and I. Verbauwhede. A pay-per-use licensing scheme for hardware IP cores in recent sram-based fpgas. *IEEE Trans. Information Forensics and Security*, 7(1) :98–108, 2012.

[104] R. Maes, P. Tuyls, and I. Verbauwhede. Intrinsic PUFs from flip-flops on reconfigurable devices. In *Benelux workshop on information and system security*, volume 17, 2008.

[105] R. Maes and I. Verbauwhede. *Physically Unclonable Functions : A Study on the State of the Art and Future Research Directions*, chapter 1. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[106] S. Mahapatra, D. Saha, D. Varghese, and P. B. Kumar. On the generation and recovery of interface traps in MOSFETs subjected to NBTI, FN, and HCI stress. *IEEE Trans. Electron Devices*, 53(7) :1583–1592, 2006.

[107] A. Maiti, J. Casarona, L. McHale, and P. Schaumont. A large scale characterization of RO-PUF. In *International Symposium on Hardware-Oriented Security and Trust*, pages 94–99, 2010.

[108] A. Maiti, V. Gunreddy, and P. Schaumont. A Systematic Method to Evaluate and Compare the Performance of Physical Unclonable Functions. *Embedded Systems Design with FPGAs*, pages 245–267, 2013.

[109] A. Maiti, I. Kim, and P. Schaumont. A robust physical unclonable function with enhanced challenge-response set. *IEEE Trans. Information Forensics and Security*, 7(1) :333–345, 2012.

[110] M.A. Mak. Dod needs to improve reporting and oversight to reduce supply chain risk. Technical report, United States Government Accountability Office, 2016.

[111] S. Malik, G.T. Becker, C. Paar, and W.P. Burleson. Development of a layout-level hardware obfuscation tool. In *Computer Society Annual Symposium on VLSI*, pages 204–209, 2015.

[112] A. Markman, B. Javidi, and M. Tehranipoor. Photon-Counting Security Tagging and Verification Using Optically Encoded QR Codes. *IEEE Photonics Journal*, 6(1) :1–9, 2014.

[113] G. Masalskis and R. Navickas. Reverse engineering of CMOS integrated circuits. *Elektronika ir elektrotechnika*, 88(8) :25–28, 2015.

[114] S. Maynard. Trusted foundry - Be Safe. Be Sure. Be Trusted. *Tursted Manufacturing of Integrated Circuits for the Department of Defenses. NDIA Manufacturing Division Meeting*, 2010.

[115] I. McLoughlin. Reverse engineering of embedded consumer electronic systems. In *International Symposium on Consumer Electronics*, pages 352–356, 2011.

[116] B.J Mohd, T. Hayajneh, and A.V. Vasilakos. A survey on lightweight block ciphers for low-resource devices : Comparative study and open issues. *Journal of Network and Computer Applications*, 58 :73–93, 2015.

[117] A. Moradi, M. Kasper, and C. Paar. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the xilinx virtex-4 and virtex-5 bitstream encryption mechanism. In *Conference on Topics in Cryptology - CT-RSA*, pages 1–18, 2012.

[118] A. Moradi, D. Oswald, C. Paar, and P. Swierczynski. Side-channel attacks on the bitstream encryption mechanism of altera stratix II : facilitating black-box analysis using software reverse-engineering. In *International Symposium on Field Programmable Gate Arrays*, pages 91–100, 2013.

[119] R. Moudgil. *A statistical and circuit based technique for counterfeit detection in existing ICs*. PhD thesis, Virginia Tech, 2013.

[120] R. Moudgil, D. Ganta, L. Nazhandali, M.S. Hsiao, C. Wang, and S. Hall. A novel statistical and circuit-based technique for counterfeit detection in existing ics. In *Great Lakes Symposium on VLSI*, pages 1–6, 2013.

[121] M. Nakamura and S. Hieda. Method of laser-marking semiconductor devices. Patent, 1990.

[122] M. Neve and J.P. Seifert. Advances on access-driven cache attacks on AES. In *International Workshop on Selected Areas in Cryptography*, pages 147–162, 2006.

[123] T. Nie. Performance Evaluation for IP Protection Watermarking Techniques. *In Tech Open Access publisher*, 2, 2012.

[124] T. Nie, T. Kisaka, and M. Toyonaga. A watermarking system for IP protection by a post layout incremental router. In *Design Automation Conference*, pages 218–221, 2005.

[125] K. Nohl, D. Evans, S., and H. Plötz. Reverse-engineering a cryptographic RFID tag. In *USENIX Security Symposium*, pages 185–194, 2008.

[126] US Department of Homeland Security. https ://www.cbp.gov/trade/priority-issues/ipr/statistics.

[127] National Institute of Standards and Technology. Data Encryption Standard. *Federal Information Processing Standards (FIPS), Publication 46*, 1977.

[128] E. Ohbuchi, H. Hanaizumi, and L. Ah Hock. Barcode readers using the camera device in mobile phones. In *International Conference on Cyberworlds*, pages 260–265, 2004.

[129] A. L. Oliveira. Techniques for the creation of digital watermarks in sequential circuit designs. *IEEE Trans. CAD of Integrated Circuits and Systems*, 20(9) :1101–1117, 2001.

[130] M. Pecht and S. Tiku. Bogus : electronic manufacturing and consumers confront a rising tide of counterfeit electronics. *IEEE Spectrum*, 43(5) :37–46, 2006.

[131] F. A.P. Petitcolas, R.J. Anderson, and M.G. Kuhn. Information hiding-a survey. *Proceedings of the IEEE*, 87(7) :1062–1078, 1999.

[132] G. Qu and M. Potkonjak. Analysis of watermarking techniques for graph coloring problem. In *International Conference on Computer-Aided Design*, pages 190–193, 1998.

[133] M. Quéméner and J.P. Pinte. *Cybersécurité des acteurs économiques : Risques, réponses stratégiques et juridiques.* Hermès Sciences Publications, 2012.

[134] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Logic encryption : A fault analysis perspective. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 953–958, 2012.

[135] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security analysis of logic obfuscation. In *Design Automation Conference*, pages 83–89, 2012.

[136] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri. Security analysis of integrated circuit camouflaging. In *Conference on Computer and Communications Security*, pages 709–720, 2013.

[137] J. Rajendran, O. Sinanoglu, and R. Karri. Is split manufacturing secure ? In *Design, Automation and Test in Europe*, pages 1259–1264, 2013.

[138] L. M. Reyneri, D. Del Corso, and B. Sacco. Oscillatory metastability in homogeneous and inhomogeneous flip-flops. *IEEE Journal of Solid-State Circuits*, 25 :254–264, 1990.

[139] N. Robson, J. Safran, C. Kothandaraman, A. Cestero, X. Chen, R. Rajeevakumar, A. Leslie, D. Moy, T. Kirihata, and S.S. Iyer. Electrically programmable fuse (efuse) : From memory redundancy to autonomic chips. In *Custom Integrated Circuits Conference*, pages 799–804, 2007.

[140] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri. Hardware security : threat models and metrics. In *International Conference on Computer-Aided Design*, pages 819–823, 2013.

[141] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri. Hardware security : Threat models and metrics. In *International Conference on Computer-Aided Design*, pages 819–823, 2013.

[142] J.A. Roy, F. Koushanfar, and I.L. Markov. EPIC : ending piracy of integrated circuits. In *Design, Automation & Test in Europe Conference & Exhibition*, pages 1069–1074, 2008.

[143] J.A. Roy, F. Koushanfar, and I.L. Markov. Protecting bus-based hardware IP by secret sharing. In *Proceedings of the 45th Design Automation Conference*, pages 846–851, 2008.

[144] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *Conference on Computer and Communications Security*, pages 237–249, 2010.

[145] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, and E. Barker. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, DTIC Document, 2001.

[146] G. Ira S. Hot die marking device. Patent, 1951.

[147] D. Samyde, S.P. Skorobogatov, R.J. Anderson, and J.J. Quisquater. On a new way to read data from memory. In *International IEEE Security in Storage Workshop*, pages 65–69, 2002.

[148] L. Sauvage, S. Guilley, and Y. Mathieu. Electromagnetic radiations of fpgas : High spatial resolution cartography and attack on a cryptographic module. *ACM Trans. Reconfigurable Technology and Systems*, 2009.

[149] COMMITTEE ON ARMED SERVICES UNITED STATES SENATE. INQUIRY INTO COUNTERFEIT ELECTRONIC PARTS IN THE DEPARTMENT OF DEFENSE SUPPLY CHAIN, Mai 2012.

[150] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo : An ultra-lightweight blockcipher. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 342–357, 2011.

[151] A.J. Shils and J.P. Ianni. Laser written chip identification method. Patent, 1985.

[152] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In *International Workshop on Fast Software Encryption*, pages 181–195, 2007.

[153] P. Simons, E. van der Sluis, and V. van der Leest. Buskeeper PUFs, a promising alternative to D Flip-Flop PUFs. In *International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 7–12, 2012.

[154] S. Skorobogatov and C. Woods. In the blink of an eye : There goes your AES key. *IACR Cryptology ePrint Archive*, 2012.

[155] B. Sood, D. Das, and M. Pecht. Screening for counterfeit electronic parts. *Journal of Materials Science : Materials in Electronics*, 22(10) :1511–1522, 2011.

[156] B. Stamme. Anti-fuse memory provides robust, secure NVM option. *EE-Times*, 2012.

[157] F.X. Standaert, G. Piret, N. Gershenfeld, and J.J. Quisquater. SEA : A scalable encryption algorithm for small embedded applications. In *International Conference on Smart Card Research and Advanced Applications*, pages 222–236, 2006.

[158] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik. Reverse engineering digital circuits using functional analysis. In *Design, Automation and Test in Europe Conference & Exhibition*, pages 1277–1280, 2013.

[159] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In *International Conference on Selected Areas in Cryptography*, pages 339–354, 2012.

[160] M. Tehranipoor, U. Guin, and D. Forte. *Chip ID*, pages 243–263. Springer International Publishing, 2015.

[161] M. Tehranipoor, U. Guin, and D. Forte. *Combating Die and IC Recycling*, pages 175–201. Springer International Publishing, 2015.

[162] M. Tehranipoor, U. Guin, and D. Forte. *Counterfeit Integrated Circuits*, pages 15–36. Springer International Publishing, 2015.

[163] M. Tehranipoor and F. Koushanfar. A survey of hardware trojan taxonomy and detection. *IEEE Design & Test of Computers*, 27(1) :10–25, 2010.

[164] M. Tehranipoor and C. Wang. *Introduction to FPGA security and trust.* Springer, 2011.

[165] W.R. Tonti. eFuse Design and Reliability. In *Workshop on Integrated Reliability*, 2008.

[166] R. Torrance and D. James. The state-of-the-art in semiconductor reverse engineering. In *Design Automation Conference*, pages 333–338, 2011.

[167] I. Torunoglu and E. Charbon. Watermarking-based copyright protection of sequential functions. *IEEE Journal of Solid-State Circuits*, 35(3) :434–440, 2000.

[168] B. Tudor, J. Wang, Z. Chen, R. Tan, W. Liu, and F. Lee. An accurate MOSFET aging model for 28 nm integrated circuit simulation. *Microelectronics Reliability*, 52(8) :1565–1570, 2012.

[169] N. Tuzzio, K. Xiao, X. Zhang, and M. Tehranipoor. A zero-overhead IC identification technique using clock sweeping and path delay analysis. In *Great Lakes Symposium on VLSI*, pages 95–98, 2012.

[170] K. Uwasawa, T. Yamamoto, and T. Mogami. A new degradation mode of scaled p+ polysilicon gate pMOSFETs induced by bias temperature (BT) instability. In *International Electron Devices Meeting*, pages 871–874, 1995.

[171] V. van der Leest, G.J. Schrijen, H. Handschuh, and P. Tuyls. Hardware Intrinsic Security from D Flip-flops. In *Workshop on Scalable Trusted Computing*, pages 53–62, 2010.

[172] M. Varchola and M. Drutarovský. New high entropy element for FPGA based true random number generators. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 351–365, 2010.

[173] J. Vliegen, N. Mentcns, and I. Verbauwhede. A single-chip solution for the secure remote configuration of FPGAs using bitstream compression. In *International Conference on Reconfigurable Computing and FPGAs*, pages 1–6, 2013.

[174] B. Škorić, S. Maubach, T. Kevenaar, and P. Tuyls. Information-theoretic analysis of capacitive physical unclonable functions. *Journal of Applied Physics*, 100, 2006.

[175] X. Wang and M. Tehranipoor. Novel Physical Unclonable Function with Process and Environmental Variations. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2010.

[176] S. Wei, F. Koushanfar, and Miodrag M. Potkonjak. Integrated Circuit Digital Rights Management Techniques Using Physical Level Characterization. In *ACM Workshop on Digital Rights Management*, pages 3–14, 2011.

[177] A. Winstanley and M.R. Greenstreet. Temporal properties of self-timed rings. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 140–154, 2001.

[178] M. Woelki and J.H. Scaroni. Writing on silicon wafers. Patent, 1994.

[179] C. Wolfe, J.L. Wong, and M. Potkonjak. Watermarking graph partitioning solutions. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(10) :1196–1204, 2002.

[180] W. Wu and L. Zhang. Lblock : A lightweight block cipher. In *International Conference in Applied Cryptography and Network Security*, pages 327–344, 2011.

[181] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor. Bit selection algorithm suitable for high-volume production of SRAM-PUF. In *IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 101–106, 2014.

[182] J. Xuehua. Digital Watermarking and its Application in Image Copyright Protection. In *International Conference on Intelligent Computation Technology and Automation*, pages 114–117, 2010.

[183] N. Yamauchi. Semiconductor integrated circuit with selective read and write inhibiting. Patent, 1992.

[184] A.S. Zeineddini and K. Gaj. Secure partial reconfiguration of FPGAs. In *International Conference on Field-Programmable Technology*, pages 155–162, 2005.

[185] X. Zhang and M. Tehranipoor. Design of On-Chip Lightweight Sensors for Effective Detection of Recycled ICs. *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 22(5) :1016–1029, 2014.

[186] X. Zhang, N. Tuzzio, and M. Tehranipoor. Identification of recovered ICs using fingerprints from a light-weight on-chip sensor. In *Design Automation Conference*, pages 703–708, 2012.

[187] X. Zhang, K. Xiao, and M. Tehranipoor. Path-delay fingerprinting for identification of recovered ICs. In *International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pages 13–18, 2012.

[188] X. Zhuang, T. Zhang, H.H.S. Lee, and S. Pande. Hardware assisted control flow obfuscation for embedded processors. In *international conference on Compilers, architecture, and synthesis for embedded systems*, pages 292–302, 2004.

[189] D. Ziener and J. Teich. Power Signature Watermarking of IP Cores for FPGAs. *Journal of Signal Processing Systems*, 51(1) :123–136, 2008.

Annexe A : implantation d'une cellule TERO sur les FPGA Xilinx Spartan 6 et Altera Cyclone V

Ce chapitre annexe présente en détail comment la cellule TERO a été implantée sur les FPGA Xilinx Spartan 6 et Altera Cyclone V. Outre l'implantation matérielle pour chaque famille de FPGA, les quatre étapes de la conception de la cellule TERO sont décrites. Ces étapes, présentées dans la section 3.2 sont :

1. Écriture du code VHDL de la cellule TERO ;
2. Placement de chacun des éléments de la cellule ;
3. Vérification des délais de routage de la cellule ;
4. Création d'un bloc non modifiable par l'outil de synthèse fourni par les vendeur de FPGA.

Pour ces travaux, les logiciels Xilinx ISE 13.4 et Altera Quartus 14.0 sont utilisés.

Conception de la cellule TERO pour les FPGA Xilinx Spartan 6

Les FPGA Xilinx de la famille des Spartan 6 (CMOS 45ñm) sont composés de blocs logiques configurables appelés CLB qui contiennent deux *slices*. Le circuit utilisé (XC6SLX16) contient exactement 2278 *slices* répartis en trois catégories de la manière suivante :

- *Slice M* (569) ;
- *Slice L* (569) ;
- *Slice X* (1140).

Chaque *slice* est composé de quatre *look up tables* (LUT) avec six entrées et deux sorties ainsi que de huit registres. Les LUT des *Slices M* peuvent être configurées comme mémoire et les *Slices L* contiennent une partie logique permettant de propager une retenue en plus des LUT et des registres. Dans ce chapitre, nous n'utiliserons que les *Slices X* pour concevoir les cellules TERO. La Figure A.2 présente un schéma de la structure du FPGA.

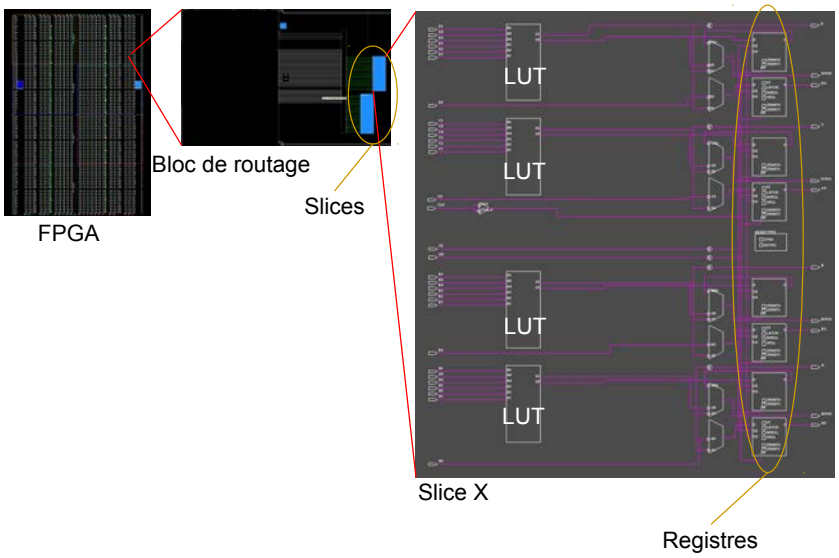

FIGURE A.2 – Structure interne d'un FGPA Xilinx Spartan 6.

Étape 1 : Écriture du code VHDL de la boucle TERO

L'implantation de la cellule TERO est faite grâce à l'utilisation du composant *LUT6* disponible dans la librairie Xilinx *UNISIM.vcomponents*. Ce composant possède six ports d'entrées, un port de sortie ainsi qu'un vecteur d'initialisation (Init) qui permet de créer n'importe quelle fonction logique comportant au plus six entrées. La cellule TERO que nous allons implanter ne comporte que des portes logiques *ET* et *NON*. La Figure A.3 présente le schéma interne du composant *LUT6* avec les noms des différents ports qui doivent être utilisés.

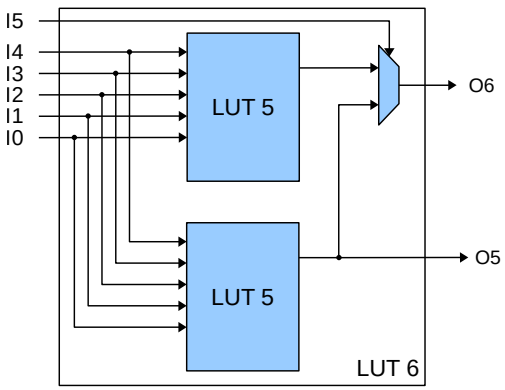

FIGURE A.3 – Structure interne d'une LUT à six entrées contenue dans un FGPA Xilinx Spartan 6.

Comme nous ne connaissons pas l'architecture interne des $LUT5$ composant chaque $LUT6$, la porte ET ainsi que la porte NON se servent en priorité de l'entrée $I5$ du composant. Ainsi, les LUT utilisées pour une cellule TERO sont configurées avec les équations suivantes : $NON(I5)$ et $I0\ ET\ I5$. Pour cela, le vecteur $INIT$ doit prendre les valeurs suivantes :

- $NON(I5) : INIT\ =$ x"00000000$FFFFFFFF$";
- $I0\ ET\ I5 : INIT\ =$ x"$AAAAAAAA$00000000".

L'architecture de la cellule TERO comporte deux branches constituées chacune de huit LUT6 regroupées dans deux *SliceX*. Ainsi, une cellule TERO utilise seize LUT6 et quatre *SliceX*. La Figure A.4 présente le schéma de cette cellule avec les noms des composants utilisés.

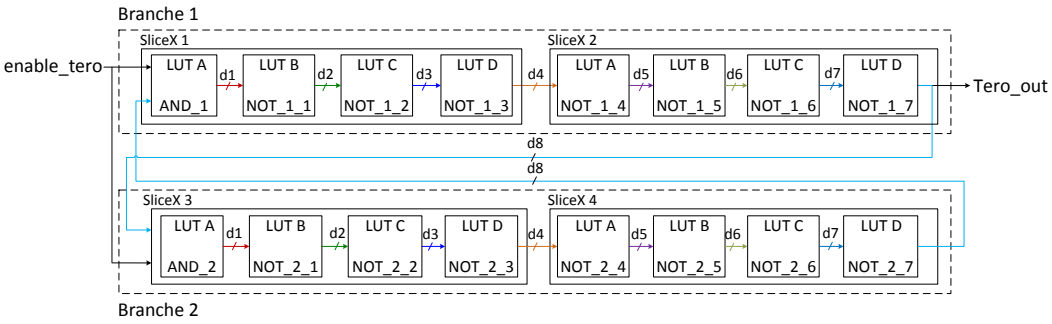

FIGURE A.4 – Schéma de l'implantation d'une cellule TERO à sept inverseurs par branche dans un FPGA Xilinx Spartan 6

Étape 2 : Placement des LUT

Une fois la description VHDL d'une cellule TERO effectuée, il est impératif de forcer le placement de chaque LUT dans le fichier de contrainte (ucf [1]). Ce placement se fait avec la syntaxe suivante :

$INST$ "**LUT à placer**" $LOC = SLICE\_X$**xx**Y**yy**$|BEL =$**LUT choisie**;

Sur cette contrainte, le texte en gras représente ce qui doit être complété. Le composant '**LUT à placer**' correspond au nom du composant choisi dans la description VHDL de la cellule, '**xx**' et '**yy**' représentent la *slice* sélectionnée dans le FPGA et la '**LUT choisie**' doit être remplacée par l'une des valeurs suivantes : $A6LUT$, $B6LUT$, $C6LUT$ ou $D6LUT$. Ces valeurs correspondent aux quatre LUT présentes dans chaque *slice*. Le reste de la ligne ne doit pas être modifié.

La Figure A.5 montre la configuration de placement choisie pour implanter la cellule TERO (capture d'écran du logiciel Xilinx FPGA editor). Cette configuration correspond à celle présentée dans la Figure A.4.

1. Ce format n'est plus supporté par le logiciel Xilinx VIVADO qui utilise le format .xdc qui n'est pas directement compatible.

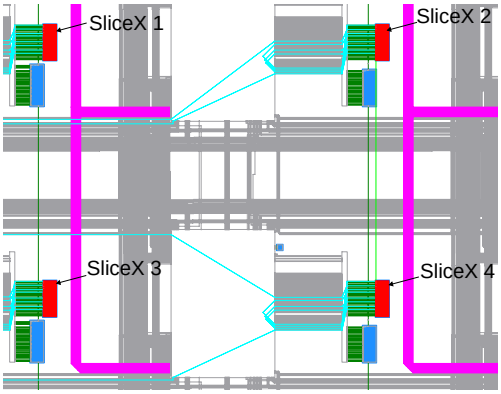

FIGURE A.5 – Vue de la cellule TERO dans le logiciel de conception Xilinx FPGA editor.

Étape 3 : Vérification des délais de routage de la cellule TERO

Pour vérifier que les délais de notre cellule TERO sont identiques dans chaque branche, le logiciel Xilinx FPGA Editor est utilisé. Il donne une représentation graphique du placement et du routage de la cellule TERO. De plus, ce logiciel permet d'estimer le délai du routage entre chaque LUT du circuit. Il s'agit uniquement d'une estimation mais il n'est pas possible d'obtenir des résultats plus précis sans les mesurer directement sur le circuit.

Dans le cas de la cellule TERO, il est nécessaire que toutes les connexions soient identiques deux à deux en ce qui concerne les délais. Le Tableau A.1 montre que le placement du TERO présenté dans la Figure A.5 remplit cette condition. Les noms des connections correspondent à ceux de la Figure A.4.

Tableau A.1 – Délais des connections de la cellule TERO implantée sur FPGA Xilinx Spartan 6 et estimés par le logiciel *Xilinx FPGA Editor*.

connection	délai dans la branche 1 (ns)	délai dans la branche 2 (ns)
$d1$	0.143	0.143
$d2$	0.352	0.352
$d3$	0.230	0.230
$d4$	0.494	0.494
$d5$	0.143	0.143
$d6$	0.352	0.352
$d7$	0.230	0.230
$d8$	0.626	0.626

Étape 4 : Création de la *Hard Macro*

La dernière étape de la conception de la cellule TERO consiste à fixer le composant dans un bloc considéré comme une boite noire par l'outil de synthèse de Xilinx. La création de ce nouveau composant se fait également avec le logiciel FPGA Editor et contient la cellule TERO placée et routée comme présenté dans la Figure A.5. Afin de créer la *hard macro*, les étapes suivantes doivent être respectées :

1. Autoriser la modification du circuit : File -> Main properties ;

2. Supprimer les ports du circuit ;

3. Sauvegarder le circuit en tant que *hard macro* : File -> Save as ;

4. Créer les ports du nouveau composant : Edit -> Add Hard Macro External Pin ;

5. Choisir une *slice* de référence : Edit -> Set Macro Reference Comp ;

6. Sauvegarder.

Une fois toutes ces étapes terminées, la *hard macro* est réalisée et peut être utilisée comme un composant dans un fichier VHDL. Le nom de ce composant correspond au nom du fichier de la *hard macro* (nmc) et les ports du composant correspondent à ceux créés dans FPGA Editor. Enfin, seul le placement de la *slice* de référence (étape 5.) doit apparaître dans le fichier de contraintes.

L'avantage de l'utilisation d'une *hard macro* pour concevoir la cellule TERO est la possibilité de dupliquer le composant autant de fois que nécessaire. De plus, chaque cellule est théoriquement identique. En revanche, les *hard macros* sont des composants spécifiques à un circuit et ne sont pas portables sur une autre famille de FPGA. Cela signifie que pour implanter la cellule TERO sur un autre FPGA Xilinx que Spartan 6, il faudra au minimum recréer une *hard macro* et peut-être même réécrire le fichier VHDL si les LUT du nouveau FPGA n'ont pas la même structure que celles des FPGA Xilinx Spartan 6.

De plus, cet outil doit être utilisé uniquement pour créer des petits composants, et il est préférable de limiter l'utilisation des registres, nécessitant une horloge ainsi que celle d'équations logiques nécessitant des constantes. En effet, ces deux types de signaux (horloge et constante) doivent être sortis du composant grâce à des ports externes et dupliqués autant de fois que les signaux apparaissent. Par exemple, si un composant comporte quatre registres reliés à l'horloge, il faudra créer quatre ports externes dédiés à l'horloge dans la *hard macro*. Cet outil est donc bien adapté dans le cas de l'implantation de la cellule TERO seule, mais ne l'est pas pour fixer l'ensemble de la TERO-PUF dans le FPGA.

Après la présentation de l'implantation de la cellule TERO sur FPGA Xilinx Spartan 6, la prochaine sous-section détaille son implantation pour les FPGA Altera Cyclone V.

Conception de la cellule TERO pour les FPGA Altera Cyclone V

Les FPGA Altera de la famille cyclone V (CMOS 28*nm*) sont composés de tableau de bloc logique (LAB), chaque LAB comporte dix modules logiques configurables (ALM) qui contiennent deux LUT chacun. Comme pour les FPGA Xilinx Spartan 6, ces LUT ont six entrées et deux sorties. Le FPGA utilisé dans ce chapitre (EP5CEBA4F17C8N) contient 18480 ALM. La Figure A.6 présente un schéma de la structure interne de ce FPGA.

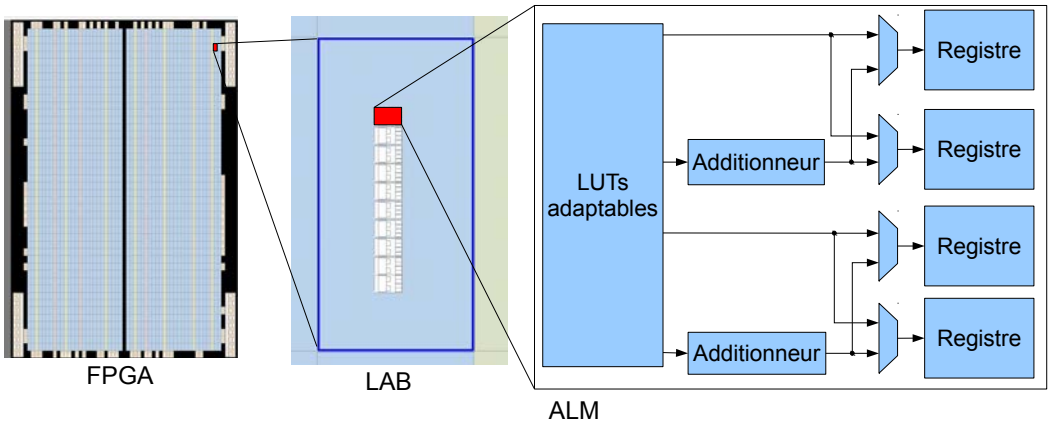

FIGURE A.6 – Structure interne d'un FGPA Altera Cyclone V.

Malheureusement, il n'existe pas d'équivalent direct au *hard macro* pour les FPGA Altera, ce qui rend l'implantation des 256 cellules TERO plus longue, car il est nécessaire d'instancier 256 fois la cellule TERO et donc de placer tous les éléments de toutes les cellules TERO dans le fichier de contrainte (qsf). L'outil Altera qui se rapproche le plus des *hard macros* s'appelle *LogicLock* mais il permet uniquement de dupliquer un bloc une fois. En revanche, la création d'un tel bloc assure au concepteur qu'il n'y aura ni ressources logiques, ni ressources de routage, d'une autre partie du système dans le bloc "verrouillé".

Étape 1 : Écriture du code VHDL de la boucle TERO

Plusieurs composants présents dans la libraire *altera.mf* peuvent être utilisés pour concevoir la cellule TERO. Les premiers permettent de délimiter les frontières d'une LUT. Ils s'appellent *lut_input* et *lut_output*. Le composant *lut_input* représente une et une seule entrée d'une LUT et le composant *lut_output* représente une sortie d'une LUT. L'équation logique contenu dans la LUT est directement appliquée sur les signaux de sortie des composants *lut_input*, instanciés dans le code VHDL, et le résultat de cette équation est envoyé à l'entrée du composant *lut_output*. Il est donc en théorie possible de concevoir la cellule TERO uniquement

à partir de ces deux composants. La figure A.7 montre schématiquement comment instancier les portes *NON* (a) et *ET* (b) qui constituent la cellule TERO.

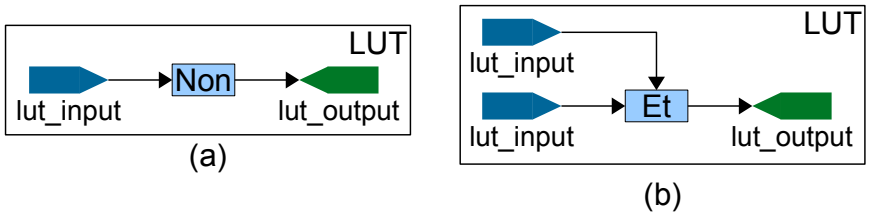

FIGURE A.7 – Schéma de l'utilisation des composants *lut_input* et *lut_output*.

Malheureusement, l'outil de synthèse fourni par Altera semble ne pas tenir compte des contraintes attachées à ce type de composants et optimise systématiquement l'ensemble du circuit. Pour palier ce problème, il faut utiliser d'autres composants de la librairie Altera appelés *LCELL*. Une *LCELL* représente un délai et n'est pas supprimé lors de la synthèse du circuit. Ainsi, la cellule TERO conçue pour les FPGA est composée d'une porte *ET* et d'une seule porte *NON* ainsi que d'une chaine de délais entre les deux portes pour chaque branche (Figure A.8). Afin de comparer la TERO-PUF sur les deux familles de FPGA, il est nécessaire que les cellules implantées dans chaque famille aient la même configuration. Pour cela, il faut que la cellules TERO implantée sur FPGA Altera Cyclone V possèdent une porte *ET* ainsi que sept délais par branche. Nous avons remarqué au cours des expériences d'implantation de la cellule TERO sur cette famille que la *LCELL* située juste après la porte *ET* ainsi que celle située juste avant la porte *NON* sont automatiquement fusionnées avec les deux portes logiques. Ces portes logiques occupent donc les deux LUT d'un ALM. Ainsi, pour avoir exactement le même nombre de délais dans chaque branche que pour la cellule TERO conçue pour les FPGA Xilinx Spartan 6, il faut que celle pour les FPGA Altera Cyclone V comporte deux branches composées de : une porte *ET*, une porte *NON* et huit *LCELL*. La Figure A.8 montre la structure de la cellule TERO conçue pour les FPGA Altera Cyclone V.

Étape 2 : Placement des éléments de la cellule TERO

Pour placer chaque porte et chaque délai de la cellule TERO, des contraintes de placement sont ajoutées au fichier de contrainte : *quartus setting file* (qsf). Toutefois, comme les portes logiques sont automatiquement fusionnées avec une *LCELL* même si un placement leur est attribué, il n'est pas nécessaire de les placer. Seules les *LCELL* sont donc à placer pour contraindre le placement de la cellule TERO. Il se fait avec la syntaxe suivante :
*set_location_assignmentLABCELL_X***xx***_Y***yy***_N***nn***−to*"**LCELL à placer**"

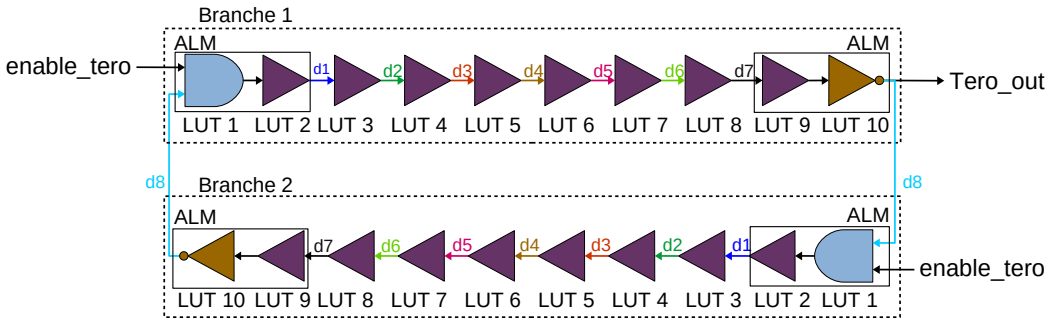

FIGURE A.8 – Schéma de la structure de la cellule TERO implantée sur FPGA Altera Cyclone V.

Compte tenu de la structure du FPGA et du nombre d'éléments à placer pour une cellule TERO, plusieurs placements sont possibles. En effet, un seul LAB contient exactement vingt LUT et il faut placer seize *LCELL* pour concevoir une cellule. Il est donc possible d'utiliser un seul LAB pour implanter la cellule TERO. Cependant, il est également possible d'utiliser un LAB par branche de la cellule. Ces deux LAB peuvent être situé l'un au dessus de l'autre ou l'un à côté de l'autre.

Étape 3 : Vérification des délais de la cellule TERO

Afin de vérifier si un placement de la cellule TERO satisfait la contrainte sur les délais, l'outil *Quartus Timing Quest Analyzer* fourni dans la suite de développement Altera est utilisé. Cet outil permet d'estimer les délais de routage entre chaque élément du circuit. Ainsi, il est possible de récupérer ceux de la cellule TERO. Après avoir testé toutes les différentes combinaisons de placement utilisant un seul LAB, nous avons constaté qu'aucun placement ne satisfait la contrainte sur les délais et la différence entre les deux branches n'est pas négligeable : supérieure à $1ns$.

Deux LAB doivent donc être utilisés pour placer correctement la cellule TERO dans les FPGA Altera Cyclone V. Après avoir testé les différentes combinaisons de placement possibles utilisant deux LAB, la configuration qui a été retenue possède une différence de 0.35ns entre les deux branches de la cellule. Les délais présents entre chaque élément de la cellule TERO sont rapportés dans le Tableau A.2. Dans ce tableau, les noms des délais correspondent à ceux inscrits sur la Figure A.8. Cette configuration utilise deux LAB côte à côte avec les *LCELL* placées comme sur la Figure A.9.

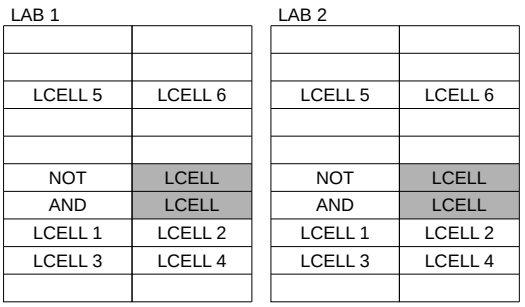

FIGURE A.9 – Configuration interne utilisée pour la cellules TERO dans les FPGA Altera Cyclone V. Les cellules grisées représentent les LCELL fusionnées avec les portes logiques *et* et *non*.

Tableau A.2 – Délais des connections de la cellule TERO implantée sur FPGA Altera Cyclone V et estimés par le logiciel *Timing Quest Analyzer*.

connection	délai dans la branche 1 (ns)	délai dans la branche 2 (ns)
$d1$	0.143	0.143
$d2$	0.352	0.352
$d3$	0.230	0.230
$d4$	0.494	0.494
$d5$	0.143	0.143
$d6$	0.352	0.352
$d7$	0.230	0.230
$d8$	0.626	0.626

Étape 4 : Implantation de toutes les cellules et création d'une *LogicLock Region*

Maintenant que la cellule TERO a été conçue pour les FPGA Altera Cyclone V, il est impératif d'instancier le nombre de cellules qui doivent être implantées pour la caractérisation de la TERO-PUF. En effet, comme il n'existe pas d'équivalent direct au *hard macro* pour les FPGA Altera, il faut placer manuellement chaque *LCELL* de chaque cellule TERO implantée. Avec 256 cellules, cela signifie qu'il faut placer 4096 *LCELL*. Un script lancé sur le logiciel R a été utilisé pour créer automatiquement le fichier de contrainte contenant le placement de toutes les cellules implantées. Ce fichier à ensuite été inclus dans le fichier de contrainte complet du projet contenant la TERO-PUF.

Enfin, une *LogicLock Region* a été créée autour de l'ensemble des cellules TERO afin de s'assurer que l'outil de synthèse fourni par Altera n'ajoute aucun élément dans les parties non utilisées des LAB occupés par les cellules TERO.