Sécurité des systèmes sur puce complexes hétérogènes

El Mehdi Benhani

N° d'ordre NNT : 2020LYSES016

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
Opérée au sein de
L'UNIVERSITE JEAN MONNET

Ecole Doctorale N° 488
Sciences Ingénierie Santé

Spécialité de doctorat :
Microélectronique

Soutenue publiquement le 06/07/2020, par :
El Mehdi Benhani

Sécurité des systèmes sur puce complexes hétérogènes

Devant le jury composé de :

DANGER, Jean-Luc	Professeur à Telecom Paris	Président
LAGADEC, Loïc	Professeur à l'ENSTA Bretagne	Rapporteur
HELY, David	Maître de Conférence HDR à l'INP Grenoble	Rapporteur
LE GAL, Bertarnd	Maître de Conférence à l'IPB	Examinateur
FRESSE, Verginie	Maître de Conférence HDR à l'UJM	Examinatrice
AUBERT, Alain	Maître de Conférence à l'UJM	Encadrent de thèse
BOSSUET, Lilian	Professeur à l'UJM	Directeur de thèse

Remerciements

La vie est un roman composé de plusieurs chapitres. Ma thèse fait partie des beaux chapitres de mon livre. Ses pages racontent une étape de ma vie qui a duré quatre ans, quatre ans de dur labeur, quatre ans de hauts et de bas. La thèse est une panoplie d'émotions, on jongle entre stress et fatigue, puis joie et fierté ! Ce qui rend cette étape encore plus spéciale, c'est les opportunités qu'elle a pu m'offrir. La rencontre de nouvelles personnes et la découverte de nouvelles villes notamment Nancy, Montpelier, Munich, Tanjin, Londre, une main sur le mur de Berlin et un pied sur la muraille de chine, sans oublié la découverte de nouveaux sports comme le ski ou la plongé, et j'en passe.

Ce chapitre de ma vie avait été élaboré avec l'aide et le soutien de plusieurs personnes indispensables que je tiens à remercier infiniment.

Tout d'abord, j'exprime ma très grande reconnaissance au Professeur Lilian Bossuet, qui a dirigé mes travaux avec attention, rigueur et bienveillance. Ses conseils, critiques et recommandations formulés ont soutenu la réalisation de ce travail. Je le remercie également de m'avoir laissé assez d'autonomie pour que je puisse me développer pleinement dans mes quatre ans de thèse.

Je remercie également l'ensemble des membres de l'équipe SESAM pour leur convivialité, leur dynamisme et leurs pauses café qui ont laissé des traces sur les feuilles de mon chapitre. Je les remercie également pour leur accompagnement durant mes premières glisses à la station de ski Chalmazel.

Je tiens à remercier M. Cuauhtemoc Mancillas et M. Cédric Marchand, pour leur encadrement et leurs conseils qui m'ont aidé à développer des parties de cette thèse.

Je tiens à remercier M. Loïc Lagadec et M. David Hély d'avoir accepté d'être les rapporteurs de cette thèse et pour la qualité de leurs retours. Je remercie M. Jean-Luc Danger qui a accepté

de présider le jury ainsi que M. Bertarnd Le gal et Mme Verginie Fresse qui ont montré tout l'intérêt qu'ils portaient à mes travaux en participant à ce jury.

Je remercie aussi toutes les personnes que j'ai rencontré à Saint-Etienne (la plus belle ville au monde) et avec qui j'ai partagé des moments inoubliables dans cette ville.

Un grand merci à ma femme Sara, qui est d'ailleurs. Je l'ai rencontré durant ma dernière année de thèse, elle m'a beaucoup soutenu et aidé durant la rédaction. Je la remercie de tout mon cœur, « je t'aime ma puce ».

Pour finir, je remercie ma famille qui ont cru en moi et qui m'ont soutenue dès mes premiers pas dans ce monde, je remercie fortement ma mère qui a fait des grands sacrifices pour m'aider à continuer mes études supérieures en France et me convaincre de ne pas abandonner. Je remercie aussi ma sœur avec qui j'ai partagé des bons souvenir tout le long de ces années. Merci beaucoup ma sœur.

Table des matières

Remerciements i
Table des matières iii
Table des figures vi
Introduction 1
Chapitre 1 : Les SoCs complexes hétérogènes et leur sécurité 8
1. Les SoCs complexes hétérogènes 9
1.1. Le bus à l'interface logiciel/matériel dans un SoC complexe hétérogène: 10
1.2. La hiérarchie mémoire dans un SoC complexe hétérogène 13
1.3. Le système de gestion d'énergie dans les SoCs complexes hétérogènes 17
2. La technologie ARM TrustZone dans les SoCs complexes hétérogènes 18
2.1. La technologie ARM TrustZone dans un cœur du processeur ARM 18
2.2. Extension de la technologie ARM en dehors des cœurs du processeur ARM 20
2.3. Le déploiement de la technologie ARM TrustZone dans la partie reconfigurable d'un SoC complexe hétérogène 22
3. Etat de l'art des attaques ciblant les SoC complexes hétérogènes 24
3.1. Les attaques visant le Secure boot 24
3.2. Les attaques logicielles visant le système d'exploitation de confiance (TEE) 26
3.3. Les attaques matérielles visant le TEE 26
4. Conclusion 31
Chapitre 2 : Les attaques matérielles visant la sécurité du bus 32
1. Rappel 33
2. L'utilisation de la technologie TrustZone dans la partie reconfigurable 33
3. La mise en œuvre des attaques matérielles visant la sécurité du bus AXI 36
3.1. Compromettre les signaux de sécurité du bus AXI 37
3.2. Compromettre les signaux de réponse BRESP/RRESP 40

3.3. Compromettre l'AXI Interconnect 42
4. Scénario complet d'attaque visant l'outil CAO Vivado de Xilinx 45
4.1. Fichiers de descriptions matérielles dans un projet Vivado 45
4.2. Scénario d'attaque complet 46
5. Conclusion : 47
Chapitre 3 : Attaques DMA et par manipulation malicieuse du DVFS 49
1. Design expérimental 50
2. Attaque DMA 51
3. Attaques par canal dérobé basé sur l'utilisation malicieuse du système DFS 52
3.1. Preuve de concept #1 : Transfert de données sensibles du cœur ARM sécurisé vers l'extérieur du SoC 53
3.2. Preuve de concept #2 : Transfert de données sensibles du cœur sécurisé vers le cœur non-sécurisé 57
3.3. Preuve de concept #3 : Transfert des données sensibles du cœur ARM sécurisé vers l'IP non-sécurisée 59
3.4. Preuve de concept #4 : Transfert de données sensibles de l'IP sécurisée vers le cœur ARM non-sécurisé 61
4. Conclusion : 63
Chapitre 4 : Exploitation malicieuse du mécanisme de cohérence de la mémoire cache dans un SoC complexe hétérogène 64
1. La cohérence de la mémoire cache depuis la partie reconfigurable dans un SoC complexe hétérogène 65
1.1. Les signaux AxCache[3:0] et AxUser[4:0] 66
1.2. Le signal AxPort[2:0] 66
2. Les éléments de l'attaque 67
2.1. Distinguer un échec d'un succès de la mémoire cache 67
2.2. Opération de maintenance de la mémoire cache 71
3. Scénario d'attaque 73
3.1. Attaque par canal auxiliaire 74
3.2. Attaque par canal dérobé 78
4. Conclusion : 80
Chapitre 5 : Contre-mesures 81
1. Protection #1 : Isolation des IP sécurisées de la partie reconfigurable du SoC 82

2. Protection #2 : Vérification des règles de conception 84
3. Protection #3 : Chiffrement des transactions du bus AXI 86
3.1. Architecture logicielle et matérielle du système de chiffrement du bus AXI 86
3.2. Le chiffrement léger de flux (Trivium) 87
3.3. Performances des implémentations logicielles et matérielles de Trivium 88
3.4. Discussion 90
4. Conclusion 91
Conclusion et perspectives 92
Publications et communications 94
Références bibliographies 95

Table des figures

Figure 1 : Trois réalisations matérielles possibles d'une TEE pour un système embarqué [21] 4
Figure 2 : Architecture d'un SoC complexe hétérogène, a- Architecture du SoC Xilinx Zynq Ultrascale+, b- Architecture du SoC Intel Stratix 10 SX 10
Figure 3: Structure d'un bus 11
Figure 4: Les canaux de communication du bus AXI entre une interface maître et une interface esclave 11
Figure 5 : Représentation schématique de la hiérarchie mémoire dans un SoC complexe hétérogène, tels que les SoC Xilinx Zynq-7000 et Intel Cyclone V 13
Figure 6 : Les unités de gestion de mémoire dans un SoC FPGA 14
Figure 7 : MMU, processus de traduction d'une adresse virtuelle en adresse physique 15
Figure 8 : Structure d'une matrice de cellules mémoire (Bank) DRAM 16
Figure 9 : Décodage d'une adresse physique dans le SoC Zynq-7000 17
Figure 10 : Anatomie d'un CMOS 17
Figure 11: Illustration d'un AXI Interconnect couplant des interfaces maître et esclave 24
Figure 13: Design créé avec l'outil Vivado de Xilinx 34
Figure 14: Commande TCL Activation de la propagation des signaux de sécurité à la partie reconfigurable 35
Figure 15: Commande TCL malveillante pour changer l'état de sécurité de l'interface AXI_M00 à non-sécurisée 36
Figure 16 : Communication entre interface maître et esclave 37
Figure 17 : accéder aux ressources sécurisées depuis le monde non-sécurisé 37
Figure 18: Quatre exemples de modifications matérielles malicieuses ciblant le signal de sécurité AWPROT[1] 38
Figure 19: Deux commandes TCL à ajouter au script pour modifier automatiquement et discrètement le code RTL 40
Figure 20: Exemple de modification matérielle malveillante du signal BRESP 41
Figure 21: Commande TCL pour modifier le vecteur d'initialisation d'une LUT. 41
Figure 22: Modification malveillante du code Verilog du Crossbar de l'AXI Interconnect 43
Figure 23: Commande TCL malveillante pour changer l'état de sécurité de l'interface ''M00'' 44
Figure 24: L'insertion d'une FIFO malveillante au niveau de l'AXI Interconnect pour espionner les données envoyées vers une IP sécurisée 45
Figure 25: Design créé à l'aide de l'outil Vivado 46
Figure 26: La description hiérarchique du design de la figure 25 46
Figure 27: Fichier accessible en lecture seulement 46
Figure 28: L'emplacement du fichier cible 47
Figure 29: Design expérimental 50
Figure 30: Attaque DMA 51

Figure 31: Transfert des données sensibles vers l'extérieur du SoC FPGA 54
Figure 32: Analyse spectrale en temps réel de la fuite électromagnétique de données secrètes 54
Figure 33: Décodage des données sur l'écran de l'analyseur de spectre en temps réel, a - bande passante = 1,42.105 bps, b - bande passante = 3,33.105 bps 56
Figure 34: Transfert des données sensibles du cœur ARM sécurisé vers le cœur ARM non-sécurisé dans SoC FPGA 57
Figure 35: Transfert des données sensibles du cœur ARM sécurisé vers l'IP non-sécurisés dans un SoC FPGA 60
Figure 36: Décodage des données reçues 61
Figure 37: Transfert des données sensibles de l'IP sécurisée vers le cœur ARM non-sécurisé dans un SoC FPGA 62
Figure 38: Manipulation de l'horloge par accès direct à la mémoire externe via l'interface ACP 62
Figure 39: Un échec et un succès d'une requête de lecture cohérente 69
Figure 40: Histogramme des échecs et des succès d'une requête de lecture cohérente 69
Figure 41: L'effet de la fréquence sur le seuil 70
Figure 42: Histogramme des échecs et des succès avec deux politiques de remplacement différentes (aléatoire, round-robin) 71
Figure 43: Schéma de l'attaque par canal auxiliaire 75
Figure 44: Attaque Flush+Reload exploitant la cohérence de cache, ... 76
Figure 45: Attaque Evict+Time depuis la partie reconfigurable $k0 = 0x51$ 77
Figure 46: le décodage de l'information dérobée 79
Figure 47: L'impact de la politique de remplacement sur la bande passante 80
Figure 48: L'isolation des IP sécurisées dans la partie reconfigurable du SoC 82
Figure 49: Vérification des règles de conception, a- design à vérifier , b- connexion corrompu entre l'interface M00_AXI de l'IP ps7_0_axi_periph et l'interfaces S_AXI de l'IP axi_gpio_0 85
Figure 50: Architecture de la crypto IP matérielle implantée dans la partie reconfigurable du SoC 87
Figure 51: Temps nécessaire pour chiffrer 1 Méga-octets de données avec les quatre implémentations de Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits), l'AES-128 et l'AES t-table exécutées sur la partie processeur du SoC cadencée à 650 MHz. 89
Figure 52: Performance des quatre implémentations matérielles de l'algorithme de chiffrement de flux Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits) dans la partie reconfigurable du SoC, a- nombre de bascules D nécessaires à l'implémentation matérielle, b- nombre de LUT nécessaires à l'implémentation matérielle 90

Introduction

Contexte actuel

Aujourd'hui, les objets connectés sont omniprésents, notamment dans les applications critiques (véhicules autonomes, aéronautique, équipements médicaux, militaires, etc.) ou les applications de sécurité (contrôle d'accès, etc.). Leur nombre est en croissance continue, il dépassera les 26 milliards objets en 2020 selon les prévisions de Gartner. Les objets connectés stockent et échangent une grande quantité de données utilisateur, y compris des informations confidentielles (photos personnelles, répertoire téléphonique, etc.). Ces données attirent l'attention des attaquants qui exploitent en permanence les vulnérabilités de ces systèmes, leurs actions peuvent avoir de lourdes conséquences. Elles peuvent causer des pertes financières et nuire à la réputation d'une entreprise. Par exemple, les deux attaques découvertes en début 2018, Spectre [1] et Meltdown [2], ont causé des pertes financières chez plusieurs fournisseurs de systèmes-sur-puce. Elles peuvent aussi mettre en danger la vie des gens. Par exemple, la prise de contrôle à distance de la voiture électrique Tesla [3] peut causer un accident très grave.

La sécurité des objets connectés est un enjeu primordial. Cette tâche constitue un défi de taille pour les industriels. Les objets connectés sont complexes car ils combinent des fonctionnalités logicielles sophistiquées et des ressources matérielles très hétérogènes, telles que les SoCs (System-on-Chip, système-sur-puce en français). Cette complexité rend parfois impossible le test et la vérification de l'intégrité de l'ensemble du système. La conception d'un système embarqué s'appuie sur des composants, logiciels et matériels, fournis par des tiers qui ne sont pas toujours fiables et qui peuvent dissimuler des éléments malveillants (chevaux de Troie, portes dérobées) afin de les utiliser pour attaquer le système. La conception d'un système embarqué est également fortement contrainte à la réduction des coûts en surface et de la consommation d'énergie, ce qui complique le choix d'une solution de sécurité car la majorité des solutions sont coûteuses.

Les attaques visant les systèmes embarqués sont de deux types, les attaques physiques et les attaques logicielles :

- **Les attaques physiques** sont divisées en deux classes :
    - Les attaques par observation : se contentent de mesurer des paramètres physiques du matériel pendant son fonctionnement normal. Exemples d'attaque : Les attaques par analyse de consommation, comme l'analyse différentielle de consommation (DPA) [4] et l'analyse du courant par corrélation (CPA) [5].
    - Les attaques par perturbation : cherchent à modifier les conditions physiques dans lesquelles l'exécution a normalement lieu afin de provoquer un calcul incorrect et de changer ainsi le comportement du programme. Exemples d'attaque : Les attaques par injection de fautes par variation de l'alimentation [6], par l'utilisation d'un faisceau laser [7]–[9] ou par la création d'ondes électromagnétiques intenses à proximité du circuit [10], [11].

Par le passé, de nombreuses études ont été menées sur la protection des SoCs et elles ont proposé des contremesures comme la réduction du signal [12], [13] et l'augmentation de bruit [14], [15] pour protéger contre les attaques par observation. L'utilisation de capteurs physiques [16] pour la détection de faute et le contrôle de flot d'exécution [17] pour protéger contre les attaques par perturbation. Cette thèse ne se focalise pas sur l'étude de la protection des SoCs mais plutôt sur l'exploration de nouveaux chemins d'attaques dans les SoCs complexes et hétérogènes.

- **Les attaques logicielles** ou les attaques par infections informatiques (logiciels ou codes malveillants) exploitent de nombreuses failles, comme les erreurs de conception matérielle ou les vulnérabilités logicielles (buffer overflow, stack overflow, etc.), dans le but de récupérer discrètement de l'information, ou de perturber le fonctionnement du système, ou de supprimer des données, etc. Exemples d'attaque : Les attaques Spectre [1] et Meltdown [2] qui exploitent une faille de sécurité dans l'exécution spéculative des instructions de certains processeurs.

L'une des solutions pour protéger les systèmes embarqués contre les attaques logicielles consiste à séparer et isoler les applications nécessitant de la sécurité (gestion de mots de passe, services cryptographiques, etc.) des applications générales. Pour ce faire, un environnement

d’exécution de confiance (en anglais Trusted Execution Environment, TEE) peut être mis en œuvre.

Les environnements d’exécution de confiance (TEE)

Le concept de TEE est normalisé par l’organisme GlobalPlatform, il s’agit de faire cohabiter deux environnements d’exécution dans un même système avec des propriétés de sécurité différentes [18]. Pour cela, l'exécution des applications logicielles s'effectue dans deux mondes distincts : le monde sécurisé et le monde non-sécurisé. Le monde sécurisé est utilisé pour protéger les applications et les données sensibles des applications du monde non-sécurisé. Les deux mondes ne sont pas symétriques, le monde sécurisé a un accès absolu à toutes les ressources du système, tandis que le monde non-sécurisé a un accès restreint aux ressources nécessaires à son fonctionnement et son accès direct aux données sécurisées est rejeté. GlobalPlatform a défini des interfaces de programmation [19] qui permettent au monde non-sécurisé de lancer des services sécurisés qui s’exécuteront dans le monde sécurisé et d’échanger avec ce dernier.

Architectures support d’un TEE

Le TEE s’appuie sur des ressources matérielles dédiées qui permettent de protéger les applications et les données du monde sécurisé. La figure 1 illustre trois possibilités de réalisation d'un TEE. Ces possibilités diffèrent en ce qui concerne la position du support matériel par rapport au système-sur-puce. La première solution, Figure 1-a, consiste à utiliser un coprocesseur extérieur au SoC pour une exécution physiquement isolée de la TEE. Le coprocesseur est réalisé par un élément sécurisé matériel externe (Secure Element) qui peut être une carte à puce ou un module matériel sécurisée (un Trusted Platform Module ou TPM en anglais) regroupant des ressources cryptographiques dans un environnement sécurisé vis-à-vis des attaques logicielles et matérielles. Une autre solution intermédiaire, Figure 1-b, consiste à intégrer l'élément sécurisé dans le SoC en tant qu'entité isolée des autres ressources. Enfin, la dernière solution, figure 1-c, consiste à utiliser une technologie d’isolation matérielle comme la technologie ARM TrustZone [20]. La technologie d’isolation matérielle permet de renforcer l’isolation entre les deux mondes et protéger les applications sécurisées contre les attaques provenant des applications générales. C’est cette technologie et son extension dans un SoC complexe et hétérogène (SoC qui regroupent des processeurs ARM et des FPGAs) qui est au cœur des travaux de cette thèse.

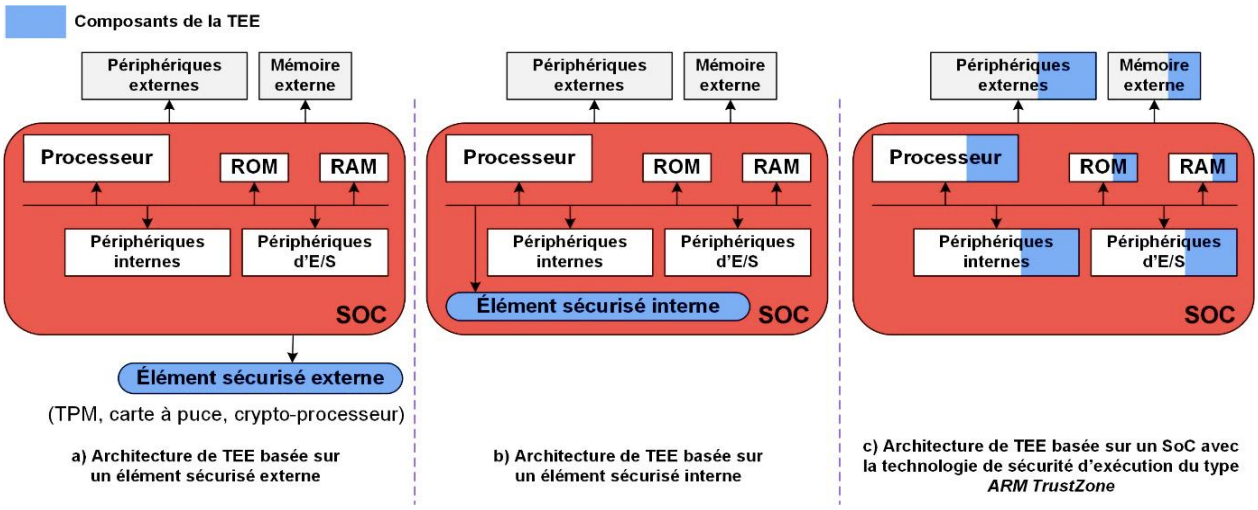


Figure 1 : Trois réalisations matérielles possibles d'une TEE pour un système embarqué [21]

La technologie ARM TrustZone

La technologie ARM TrustZone est largement disponible sur les appareils mobiles courants. Elle renforce l'isolation entre le monde sécurisé et le monde non-sécurisé en utilisant des contrôleurs matériels qui protègent l'accès à la mémoire, aux périphériques et d'autres éléments du système. Les contrôleurs matériels rendent inaccessibles les données du monde sécurisé au monde non-sécurisé ce qui permet de protéger les applications et les données sensibles. Elle fournit les bases de sécurité pour des TEEs, comme Samsung Knox [22], Qualcomm QSEE [23], Trustonic Kinibi [24], SierraTEE [25], OP-TEE [26], OPEN-TEE [27] et TOPPERS de SafeG [28].

La technologie ARM TrustZone est une solution efficace contre les attaques logicielles provenant du monde non-sécurisé. Malheureusement, cette technologie est restée dans un état d'obscurité relative pendant un certain temps [29]. Les fabricants de systèmes sur puce intégrant cette technologie étaient quelque peu réticents à la divulgation de détails techniques, obligeant souvent les développeurs à signer des accords de non-divulgation avant que des détails architecturaux puissent leur être communiqués. Les recherches impliquant la technologie ARM TrustZone ont également été ralenties par la disponibilité limitée de plates-formes de développement sur lesquelles toutes les fonctionnalités de technologie ARM TrustZone ont été déverrouillées.

Aujourd'hui, la technologie ARM TrustZone attire l'attention des académiques et des industriels. Les fabricants de SoC hétérogènes (regroupant des processeurs ARM et des FPGA), comme Xilinx et Intel, encouragent la recherche et le développement en publiant les détails

techniques de leurs implémentations de la technologie ARM TrustZone et en vendant des cartes de développement avec un accès à toutes les fonctionnalités de la technologie ARM TrustZone. **Les travaux présentés dans ce manuscrit étudient les propriétés de sécurité des SoCs hétérogènes utilisant la technologie ARM TrustZone et mettent en lumières les vulnérabilités importantes qui nécessitent des recherches supplémentaires pour concevoir des solutions efficaces. Ces travaux ont été réalisés dans le cadre du projet collaboratif FUI TEEVA.**

Le projet TEEVA

Les travaux présentés dans ce manuscrit ont été réalisés dans le cadre du projet TEEVA[1] (Trusted Execution Environment eVAluation, projet français FUI n ° 20). Le projet TEEVA avait pour mission d'évaluer la sécurité de deux solutions utilisées dans les plates-formes mobiles pour la protection des applications sensibles et des données privées. La première solution est purement logicielle, la cryptographie à boîte blanche (cryptographie WhiteBox) [30] qui repose sur l'obscurcissement de la clé cryptographique en la stockant, non seulement sous forme de données mais aussi sous forme de code, avec des données aléatoires instanciées au moment de la compilation dans une composition à partir de laquelle il est difficile de déduire la clé d'origine. La deuxième solution étudiée est une combinaison de composants logiciels et matériels, le TEE basé sur l'utilisation de la technologie d'isolation matérielle, la technologie ARM TrustZone, qui est disponible sur la plupart des périphériques mobiles actuels.

Les partenaires du projet TEEVA étaient :

- La société Gemalto (coordinateur), qui fournit des solutions logicielles et matérielles de sécurité
- La société Trustonic, qui développe et fournit une solution de TEE
- La société Phonesec, qui fait du conseil sur la sécurité des systèmes de communication
- Le laboratoire d'informatique, de robotique et de microélectronique de Montpellier
- L'école des Mines de Saint-Étienne
- Le laboratoire Hubert Curien

[1] https://www.pole-scs.org/projets/teeva/

Chacun de ces partenaires avait comme objectif l'analyse précise du niveau de sécurité d'une des deux solutions ou des deux dans son domaine d'expertise.

Objectifs et contributions de la thèse

Le premier objectif de cette thèse est la mise en place d'un environnement d'exécution de confiance dans un SoC FPGA embarquant la technologie ARM TrustZone. Le deuxième objectif est l'évaluation de la technologie ARM TrustZone en réalisant des attaques matérielles depuis la partie reconfigurable d'un SoC FPGA. Enfin, le troisième objectif est la proposition de contre-mesures d'attaques réussies afin de les tester et d'arriver à des solutions parfaitement caractérisées du point de vue de la sécurité. Ces objectifs sont fixés par le consortium du projet TEEVA au laboratoire Hubert Curie, mais les contributions de cette thèse sont au-delà de ces objectifs.

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

- L'implémentation documentée d'un TEE dans un SoC FPGA, dont les sources sont mises à disposition de la communauté.
- L'analyse de la sécurité de l'extension de la technologie ARM TrustZone dans un SoC hétérogène et la mise en lumière de failles de sécurité depuis la partie FPGA du SoC. Cela passe par la réalisation d'attaques exploitant des modifications malveillantes de l'architecture de la partie reconfigurable du SoC-FPGA. Ces attaques visent la sécurité des bus de communication internes au SoC et la corruption de l'exécution d'application dans le monde sécurisé.
- La réalisation d'attaques ciblant la zone de la mémoire réservée au monde sécurisé depuis une IP malveillante embarquée dans la partie reconfigurable (FPGA) du SoC.
- La mise en évidence d'un nouveau canal auxiliaire au sein du SoC exploitant le système de management de la consommation de la puissance du SoC (DVFS, en anglais Dynamic Voltage and Frequency Scaling). Ce canal permet la communication interne et externe de données entre des entités du SoC non-autorisées à communiquer par le système.
- La preuve de concept de la réalisation d'attaques par canaux cachés basées sur la cohérence de cache entre la partie reconfigurable (FPGA) et un cœur ARM du SoC.

- Enfin, l'étude du chiffrement matériel des communications interne au SoC à l'aide d'algorithmes de chiffrement légers.

Organisation du manuscrit

Outre cette introduction et la conclusion finale, ce document est composé de 4 chapitres.

- Le chapitre 1 présente le contexte des travaux présentés dans ce manuscrit. Il présente aussi un état de l'art des attaques qui visent la sécurité des SoCs embarquant la technologie ARM TrustZone, des attaques qui visent à compromettre la chaine de confiance dans son démarrage (attaque visant le Secure boot), et des attaques exploitant des vulnérabilités dans l'architecture de mémoire ou la technique DVFS.
- Le chapitre 2 présente des attaques matérielles visant la sécurité du bus AXI dans un SoC complexe hétérogène embarquant la technologie TrustZone, comme le SoC Xilinx Zynq-7000. Ces attaques sont mises en place durant la phase de conception en exploitant soit un script TCL malicieux, soit une IP tiers malicieuse, soit un outil CAO de non de confiance.
- Le chapitre 3 présente une attaque DMA visant les régions sécurisées de la mémoire externe et les registres de configurations dédiés à la technologie ARM TrustZone. Le chapitre présente aussi des attaques exploitant technique DFS pour faire de la modulation de fréquence et dérober les données sensibles du monde sécurisé.
- Le chapitre 4 présente une exploitation malicieuse de la cohérence de cache depuis la partie reconfigurable. Le chapitre introduit deux méthodes à utiliser depuis la partie reconfigurable pour mettre en place une attaque de cache. Les deux méthodes sont une méthode pour mesurer le temps d'accès à une donnée cohérente et une autre pour expulser une donnée cohérente de la mémoire cache. Le chapitre présente aussi des attaques de cache contre un algorithme cryptographique et une attaque par canal dérobé.
- Le chapitre 5 présente des solutions pour améliorer la sécurité des SoC FPGA embarquant la technologie ARM TrustZone. Il présente une méthodologie d'isolation des IP sécurisées, mais aussi un vérificateur de règle de conception et une étude de l'utilisation d'un chiffrement de flux pour sécuriser les transactions dans le bus AXI.

Chapitre 1 : Les SoCs complexes hétérogènes et leur sécurité

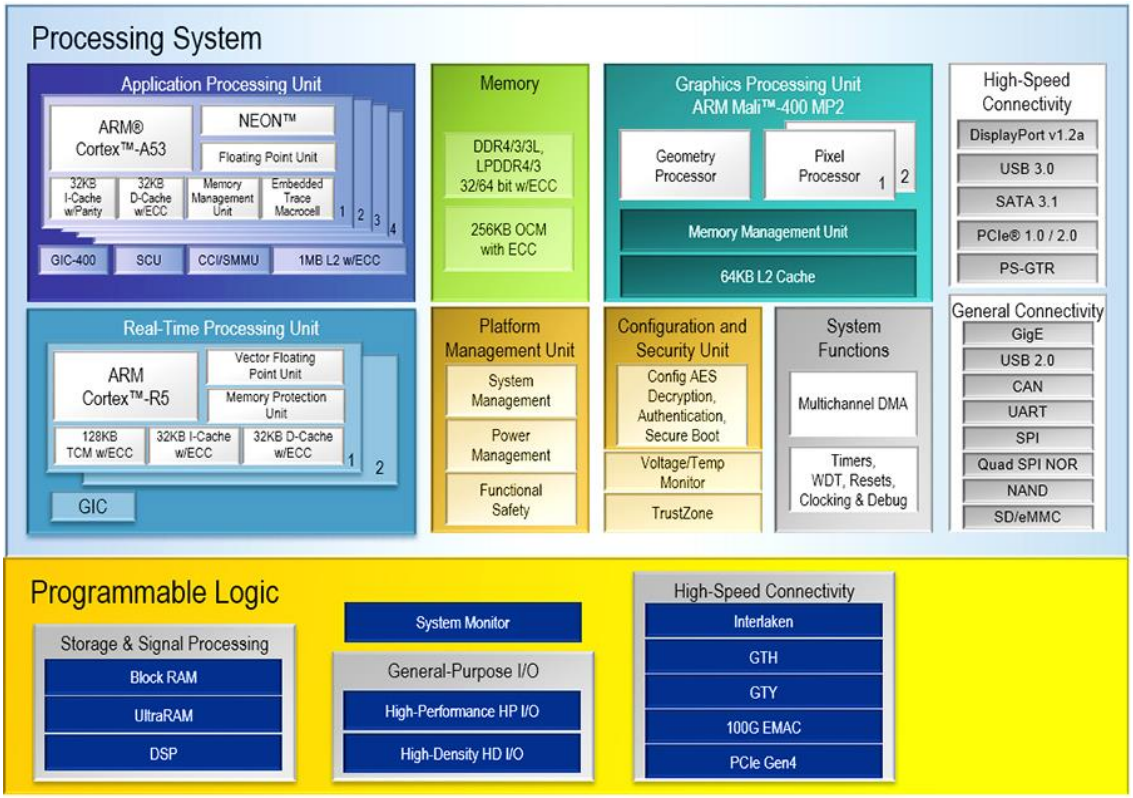
Ce chapitre présente le contexte des travaux présentés dans ce manuscrit. Au travers de celui-ci, la complexité des SoCs hétérogènes modernes, leurs architectures de bus et celles de leurs mémoires sont présentées. La technique du DVFS (Dynamic Voltage and Frequency Scaling) utilisée pour réduire la consommation d'énergie et la technologie ARM TrustZone utilisée pour protéger les ressources et les applications critiques d'un système sont abordées également. Cette partie du chapitre peut sembler fastidieuse à lire, ce pendant elle contient l'ensemble des informations techniques nécessaires à la compréhension des attaques et des protections présentées dans la suite de ce chapitre et de tout le manuscrit.

L'objectif de cette thèse est d'évaluer la sécurité des SoCs embarquant la technologie ARM TrustZone. La fin de ce chapitre propose un état de l'art des attaques qui visent la sécurité des SoCs embarquant la technologie ARM TrustZone, des attaques qui visent à compromettre la chaine de confiance dans son démarrage (attaque visant le Secure boot), et des attaques exploitant des vulnérabilités dans l'architecture de mémoire ou la technique DVFS.

1. Les SoCs complexes hétérogènes

Un SoC complexe hétérogène est un ensemble de composants matériels et logiciels conçus et intégrés dans une seule puce électronique pour réaliser un ensemble de fonctionnalités. Typiquement, les SoCs complexes hétérogènes modernes intègrent (mais pas toujours) plusieurs processeurs de différentes natures (processeurs pour les opérations temps réel, processeurs pour les applications générales, …), plusieurs composants complexes et différents tels que des unités de calcul spécifiques programmables et/ou non programmables (DSP, ASICIP, FPGA), des composants de mémorisation variés, des périphériques E/S, et des réseaux de communication complexes (bus, NoC).

La figure 2 présente deux architectures de SoC complexe hétérogène des deux leaders mondiaux du marché des circuits embarquant une matrice de FPGA et un système multicœurs, Xilinx et Intel. La figure 2-a présente le SoC Xilinx Zynq Ultrascale+ [31] et la figure 2-b présente le SoC Intel Stratix 10 SX [32]. Les deux SoCs intègrent des composants de technologie différente, quatre processeurs pour les applications générales (Cortex-A53), différents type de mémoires, une partie reconfigurable (FPGA UltraScale) et de nombreuses interfaces.



a-

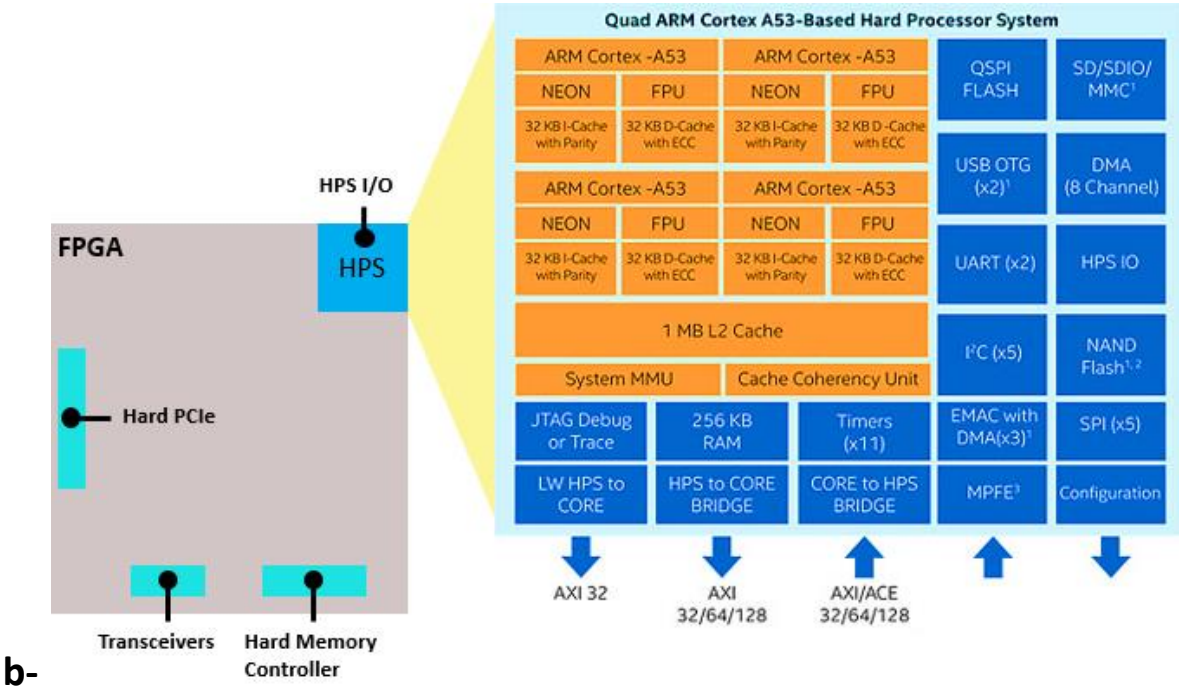


Figure 2 : Architecture d'un SoC complexe hétérogène, a- Architecture du SoC Xilinx Zynq Ultrascale+, b- Architecture du SoC Intel Stratix 10 SX

Aujourd'hui, les SoCs complexes hétérogènes sont de plus en plus performants avec une réduction de la consommation d'énergie, du coût en surface de silicium. Leur partie reconfigurable (FPGA) constitue un support de développement pour une grande variété d'applications très diverses car il est possible d'implémenter n'importe quel circuit puis de configurer le réseau de portes logiques pour changer le fonctionnement de circuit. Elle permet également d'accélérer d'un facteur important le traitement des données par rapport à un logiciel exécuté par les processeurs [33]. Pour ces avantages, les SoCs complexes hétérogènes sont très répandus dans plusieurs domaines, on les trouve dans le domaine grand public (smartphone, console de jeux, appareil photos, …), dans le domaine des transports (système de navigation, conduite autonome, …), dans le domaine médical (imagerie, opération robotisée, …), dans le domaine des télécommunications (mobile, routeur, satellite, …), dans l'industrie (commande, capteurs intelligents, robot, …), etc.

1.1. Le bus à l'interface logiciel/matériel dans un SoC complexe hétérogène:

Le bus à l'interface logiciel/matériel dans un SoC complexe hétérogène est un élément très important, il relie les divers composants du SoC, la mémoire, les processeurs, les périphériques E/S, …etc. Le bus relie les interfaces maîtres et esclaves des composants matériels du type IP

(certaine IP peuvent avoir plusieurs interfaces à la fois, comme l'$IP_{\#2}$ dans la figure 3). Les interfaces maîtres sont responsables d'initialiser des requêtes d'écriture ou de lecture destinées à des interfaces esclaves qui ont seulement le droit d'exécuter les requêtes. Le bus utilise un système d'ID pour identifier l'émetteur et le récepteur d'une requête et un protocole d'arbitrage pour gérer les communications et résoudre les conflits d'accès.

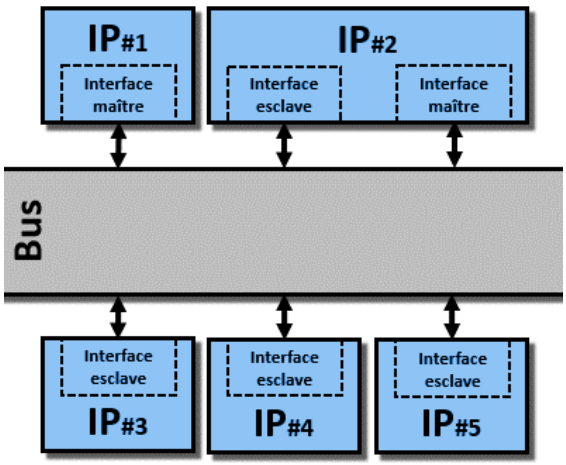


Figure 3: Structure d'un bus

Un SoC complexe hétérogène peut contenir plusieurs types de bus qui peuvent être différents au niveau du protocole utilisé, la largeur du bus, le débit, l'utilisation des signaux de control, etc. Le passage d'un type à un autre nécessite des convertisseurs de protocoles et des FIFOs. Parmi les types de bus, on trouve le bus AXI (Advanced eXtensible Interface) qui a vu le jour dans la spécification du standard AMBA3 (Advanced Microcontroller Bus Architecture 3), il est utilisé par Xilinx et Intel pour connecter certaines des IP matérielles intégrées dans leurs SoC. Le bus AXI est composé de 5 canaux qui connectent une interface maître et une interface esclave, comme illustré dans la figure 4 :

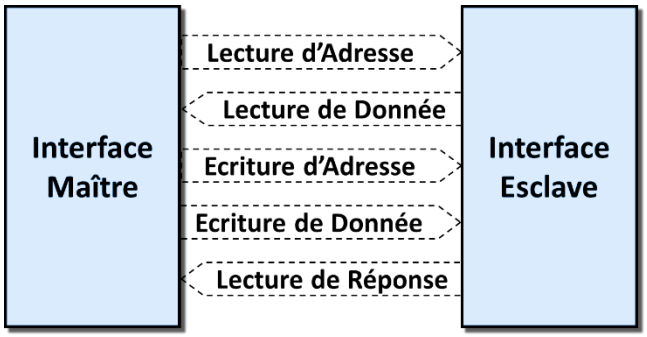


Figure 4: Les canaux de communication du bus AXI entre une interface maître et une interface esclave

C'est 5 canaux sont les suivants :

- **Le canal d'adresse de lecture** est utilisé par l'interface maître pour transmettre à l'interface esclave l'adresse à lire. Parmi les signaux utilisés dans ce canal, on trouve ARADDR[Largeur du bus d'adressage : 0], ARVALID, ARREADY, et ARPROT[2 : 0] qui sera détaillé dans la suite de ce chapitre.
- **Le canal de lecture de donnée** est utilisé par l'interface esclave pour transmettre la donnée enregistrée à l'adresse reçue par le canal d'adresse de lecture. Parmi les signaux utilisés dans ce canal, on trouve RDATA[Largeur du bus de donnée : 0], RVALID, RREADY, RRESP[1 : 0].

  A la fin de la transaction, l'interface esclave envoie une réponse à l'aide des signaux RRESP[1 : 0] pour informer l'interface maître de l'échec ou le succès de la transaction.
- **Le canal d'adresse d'écriture** est utilisé par l'interface maître pour transmettre à l'interface esclave l'adresse à modifier. Parmi les signaux utilisés dans ce canal, on trouve AWADDR[Largeur du bus d'adressage : 0], AWVALID, AWREADY, BRESP[1 : 0], BVALID, BREADY et AWPROT[2 : 0] qui sera également détaillé dans la suite de ce chapitre.
- **Le canal d'écriture de donnée** est utilisé pour recevoir la donnée à enregistrer dans l'adresse envoyée par l'interface maître. Parmi les signaux utilisés dans ce canal, on trouve WDATA[Largeur du bus de donnée : 0], WVALID, WREADY

  Le canal d'écriture de réponse est utilisé pour informer l'interface maître de l'acceptation ou du rejet de la requête. Parmi les signaux utilisés dans ce canal, on trouve BRESP[1 : 0], BVALID et BREADY.

Les signaux …VALID et …READY dans chaque canal sont des signaux de handshake utilisés par les deux interfaces, maître et esclave, pour indiquer à quel moment ils sont prêts à échanger des données. Les signaux BRESP[1 : 0] et RRESP[1 : 0] sont utilisés pour envoyer une réponse à la requête initiée par l'interface maître (échec ou succès). Si la transaction est correcte, l'interface esclave envoie un OK en envoyant la valeur binaire "00''. Si la transaction échoue, l'interface esclave peut envoyer deux types d'erreur [34]:

- L'erreur DECERR (la valeur binaire "11") qui est utilisée pour informer l'interface maître que le bus n'a pas réussi à accéder à l'interface esclave.

- L'erreur SLVERR (la valeur binaire "10") qui est utilisée pour informer l'interface maître d'un dépassement de FIFO dans l'interface esclave, de la non prise en charge de la taille des données, de la tentative d'accès en écriture à une région déclarée en lecture seule, …

Dans le deuxième chapitre de ce manuscrit, on montrera que la sécurité du bus de communication est très importante et qu'une simple manipulation malicieuse peut altérer le bon fonctionnement du bus.

1.2. La hiérarchie mémoire dans un SoC complexe hétérogène

Dans les SoCs modernes, il y a une grande différence entre le temps d'exécution d'un processeur et la latence d'accès à la mémoire. Pour combler cette différence, la mémoire et structuré en une hiérarchie de mémoires de différentes latences, tailles, débits et coûts. La figure 5 présente les éléments de la hiérarchie de mémoires intégrées dans les SoCs complexes hétérogènes, Xilinx Zynq-7000 et Intel Cyclone V. Dans cette section on détaillera les éléments de la hiérarchie mémoire en utilisant les noms utilisés dans la documentation de Xilinx

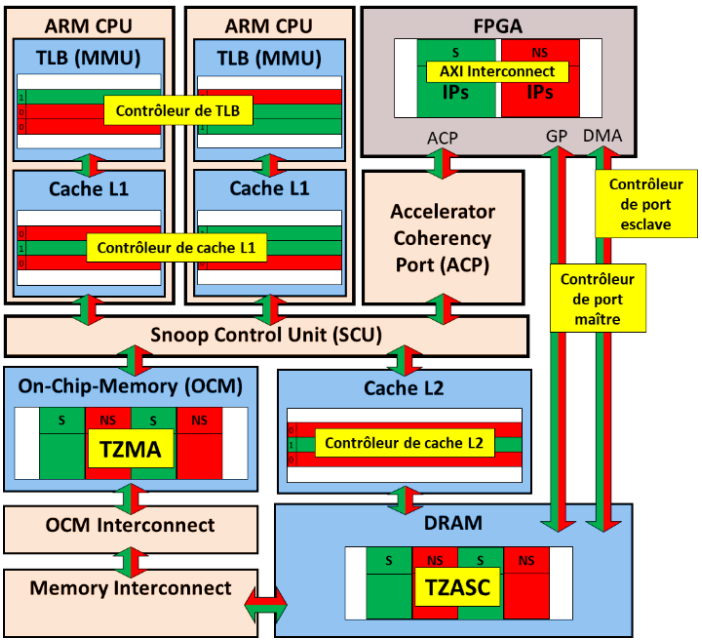


Figure 5 : Représentation schématique de la hiérarchie mémoire dans un SoC complexe hétérogène, tels que les SoC Xilinx Zynq-7000 et Intel Cyclone V.

1.2.1. Unités de gestion de la mémoire (MMU et IOMMU)

Dans un cœur ARM, l'unité de gestion de la mémoire (MMU) [35] fonctionne en étroite collaboration avec les mémoires cache L1 et L2. Le matériel de la MMU comprend une TLB (Translation Lookaside Buffer qui n'est qu'une mémoire cache pour la MMU), une logique de

contrôle d'accès et une logique de déplacement de tables de traduction. La MMU convertit les adresses virtuelles en adresses physiques tout en contrôlant les autorisations d'accès à la mémoire. La MMU signale toute violation d'accès au cœur ARM en déclenchant une exception.

Par exemple, la MMU est utilisée par les systèmes pour protéger l'espace mémoire d'un processus en interdisant l'accès aux autres processus. Elle protège aussi les régions mémoires contenant des données sensibles en déclarant ces régions accessibles en lecture seulement.

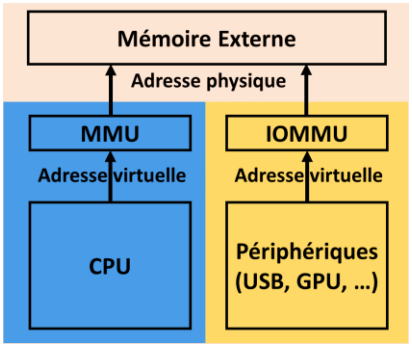


Figure 6 : Les unités de gestion de mémoire dans un SoC FPGA

D'après la figure 6, les SoC modernes intègrent deux types de MMU, une MMU qui protège le système des codes malveillant exécutés par le CPU (MMU détaillée dans la section 1.2.1) et une MMU qui protège le système des périphériques E/S malveillants. Ce deuxième type de MMU est nommé l'unité de gestion de mémoire d'entrée-sortie (IOMMU) [36]. L'IOMMU connecte le bus de périphérique E/S à la mémoire principale pour un accès direct à la mémoire (DMA, Direct Memory Access). L'IOMMU traduit les adresses virtuelles utilisées par les périphériques en adresses physiques tout en contrôlant les droits d'accès. Le SoC Zynq UltraScale+ et le SoC Intel Stratix 10 SX intègrent une SMMU (System MMU), un équivalant de l'IOMMU, pour protéger le système contre les attaques DMA provenant de la partie reconfigurable et des périphériques E/S.

1.2.2. La mémoire cache TLB

La mémoire cache TLB qui n'est rien d'autre qu'une mémoire cache spéciale pour la MMU utilisée pour garder la trace des entrées récemment utilisées. Une entrée est un vecteur de bits indiquant les informations utiles pour la traduction d'une adresse virtuelle et les droits d'accès.

Dans la plupart des processeurs ARM, la TLB est une mémoire cache à deux niveaux comme illustré dans la figure 7. Un premier niveau appelé micro-TLB divisée en deux parties, une pour les instructions et une autre pour les données. Un deuxième niveau appelé main-TLB plus grand que le premier intègre un tableau d'entrées associatives et des entrées verrouillées. Comme toute

mémoire cache, la micro-TLB et le main-TLB ont des succès (existence de la donnée dans la mémoire cache) et échec (absence de la donnée dans la mémoire cache).

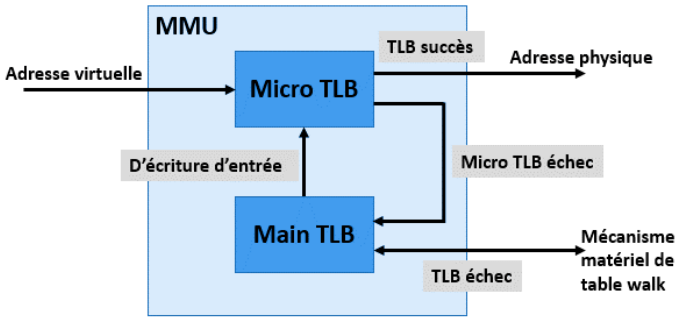


Figure 7 : MMU, processus de traduction d'une adresse virtuelle en adresse physique

1.2.3. Les mémoires cache L1 et L2

Les mémoires cache L1 et L2 sont des mémoires qui enregistrent temporairement des copies de données afin de diminuer le temps d'accès à la mémoire réalisé par le processeur ARM. Le cache L1 est plus petit, plus rapide et plus proche du cœur ARM que le cache L2. Dans une architecture multi-cœur, chaque cœur a sa propre mémoire cache L1 et la mémoire cache L2 est partagée entre les différents cœurs et avec d'autres éléments du SoC complexe hétérogène, comme la partie reconfigurable et le processeur graphique.

Les mémoires cache L1 et L2 ont aussi des échecs et des succès. Dans certaines configurations de la mémoire cache, la mémoire cache L1 récupère des données de la mémoire cache L2 en cas d'échec. Et si la mémoire cache L2 ne contient pas la donnée (échec), la mémoire externe est sollicitée.

1.2.4. Snoop Control Unit (SCU)

Dans un système où la mémoire cache L2 est partagée entre plusieurs éléments, la mémoire cache L2 est connectée à une unité qui utilise des algorithmes de maintien de cohérence de cache pour éviter l'utilisation des opérations de maintenance de la mémoire cache à chaque accès à la mémoire. Dans les processeurs ARM cette unité est nommée le Snoop Control Unit (SCU) [37].

1.2.5. Interface ACP (Accelerator Coherency Port)

Dans un SoC complexe hétérogène, l'interface ACP est défini comme une interface esclave. Il est utilisé par les IP de la partie reconfigurable et les processeurs graphiques pour accéder à la mémoire externe toute en gardant la cohérence de la mémoire cache L1 et L2.

D'un point de vue système, la connectivité de l'interface ACP est similaire à celle d'un cœur ARM ce qui permet aux IP de la partie reconfigurable de concurrencer les cœurs ARM du système à l'accès à la mémoire. L'interface ACP est connectée directement à l'unité de contrôle SCU qui est également connectée aux mémoires caches L1 des cœurs ARM et aux mémoires cache L2. Lors d'une requête d'écriture ACP, la SCU vérifie l'existence de l'adresse de la requête dans les différents niveaux de caches de données. Si elle est présente, le protocole de cohérence nettoie et invalide la ligne appropriée et actualise l'adresse dans la mémoire. Lors d'une requête de lecture ACP, si l'adresse réside dans la mémoire cache du système, qu'elle soit invalidée ou non, la donnée de l'adresse est renvoyée par l'interface ACP au destinataire. Sinon, la donnée est transmise directement depuis la mémoire externe vers l'interface ACP.

1.2.6. La mémoire DRAM (Dynamic Random Access Memory)

Dans la figure 5, la mémoire DRAM est connectée à la mémoire cache L2 avec une latence d'accès plus élevée et une taille très importante. La mémoire DRAM est composée de plusieurs matrices de cellules mémoires (une matrice = bank) comme illustré dans la figure 8, chaque cellule est composée d'un condensateur contrôler par un transistor. Une cellule permet de stocker un bit dont la valeur est liée au chargement et au déchargement du condensateur. Pour cela, les mémoires DRAM doivent être actualisées périodiquement (période liée à la technologie de fabrication des mémoires DRAM) pour éviter les pertes de données dues à la dispersion des charges électriques des condensateurs dans le temps.

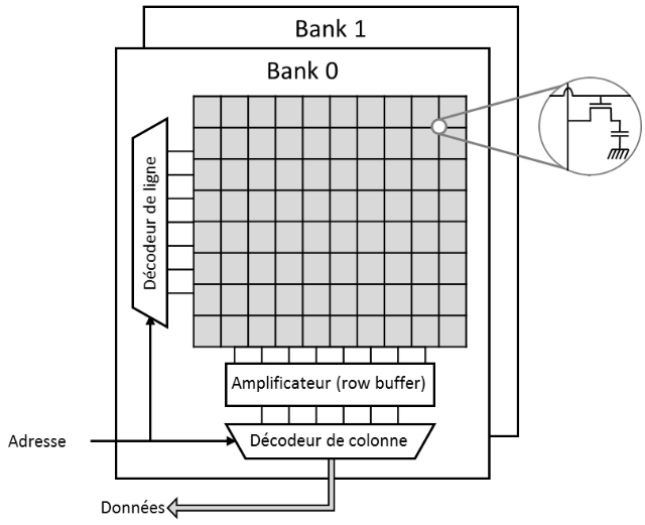


Figure 8 : Structure d'une matrice de cellules mémoire (Bank) DRAM

L'unité de fonctionnement d'une DRAM est une ligne de matrice, durant toutes les opérations (lecture ou écriture) les données transitent par un *row buffer* avant d'atterrir dans une ligne de la matrice. L'emplacement de la ligne dans la matrice est indiqué par l'adresse physique qui contient le numéro du *bank*, de la ligne et de la colonne. La figure 9 présente la fonction de décodage d'une adresse physique pour le SoC Zynq-7000, certain SoC utilisent des fonctions de décodage plus complexes (utilisant des fonctions de hachage) et non-documentées.

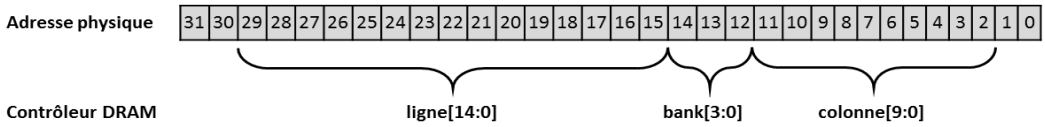


Figure 9 : Décodage d'une adresse physique dans le SoC Zynq-7000

La complexité du système mémoire des SoCs complexe hétérogène augmente considérablement leurs consommations d'énergie. La section suivante présente des méthodes utilisées dans la majorité des systèmes moderne pour optimiser leurs consommations.

1.3. Le système de gestion d'énergie dans les SoCs complexes hétérogènes

1.3.1. La consommation d'énergie d'un SoC complexe hétérogène

La figure 10 présente l'anatomie d'un transistor CMOS utilisé dans la conception des processeurs, on trouve une plaque en métal reliée à la grille appelée l'armature, un élément semi-conducteur entre la source et le drain, et un morceau d'isolant entre les deux. On peut considérer le transistor comme un condensateur composé de deux morceaux de conducteurs, la grille et la liaison drain-source, séparés par un isolant. La quantité d'énergie dissipée en une seconde par ce condensateur est définie par l'équation (1), avec C la capacité du condensateur, U la tension d'alimentation et f la fréquence de commutation.

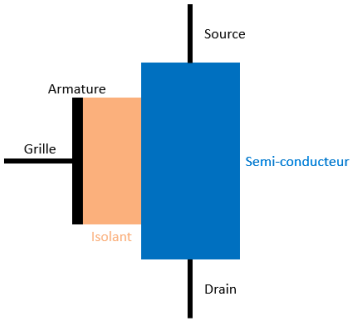


Figure 10 : Anatomie d'un CMOS

$$E = \frac{1}{2} CU^2 * f \quad (1)$$

Pour obtenir la consommation d'énergie totale d'un SoC, il suffit de multiplier celle d'un transistor par le nombre total de transistors d'un SoC. En notant le nombre total de transistors du processeur n, on a alors :

$$E = \frac{1}{2} CU^2 * f * n \quad (2)$$

D'après l'équation (2), l'énergie dissipée par le SoC est proportionnelle à la fréquence et au carré de la tension si l'on souhaite diminuer l'énergie dissipé par le SoC il faut donc agir sur ces deux paramètres (le paramètre C étant lié à la technologie utilisé).

1.3.2. L'ajustement dynamique de la fréquence et de la tension (DVFS)

Parmi les solutions industrielles utilisées pour diminuer la consommation d'un système est l'ajustement dynamique de la tension et de la fréquence (DVFS) qui est une technique basée sur la manipulation de la tension et de la fréquence pour permettre à un SoC d'effectuer les tâches nécessaires avec le minimum de puissance requise. Le DVFS est utilisé dans presque tous les systèmes modernes afin de maximiser les économies d'énergie, la durée de vie de la batterie et la longévité des périphériques tout en maintenant une disponibilité des performances de calcul prête à l'emploi.

Dans un système Linux, L'ajustement dynamique de la tension et de la fréquence (DVFS) se base sur l'utilisation d'un pilote dédié qui estime en permanence l'utilisation et la charge de travail imposée par les tâches à exécuter et en déduira s'il faut ajuster ou pas la tension et la fréquence. Le pilote prend également en considération le mode de fonctionnement imposé par l'utilisateur dans le choix de la fréquence et la tension. Ce mode peut être par exemple un mode basse consommation d'énergie ou bien un mode haut performances.

2. La technologie ARM TrustZone dans les SoCs complexes hétérogènes

2.1. La technologie ARM TrustZone dans un cœur du processeur ARM

À partir des architectures ARMv6 [38], l'extension matérielle de sécurité TrustZone a été introduite dans les processeurs ARM. Cette extension permet d'isoler matériellement un environnement d'exécution sécurisé à l'aide de contrôleurs matériels dédiés (détaillé dans la suite

de ce chapitre). Ces contrôleurs matériels permettent d'interdire l'accès à certains composants du SoC pour les autres environnements d'exécution. L'environnement d'exécution sécurisé est utilisé lorsque le cœur ARM est dans son état sécurisé, et cela est contrôlé par le bit d'état de sécurité (détaillé dans 2.1.1.). Dans la suite de cette section, on présente quelques éléments de la technologie ARM TrustZone dans un cœur ARM.

2.1.1. Le bit d'état de sécurité

Dans un processeur ARM embarquant la technologie ARM TrustZone, le registre SCR (*Secure Configuration Register*) du coprocesseur CP15 (un coprocesseur de contrôle système) du cœur ARM inclut un bit d'état de sécurité appelé le bit NS (Non-Secure). Ce bit contrôle l'état de sécurité du cœur et de toutes les ressources du SoC dans le cas de l'extension de la technologie ARM TrustZone en dehors du processeur ARM. Si le bit NS est valide (à l'état haut), le système fonctionne dans le monde non-sécurisé. Dans le cas contraire, le système fonctionne dans l'autre monde, le monde sécurisé. Le bit NS permet de diviser le cœur ARM physique en deux cœurs virtuels, un cœur virtuel sécurisé utilisé par le monde sécurisé et un cœur non-sécurisé utilisé par le monde non-sécurisé. La valeur du bit NS est modifiable seulement depuis le mode monitor (décrit dans la section suivante), toute tentative de modification par d'autres moyens déclenche une exception.

2.1.2. Un mode supplémentaire : le mode monitor

Dans un processeur ARM embarquant la technologie ARM TrustZone, le cœur ARM utilise un mode supplémentaire appelé le mode monitor pour effectuer et autoriser le basculement du monde non-sécurisé au monde sécurisé et vice-versa. Le mode monitor joue le rôle de gardien entre les deux mondes, il permet de changer l'état de sécurité (la valeur du bit NS) sans affecter l'exécution du système, d'annuler la fuite de données entre les deux mondes. Ce mode est accessible depuis les deux mondes en utilisant soit l'instruction SMC (Secure Monitor Call, ou Appel moniteur surveillé), soit l'une des trois exceptions suivantes: External abort handler, FIQ handler, IRQ handler.

Un exemple de scénario de changement de monde est le suivant: après l'exécution de l'instruction SMC par l'un des deux mondes, le cœur ARM entre dans le mode monitor, enregistre le contexte du monde actuel, charge le contexte du monde futur, et change la valeur du bit NS (0 pour le monde sécurisé, et 1 pour le monde non-sécurisé). Ensuite le processeur effectue un saut de mémoire vers l'adresse de reprise d'exécution du monde futur qui a été chargée avec le

contexte. Durant ce changement de monde, le programme monitor fait le nécessaire pour éviter la fuite de données entre les deux mondes.

2.1.3. Deux MMU virtuelles

Dans un cœur ARM embarquant la technologie ARM TrustZone, la MMU physique est diviser en deux MMU virtuelles ce qui permet à chaque monde d'avoir sa propre table de traduction d'adresse virtuelle en adresse physique. Durant la traduction d'une adresse virtuelle, la MMU charge l'entrée liée à l'adresse dans la TLB (Translation Lookaside Buffer) accompagnée d'un identifiant de table non-sécurisé (NSTID). Cet identifiant permet la coexistence des entrées sécurisées et non-sécurisées dans la TLB, et supprime la nécessité de vider la TLB à chaque commutation entre les deux mondes.

2.1.4. La mémoire cache L1

L'architecture de la mémoire cache L1 d'un cœur ARM embarquant la technologie TrustZone est modifiée pour inclure le bit supplémentaire indiquant l'état de sécurité des transactions en mémoire. Ce bit permet la coexistence des données des deux mondes dans la mémoire cache. Même si les données sécurisées de la mémoire cache ne sont pas accessibles depuis le monde non-sécurisé, les deux mondes sont égaux lorsqu'ils se font concurrence pour l'utilisation d'une ligne de cache. Ainsi, lors d'un changement de monde, les lignes de cache n'ont pas besoin d'être vidées, car un remplissage d'une ligne sécurisée de la mémoire cache peut expulser une ligne non sécurisée de cette même mémoire, et inversement. Cette conception de la cohérence du cache améliore les performances du système en éliminant la nécessité de vider le cache lors d'un changement de monde.

2.2. Extension de la technologie ARM en dehors des cœurs du processeur ARM

L'une des fonctionnalités les plus utiles de la technologie ARM TrustZone est la possibilité de sécuriser l'ensemble du SoC, et pas seulement le cœur ARM. L'état de sécurité du cœur ARM est propagé dans l'ensemble du SoC en utilisant les signaux de sécurité du bus de communication (ARPROT[2 : 0], AWPROT[2 : 0] pour le bus AXI). Ces signaux de sécurité sont utilisés par les contrôleurs matériels TrustZone qui assurent la sécurité des ressources, comme la mémoire et les périphériques. La suite de cette section présente les différents contrôleurs matériels TrustZone utilisés pour l'extension de la technologie ARM TrustZone en dehors du processeur ARM.

2.2.1. TrustZone Protection Controller (TZPC)

Le TZPC étend la sécurité TrustZone aux périphériques du système. Ce contrôleur matériel possède une interface de programmation dynamique qui permet de le configurer depuis le monde sécurisé durant l'exécution du système. Le TZPC permet de restreindre l'accès du monde non-sécurisé à certains périphériques. Il contrôle aussi les violations de sécurité concernant l'accès aux périphériques et remonte les erreurs au cœur ARM qui déclenche l'exception liée à l'erreur.

2.2.2. TrustZone Address Space Controller (TZASC)

Le TZASC étend la sécurité TrustZone à l'infrastructure de la mémoire. Le TZASC partitionne la mémoire DRAM (*Dynamic Random Access Memory*) en différentes régions de mémoire. Ce contrôleur matériel possède aussi une interface de programmation dynamique qui lui permet une configuration durant l'exécution du système. Le TZASC permet de restreindre l'accès du monde non-sécurisé à certaines régions de la mémoire. Par défaut, les applications sécurisées peuvent accéder aux régions de mémoire non-sécurisées, mais l'inverse n'est pas possible.

2.2.3. TrustZone Memory Adapter (TZMA)

Le TZMA fournit des fonctionnalités similaires à celles du TZASC pour la mémoire sur puce. Cela signifie qu'il permet de sécuriser une région dans une mémoire statique sur puce, telle qu'une ROM ou une mémoire SRAM. Le TZMA permet de partitionner une seule mémoire statique pouvant atteindre 2 Mo en deux régions, il ne peut pas être utilisé pour partitionner des mémoires DRAM, ou des mémoires qui nécessitent plusieurs régions sécurisées.

2.2.4. Direct Memory Access Controler (DMAC)

Le DMAC est utilisé pour déplacer des données dans la mémoire, au lieu d'utiliser le processeur pour effectuer cette tâche. Le DMAC peut supporter simultanément les canaux de transfert sécurisés et non-sécurisés, chacun avec des événements d'interruption indépendants et contrôlés par une interface dédiée. Une transaction non-sécurisée impliquant un transfert DMA (*Direct Memory Access*) vers ou depuis une mémoire sécurisée entraînera l'échec du transfert DMA et déclenchera une exception.

2.2.5. Generic Interrupt Controller (GIC)

Dans un SoC embarquant la technologie ARM TrustZone, le GIC affecte à chaque monde les interruptions nécessaires pour ses applications. Pour cela, il utilise le bit de configuration de l'état de sécurité lié à chaque interruption du système. Le GIC prend en charge aussi la hiérarchisation

des interruptions, permettant la configuration d'interruptions sécurisées avec une priorité plus élevée que les interruptions non-sécurisées.

2.2.6. Le contrôleur de la mémoire cache L2

Le contrôleur de la mémoire cache L2 associe un bit NS à toutes les données stockées dans la mémoire cache L2 et dans les mémoires tampons internes. Une transaction non-sécurisée ne peut pas accéder à des données sécurisées. Par conséquent, le contrôleur traite les données sécurisées et non-sécurisées comme faisant partie de deux espaces mémoires différents. Lorsque le monde non-sécurisé accède (lecture/écriture) à des données sécurisées, le contrôleur de la mémoire cache L2 traite l'accès comme un échec. Pour une opération de lecture depuis le monde non-sécurisé d'une donnée sécurisée, le contrôleur de cache envoie une commande de remplissage de ligne à la mémoire externe, et propage l'erreur de sécurité de la mémoire externe vers le processeur et n'alloue pas de ligne dans le cache.

2.3. Le déploiement de la technologie ARM TrustZone dans la partie reconfigurable d'un SoC complexe hétérogène.

Dans un SoC embarquant une partie reconfigurable, l'état de sécurité du processeur peut être propagé optionnellement vers la partie reconfigurable en utilisant les signaux de sécurité inclus dans le bus de communication. Cette partie présente les éléments qui permettent l'extension de la technologie ARM TrustZone à la partie reconfigurable.

2.3.1. Les signaux de sécurité du bus AXI

En général, les signaux ARPROT[1] (du canal d'adresse de lecture) et AWPROT[1] (du canal d'adresse d'écriture) sont des bits NS qui transmettent l'état de sécurité de l'interface maître dans le bus AXI. L'interface esclave utilise ces deux signaux pour s'assurer qu'aucune violation de sécurité ne se produit durant la transaction. Pendant une transaction (lecture ou écriture), l'interface maître impose la sécurité de la transaction, il choisit la valeur à appliquer à ces deux signaux. Pour une transaction sécurisée, les deux signaux sont à l'état logique bas, et à l'état logique haut pour une transaction non-sécurisée. Dans le cas d'une requête issue d'une interface maître non-sécurisée et destinée à une interface esclave sécurisée, l'interface esclave envoie l'erreur DECERR à l'interface maître en utilisant les signaux d'erreur du bus.

Dans le cas d'un SoC intégrant une partie reconfigurable, les signaux ARPROT[1] et AWPROT[1] peuvent être utilisés pour propager l'état de sécurité du processeur ARM à la partie

reconfigurable. Dans ce cas, les signaux ARPROT[1] et ARPROT[1] ont la même valeur que le bit NS du registre SCR.

2.3.2. L'AXI Interconnect (AXI bridge pour Intel)

Dans la partie reconfigurable, l'AXI Interconnect (AXI bridge pour Intel) est utilisé pour connecter les IP matérielles au bus AXI. Dans un SoC embarquant la technologie ARM TrustZone, l'AXI Interconnect inclut une fonctionnalité optionnelle qui permet de déclarer statiquement durant la phase de conception les interfaces maîtres connectant les interfaces esclave des IP au bus comme sécurisée ou non. La figure 11 illustre un AXI Interconnect qui connecte la partie processeur (le processeur ARM et le processeur graphique, les périphérique E/S, …) avec des IP matérielles, les interfaces maîtres M1_AXI et M2_AXI connectant l'$IP_{\#1}$ et l'$IP_{\#2}$ respectivement au bus sont sécurisées, les interfaces maîtres M0_AXI et M3_AXI connectant la partie processeur et l'$IP_{\#4}$ respectivement au bus sont non-sécurisées.

Les deux interfaces esclaves S0_AXI et S1_AXI connectant la partie processeur et l'$IP_{\#3}$ respectivement au bus ont un état de sécurité neutre. Effectivement, la partie processeur et l'$IP_{\#3}$, par ce qu'elles sont maîtres, choisissent l'état de sécurité de la requête à transmettre à une interface esclave en cours d'exécution et décident de l'état de sécurité des deux interfaces S0_AXI et S1_AXI alors dynamiquement.

Comme illustré dans la figure 11, à l'intérieur de l'AXI Interconnect, on trouve le Crossbar qui permet de connecter les différentes interfaces entre elles et de distribuer les communications en utilisant un système d'ID. On trouve aussi une série facultative d'infrastructures AXI (''Coupleurs'' dans la figure 11) qui permet d'effectuer la conversion de protocole si nécessaire. Cette série facultative d'infrastructures AXI utilisent parfois des FIFOs, si les deux protocoles ont des débits différents.

Si la fonction de sécurité est activée, le Crossbar est chargée de vérifier l'état de sécurité des transactions dans la partie reconfigurable. Il compare les signaux de sécurité du bus AXI avec l'état de sécurité statique de l'IP destinée. En cas de violation de la sécurité, le Crossbar renvoie une erreur au coupleur approprié qui arrête la transaction et transfère l'erreur au système processeur qui déclenche alors une exception.

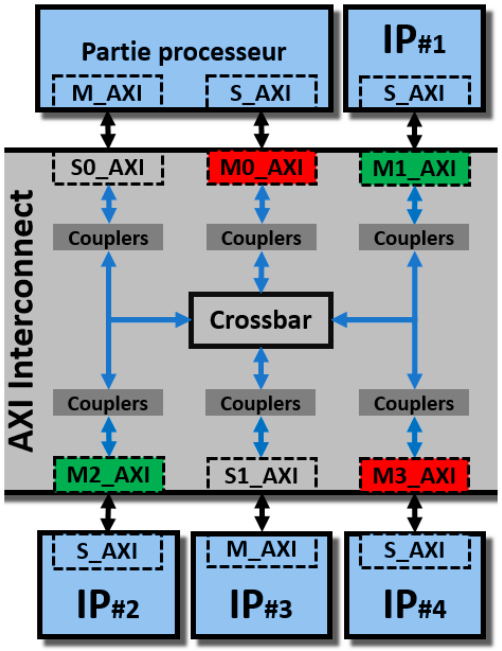


Figure 11: Illustration d'un AXI Interconnect couplant des interfaces maître et esclave

3. Etat de l'art des attaques ciblant les SoC complexes hétérogènes

3.1. Les attaques visant le Secure boot

Les attaques visant la chaîne de confiance, assez tôt dans le processus de Secure boot, sont une grande problématique pour la sécurité des systèmes embarqués car elles ont une grande portée. Elles peuvent toucher à la propriété intellectuelle, si l'attaquant récupère les clés cryptographiques utilisées pour le déchiffrement des images binaire. Elles peuvent compromettre l'intégralité du système, si l'attaquant charge une image binaire malveillante depuis un support de boot malicieux.

3.1.1. Attaques logicielles

Les attaques logicielles visant le Secure boot exploitent les vulnérabilités des programmes [39], [40] utilisés durant le processus de boot. Ces vulnérabilités peuvent être un dépassement de tampon (buffer overflow), un dépassement de tas (heap overflow), un dépassement d'entier, etc. Les sources de ces vulnérabilités sont les erreurs de programmation et l'absence de méthode de vérification efficace. Avant d'exploiter la vulnérabilité, l'attaquant doit réaliser une étape de rétro-ingénierie pour la trouver et comprendre le fonctionnement du système.

3.1.2. Attaques matérielle

3.1.2.1. Attaque par injection de faute

Les attaques par injection de faute consistent à modifier l'état de la mémoire afin de provoquer un changement du chemin d'exécution vers un chemin souhaité. Les fautes peuvent être injectées, via les bus de communication, directement dans les registres, etc. Par contre, ce type d'attaque a une probabilité plus grande de provoquer un comportement inattendu que d'aboutir à un chemin exploitable [41]. Une attaque en faute est plus complexe à mettre en œuvre par rapport à une attaque logicielle. Et il est compliqué de trouver les paramètres de l'attaque, l'emplacement, le moment et la puissance de l'injection pour rendre l'attaque efficaces [42] [43].

Parmi les méthodes utilisées pour les attaques par injection de faute visant le Secure boot, on trouve le *Glitch* de tension [6], l'impulsion électromagnétique [10], [11], l'injection par laser [8]. Ces méthodes sont utilisées pour sauter la vérification de la signature des images binaires chargées durant le processus de boot et charger un programme malveillant depuis un support de boot malicieux.

3.1.2.2. Attaque par analyse de canaux auxiliaires physiques

Une attaque par canaux auxiliaires est une attaque qui recherche et exploite des failles, logicielle ou matérielle, dans l'implémentation des méthodes et procédures de sécurité sans remettre en cause leurs robustesse théorique. Parmi les méthodes utilisées pour les attaques par canal auxiliaire visant le Secure boot, on trouve l'analyse de consommation [44], [45] et l'analyse des émissions de photons [9]. Les deux méthodes ont été utilisées pour trouver la clé de déchiffrement du bitstream durant le Secure boot.

3.1.2.3. Cheval de Troie matériel

En 2017, Jacob et el [46] ont réussi à corrompre le processus de Secure boot en utilisant un cheval de Troie matériel avec un accès direct à la mémoire (DMA) et implémenté dans la partie reconfigurable d'un SoC FPGA. Ils ont réussi à injecter un code malicieux dans le bootloader de deuxième niveau pour charger une image binaire d'un noyau malicieux depuis un serveur distant.

3.2. Les attaques logicielles visant le système d'exploitation de confiance (TEE)

La plupart des vulnérabilités qui touchent la sécurité du TEE, sont liées à des bugs existants soit dans le noyau du système d'exploitation de confiance, soit dans les pilotes des périphériques, soit dans les applications de confiance. Parmi les vulnérabilités touchant les TEE basées sur la TrustZone, on trouve l'absence de validation des applications de confiance, les dépassements de mémoire, et les variables non initialisées. La majorité de ces vulnérabilités n'est pas bien documentée et met du temps à être publiée. Par exemple, la vulnérabilité CVE-2016-10238 a été découverte en 2016 mais elle n'a été publiée qu'en mars 2017, et sa description est très brève: "... Les détails techniques sont inconnus et un exploit n'est pas publiquement disponible". Li est al. [47] ont présenté une analyse complète des vulnérabilités récentes, et les ont classée en trois catégories:

- Des vulnérabilités conduisant à des fuites de données ou exécution de code malveillant (par exemple : la CVE-2017-0518/0519, une vulnérabilité d'élévation de privilège qui touche le pilote du détecteur d'empreintes digitales Qualcomm. La vulnérabilité permet à une application locale malveillante d'exécuter du code dans le noyau TEE).
- Des vulnérabilités permettant à une application sécurisée d'affecter la sécurité des autres applications sécurisées (par exemple : CVE-2016-0825, l'application Widevine de Qualcomm QSEE peut exploiter le noyau TEE pour récupérer et fuiter des données stockées dans le Secure Storage).
- Des vulnérabilités permettant de réaliser une élévation de privilèges dans le système d'opération du monde non-sécurisé (par exemple : CVE-2017-0518/0519, un pilote TEE présente une vulnérabilité de validation d'entrée. Un attaquant peut l'exploiter pour obtenir un privilège root dans le monde non-sécurisé, ainsi que l'exécution du code malveillant dans le TEE).

3.3. Les attaques matérielles visant le TEE

Un certain nombre de vulnérabilités liées au matériel ont également été découvertes au cours des dernières années. Les vulnérabilités signalées affectent différentes parties matérielles du système, en particulier les composants qui constituent la racine de confiance, les différents types de mémoires (cache, TLB, DRAM), les mécanismes de gestion d'énergie et la partie reconfigurable d'un SoC FPGA. Cette section présente en premier temps les différentes attaques

visant la mémoire. Ensuite, elle présente les attaques matérielles exploitant les mécanismes de gestion d'énergie utilisés dans un SoC FPGA.

3.3.1. Les attaques visant la hiérarchie de la mémoire dans un SoC FPGA

3.3.1.1. Les attaques visant la mémoire DRAM

- **Attaque rowhammer**

L'attaque rowhammer exploite un défaut dans les mémoires DRAMs modernes, L'accès répété en lecture à une ligne d'un *bank* provoque une fuite de charge électrique dans une cellule mémoire des lignes voisines, et donc un inversement de bit. La vulnérabilité est dû à la réduction de la taille de technologie utilisée pour la fabrication des DRAMs (<40nm) et de l'espace entre les cellules de mémoire (couplage électrique entre les lignes). Ce défaut exploité par un programme malveillant permet de modifier des instructions assembleur [48] et de réaliser une élévation de privilèges dans le noyau Linux [49].

- **Attaque Cold boot (attaque par démarrage à froid)**

L'attaque *Cold boot* [50] est une attaque physique qui permet de récupérer des données sensibles de la mémoire DRAM après la coupure de l'alimentation. Cette attaque nécessite un accès physique pour refroidir la mémoire DRAM avant la mise hors tension du système. Le froid ralentit la décharge des condensateurs, donc une grande partie des données est toujours présente dans la DRAM hors tension. Pour extraire ces données, l'attaquant doit utiliser un programme malveillant qui vide la DRAM sans l'initialiser au démarrage.

Après le refroidissement de la DRAM, La perte de charge dans un condensateur est de l'ordre de 0,01% toutes les minutes [50]. En cas de perte importante de données, il existe des algorithmes capables de récupérer les données perdues en utilisant la manière dont les données sont stockées dans la mémoire.

- **DRAMA**

L'attaque DRAMA [51] applique les attaques Flush+Flush et Flush+Reload (attaques détaillées en section 3.3.1.2) sur le *row buffer* qui garde trace de la dernière ligne chargée. DRAMA mesure le temps d'accès à la donnée dans la mémoire DRAM, pour vérifier l'existence ou pas de ligne contenant la donnée dans le *row buffer*.

- **L'accès direct à la mémoire (DMA)**

L'accès direct à la mémoire (DMA) est le mécanisme qui permet d'accéder à la mémoire externe sans passer par le processeur. Cette méthode est utilisée par différents périphériques pour des raisons de performance, au détriment de la visibilité par le processeur sur les échanges mémoires effectués. Il libère les processeurs de la charge de travail générée par de simples transferts de données. Les transferts de données sont effectués sans supervision des processeurs entre la mémoire et les différents éléments du système.

Les transferts DMA non-supervisés peuvent mettre le système en danger, si ce dernier contient des périphériques ou des IP (implémentées dans la partie reconfigurable) malveillants. Ces éléments malveillants peuvent injecter des programmes malicieux dans la mémoire [46], ou profiler la mémoire à la recherche de données sensibles, etc. Dans le cas où la cohérence de cache est activée, les attaques DMA depuis la partie reconfigurable peuvent être utilisées pour réaliser des attaques visant la mémoire cache [52] ou des attaques visant le mécanisme ASLR (*Address Space Layout Randomization*) ou encore une attaque Rowhammer.

3.3.1.2. Les attaques visant la mémoire cache (TLB, L1 et L2)

Les mémoires cache (TLB, cache L1 et L2) sont considérées comme des mémoires partagées, elles peuvent contenir des données utilisées par plusieurs processus à la fois. Dans [53]–[57], les mémoires cache ont permis à des processus malicieux d'étudier le comportement d'un processus victime (par exemple, un programme effectuant des opérations cryptographiques). Les processus malicieux se basent sur le succès et l'échec de la mémoire cache pour déterminer, si le processus victime à utiliser ou pas la donnée partagée. Par exemple, les attaques visant la TLB ont réussi à contourner la protection ASLR [53]. Les attaques visant la mémoire cache L2 ont permis à un processus d'espionner des processus lancés sur d'autres cœurs et à des IP malicieux utilisant l'interface ACP [52] d'espionner des programmes exécuté par des cœurs ARM.

Dans une attaque pour différencier entre les échecs et les succès de la mémoire cache, les processus malicieux peuvent utiliser le PMU (Performance Monitoring Unit) d'ARM accessible seulement depuis l'espace noyau. Le PMU permet de compter les événements comme les échecs de TLB, les échecs de cache L1, le nombre d'accès au cache, etc. Les processus malveillants peuvent également utiliser le temps d'accès aux données pour différencier les échecs des succès. Le temps d'accès est plus court pour les données mises en cache (succès du cache) que pour les données chargées à partir de DRAM (échec de cache). Cette deuxième méthode est plus

avantageuse, car il existe plusieurs *timers* accessibles depuis l'espace utilisateur. Par contre, il faut trouver le seuil qui permet de différencier entre un échec et un succès avant de commencer l'attaque.

Il existe différents types d'attaques de cache basées sur la mesure de temps d'accès et voici leurs principales étapes :

- **Evict+Time** [57]

Etape 1. Mesurer le temps d'exécution du programme victime.

Etape 2. Expulser un ensemble spécifique de N lignes de cache.

Etape 3. Mesurer à nouveau le temps d'exécution du programme victime.

- **Prime+Probe** [57]

Etape 1. Occuper des ensembles spécifiques de N lignes de cache.

Etape 2. Planifier l'exécution du programme victime.

Etape 3. Déterminez quels sets de cache sont encore occupés.

- **Flush+Flush** [54]

Etape 1. Mapper le fichier binaire (par exemple, un objet partagé) dans un espace adresse.

Etape 2. Vider une ligne de cache (code ou données) du cache.

Etape 3. Exécuter le programme victime.

Etape 4. Mesurer Videz une ligne de cache (code ou données) du cache.

- **Flush+Reload** [55]

Etape 1. Mapper le fichier binaire (par exemple, un objet partagé) dans un espace adresse.

Etape 2. Vider une ligne de la mémoire cache (code ou données).

Etape 3. Exécuter le programme victime.

Etape 4. Vérifier si la ligne de l'étape 2 a été chargée par le programme victime.

Dans tous les types d'attaques de la mémoire cache, le processus malicieux a besoin d'une méthode pour expulser les données du cache avant d'effectuer un sondage du cache. Pour cela, le processus malicieux peut utiliser les instructions de maintenance de la mémoire cache (TLB,

cache L1 et L2) accessibles seulement depuis l'espace noyau. Le processus malicieux peut également utiliser une des méthodes présentées dans la littérature [54], [58] pour expulser une ligne de cache depuis l'espace utilisateur. Dans le cas d'un cache de type *n-way associative*, une des techniques est l'accès aux adresses congruentes pour expulser une ligne de cache, car une mémoire cache *n-way associative* ne peut pas contenir plus de n adresses congruentes.

3.3.1.3. Spectre et Meltdown

Pour des raisons de performance, les processeurs modernes exécutent des instructions en parallèle. Ils exécutent spéculativement des instructions futures (*Out-of-order*) et enregistrent leurs résultats dans le cache. En 2018, des chercheurs ont démontré que certains processeurs peuvent exécuter des instructions non-autorisées. Ils ont présenté les deux attaques, Spectre [1] et Meltdown [2], qui se basent sur les attaques de cache pour vérifier l'existence de l'instruction non-autorisée.

3.3.2. Les défaillances de sécurité des systèmes de distribution et de gestion d'énergie dans les SoCs

3.3.2.1. Attaque par injection de faute

Dans la littérature, on trouve plusieurs utilisations malicieuses du mécanisme DVFS. Il est utilisé pour réaliser une attaque par injection d'un Glitch d'horloge ou par injection d'un Glitch d'alimentation. Le DVFS permet de réaliser ces Glitchs en interne sans l'aide de matériel externe sophistiqué. Dans CLKSCREW [59], les chercheurs l'ont utilisé pour injecter une faute dans un algorithme cryptographique sécurisé depuis le monde non-sécurisé.

Dans FPGAHAMMER [60], les chercheurs ont présenté une méthode basé sur les oscillateurs en anneau pour injecter des fautes dans les algorithmes cryptographique implémentés dans la partie reconfigurable. Les chercheurs ont exploité la sensibilité du réseau de distribution d'énergie dans les FPGAs.

3.3.2.2. Attaque par canal auxiliaire

Dans [61], les chercheurs ont également exploité la sensibilité du réseau de distribution d'énergie dans les SoC FPGA. Ils ont démontré que les oscillateurs en anneau peuvent également être utilisés pour la réalisation des attaques par canal auxiliaire depuis la partie reconfigurable sur des algorithmes cryptographiques qui sont exécutés par la partie processeur. Les oscillateurs en anneau sont connus pour être très sensibles aux variations de la tension et de la fréquence, dans une attaque ils peuvent être utilisés comme des sondes internes au SoC. Schellenberg et al.

[58] ont démontré également que les oscillateurs en anneau peuvent capturer les variations de tension générées par un autre SoC qui partage la même source d'énergie, donc de réaliser l'attaque au niveau d'une carte électronique : un circuit intégré de la carte attaque par analyse de consommation un autre circuit de la même carte.

Dans [62], les chercheurs ont également utilisé les oscillateurs en anneau pour transmettre des données discrètement en dehors d'un FPGA par émission électromagnétique. Ils utilisent un analyseur de spectre en temps réel pour capturer les données fuitées du SoC.

4. Conclusion

Ce premier chapitre constitue une description du contexte des travaux présentés dans ce manuscrit. Le chapitre décrit la complexité des SoCs hétérogènes modernes, leurs architectures de bus et celles de leurs de mémoires. La technique du DVFS (Dynamic Voltage and Frequency Scaling) et la technologie ARM TrustZone utilisée pour protéger les ressources et les applications critiques d'un système, sont abordées également. A la fin du chapitre, un nombre d'attaques visant la sécurité d'un SoC complexe hétérogène, des attaques visant le Secure Boot (la chaine de confiance à son démarrage), des attaques visant la hiérarchie de mémoire, et des attaques manipulant malicieusement la technique DVFS sont présentées.

En comparaison avec les travaux présentés dans ce chapitre, ce manuscrit présente pour la première fois une évaluation matérielle de la technologie ARM TrustZone dans un SoC complexe hétérogène. Le manuscrit présente une analyse de la sécurité du bus de communication, des attaques utilisant la technique DVFS pour permettre aux deux mondes de communiquer secrètement, et des attaques exploitant la cohérence du cache. Le manuscrit présente aussi les bonnes méthodes à suivre pour développer un système sécurisé utilisant la technologie ARM TrustZone dans un SoC FPGA.

Chapitre 2 : Les attaques matérielles visant la sécurité du bus

Ce chapitre présente les attaques matérielles développées dans le cadre de cette thèse visant la sécurité du bus AXI dans un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Le chapitre commence par un rappel des aspects de la technologie ARM TrustZone liés au bus AXI. Ensuite, il présente la méthode à suivre pour utiliser la technologie ARM TrustZone dans la partie reconfigurable d'un SoC FPGA. A la fin du chapitre, des attaques qui visent à compromettre la sécurité du bus AXI sont présentées. Ces attaques sont mises en place durant la phase de conception du circuit à l'aide de différentes méthodes, notamment l'utilisation de script TCL malicieux, la modification malicieuse du code source d'une IP, ou l'utilisation d'un outil CAO de non-confiance.

Les attaques présentées dans ce manuscrit sont mises en place à l'aide du SoC Xilinx Zynq-7000 vu son emploi dans plusieurs projets académiques et industriels dans différents domaines. Mais ces attaques peuvent aussi ciblées tous les SoCs complexes hétérogènes embarquant la technologie ARM TrustZone.

1. Rappel

Dans un SoC complexe hétérogène embarquant la technologie ARM TrustZone :

- Les interfaces maîtres contrôlent l'état de sécurité des requêtes émises au bus.
- Le bus utilise les signaux de sécurité ARPROT[1] et AWPROT[1] (détaillés dans la section chap1-2.3.1) pour transmettre l'état de sécurité aux interfaces esclaves du système.
- La valeur des signaux de sécurité ARPROT[1] et AWPROT[1] (l'état de sécurité actuelle de l'interface maître émettrice de la requête) est comparée avec l'état de sécurité de l'interface esclave destinatrice. Selon l'architecture du système, cette comparaison est effectuée soit au niveau de l'AXI Interconnect, soit au niveau d'une IP TrustZone (détaillé dans la section chap1-2.2), soit directement au niveau de l'interface esclave de l'IP destinataire.
- Une requête sécurisée est acceptée par toutes les interfaces esclaves du système peu importe leurs états de sécurité. Si elle n'est pas sécurisée, une requête n'est acceptée que par les interfaces esclaves non-sécurisées du système.
- Si une requête non-sécurisée essaye de communiquer avec une interface esclave sécurisé, la requête est rejetée et l'erreur DECERR est envoyée à l'interface maître émettrice en utilisant les signaux de réponse du bus BRESP ou RRESP (détaillés dans la section chap1-1.1).

Les attaques introduites dans la suite de ce chapitre exploitent les vulnérabilités liées au bus AXI dans un SoC complexe hétérogène embarquant la technologie ARM TrustZone.

2. L'utilisation de la technologie TrustZone dans la partie reconfigurable

Comme indiqué dans la section chap1-2.3, l'utilisation de la technologie ARM TrustZone dans la partie reconfigurable est optionnelle dans tous les SoCs FPGA existants. Cette section présente en détail les étapes à suivre pour activer la propagation de l'état de sécurité d'une requête de communication émise vers la partie reconfigurable et le partitionnement des IP dans la partie reconfigurable du SoC FPGA Xilinx Zynq-7000. Le choix a été porté sur l'utilisation du SoC FPGA Xilinx Zynq-7000 vu son emploi dans plusieurs projets académiques et industriels dans

différents domaines. Cependant, les autres SoC FPGA du marché, comme ceux d'Intel, sont très proches.

L'utilisation de la technologie ARM TrustZone dans la partie reconfigurable est présentée dans ce chapitre pour son intérêt dans la compréhension de la suite de ce manuscrit.

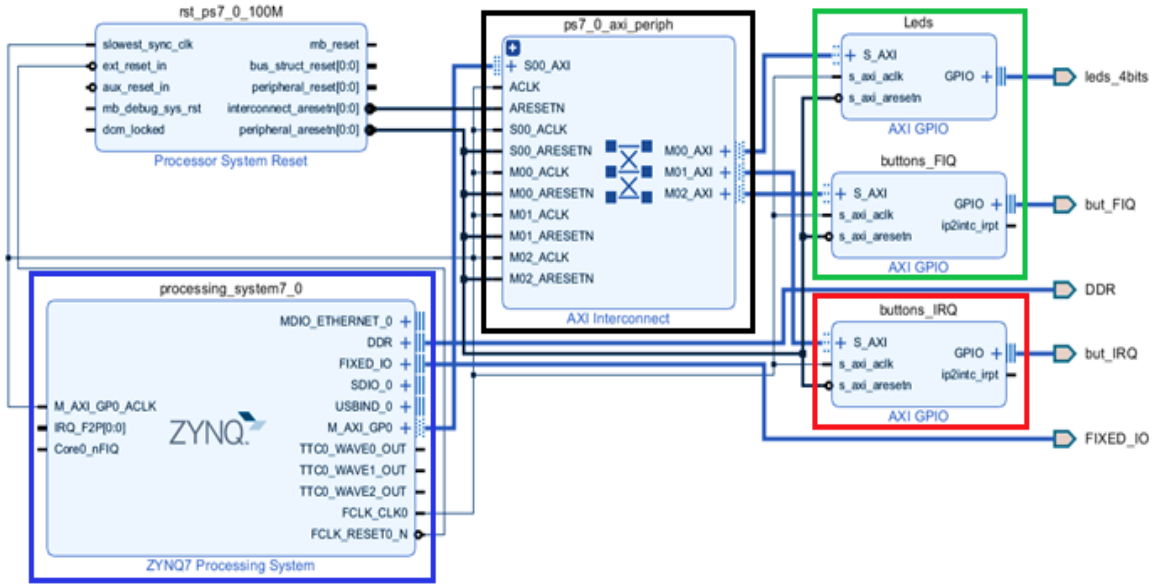


Figure 12: Design créé avec l'outil Vivado de Xilinx

La figure 13, présente le design créé à l'aide de l'outil CAO Vivado de Xilinx. Pour créer ce design nous avons suivi un processus bien précis que nous avons détaillé dans le tutoriel [63] que nous avons rendu disponible à toute la communauté. En suivant scrupuleusement ce tutoriel il est possible d'implémenter un système utilisant la technologie ARM TrustZone dans un SoC FPGA Xilinx Zynq.

Dans la figure 13, le rectangle bleu encercle l'IP ZYNQ7 Processing System qui représente la partie processeur du SoC Xilinx Zynq-7000. Cette IP permet d'effectuer des configurations système, comme la configuration des horloges, l'activation de la propagation de l'état de sécurité des cœurs ARM (l'état de sécurité de la partie processeur) vers la partie reconfigurable, etc. Le reste des blocs constituent les éléments qui sont implémentés dans la partie reconfigurable du SoC. Le rectangle noir encercle l'AXI Interconnect qui permet de connecter les IP entre elles. Le rectangle vert entour les IP déclarées comme sécurisées à l'aide de la fonctionnalité de sécurité de l'AXI Interconnect (fonctionnalité détaillée dans la section chap1-2.3.2), le rectangle rouge entour l'IP déclarée non-sécurisée.

Pour utiliser la technologie TrustZone dans la partie reconfigurable du SoC Xilinx Zynq-7000, il faut passer par les deux étapes suivantes :

- **Etape 1 : L'activation de la propagation de l'état de sécurité du cœur ARM vers la partie reconfigurable du SoC Zynq-7000**

La propagation des signaux de sécurité dans le SoC FPGA Xilinx Zynq-7000 doit être activée pendant la phase de conception. Pour cela, le concepteur doit changer la valeur du paramètre ''***AXI Non Secure Enablement***'' de l'IP ZYNQ7 Processing System (rectangle bleu figure 13) à 1. Sous l'outil Vivado, Le concepteur peut configurer ce paramètre en ligne de commande. Pour cela, le concepteur doit utiliser la commande TCL de l'outil Vivado '***set_property [-dict args] [-quiet] [-verbose] name value objects***'. Cette commande permet de configurer les paramètres d'un objet (l'IP processing_system7_0 dans notre exemple, figure 13) en ligne de commande. La figure 14 présente la commande à exécuter pour changer la valeur du paramètre '***CONFIG.PCW_USE_AXI_NONSECURE***' qui permet d'activer la propagation de l'état de sécurité des requêtes émise vers la partie reconfigurable.

```
set_property -dict [list CONFIG.PCW_USE_AXI_NONSECURE {1}] [get_bd_cells processing_system7_0]
```

Figure 13: Commande TCL Activation de la propagation des signaux de sécurité à la partie reconfigurable

- **Etape 2 : Affecter un état de sécurité aux IP de la partie reconfigurable**

Comme indiqué dans la section chap1-2.2, La technologie TrustZone se base sur l'utilisation de contrôleurs matériels pour partitionner la totalité des ressources du système en sécurisés et non-sécurisés. Pour partitionner les IP en IP sécurisées et IP non-sécurisées dans notre exemple, l'AXI Interconnect intègre une fonctionnalité optionnelle (détaillée dans la section chap1-2.3.2) qui permet de déclarer une IP comme sécurisée. Si cette fonctionnalité n'est pas activée la totalité des IP de la partie reconfigurable sont considérer comme non-sécurisées.

Une fois que la propagation de l'état de sécurité de la partie processeur vers la partie reconfigurable du SoC Xilinx Zynq-7000 est activée (Etape 1 réalisée). Il reste au concepteur à déclarer l'interface maître qui connecte cette IP à l'AXI Interconnect comme sécurisée. Pour réaliser cela en ligne de commande, le concepteur doit utiliser la commande TCL 'set_property' utilisée précédemment pour activer la propagation des signaux de sécurité. Cette commande sera utilisée pour déclarer l'interface M00_AXI et M01_AXI sécurisées et l'interface M02_AXI non-sécurisée. Par exemple pour sécuriser l'interface maître M00_AXI, il faut changer la valeur du

paramètre ***CONFIG.M00_Secure*** de l'AXI Interconnect (***ps7_0_axi_periph***, rectangle noir figure 13) en ***true*** en ligne de commande. La figure 15 présente les 3 lignes de commandes à utiliser pour la réalisation du design figure 13.

```
set_property CONFIG.M00_Secure true [get_bd_cells ps7_0_axi_periph]
set_property CONFIG.M01_Secure true [get_bd_cells ps7_0_axi_periph]
set_property CONFIG.M02_Secure false [get_bd_cells ps7_0_axi_periph]
```

Figure 14: Commande TCL malveillante pour changer l'état de sécurité de l'interface AXI_M00 à non-sécurisée

3. La mise en œuvre des attaques matérielles visant la sécurité du bus AXI

Cette section présente les attaques visant la sécurité du bus AXI dans un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Les attaques, que nous présentons [64], [65], visent les signaux de sécurité AWPROT[1]/ARPROT[1], les signaux de réponse BRESP/RRESP, et l'AXI Interconnect.

Dans la suite de ce chapitre, les attaques sont implémentées à l'aide du SoC Xilinx Zynq-7000, mais celles-ci peuvent toucher tous les SoC embarquant la technologie ARM TrustZone. La figure 16 est une simplification du design de la figure 13. La figure présente deux interfaces, une maître et une esclave, connectées par le bus AXI qui inclus l'AXI Interconnect (détaillé dans la section chap1-2.3.2). L'interface maître contrôle l'état de sécurité des requêtes de communication destinées à l'interface esclave. Le bus AXI transmet la requête à l'AXI Interconnect qui compare l'état de la requête avec l'état de sécurité statique de l'interface esclave. Si la comparaison est correcte, l'AXI Interconnect transmet la requête à l'interface esclave avec un état de sécurité neutre et envoie un OK comme réponse à l'interface maître. Dans le cas contraire, l'AXI Interconnect rejette la requête et envoie une erreur à l'interface maître. L'état de sécurité de l'interface esclave est statiquement fixé durant la phase de conception en utilisant la fonctionnalité de l'AXI Interconnect.

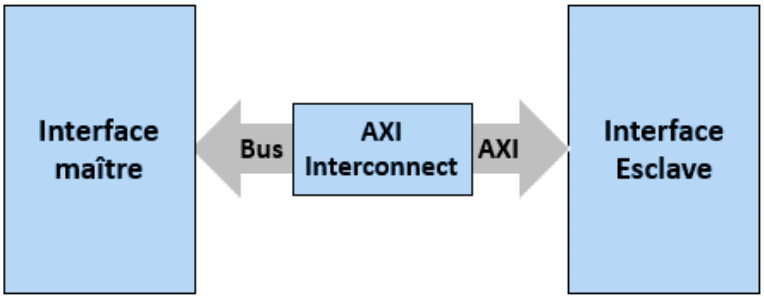


Figure 15 : Communication entre interface maître et esclave

Durant une transaction normale du bus AXI, si l'état logique des signaux AWPROT[1]/ARPROT[1] est à un niveau logique bas à l'entrée de l'AXI Interconnect, l'interface maître peut accéder à toutes les IP connectées à cet AXI Interconnect. Dans le cas contraire, l'accès par l'interface maître aux IP non-sécurisées est restreint. En cas de violation d'accès, l'AXI Interconnect envoie une erreur à l'aide des signaux de réponse à l'interface maître ou au contrôleur du système. Le récepteur de l'erreur effectue le nécessaire (déclencher une interruption par exemple) pour informer le système de l'erreur.

3.1. Compromettre les signaux de sécurité du bus AXI

Pour contrôler les signaux de sécurité, AWPROT[1]/ARPROT[1], un cheval de Troie est placé avant l'AXI Interconnect (le contrôleur TrustZone qui compare les états de sécurité) comme indiqué dans la figure 17. Le cheval de Troie modifie les signaux avant leur comparaison avec l'état de sécurité statique de l'IP ciblée (Etat fixé durant la conception par le designer). Si le cheval de Troie fixe la valeur des deux signaux à un niveau logique bas, l'attaque permettra une escalade de privilège, elle autorisera l'interface maître non-sécurisée à utiliser les IP sécurisées connectées à l'AXI Interconnect. Dans le cas contraire, l'attaque crée un déni de service. Les IP déclarées sécurisées sont condamnées pour la vie du système, tout accès les ciblant (même par une interface maître sécurisée) génère un message d'erreur et la requête de communication est rejeter par le contrôleur.

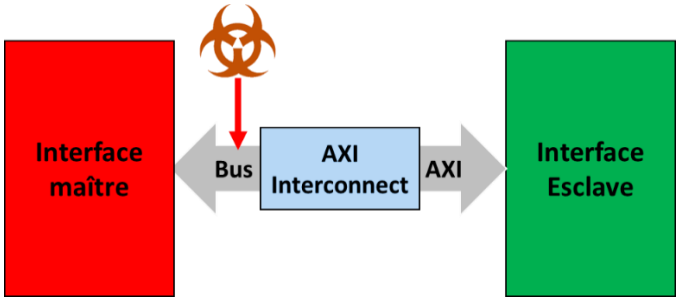


Figure 16 : accéder aux ressources sécurisées depuis le monde non-sécurisé

3.1.1. Exemple de modification matérielle malveillante visant les signaux de sécurité AWPROT[1]/ARPROT[1] du bus AXI

Dans la figure 18, l'interface maître contrôle la sécurité des requêtes émises vers l'interface esclave. L'interface esclave est l'interface esclave du contrôleur TrustZone (l'AXI Interconnect dans la figure 17) qui compare la valeur des signaux de sécurité avec l'état de sécurité de l'IP destinatrice. La figure 18 présente quatre modifications matérielles malveillantes pour réaliser l'escalade de privilège et le déni de service. Les modifications malveillantes visent à modifier le signal de sécurité AWPROT[1] du canal d'adresse d'écriture (les 5 canaux du bus AXI sont détaillés dans la section chap1-1.1) avant sa comparaison avec l'état de sécurité de l'IP destinatrice. Les mêmes modifications peuvent être utilisées pour compromettre le signal ARPROT[1] du canal d'adresse de lecture du bus AXI. Certaines des modifications matérielles malveillantes utilisent un trigger qui permet de déclencher la fonction malveillante uniquement au moment où on en a besoin, et revenir au fonctionnement normal. Le trigger permet de cacher la fonction malveillante durant le test structurel du circuit et éliminer les soupçons.

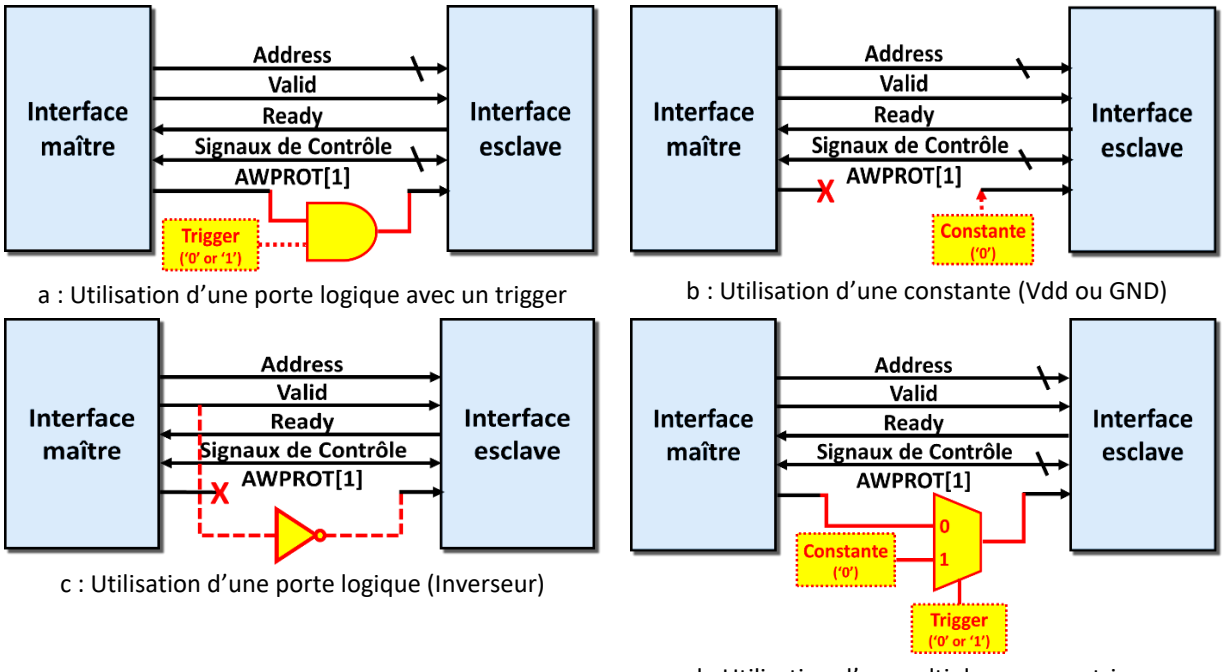


Figure 17: Quatre exemples de modifications matérielles malicieuses ciblant le signal de sécurité AWPROT[1]

Dans la modification matérielle malicieuse de la figure 18-a, le signal AWPROT[1] est contrôlé par une porte logique ''et'' entre l'interface maître et l'interface esclave du contrôleur TrustZone. La porte logique ''et'' a comme entrée, un signal trigger utilisé pour activer la fonction

malveillante et le signal AWPROT[1]. Si le trigger est à l'état logique haut, le système fonctionne normalement. Dans le cas contraire, toutes les transactions émises par l'interface maître seront vues comme sécurisées par le contrôleur TrustZone (escalade de privilège). Cette modification malicieuse permet au monde non-sécurisé d'utiliser toutes les ressources sécurisées connectées au contrôleur TrustZone visé par l'attaque.

Dans La modification matérielle malicieuse de la figure 18-b, le signal AWPROT[1] est corrompu entre l'interface maître et l'interface esclave. Et l'entrée du signal AWPROT[1] sur l'interface esclave est connectée à une source constante, soit GND pour le fixer au niveau logique bas, soit Vdd pour le fixer au niveau logique haut. Si la constante est fixée à GND, la modification malicieuse permet de réaliser une escalade de privilège depuis le monde non-sécurisé. Dans le cas contraire, la modification malicieuse permet de réaliser un déni de service, condamnation des IP sécurisées.

Dans la modification matérielle malicieuse de la figure 18-c, le signal AWPROT[1] est corrompu entre les deux interfaces comme dans la modification de la figure 18-b, et l'entrée du signal AWPROT[1] est connectée à la sortie d'un inverseur du signal Valid du canal d'adresse d'écriture. Cette modification se base sur le fait que durant le handshake (signal Valid active), l'interface esclave fait une capture de l'état des signaux de contrôle (enregistre la valeur des signaux dans des registres dédiés), signale AWPROT[1] inclus. Du coup, si l'interface esclave récupère l'inverse du signal Valid pour le signal de sécurité AWPROT[1], toutes les transactions provenant de l'interface maître seront sécurisées pour l'AXI Interconnect. Cette modification permet de réaliser seulement une escalade de privilège.

Dans la modification matérielle malicieuse de la figure 18-d, le signal AWPROT[1] est contrôlé par un multiplexeur avec deux entrées, le signal AWPROT[1] et une constante, et un trigger comme signal de contrôle du multiplexer. Le trigger permet de choisir entre le fonctionnement normal ou malveillant. Le choix de la constante (Vdd ou GND) dépend de l'attaque à implémenter.

3.1.2. L'utilisation d'un script malicieux

Cette section présente l'utilisation d'un script malicieux pour réaliser automatiquement les modifications présentées dans la figure 18. Le script malicieux peut être généré par un outil d'automatisation de flot de conception tiers malveillants (Outil de CAO), ou développer par un concepteur compromis. Les scripts TCL sont utilisés partout durant la phase de conception et cela

pour plusieurs raisons, notamment pour l'automatisation du processus d'implémentation, la vérification de l'exactitude d'une conception (DRC), la livraison d'un design à un autre utilisateur, etc. Un script malicieux constitue un chemin d'attaque efficace et une grande menace durant la conception d'un SoC. Mais la plus grande menace est l'exécution des scripts provenant de tiers non fiable sans vérification, soit par un manque de temps, soit par un manque d'expérience.

Par exemple, il n'est pas difficile de réaliser la modification de la figure 18-b en utilisant les commandes TCL. Un attaquant peut utiliser les deux commandes TCL présentées dans la figure 19, la première commande déconnecte le signal AWPROT[1] de l'entrée de l'AXI Interconnect, et la deuxième commande connecte l'entrée débranchée au GND afin de faire apparaitre comme des transactions sécurisées toutes les transactions ciblant les IP connectées à l'AXI Interconnect. Pour ne pas laisser de trace dans le journal des commandes de l'outil CAO, certaines commandes ont des options notamment ***–notrace*** et ***–quiet*** qui permettent de ne pas apparaitre aux yeux du concepteur.

Disconnect_net **–quiet** –net [get_nets *design/signal_AXI_AWPROT[1]*] –objets [get_pins *design/AXI_Interconnect/S_AXI_AWPROT[1]*]

Connect_net –hier **–quiet** –net *design/<constant0>* –objets [get_pins *design/AXI_Interconnect /S_AXI_AWPROT[1]*]

Figure 18: Deux commandes TCL à ajouter au script pour modifier automatiquement et discrètement le code RTL

3.2. Compromettre les signaux de réponse BRESP/RRESP

Après la vérification des signaux de sécurité par l'AXI Interconnect, une réponse est envoyée à l'interface maître en utilisant les signaux de réponse à 2 bits BRESP et RRESP (détaillés dans la section chap1-1.1). Ces signaux sont essentiels pour la gestion de la sécurité dans un SoC complexe hétérogène embarquant la TrustZone. En conséquence, ces signaux sont des cibles d'attaque appropriées.

3.2.1. Exemple de modification matérielle visant les signaux BRESP/RRESP

La figure 20 montre une modification matérielle malveillante possible du signal BRESP du canal de réponse du bus AXI (cette modification peut être utilisée aussi pour attaquer le signal RRESP du canal lecture de donnée). Certaines de ces modifications matérielles malveillantes présentées dans la figure 18 sont également valables dans ce contexte. Les deux bits du signal renvoyés à l'interface maître comme réponse sont contrôlés par un multiplexeur 4x1 avec un trigger comme signal de contrôle. Le trigger permet de choisir de forcer ou non la réponse de

l'esclave. En fonction du scénario d'attaque, ce multiplexeur pourrait être plus petit et capable de ne forcer qu'une ou deux des réponses possibles.

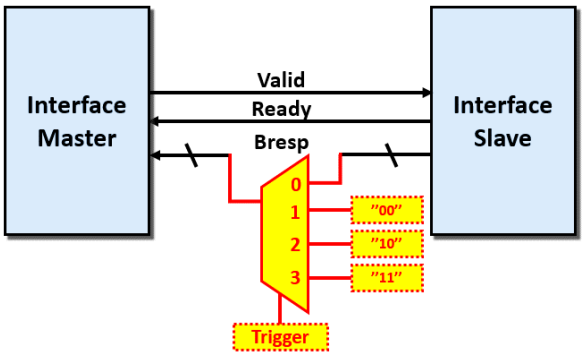


Figure 19: Exemple de modification matérielle malveillante du signal BRESP

Lorsque le signal BRESP est forcé sur la valeur binaire "00", la réponse de l'esclave est forcée sur OK même en cas de violation de l'état de sécurité. Dans cette attaque, la requête de communication est rejetée par l'AXI Interconnect et l'interface maître n'est pas informé pour prendre les mesures nécessaires contre la violation de sécurité.

Lorsque le signal BRESP est forcé sur les valeurs binaires "10" ou "11", la réponse de l'esclave est forcée sur SLVERR ou DECERR, même dans le cas d'une transaction correcte. Cette attaque permet de créer un déni de service, l'interface maître reçoit un message d'erreur pour chaque requête qu'elle émet au bus.

3.2.2. Modification de la configuration de la LUT

Pour compromettre le signal BRESP, une autre forme d'attaque possible consiste à modifier malicieusement la configuration des LUT (LookUp Table) après synthèse du design. En effet, le vecteur d'initialisation stocké dans une LUT est responsable de sa configuration et donc de la fonction logique entre les entrées et les sorties de la LUT. Un développeur interne malveillant qui a accès au design après synthèse peut modifier malicieusement les vecteurs d'initialisation des LUT critiques, ce qui modifie leur fonction logique ainsi que le comportement normal du design. La modification peut être effectuée directement depuis l'onglet de configuration de la LUT dans l'outil CAO, ou en utilisant la commande TCL présentée dans la figure 21.

```
Set_property INIT "32'h00000000" [get_cells cell_name]
```

Figure 20: Commande TCL pour modifier le vecteur d'initialisation d'une LUT.

Cette attaque demande une étape de rétro-ingénierie pour choisir les LUT critiques à cibler. Pour cela, différentes configurations de l'AXI Interconnect sont synthétisé pour choisir les LUT

critiques qui sont stables et qui ne changent ni de vecteur d'initialisation (la même fonction logique) ni de nom. Par la suite, les LUT qui affectent la valeur du signal BRESP sont sélectionnées. Enfin, deux LUT avec les noms ***s_axi_bresp[0]_INST_0*** et ***s_axi_bresp[1]_INST_0*** sont ciblées. Ces deux LUT sont initialisés par une fonction logique qui contrôle le signal BRESP avant de l'envoyer à l'interface maître. Pour ne pas signaler d'erreur à l'interface maître, la fonction logique des deux LUT est forcée à générer des sorties nulles (vecteur d'initialisation "***32'h00000000***") peu importe l'état de ses entrées.

Si l'attaquant ne connaît pas les noms des IP utilisées ou des LUT générées après synthèse, il peut utiliser les commandes TCL proposée par l'outil CAO Vivado [66] pour profiler le design et trouver les éléments à cibler. Ces commandes TCL peuvent être utilisées pour généraliser l'attaque et cibler un grand nombre de design.

3.3. Compromettre l'AXI Interconnect

Dans le design de la figure 13, l'AXI Interconnect joue un rôle important dans la sécurité du système. Comme contrôleur TrustZone, il compare l'état de sécurité des signaux de sécurité du bus AXI et l'état de sécurité de l'IP destinatrice, il sauvegarde les états statiques de sécurité de toutes les IP connectées et il envoie des messages d'erreur pour informer le système de toute violation de sécurité. Cette section présente des attaques qui peuvent cibler l'AXI Interconnect et donc le système complet.

3.3.1. Modification malicieuse du code du Crossbar

Le Crossbar (détaillé dans la section chap1-2.3.2) est le bloc central de l'AXI Interconnect. Il est chargé de vérifier l'état de sécurité de chaque transactions en comparant les signaux de sécurité AWPROT[1]/ARPROT[1] avec l'état de sécurité statique de l'IP matérielle ciblée. En cas de violation, il envoie une erreur à l'interface maître par le bus AXI. Pour cela, une attaque est présentée par modification malveillante du code Verilog du Crossbar dans cette section.

Avant de réaliser l'attaque, il faut effecteur une étape de rétro-ingénieur pour comprendre et analyser le code Verilog du Crossbar. Cette étape a révélé l'importance des deux lignes de code présentées dans la figure 22-a. Les deux lignes sont responsables de la vérification de sécurité, la comparaison entre le signal de sécurité AWPROT[1] (***S_APROT[P_NONSECURE_BIT]***, dans le code) et l'état de sécurité statique de l'IP ciblée (***target_secure***), et la génération d'erreur de sécurité (***any_error_i[1]***). Dans le cas de violation de sécurité la valeur du signal ***any_error_i[1]*** est égale à '1', sinon '0'. Le signal ***target_secure*** est le résultat du ''et'' logique bit à bit entre le

signal ***target_mi_hot*** qui est un vecteur contenant l'ID de l'IP ciblée, et le vecteur ***P_M_SECURE_MASK*** qui contient l'état de sécurité des interfaces maîtres connectées à l'AXI Interconnect.

```
assign target_secure = |(target_mi_hot & P_M_SECURE_MASK);
assign any_error_i[1] = target_secure && S_APROT[P_NONSECURE_BIT];
```

a. Code Verilog d'origine de l'opération de vérification

```
assign target_secure = |(target_mi_hot & P_M_SECURE_MASK);
assign any_error_i[1] = (target_mi_hot == 2'b01) ? 1'b0 : (target_secure && S_APROT[P_NONSECURE_BIT]);
```

b. Code Verilog modifié

Figure 21: Modification malveillante du code Verilog du Crossbar de l'AXI Interconnect

Plusieurs attaques sont possibles en modifiant malicieusement les deux lignes du code Verilog de la figure 22-a. Tous les bits du vecteur ***P_M_SECURE_MASK*** peuvent être validés (tous les interfaces maîtres sont non-sécurisées) pour créer une escalade de privilège, le monde non-sécurisé accède aux IP sécurisées, et le monde sécurisé a son fonctionnement normal due au fait que le monde sécurisé a accès à la totalité des ressources. Les bits du vecteur ***P_M_SECURE_MASK*** peuvent être mis à zéro (tous les interfaces maîtres sont sécurisées) pour créer un déni de service au monde non-sécurisé, tout accès depuis le monde non-sécurisé génère une erreur. La valeur du signal **any_error_i[1]** peut être fixée pour la vie du système, si la valeur fixée est à '0', le Crossbar ne remonte jamais d'erreur. Dans le cas contraire, le Crossbar envoie toujours un message d'erreur au système à l'interface maître.

La figure 22-b montre une modification qui vise le fonctionnement d'une IP sécurisée spécifique. Lorsque la transaction cible cette IP depuis une interface maître non-sécurisée, le Crossbar autorise la transaction au lieu de la rejeter et il envoie un message d'erreur. Lorsque les transactions ciblent le reste des IP, le système se comporte normalement. Le code est modifié par l'ajout d'une branche conditionnelle afin d'obtenir un accès privilège lorsque le bit du vecteur **target_mi_hot** lié à l'IP cible est valide et a un comportement normal pour les autres IP.

3.3.2. La modification malicieuse des configurations de sécurité de l'AXI Interconnect

Cette section présente une attaque qui utilise les commandes TCL pour attaquer l'AXI Interconnect. Cette attaque cible à modifier l'état de sécurité statique des IP connectées à l'AXI

Interconnect durant la phase de conception. Les commandes TCL peuvent être dissimulées dans un script d'automatisation de l'étape de synthèse ou dans l'étape de création de design.

```
Set_property CONFIG.M00_Secure false [get_bd_cells AXI_Interconnect]
```

Figure 22: Commande TCL malveillante pour changer l'état de sécurité de l'interface ''M00''

L'AXI Interconnect utilisé dans le design d'exemple de la figure 13 offre 16 emplacements d'interfaces maîtres à connecter aux IP (interface esclave). La figure 23 montre la modification de l'état de sécurité de l'interface maître numéro 0, l'interface maître ''M00''. Cette commande change l'état de l'IP connectée à l'interface maître ''M00'' en non-sécurisée. La variable ***P_M_SECURE_MASK*** présentée dans la section précédente est affectée car elle sauvegarde les états de sécurité statique des 16 interfaces dans le code Verilog du Crossbar. La modification de la figure 23 conduit à une escalade de privilège, elle permet à une interface maître non-sécurisée d'utiliser l'IP connectée à l'interface ''M00'' de l'AXI Interconnect.

3.3.3. L'insertion d'une FIFO malicieuse

En 2016, Fern et al. [67] ont présenté pour la première fois l'insertion d'une FIFO malicieuse dans l'AXI Interconnect pour fuiter des données sensibles. Comme dans [67], une FIFO malicieuse est utilisée pour espionner les transactions sécurisée entre l'interface maître et une IP sécurisée (interface esclave verte, figure 24). Supposons que la FIFO malicieuse est insérée dans le code RTL de l'AXI Interconnect durant la phase de conception. La taille de la FIFO malicieuse dépend de la quantité de données à divulguer, mais elle doit être aussi petite que possible pour assurer le succès de cette attaque et le non détection de la modification.

La figure 24 présente les modifications malicieuses apportées à l'AXI Interconnect pour l'insertion de la FIFO malicieuse. En générale, une FIFO a besoin de deux conditions d'activation, une pour la lecture de donnée et une autre pour l'écriture de donnée. Dans ce scénario d'attaque, le handshake du canal d'écriture connecté à l'interface esclave sécurisée est utilisé comme condition d'écriture et le handshake du canal d'écriture connecté à l'interface esclave non-sécurisée est utilisé comme condition de lecture. L'attaquant peut ajouter des signaux de contrôle pour indiquer le début et la fin de l'enregistrement ou pour indiquer le remplissage de la FIFO malicieuse.

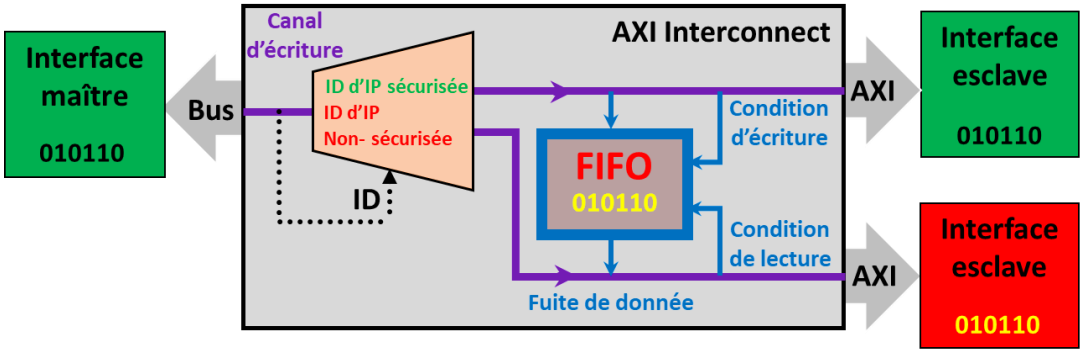


Figure 23: L'insertion d'une FIFO malveillante au niveau de l'AXI Interconnect pour espionner les données envoyées vers une IP sécurisée

Durant une transaction entre l'interface maître et l'interface esclave sécurisée, la FIFO malicieuse enregistre les données échangées. Par la suite, l'interface esclave non-sécurisée récupère les données enregistrées dans la FIFO durant une requête de lecture envoyée par l'interface maître (trame de donnée ''010110'' en jaune, figure 24).

4. Scénario complet d'attaque visant l'outil CAO Vivado de Xilinx

Les sections précédentes de ce deuxième chapitre ont déjà discuté la problématique des attaques basées sur l'utilisation des outils CAO qui ne sont pas de confiance sans présenter de scénario complet d'attaque. Cette section présente un scénario d'attaque complet visant l'outil CAO Vivado de Xilinx. Cette section démontre aussi l'importance de la sécurité des outils CAO utilisés pour la création des SoC hétérogènes de confiance.

4.1. Fichiers de descriptions matérielles dans un projet Vivado

La figure 25 présente un exemple de design créé par l'outil Vivado. Le design contient l'IP Zynq7 Processing System qui permet de configurer la partie processeur du Zynq-7000, l'AXI Interconnect, le Processeur System Reset et un AXI GPIO qui contrôle en sortie quatre Leds.

La figure 26 présente la description hiérarchique du design de la figure 25, cette description est générée dans l'onglet source de l'outil CAO Vivado en même temps que la création du design. La description hiérarchique présente la hiérarchie du design avec les noms (le nom est indiqué entre parenthèse devant le nom du bloc intégré dans le design) des fichiers contenant la description matérielle des IP intégrées dans le design. Ces fichiers sont copiés depuis la bibliothèque standard fournie par Xilinx dans le dossier du projet.

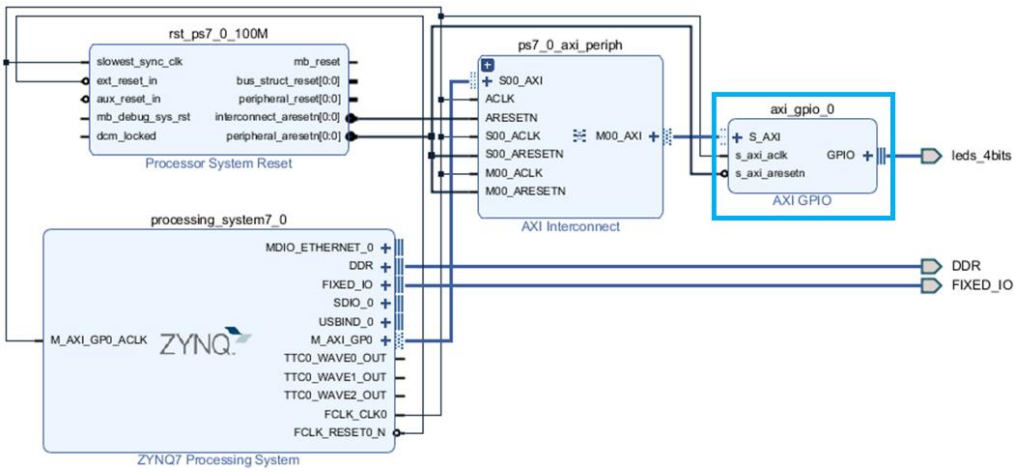


Figure 24: Design créé à l'aide de l'outil Vivado

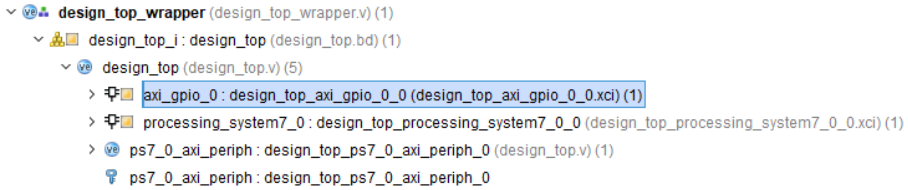


Figure 25: La description hiérarchique du design de la figure 25

Les fichiers contenant la description matérielle d'une IP Xilinx sont séparés en deux types. Des fichiers contenant la description matérielle en Vhdl ou Verilog de l'IP et qui sont accessible en lecture seule au concepteur dans l'interface de l'outil Vivado, comme le montre le rectangle bleu en haut à droite de la figure 27. Le deuxième type est des fichiers sécurisés par l'outil Vivado pour protéger la propriété intellectuelle. Ce deuxième type de fichiers contient seulement les entités des sous blocs constituant l'IP.

Figure 26: Fichier accessible en lecture seulement

4.2. Scénario d'attaque complet

Le scénario d'attaque présenté dans la figure 28 vise à modifier malicieusement le code source de l'IP GPIO intégrée dans le design de la figure 25. Comme indiqué dans la section précédente les fichiers de l'IP GPIO ciblé par l'attaque ne peuvent pas être modifiés par le

concepteur dans l'outil Vivado car ce dernier les ouvre seulement en lecture dans son interface. Par contre, ils sont modifiables en dehors de l'outil Vivado à l'aide d'un simple éditeur de texte. Dans ce scénario d'attaque un programme malicieux installé dans le système est utilisé pour modifier les fichiers de l'IP GPIO.

Figure 27: L'emplacement du fichier cible

L'attaque demande une étape de rétro-ingénierie pour comprendre la description matérielle de l'IP GPIO et trouver les lignes de code critique à modifier. L'attaque commence par rechercher l'emplacement du fichier à cibler, pour cela le programme malicieux utilise le fichier ***nom_du_projet.xpr*** qui se trouve dans la racine du dossier du projet Vivado. Ce fichier est un fichier XML qui contient toutes les informations concernant le projet, comme la configuration de l'environnement de travail, les différents fichiers utilisés dans le projet, etc. La figure 28 présente le bout du code XML qui présente la liste des différents IP utilisés dans ce projet et l'emplacement de leurs fichiers. Une fois le fichier cible trouvé, le programme malicieux peut éditer les lignes du code à modifier.

Pour le succès de ce scénario d'attaque, le programme malicieux doit exécuter sa routine avant l'étape de synthèse et à chaque modification du design car l'outil Vivado recopie à nouveau les fichiers des IP utilisées dans le design depuis la bibliothèque standard. La recopie écrase la version du fichier modifié par le programme malicieux.

5. Conclusion :

Ce chapitre présente les méthodes à suivre pour propager l'utilisation de la technologie ARM TrustZone dans la partie reconfigurable d'un SoC FPGA. Ce chapitre présente également des

attaques qui visent la sécurité du bus AXI dans un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Les attaques démontrent l'importance de la sécurité des signaux du bus AXI et de ses composantes (le Crossbar par exemple). Le chapitre montre que les outils CAO et les IP tiers provenant des fournisseurs qui ne sont pas de confiance sont une menace de sécurité pour les systèmes.

Chapitre 3 : Attaques DMA et par manipulation malicieuse du DVFS

Ce chapitre présente une attaque DMA visant la région sécurisée de la mémoire externe et les registres de configurations de la TrustZone. Le chapitre présente aussi des attaques par canal dérobé visant la sécurité d'un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Les attaques par canal dérobé présentées se basent sur l'utilisation malicieuse de la technique DFS qui fait varier seulement la fréquence (modifie les paramètres de la PLL) pour réduire la consommation d'énergie.

1. Design expérimental

Cette section présente le design expérimental utilisé pour la mise en œuvre des attaques présentées dans la suite de ce manuscrit. Le design expérimental est implémenté à l'aide du SoC Xilinx Zynq-7000, mais il peut être implémenté dans tous les SoC FPGA embarquant la technologie ARM TrustZone.

La figure 29 présente le design expérimental implémenté dans le SoC Xilinx Zynq-7000. Les IP matérielles de la partie reconfigurable du SoC FPGA sont partitionnées en deux : IP sécurisées et IP non-sécurisées, en utilisant la méthode détaillée dans la section chap2-2. Les deux IP ont un accès direct à la mémoire en utilisant l'interface ACP (détaillé dans la section chap1-1.2.5). La partie processeur du SoC Xilinx Zynq-7000 inclue deux cœurs ARM, chacun de ces deux cœurs est dédié à un monde, le cœur ARM sécurisé exécute les applications critiques, le cœur ARM non-sécurisé exécute les applications normales. La mémoire externe est partitionnée aussi en une région sécurisée et une non-sécurisée en utilisant les registres de configuration du TZMA (détaillé dans la section chap1-2.2.3). La région sécurisée de la mémoire externe enregistre les applications critiques et la région non-sécurisée contient le reste des applications.

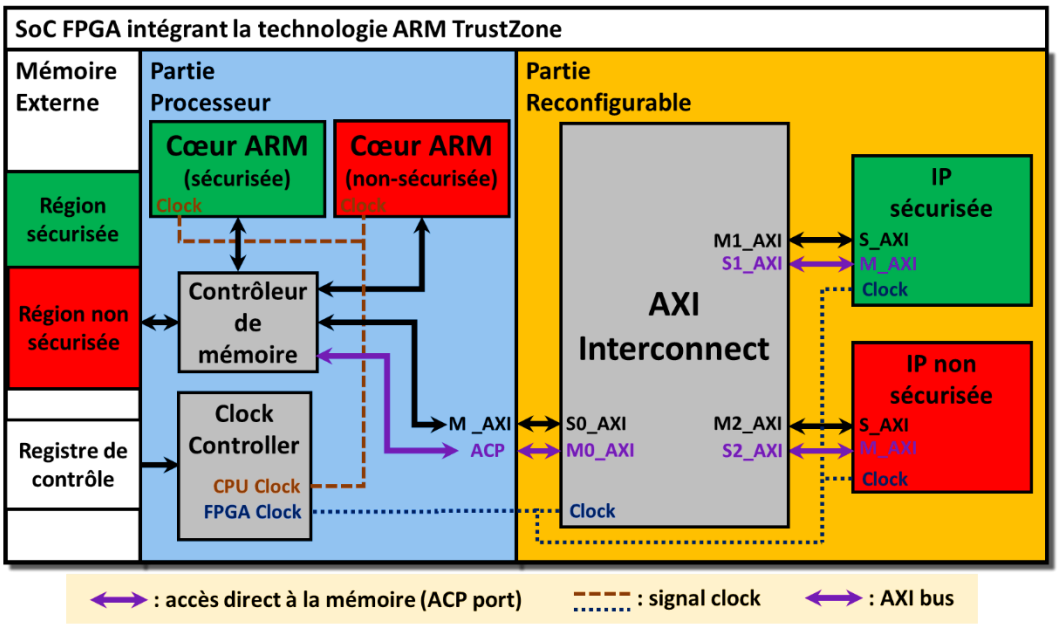


Figure 28: Design expérimental

2. Attaque DMA

Cette section présente l'attaque DMA, que nous avons proposé [65], qui vise les applications et les données sensible du monde sécurisé dans un SoC FPGA. La figure 30 présente le chemin de l'attaque DMA, l'IP non-sécurisée inclut un cheval de Troie qui utilise l'accès direct à la mémoire pour compromettre la sécurité du monde sécurisé. L'IP non-sécurisée (le cheval de Troie) est connectée à la partie processeur à l'aide de l'interface ACP qui permet d'accéder à la totalité de la mémoire externe, inclue les registres de configuration. Les transactions entre l'IP non-sécurisée et la mémoire externe ne sont pas contrôlées par le processeur ARM. Cela met en danger la sécurité du système complet, si un processus malveillant profite de l'accès direct non-supervisé à la mémoire pour installer un malware ou fuiter les données sensibles.

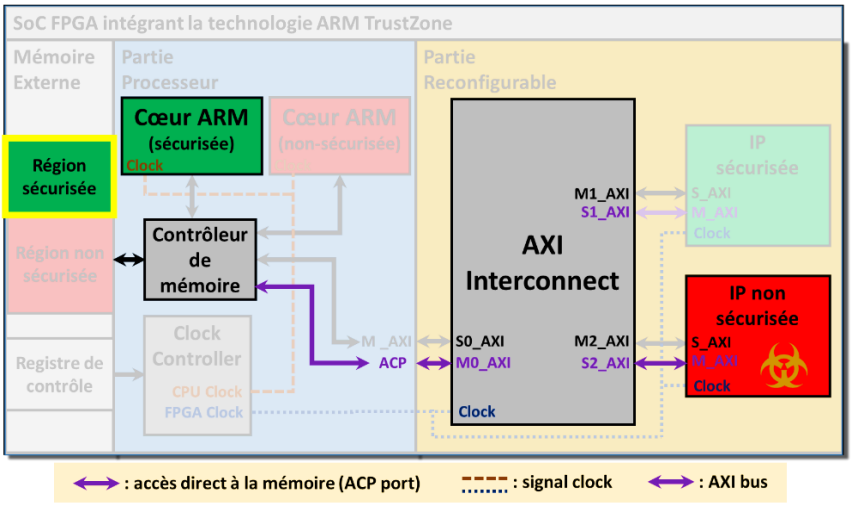


Figure 29: Attaque DMA

Dans le chemin d'attaque présenté dans la figure 30, l'IP non-sécurisée utilise une interface maître pour communiquer avec l'interface esclave (l'interface ACP) de la partie processeur. Comme indiqué dans la section chap2-1, une interface maître contrôle l'état de sécurité des requêtes envoyées vers l'interface esclave (la mémoire externe). Donc, si l'IP non-sécurisée envoie une requête sécurisée vers la région sécurisée de la mémoire externe, le contrôleur TZMA autorise l'accès et l'IP non-sécurisée récupère les données sensibles. Ce scénario d'attaque est possible seulement si la partie processeur autorise l'accès à la région sécurisée de la mémoire externe depuis la partie reconfigurable.

Si cette attaque cible une application sécurisée du monde sécurisé, l'attaquant doit connaitre l'emplacement de l'application dans la mémoire externe et cette tâche n'est pas facile.

Sinon, l'attaquant peut viser les parties de la mémoire externe documentées par le fournisseur du SoC attaqué, comme les registres de configuration. Dans le Zynq-7000, les registres de configuration sont une bonne cible car ils ne sont pas bien protégés par le système et ils sont bien détaillés. Par exemple, si l'attaque DMA cible la configuration du registre contrôlant le partitionnement de la mémoire externe en deux, elle peut changer certaines régions en non-sécurisées et les rendre ainsi accessibles depuis le monde non-sécurisé qui peut les exploiter malicieusement. Le monde sécurisé n'est pas affecté par le changement car il a accès à la totalité des ressources du système peu-importe leurs états de sécurité.

3. Attaques par canal dérobé basé sur l'utilisation malicieuse du système DFS

Aujourd'hui, l'une des menaces auxquelles sont confrontés les SoCs complexes hétérogènes est la transmission secrète de données sensibles par canal dérobé. Cette attaque permet à un attaquant de transférer des données sensibles entre des processus qui ne sont pas autorisés à communiquer par les politiques de sécurité du système. En général, une transmission par canal dérobé utilise un processus espion qui infiltre le système et transfère les données sensibles à un processus récepteur qui les décode et les utilise à des fins malveillantes.

Dans la littérature, il existe plusieurs méthodes pour créer un canal dérobé, mais la majorité des méthodes utilisent les ressources partagées telles que la mémoire partagée. Dans [58], Lipp et al. ont utilisé une bibliothèque partagée (une mémoire partagée) et les attaques de mémoire de cache pour échanger des données sensibles entre deux processus non privilégiés. Le moyen de communication secrète entre les deux processus, espion et récepteur, se base sur l'utilisation soit de l'attaque de la mémoire cache Flush+Reload soit l'attaque de la mémoire cache Flush+Flush (détaillées dans la section chap1-4.3.1.2). Dans le cas de l'utilisation de l'attaque de cache Flush+Reload, leur processus espion envoie un 1 logique en accédant à une adresse de la bibliothèque partagée et envoie un 0 logique en renvoyant l'adresse de la mémoire cache. Leur processus récepteur mesure seulement le temps d'accès à l'adresse accédée par le processus espion pour décoder les états logiques 0 et 1.

Dans [68], Masti et al. ont démontré la faisabilité d'un canal dérobé thermique. Ils ont utilisé le capteur thermique inclus dans un cœur du processeur pour faire communiquer deux processus qui sont exécutés sur deux cœurs différents du même processeur. Pour envoyer un 1 logique, leur processus espion stresse le cœur qu'il utilise pour chauffer le processeur. Pour envoyer un 0

logique, le processus espion diminue la charge de travail du cœur. Pour décoder les données, leur processus récepteur effectue seulement des lectures de la température du cœur appartenant au même processeur.

Dans [69], Alagappan et al. ont démontré la faisabilité d'un canal dérobé en utilisant la modulation de fréquence. Ils ont utilisé la technique DFS pour transférer des données sensibles entre le processus espion et le processus récepteur. Pour envoyer un 1 logique, leur processus espion stresse le processeur comme dans [68], cela pousse le système à changer sa fréquence pour répondre à la charge de travail exécuté par leur processus espion. La fréquence choisie dépend du mode de gouverneur de fréquence utilisé par le système (performances, économie d'énergie, espace utilisateur, en demande, conservateur). Pour envoyer un 0 logique, leur processus espion diminue la charge de travail appliquée au processeur. Le processus récepteur effectue une simple lecture de la fréquence pour décoder les 0 et les 1 logiques.

Comme dans [69], les preuves de concepts présentées dans la suite de ce chapitre utilisent également la modulation de fréquence pour envoyer des données sensibles entre un processus espion et un processus récepteur. Mais contrairement à [69], les deux processus ont des statuts de sécurité différents dans un SoC FPGA embarquant la technologie ARM TrustZone. De plus, les quatre preuves de concept utilisent une modification directe du registre lié au régulateur de fréquence. Ce chapitre introduit également pour la première fois une communication secrète dans un SoC FPGA, qui est la communication secrète entre une IP matérielle embarquée dans la partie reconfigurable et un cœur ARM de la partie processeur.

3.1. Preuve de concept #1 : Transfert de données sensibles du cœur ARM sécurisé vers l'extérieur du SoC

La figure 31 présente le chemin d'attaque de la première preuve de concept (chemin d'attaque n° 1, figure 31), les données sensibles exécutées par le cœur ARM sécurisé sont transférées vers l'extérieur du SoC FPGA par émission électromagnétique. En 2015, Bossuet et al. [70] ont démontré que le canal électromagnétique est un canal dérobé puissant pour la transmission discrète de donnée sensible vers l'extérieur d'un SoC. Ils ont utilisé un circuit malveillant exploitant un oscillateur en anneau configurable (processus espion) pour faire de la modulation de fréquence, et un analyseur de spectre en temps réel (processus récepteur) pour démoduler les données fuitées.

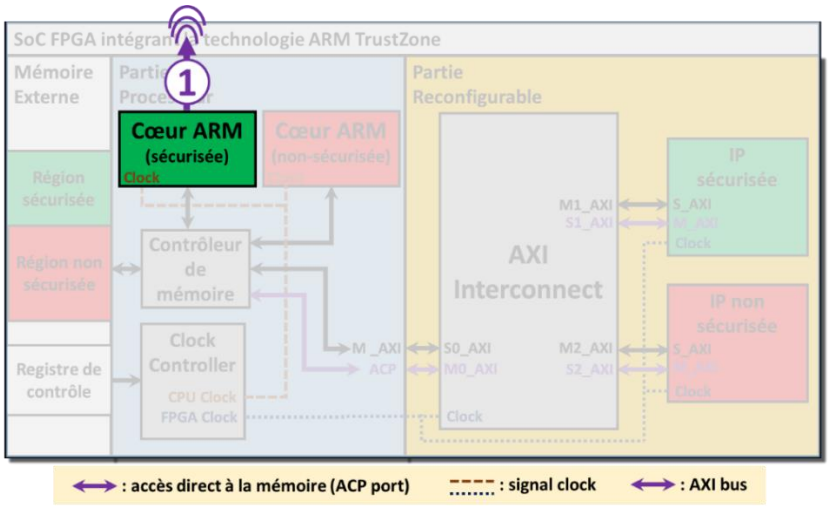


Figure 30: Transfert des données sensibles vers l'extérieur du SoC FPGA

Dans nos travaux [71], contrairement à [70], la preuve de concept ne se base pas sur une modification malveillante introduite dans le SoC durant sa conception. La preuve de concept utilise la technique DFS comme dans [69] pour faire de la modulation de fréquence et fuiter les données par émission électromagnétique. Par contre, le processus récepteur est le même que celui utilisé dans [70]. La figure 32 présente le dispositif pour le décodage de l'information fuitée, une sonde électromagnétique pour capter les émissions électromagnétiques du SoC et un analyseur de spectre en temps réel pour décoder le flux de données.

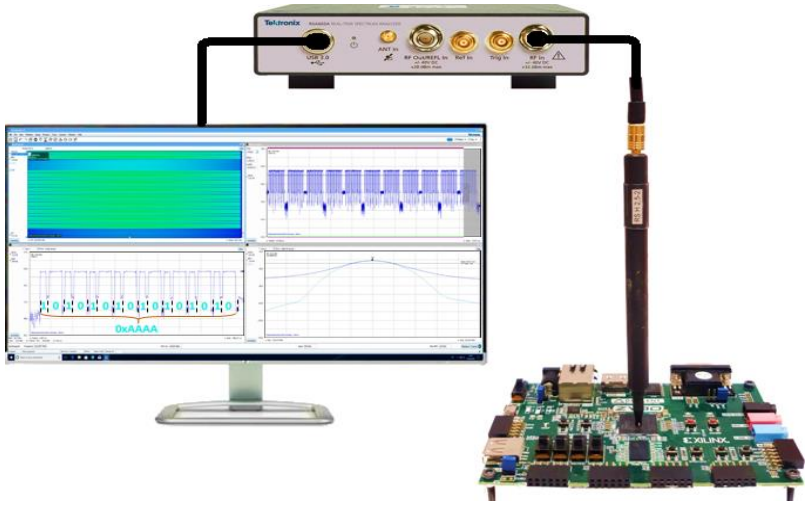


Figure 31: Analyse spectrale en temps réel de la fuite électromagnétique de données secrètes

Dans la majorité des attaques basées sur les émissions électromagnétiques, les attaquants ont besoin d'effectuer une cartographie électromagnétique pour localiser l'emplacement de la modification malicieuse au niveau du SoC, et du prétraitement du flux de donnée pour réduire le

bruit et extraire l'information utile. Dans la preuve de concept #1, l'attaquant a besoin d'un simple balayage manuel pour révéler la position de la fuite et capter un signal puissant sur l'analyseur de spectre. La puissance du signal permet de décoder directement le flux de données sur l'écran de l'analyseur de spectre en temps réel sans prétraitement. Ce gain de temps est dû au fait que la technique DFS manipule une horloge qui alimente la majorité des composants de la partie processeur, les files de l'horloge sont enracinés dans la partie processeur.

> Dans une cartographie électromagnétique, la cible sous test est positionnée sur une table XY (XYZ parfois) qui la fait bouger par rapport à une sonde qui capture à chaque pas ses émissions électromagnétique. Le pas est de quelque micromètre.

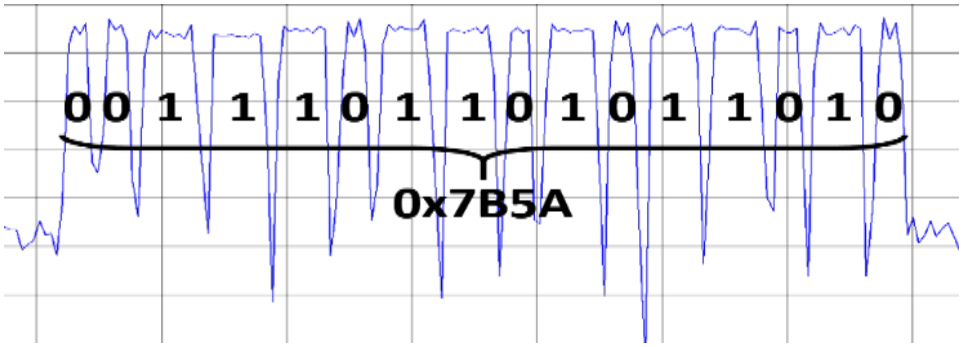
Pour le processus espion, le pilote de la technique DFS du système d'exploitation de confiance inclus le code malveillant présenté dans l'algorithme 1. Le code malveillant utilise la modulation de fréquence pour transférer les données sensibles. Pour transférer un 1 logique, l'algorithme 1 force le système DFS à garder une fréquence *freq_1* (*freq_actuelle = freq_1*) pour une temporisation *tempo_1*. Pour transférer un 0 logique, l'algorithme 1 force le système DFS à garder la même fréquence mais cette fois ci pour une temporisation *tempo_2*. Entre l'envoie de deux bits successifs, l'algorithme 1 force le système DFS à grader une fréquence *freq_2* (*freq_actuelle = freq_2*) pour une courte temporisation (*Tempo_3*). Après l'envoie de toute la trame de donnée, l'algorithme 1 remets la fréquence normale de fonctionnement du système (*freq_actuelle = freq_normal*).

Algorithme 1: Modulation de fréquence

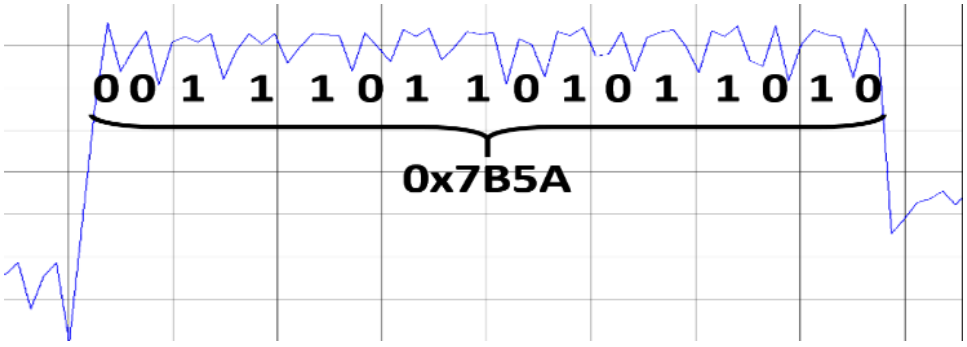
```
Input: donnée_à_transférer
    For i = donnée_à_transférer_size To 0 Do
        If (donnée_à_transférer[i] = 1) Then
          freq_actuelle = freq_1
          loop for Tempo_1
        Else
          freq_actuelle = freq_1
          loop for Tempo_2
          End If
        freq_actuelle = freq_2
        loop for Tempo_3
     End For
     freq_actuelle = freq_normal
```

L'utilisation d'une même fréquence (*freq_1*) pour transférer les 1 et les 0 logiques permet de centrer l'analyseur de spectre en temps réel sur une seule fréquence pour la réception de la

totalité des données fuitées. La différentiation entre un 1 et un 0 logique se fait par les deux temporisation *tempo_1* et *tempo_2* qui permettent d'avoir deux motifs distingués par leurs longueurs comme illustrée dans la figure 33-a. La figure montre aussi l'importance des paramètres *Tempo_3* et la *freq_2* pour différencier entre deux bits successifs de la même valeur et pour décoder directement les données à l'écran de l'analyseur de spectre en temps réel. La figure 33-a montre le décodage à l'écran de la trame 0x7B5A, si le décodage du motif large est pour un 1 logique et le court pour un 0 logique.



a: *freq_1* = 325MHz, *freq_2* = 433MHz, *Tempo_1* = 400, *Tempo_2* = 200, *Tempo_3* = 200



b: *freq_1* = 325 MHz, *freq_2* = 433 MHz, *Tempo_1* = 200, *Tempo_2* = 100, *Tempo_3* = 25

Figure 32: Décodage des données sur l'écran de l'analyseur de spectre en temps réel, a - bande passante = 1,42.105 bps, b - bande passante = 3,33.105 bps

Le choix des temporisations (*Tempo_1, Tempo_2 et Tempo_3*) dans l'algorithme 1 joue un rôle important sur la taille de la bande passante du canal dérobé. Si les temporisations choisies sont trop élevées (figure 33-a), il est simple de décoder les données reçues directement à l'écran mais la bande passante est plus petite. Si les temporisations choisies sont trop petites (figure 33-b), le signal devient très bruité ce qui rend difficile le décodage direct de l'information transmise. La figure 33-b montre la limite inférieure des trois temporisations pour la *freq_1. E*n dessous de cette limite l'analyseur de spectre n'affiche que du bruit.

Le choix de la fréquence *freq_1* joue aussi sur la taille de la bande passante du canal dérobé. Si elle est élevée avec des temporisations petites, la bande passante est grande. Dans le cas contraire la bande passante est petite. Le choix de la fréquence *freq_1* peut être aussi limité par la cible attaquée car certain SoC ont une plage de fonctionnement fixée par les concepteurs. En dehors de cette plage de fonctionnement le SoC ne fonctionne pas correctement.

3.2. Preuve de concept #2 : Transfert de données sensibles du cœur sécurisé vers le cœur non-sécurisé

La preuve de concept #2 est un transfert de données sensibles du cœur ARM sécurisé vers le cœur non-sécurisé (Chemin d'attaque n° 2, figure 34). Les deux cœurs Cortex-A9 du SoC Xilinx Zynq-7000 ne sont pas bien isolés au niveau de la gestion d'énergie, les deux cœurs partagent la même source d'horloge, si l'un des cœurs modifie la fréquence (les paramètres de la PLL), la fréquence change pour les deux cœurs.

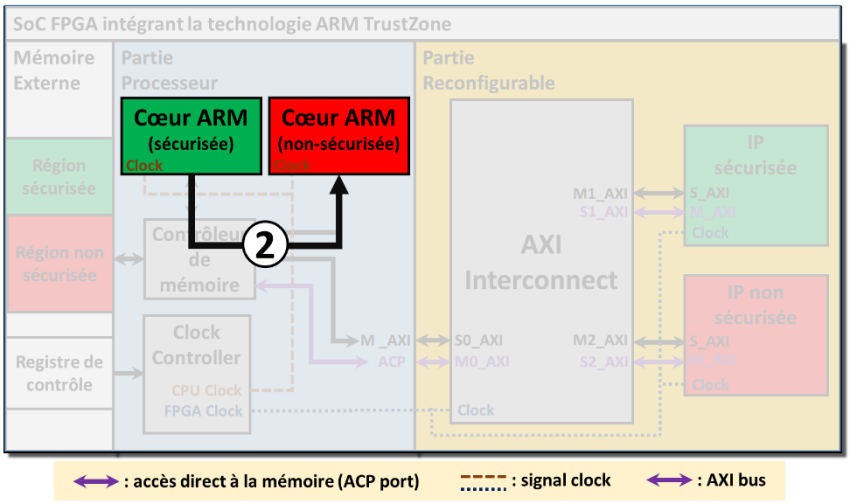


Figure 33: Transfert des données sensibles du cœur ARM sécurisé vers le cœur ARM non-sécurisé dans SoC FPGA

La preuve de concept #2 démontre qu'une horloge partagée entre deux éléments d'un SoC peut être utilisée comme support de communication pour une attaque par canal dérobé. Pour cela, la preuve de concept #2 utilise un processus espion inclus dans le pilote du système DFS du système d'exploitation de confiance et un processus récepteur inclus dans le système d'exploitation du monde non-sécurisé. Les deux processus interagissent directement (écriture/lecture) avec les registres de configuration liés à la PLL (la fréquence), source de l'horloge qui alimente les deux processeurs.

Dans cette preuve de concept, le processus récepteur n'a pas accès aux régions sécurisées de la mémoire externe et le processus espion (monde sécurisé) n'a aucune information sur l'adressage du système d'exploitation du monde sécurisé pour choisir une adresse et communiquer directement avec le processus récepteur.

Le processus espion utilise l'algorithme 2 pour faire de la modulation de fréquence. Pour envoyer un 1 logique, l'algorithme change la fréquence actuelle (*freq_actuelle*) de la fréquence *freq_1* à *freq_2*. Pour l'envoi d'un 0 logique, l'algorithme fait l'inverse, il change la fréquence actuelle de *freq_2* à *freq_1*. A chaque changement de fréquence l'algorithme conserve les deux fréquences pendant une période de temps, pour que le processus récepteur détecte le changement de fréquence et l'interprète. Après l'envoie de toute la trame de donnée, l'algorithme 2 remets la fréquence normale de fonctionnement du système (*freq_actuelle = freq_normal*) comme dans l'algorithme 1.

Algorithme 2: Modulation de fréquence

```
Input: donnée_à_transférer
    For i = donnée_à_transférer_size To 0 Do
        If (donnée_à_transférer[i] = 1) Then
          freq_actuelle = écriture_reg(freq_1);
          loop for Tempo_1;
          freq_actuelle = écriture_reg(freq_2);
          loop for Tempo_1;
        Else
          freq_actuelle = écriture_reg(freq_2);
          loop for Tempo_1;
          freq_actuelle = écriture_reg(freq_1);
          loop for Tempo_1;
        End If;
        freq_actuelle = écriture_reg(freq_3);
    End For;
    freq_actuelle =  freq_normal;
```

La méthode de modulation de fréquence utilisée dans la section précédente fonctionne également. Mais si la trame de données à transférer est trop longue, la méthode a un taux d'erreur trop élevé et il est difficile de synchroniser les deux cœurs ARM. La méthode de l'algorithme 2 permet d'atteindre une bande passante de 6.104bps et un taux d'erreur nul.

Algorithme 3: Décodage des données

Input: *donnée_reçues_size*
Output: *donnée_dérobée*
For i = *donnée_reçues_size* **To** 0 **Do**
loop for *Tempo_échantillonnage*
last_freq = *new_freq*;
new_freq = lecture(*freq_actuelle*);
If (*last_freq* = *freq_1* and *new_freq* = *freq_2*) **Then**
donnée_dérobées[i]= '1'
End If
If (*last_freq* = *freq_2* and *new_freq* = *freq_1*) **Then**
donnée_dérobées[i]= '0'
End If
End For
Return *donnée_volées*

Le processus récepteur utilise l'algorithme 3 pour décoder les données reçues. L'algorithme utilise une temporisation d'échantillonnage (*Tempo_échantillonnage*) au bout de laquelle il enregistre l'ancienne valeur de fréquence lue dans une variable et effectue une nouvelle lecture du registre de configuration de la PLL. Ensuite, il compare l'ancienne valeur de la fréquence avec la nouvelle. S'il détecte un changement de la fréquence *freq_1* à *freq_2*, il enregistre un 1 logique dans le tableau. Et s'il détecte un changement de la fréquence *freq_2* à *freq_1*, il enregistre un 0 logique dans le tableau. Le choix de la durée d'échantillonnage (*Tempo_échantillonnage*) est crucial pour ne pas louper aucun bit, elle doit être inférieure à la temporisation (*Tempo_1*) utilisée par le processus espion.

3.3. Preuve de concept #3 : Transfert des données sensibles du cœur ARM sécurisé vers l'IP non-sécurisée

La preuve de concept #3 consiste en un transfert de données sensibles du cœur ARM sécurisé vers l'IP non-sécurisée (chemin d'attaque n° 3, figure 35).

Comme les deux premières preuves de concept, un code malveillant inséré dans le pilote du système DFS du monde sécurisé est utilisé comme processus espion. Par contre, dans cette troisième preuve de concept, le processus espion contrôle l'une des quatre d'horloges disponibles dans la partie reconfigurable du SoC Xilinx Zynq-7000. Les quatre horloges sont limitées à 250MHz et elles ont une fonctionnalité qui leurs permet d'être actives seulement pour un nombre de cycles prédéfini dans leurs configurations. Cette fonctionnalité est utilisée pour réduire la

consommation d'énergie dans la partie reconfigurable. Le processus espion utilise cette fonctionnalité pour transférer les données sensibles.

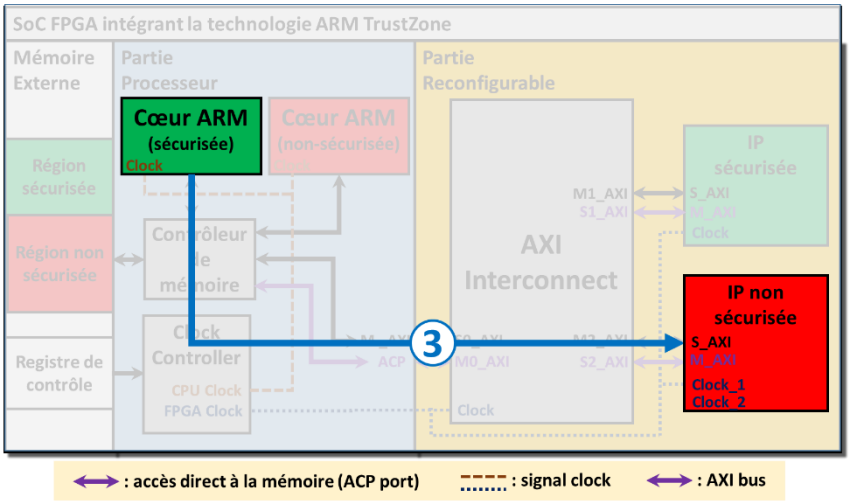


Figure 34: Transfert des données sensibles du cœur ARM sécurisé vers l'IP non-sécurisés dans un SoC FPGA

Dans cette preuve de concept, le processus récepteur n'a aucune information sur l'adressage de système de confiance et le processus espion (monde sécurisé) n'a pas accès aux registres utilisées par l'IP non-sécurisée. L'accès aux registres est protégé par le système de confiance en utilisant la MMU virtuelle du monde sécurisé.

Pour envoyer un 1 logique, le processus espion active l'horloge (horloge_1) connectée à l'IP non-sécurisée pendant 3 cycles. Pour envoyer un 0 logique, il active l'horloge_1 pour 2 cycles. Entre l'envoie de deux bits successifs, l'horloge_1 est désactivée pour une période de temps. Il n'y a aucune contrainte sur le choix du nombre de cycles d'horloge pour l'envoi d'un 1 et un 0 logiques, mais plus le nombre de cycles est petit plus la taille de la bande passante du canal dérobé est grande. Pour aider le processus récepteur à décoder les données transférées dans le canal dérobé, le processus espion contrôle une deuxième source d'horloge (horloge_2) qui oscille en continue. Le processus espion force horloge_2 à la même fréquence que horloge_1 durant le transfert secret des données et la remet à sa fréquence normale après l'envoie du dernier bit fuité.

L'IP non-sécurisée (processus récepteur) est connectée aux deux horloges (horloge_1 et horloge_2) manipulées par le processus espion pour les utiliser dans le décodage des données reçues. Le processus récepteur utilise un compteur synchronisé sur l'horloge_2 qui compte le nombre de front montant dans le signal de l'horloge_1. Le compteur est initialisé après chaque

désactivation de l'horloge_1. La figure 36 montre un exemple de décodage d'une trame de données fuitées. Sur cette figure, la présence de deux fronts successifs est traduite par un 0 logique et la présence de trois fronts successifs par un 1 logique. Le signal de l'horloge_1 (signal violet figure 36) est capturé à l'aide du débogueur des signaux numériques de l'outil Vivado.

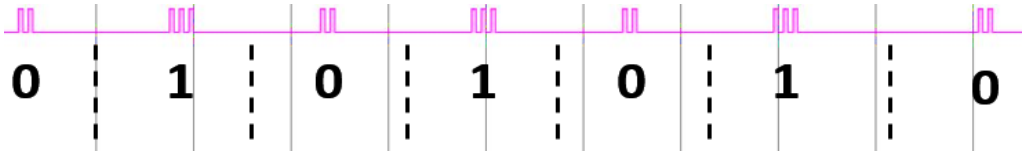


Figure 35: Décodage des données reçues

Le tableau 1 répertorie trois configurations de test et leurs bandes passantes. La taille de la bande passante est liée à la fréquence et aux nombres de cycles d'activation de l'horloge_1. Les résultats du tableau 1 ne prennent pas en considération le temps de désactivation de l'horloge_1 entre deux bits successifs.

Tableau 1 bande passante selon la fréquence et les cycles d'activation de l'horloge

La fréquence (MHz)	N ° de cycles pour 1 logique	N ° de cycles pour 0 logique	Bande passante (Mbps)
250	10	5	50
250	3	2	125
100	10	5	20

3.4. Preuve de concept #4 : Transfert de données sensibles de l'IP sécurisée vers le cœur ARM non-sécurisé

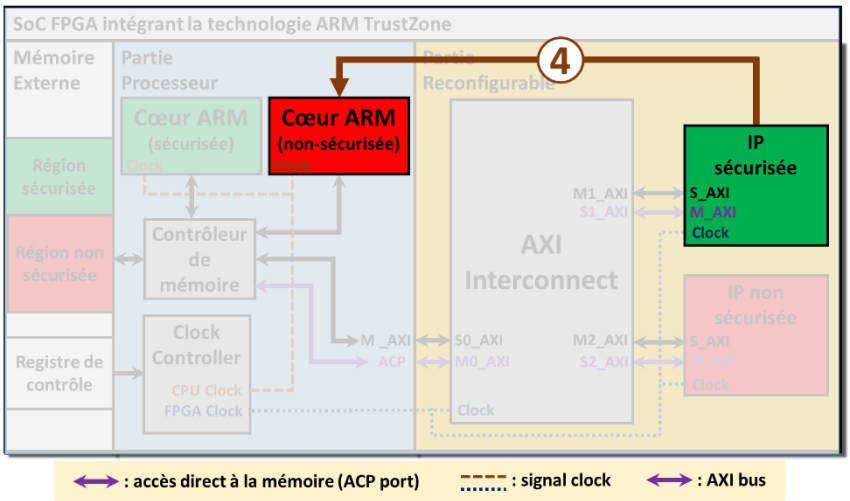


Figure 36: Transfert des données sensibles de l'IP sécurisée vers le cœur ARM non-sécurisé dans un SoC FPGA

La preuve de concept #4 consiste en un transfert de données sensibles depuis l'IP sécurisée vers le cœur ARM non-sécurisée (chemin d'attaque n° 4, figure 37). Cette preuve de concept se base sur l'utilisation d'un cheval de Troie comme dans l'attaque DMA présentée dans la section chap3.2. Analogiquement à l'attaque DMA, le cheval de Troie exploite l'accès direct à la mémoire pour compromettre la sécurité du système. Dans cette preuve de concept, le cheval de Troie est utilisé comme processus espion, il est inséré dans l'IP sécurisée et cible le registre de configuration lié à l'horloge qui cadence les opérations du cœur ARM non-sécurisé.

Dans cette preuve de concept, le processus espion n'a aucune information sur l'adressage de système d'exploitation du monde non-sécurisé et le processus non-sécurisé n'est pas autorisé à communiquer avec les IP sécurisées de la partie reconfigurable.

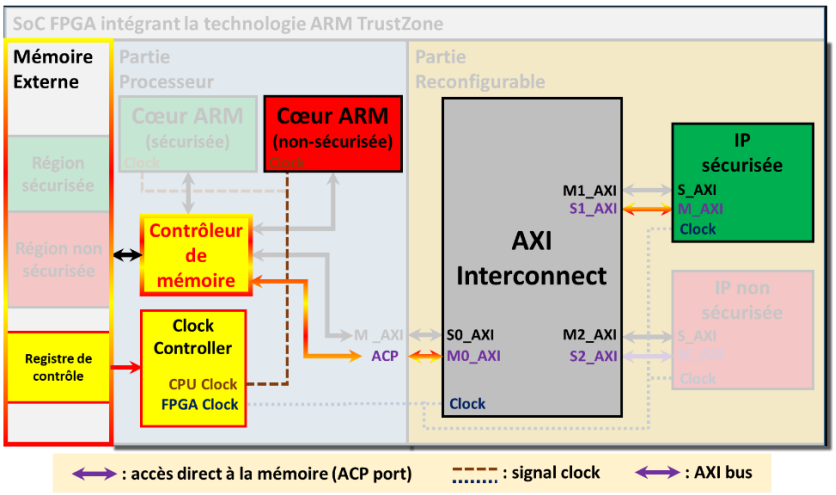


Figure 37: Manipulation de l'horloge par accès direct à la mémoire externe via l'interface ACP

Le processus espion a accès aux registres de configuration car l'IP sécurisée utilise l'interface ACP qui permet un accès direct aux registres depuis la partie reconfigurable comme illustré dans la figure 38. Le processus espion cible le registre de configuration de l'horloge qui est bien détaillé dans la documentation du SoC Xilinx Zynq-7000, ce registre n'est pas bien protégé par le système et il est accessible pour les deux processus, espion et récepteur. Dans cette preuve de concept, pour envoyer un 1 logique, le processus espion applique une *valeur_1* au registre de configuration de l'horloge. Pour l'envoie d'un 0 logique, le processus espion applique une *valeur_2* différente de la première valeur. La *valeur_1 et* la *valeur_2* sont équivalent à la fréquence *freq_1* et *freq_2*

respectivement. Le processus récepteur effectue une simple lecture du registre de configuration de l'horloge pour décoder les données dérobées.

4. Conclusion :

Ce chapitre démontre que la modulation de fréquence est un canal dérobé puissant pour transmettre discrètement des données sensibles entre deux processus qui ne sont pas autorisés à communiquer. Le chapitre démontre aussi l'importance de protéger les systèmes informatiques contre les attaques DMA. Les quatre preuves de concept présentées dans ce chapitre représentent un nouveau canal dérobé efficace basé sur l'utilisation des registres de configuration. Le chapitre illustre l'importance de protéger les registres de configuration contre les utilisations malicieuses.

Chapitre 4 : Exploitation malicieuse du mécanisme de cohérence de la mémoire cache dans un SoC complexe hétérogène

Ce chapitre présente l'exploitation malicieuse du mécanisme de cohérence de la mémoire cache d'un SoC complexe hétérogène depuis la partie reconfigurable. Le chapitre débute par une présentation des signaux de l'interface ACP qui contrôlent la cohérence d'une requête à la mémoire cache et les signaux de protection liés à la technologie ARM TrustZone. Ensuite, le chapitre présente une méthode pour mesurer le temps d'accès à une donnée cohérente depuis la partie reconfigurable du SoC et une méthode pour expulser une ligne de la mémoire cache depuis cette même partie reconfigurable. La maîtrise de ces deux méthodes est la condition sine qua non pour la mise en place d'une attaque par analyse temporelle de la mémoire cache.

La fin du chapitre présente trois scénario d'attaque exploitant malicieusement le mécanisme de cohérence de la mémoire cache depuis la partie reconfigurable d'un SoC complexe hétérogène : deux attaques par analyse temporelle de la mémoire cache ciblant l'exécution d'un algorithme de chiffrement symétrique AES exécuté dans le monde sécurisé par la partie processeur (attaques du type Flush+Reload et du type Evict+Time) et une attaque par transmission d'informations sensibles via un canal dérobé.

1. La cohérence de la mémoire cache depuis la partie reconfigurable dans un SoC complexe hétérogène

Dans un SoC complexe hétérogène, l'unité SCU maintient la cohérence de la mémoire cache entre les différents cœurs du processeur (comme cela est détaillé dans la section chap1-1.2.5). L'unité SCU arbitre les requêtes vers la mémoire cache L2 des cœurs du processeur qui sont issues des interfaces maîtres des accélérateurs matériels connectés à l'interface ACP (Accelerator Coherency Port). Une interface maître connectée à l'interface ACP peut lire la mémoire cohérente directement à partir des mémoires caches L1 et L2, mais ne peut pas écrire directement dans le cache L1. Les scénarios possibles de lecture et d'écriture d'une interface maître ACP sont les suivants :

- Requête de lecture avec des données cohérentes dans la mémoire cache L1 : l'interface maître ACP obtient les données directement de la mémoire cache L1.
- Requête de lecture avec des données cohérentes dans la mémoire cache L2 : la demande est mise en file d'attente dans le SCU et la requête est transmise à la mémoire cache L2.
- Requête de lecture avec des données cohérentes qui n'existent pas dans les mémoires cache L1 et L2 : selon l'état précédant des données, les mémoires cache L1 ou L2 peuvent lancer une requête pour obtenir les données demandées à la mémoire RAM ou bien à la mémoire principale avec la mise en attente de la requête initiale jusqu'à la disponibilité des données.
- Requête d'écriture avec des données cohérentes dans le cache L1: dans un premier temps les données au niveau de la mémoire cache L1 sont marquées comme non valides et les lignes de cache sont expulsées vers la mémoire cache L2. Puis, dans un second temps, la requête d'écriture est planifiée dans le SCU et les données sont écrites dans la mémoire cache L2. Enfin, lorsque le processeur accède à la même adresse mémoire que celle visée par la requête, un échec se produit dans la mémoire cache L1.
- Requête d'écriture avec des données cohérentes dans la mémoire cache L2 : la requête est planifiée dans le SCU puis les données sont écrites dans la mémoire cache L2.

- Requête d'écriture avec des données cohérentes qui ne sont pas dans les mémoires cache L1 et L2: la requête est destinée directement au contrôleur de la mémoire externe.

Ces scénarios sont contrôlés par les signaux AxCache[3:0] et AxUser[4:0] de l'interface ACP qui sont détaillés dans la section suivante.

1.1. Les signaux AxCache[3:0] et AxUser[4:0]

Les signaux AxCache[3:0] et AxUser[4:0] (pour chacun des signaux x = R pour lecture ou x = W pour écriture) de l'interface ACP permettent de contrôler la cohérence d'une requête vers les mémoires cache depuis la partie reconfigurable. Le signal AxUser[4:0] est composé du bit AxUser[0] qui est un bit de partage et des bits AxUser[4:1] qui contrôlent la stratégie d'écriture (Write-back [37], Write-through, … etc) adoptée par la requête. Les bits AxUser[4:0] ne sont pas interprétés par l'unité SCU et ils sont transmis directement au contrôleur de la mémoire cache L2 qui les utilise en cas d'une mémoire cache configurée en mode exclusif. Le signal AxCache[3:0] contrôle aussi la stratégie d'écriture adoptée par la requête. Il est composé des bits suivants :

- Le bit AxCache[0] qui contrôle la mise en mémoire tampon (Bufferable, B),
- Le bit AxCache[1] qui contrôle la mise en mémoire cache (Cacheable, C),
- Le bit AxCache[2] qui contrôle l'allocation sur lecture (Read-Allocate, RA),
- Le bit AxCache[3] qui contrôle l'allocation sur écriture (Write-Allocate, WA).

Selon les recommandations de ARM [68], le bit AxUser[0] et le bit AxCache[1] doivent être valides pour une requête ACP cohérente.

1.2. Le signal AxPort[2:0]

Le signal AxPROT[2:0] (pour lequel x = R pour lecture et x = W pour écriture) est un signal d'autorisation d'accès qui permet de protéger les interfaces esclaves des IP sécurisées contre les requêtes malveillantes. Il est composé des bits suivants :

- Le bit AxPROT[0] qui contrôle le niveau du privilège d'une requête,
- Le bit AxPROT[1] qui contrôle l'état de sécurité de la requête ACP,
- Le bit AxPROT[2] qui contrôle l'accès aux données ou aux instructions.

Les interfaces maîtres connectées à l'interface ACP contrôlent le signal AxPROT[2:0], en particulier le bit AxPROT[1] qui transmet l'état de sécurité de la requête à l'interface esclave. Dans un système intégrant la technologie ARM TrustZone, chaque interface maître impose la valeur du bit AxPROT[1], cela peut mettre en danger le système si l'interface maître est exploitée malicieusement par un cheval de Troie. De plus, ce scénario est particulièrement crédible lorsque les interfaces esclave de la partie processeur ne sont pas configurée pour interdire l'accès aux régions sécurisées de la mémoire principale depuis la partie reconfigurable ce qui est souvent le cas. La suite de ce chapitre présente différents types d'attaque exploitant la cohérence de la mémoire cache.

2. Les éléments de l'attaque

La section suivante présente les éléments utilisés pour la mise en place des trois attaques présentées à la fin de ce chapitre. Cette section présente tout d'abord les méthodes utilisées pour distinguer l'échec d'un succès d'une requête à la mémoire cache depuis la partie reconfigurable ou depuis la partie processeur. La section présente également les méthodes utilisées pour expulser une ligne de la mémoire cache ciblée par l'attaque. La maîtrise de ces méthodes est indispensable à la réalisation des attaques.

2.1. Distinguer un échec d'un succès de la mémoire cache

La première condition pour réussir une attaque visant la mémoire cache consiste à distinguer l'échec d'un succès d'une requête adressée à cette mémoire cache. Cette section présente des méthodes à utiliser pour distinguer l'échec d'un succès depuis la partie reconfigurable ou bien depuis la partie processeur.

2.1.1. Depuis la partie reconfigurable

La méthode présentée dans cette partie se base sur la mesure du temps d'accès à une donnée depuis la partie reconfigurable pour distinguer l'échec d'un succès d'une requête à la mémoire cache. Cette méthode utilise les signaux du bus AXI (détaillés dans la section chap1-1.1) connectant l'interface maître à l'interface ACP pour mesurer le temps d'accès. Mais avant d'utiliser cette méthode, il faut s'assurer que l'interface maître est connectée à l'interface ACP comme cela est indiqué dans la section chap3-1, que les bit AxUser[0] et AxCache[1] soient valides, et que l'unité SCU et le *symmetric multiprocessor system* (SMP) soient activés. La suite de cette section présente les canaux du bus AXI qui fuitent des informations sur la présence ou pas d'une donnée dans la mémoire cache et la méthode utilisée pour mesurer le temps d'accès.

Durant une requête d'écriture cohérente depuis la partie reconfigurable, l'interface maître envoie l'adresse dans le canal d'adresse d'écriture du bus AXI suivie de la donnée à écrire dans le canal d'écriture de données. Une fois la donnée reçue par l'interface ACP, cette dernière envoie une réponse à la requête à l'interface maître en utilisant le canal d'écriture de réponse. En mesurant le nombre de cycles d'horloge écoulés entre le handshake du canal d'adresse d'écriture et le canal d'écriture de réponse durant une requête d'écriture, on constate que le nombre de cycles d'horloge ne varie pas par rapport à la présence de la donnée dans la mémoire cache L1 ou L2. Donc, il n'est pas possible de distinguer un échec d'un succès lors d'une requête d'écriture.

Pour une requête de lecture cohérente depuis la partie reconfigurable, l'interface maître envoie tout d'abord l'adresse à lire dans le canal d'adresse de lecture du bus AXI. Ensuite, si la communication est établie avec l'interface ACP, cette dernière envoie la donnée à l'interface maître en utilisant le canal de lecture de donnée. En mesurant le nombre de cycles d'horloge écoulés entre le handshake du canal d'adresse de lecture et celui du canal de lecture de donnée, le nombre de cycles d'horloge varie selon l'existence ou pas de la donnée dans la mémoire cache L1 ou L2. Il est court, si la donnée réside dans la mémoire cache et il est long dans le cas contraire. Pour distinguer les succès d'un échec d'une requête à la mémoire cache depuis la partie reconfigurable, il suffit donc de compter le nombre de cycles d'horloge entre les deux handshake des deux canaux du bus AXI utilisés dans une requête de lecture.

Dans la suite de ce chapitre, l'algorithme 4 est utilisé pour valider les méthodes de mesure de temps et d'expulsion de la ligne de cache introduites dans ce chapitre. L'algorithme effectue en boucle, un accès à l'adresse donnée en entrée, une expulsion de la ligne de cache contenant cette adresse une itération sur deux, et une mesure de temps d'accès à cette adresse. L'algorithme 4 provoque un échec de la mémoire cache toutes les deux itérations en expulsant la ligne de cache.

Algorithme 4: Vérification de la méthode utilisée pour distinguer un échec d'un succès

```
Input: adresse
    for i = nombre_dacces to 0 do
        Accès(adresse)
        if !(i%2) then
          Expulser_ligne_cache (adresse)
        end if
        Measure_clock_cycles (Access(adresse))
    end for
```

La méthode utilisée dans la fonction ***Expulser_ligne_cache*** est détaillée dans la section chap4-2.2.1 et la méthode utilisée dans la fonction ***Measure_clock_cycles*** est celle basée sur la mesure de nombre de cycles d'horloge entre le handshake du canal d'adresse de lecture et le canal de lecture de donnée.

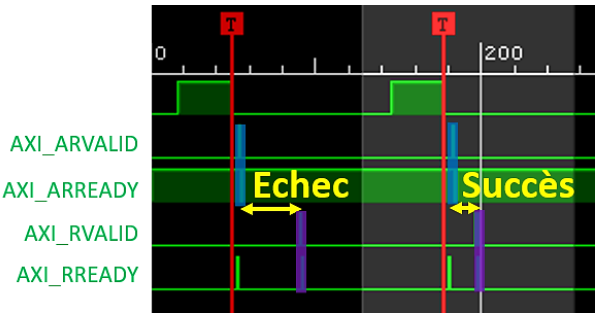


Figure 38: Un échec et un succès d'une requête de lecture cohérente

La figure 39 présente la capture des signaux du handshake (les signaux _valid et _ready) du canal d'adresse de lecture et du canal de lecture de donnée durant deux itérations successives de l'algorithme 4. La figure montre que le temps écoulé entre le handshake du canal de lecture d'adresse (AXI_ARVALID == '1' && AXI_ARREADY == '0', ligne verticale bleue figure 39) et celui du canal de lecture de donnée (AXI_RVALID == '1' && AXI_RREADY == '1', la ligne verticale violette figure 39) est plus long pour un échec que pour un succès. La figure 40 valide la méthode introduite dans cette section, les échecs et les succès de la mémoire cache sont clairement distinguables depuis la partie reconfigurable.

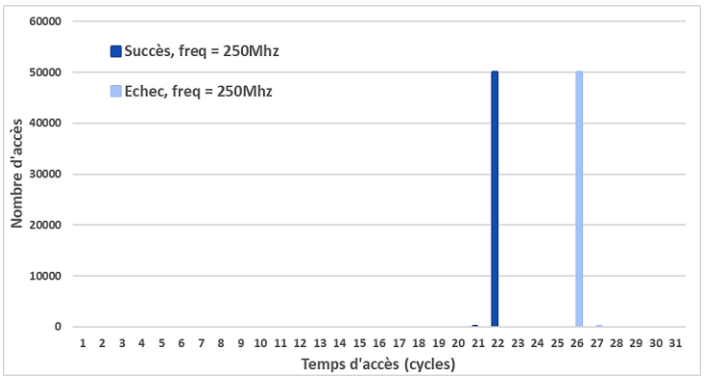


Figure 39: Histogramme des échecs et des succès d'une requête de lecture cohérente (Fréquence FPGA = 250Mhz)

Pour utiliser cette méthode dans d'autres systèmes, il faut réaliser une pré-étape de profilage (utiliser simplement l'algorithme 4) pour déterminer un seuil qui permet de différencier les échecs des succès. Comme nous l'avons vu dans le chapitre précédent, dans les SoCs modernes

il est possible de modifier la fréquence en cours d'exécution pour adapter dynamiquement la consommation de puissance du système. La fréquence d'horloge peut donc évoluer au cours du temps. Il faut donc réaliser le profilage à chaque changement de fréquence. La figure 41 montre que pour le SoC Xilinx Zynq-7000, chaque fréquence définie un seuil différent.

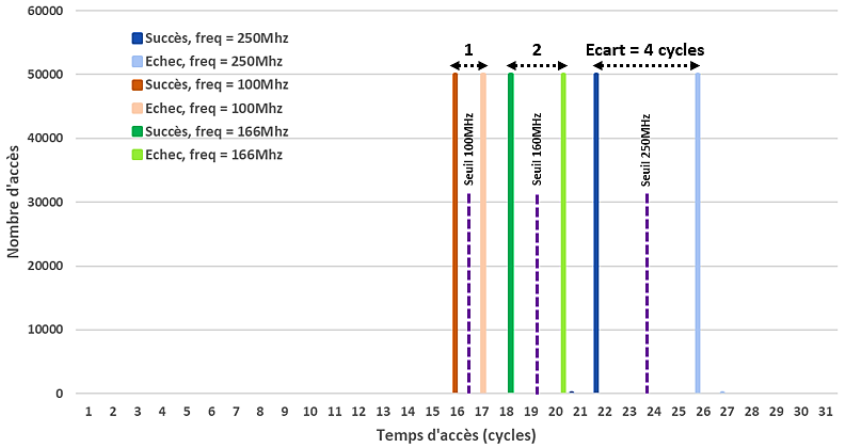


Figure 40: L'effet de la fréquence sur le seuil

La figure 41 montre que l'écart entre le nombre de cycles d'horloge pour les échecs et les succès diminue avec la fréquence d'horloge de la partie reconfigurable. Dans le SoC Xilinx Zynq-7000, l'écart passe de 4 cycles pour une fréquence = 250Mhz à un cycle pour une fréquence = 100Mhz. Cet écart est nul pour toutes les fréquences en dessous de 55Mhz, donc pour le SoC Xilinx Zynq-7000 cette méthode est limitée aux fréquences supérieures à 55Mhz. Pour l'utiliser dans un scénario d'attaque il faut prendre en considération cette limite.

La suite du chapitre démontre qu'il est possible de réaliser une attaque par analyse de la mémoire cache avec cette méthode de mesure du temps d'accès même avec un écart d'un cycle d'horloge (pour une fréquence d'horloge comprise entre 55 Mhz et 125 Mhz) entre les échecs et les succès d'une requête adressée à la mémoire cache par la partie reconfigurable.

2.1.2. Depuis la partie processeur

Pout distinguer l'échec d'un succès d'une requête à la mémoire cache depuis la partie processeur, l'utilisateur peut utiliser une des deux méthodes suivantes :

- L'utilisation du PMU (Performance Monitor Unit) qui offre 6 compteurs configurables pour compter le nombre des évènements, comme les échecs de la mémoire cache et le nombre d'accès à cette mémoire. Le PMU est accessible depuis l'espace utilisateur seulement si la configuration du système l'autorise, ce qui n'est pas le cas dans la

majorité des systèmes d'exploitation. Pour l'utiliser dans une attaque, il suffit d'initialiser les compteurs avant le lancement du programme victime et lire la valeur des compteurs une fois le programme victime terminé. Ensuite, il reste à analyser ces valeurs selon le type d'attaque mise en place.

- Mesurer le temps d'accès à une donnée dans le système mémoire. Pour mesurer le temps lorsque le système d'exploitation Linux est mise en œuvre dans le SoC, l'attaquant peut utiliser soit la fonction ***POSIX clock_gettime()***, soit l'appel système ***perf_event_open*** si le système autorise son utilisation vu qu'elle fait appelle au PMU, soit l'utilisation d'un thread qui incrémente une variable globale en boucle [58]. Cette méthode elle aussi demande une pré-étape de profilage pour chaque système pour déterminer un seuil qui permet de différencier les échecs des succès de la mémoire cache.

Comme dans la méthode précédente, pour chaque système et à chaque changement de fréquence, il faut réaliser une pré-étape de profilage (utilisation de l'algorithme 4) pour déterminer le seuil qui permet de distinguer l'échec d'un succès de la mémoire cache. Cependant, ce n'est pas que la fréquence et le changement de système qui affectent le choix du seuil. Ce seuil varie également en changeant la politique de remplacement d'une ligne de la mémoire cache. La figure 42 montre que dans le SoC Xilinx Zynq-7000, la politique de remplacement round-robin ajoute de la latence à l'accès aux données par rapport à une politique de remplacement aléatoire.

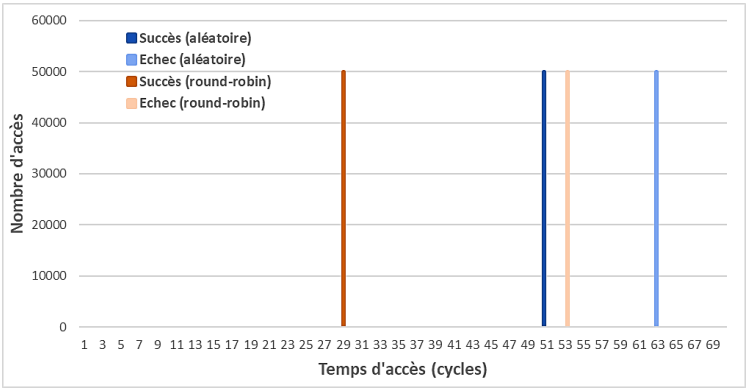


Figure 41: Histogramme des échecs et des succès avec deux politiques de remplacement différentes (aléatoire, round-robin)

2.2. Opération de maintenance de la mémoire cache

La deuxième condition pour réussir une attaque visant la mémoire cache consiste à utiliser une opération de maintenance privilégiée ou non-privilégiée pour expulser une ligne de la

mémoire cache ou pour invalider la totalité des lignes de cache. Cette section présente des méthodes à utiliser pour expulser une ligne de cache depuis la partie reconfigurable et dans la partie processeur.

2.2.1. Expulser une ligne de cache depuis la partie reconfigurable

Comme présenté dans les scénarios possibles de lecture et d'écriture d'une interface maître ACP dans la section chap4-1, une requête d'écriture cohérente force la mémoire cache L1 à invalider la ligne de cache contenant l'adresse de la requête, si la donnée cohérente réside dans la mémoire cache L1. Donc, il suffit d'envoyer une requête d'écriture cohérente pour expulser une adresse de la mémoire cache. Cependant, il faut ajouter à la requête une modification du signal WSTRB[4:0] de l'interface ACP qui contrôle le nombre d'octets valides dans le signal WDATA[31:0] et qui doivent être mis à jour dans la mémoire. Selon [72], il faut que les bits du signal soient à un état logique bas pour que les octets du signal WDATA[31:0] soient invalides. Avec cette modification, la requête d'écriture est valide, mais la valeur de l'adresse n'est pas mise à jour.

Cependant, cette technique n'arrive pas à expulser les lignes de cache si la région contenant l'adresse ciblée a une stratégie d'écriture autre que Write-back. Pour vérifier la possibilité d'utiliser cette technique d'expulsion dans une attaque de cache contre une région de mémoire, il faut utiliser l'algorithme 4 en remplaçant la fonction ***Expulser_ligne_cache*** par une requête d'écriture cohérente avec WSTRB[4:0] = b''0000''.

2.2.2. Expulser une ligne de cache depuis la partie processeur

Les processeurs ARM ont des instructions de maintenance de la mémoire cache accessibles depuis le coprocesseur CP15, mais ces instructions ne sont pas accessibles depuis l'espace utilisateur. Pour cette raison, l'attaquant peut utiliser l'une des stratégies d'expulsion présentes dans la littérature, comme celle présentée dans [58]. Dans les SoCs avec une architecture de cache simple, comme le SoC Xilinx Zynq-7000, il suffit d'accéder à un nombre N d'adresses congruentes à l'adresse à expulser. Dans un cache *n-way associative*, les adresses congruentes sont situées à une adresse à une distance proportionnelle à la **''taille du cache/nombre de *way*''** de l'adresse à expulser.

Le choix du nombre N dépend de la politique de remplacement de ligne de cache. Si la mémoire utilise une politique de remplacement round-robin, un nombre N égale au nombre de *way* est suffisant pour expulser une adresse car un cache *n-way associative* ne peut pas contenir

plus que N adresses congruent à la fois. Si la mémoire utilise une politique de remplacement aléatoire, le nombre N doit être supérieur au nombre de *way* pour parvenir à expulser l'adresse cible.

2.2.3. Invalider la totalité de la mémoire cache dans la partie processeur

Pour invalider simultanément la totalité des lignes de la mémoire cache dans la partie processeur, l'attaquant peut effectuer un changement de contexte (commutation entre deux processus du système d'exploitation). En effet, lors d'un changement de contexte, certains systèmes d'exploitation et mécanismes matériels vident automatiquement la mémoire cache pour éviter les fuites de données entre les processus.

3. Scénario d'attaque

Cette section présente trois scénarios d'attaques complètes (Flush+Reload, Evict+Time et une attaque par canal dérobé) exploitant la cohérence de cache. Les attaques sont implémentées en utilisant le SoC Xilinx Zynq-7000 qui inclut un processeur dual-cœur ARM Cortex-A9, une mémoire cache L1 4-*way set-associative* pour les instructions, et une pour les données avec une taille de 32Ko chacune. Le SoC possède également une mémoire cache L2 8-*way set-associative* de taille 512Ko et d'une ligne de cache de longueur de 32 octets. La mémoire cache L2 constitue le dernier niveau de mémoire cache LLC (Last Level Cache) pour le SoC Xilinx Zynq-7000, elle est partagée entre les deux cœurs ARM Cortex-A9 et les interfaces maîtres de la partie reconfigurable connectées à l'interface ACP.

Le design expérimental de la section chap3-1 est utilisé pour l'implémentation des trois attaques. La fréquence de la partie processeur est de 650Mhz et celle de la partie reconfigurable est de 250Mhz. Le signal AxProt[1] de l'interface ACP est fixé à un état logique bas pour permettre aux interfaces maîtres d'accéder aux régions sécurisées de la mémoire principale. Ils peuvent ainsi avoir accès aux données sensibles du système. La partie processeur autorise l'accès aux régions sécurisées de la mémoire principale depuis la partie reconfigurable.

Remarque : l'ensemble des attaques présentées dans cette section sont réalisées à l'aide d'une application bare-metal. Donc, les résultats présentés ne contiennent pas beaucoup de bruit contrairement à ce que l'on pourrait obtenir avec un système d'exploitation mis en œuvre dans le SoC.

3.1. Attaque par canal auxiliaire

Cette section présente deux attaques par analyse temporelle de la mémoire cache, l'attaques Flush+Reload et l'attaque Evict+Time, qui cible un algorithme de chiffrement symétrique. La section présente l'algorithme de chiffrement utilisé dans les attaques, ainsi que le modèle de menaces et les deux attaques.

3.1.1. AES-128 T-table

L'implémentation AES-128 T-table est une implémentation optimisée en termes de performances du standard AES-128 (Advanced Encryption Standard) [73]. Cette implémentation combine les trois fonctions d'un tour de l'AES (SubBytes, ShiftRows et MixColumns) en une seule étape en utilisant quatre tables de recherches précalculées T_0, T_1, T_2 et T_3 de 1ko (256 éléments de 32bits) pour les 9 premiers tours de l'algorithme. Le dernier tour utilise une table de recherches précalculées T_4 supplémentaire car ce tour n'utilise pas l'opération MixColumns. L'AES-128 T-table prend en entrée un texte clair de 16 octets $p = (p_0, p_1, \dots, p_{15})$, et une clé de 16 octets $k = (k_0, k_1, \dots, k_{15})$. L'équation (3) présente la structure du tour i ($1 \leq i \leq 9$) qui prend en entrée l'état intermédiaire $S_i = (s_{i,0}, s_{i,1}, \dots, s_{i,15})$ et la clé du tour $k_i = (k_{i,0}, k_{i,1}, \dots, k_{i,15})$. Le tour donne en sortie l'état intermédiaire S_{i+1}. L'état intermédiaire S_1 prend en entrée $p \oplus k = (p_0 \oplus k_0, p_1 \oplus k_1, \dots, p_{15} \oplus k_{15})$ et la clé secrète k.

$$
\begin{aligned}
S_{i+1} = \{ & T_0[s_{i,0}] \oplus T_1[s_{i,5}] \oplus T_2[s_{i,10}] \oplus T_3[s_{i,15}] \oplus \{k_{i,0}, k_{i,1}, k_{i,2}, k_{i,3}\}, \\
& T_0[s_{i,4}] \oplus T_1[s_{i,9}] \oplus T_2[s_{i,14}] \oplus T_3[s_{i,3}] \oplus \{k_{i,4}, k_{i,5}, k_{i,6}, k_{i,7}\}, \\
& T_0[s_{i,8}] \oplus T_1[s_{i,13}] \oplus T_2[s_{i,2}] \oplus T_3[s_{i,7}] \oplus \{k_{i,8}, k_{i,9}, k_{i,10}, k_{i,11}\}, \\
& T_0[s_{i,12}] \oplus T_1[s_{i,1}] \oplus T_2[s_{i,6}] \oplus T_3[s_{i,11}] \oplus \{k_{i,12}, k_{i,13}, k_{i,14}, k_{i,15}\}\}
\end{aligned}
\tag{3}
$$

L'AES-128 T-table est la cible de plusieurs attaques de cache [57], [74], [75]. Ces attaques exploitent le fait que l'état intermédiaire S_1 dépend directement du texte clair p et de la clé k. Si l'attaquant connait le texte claire p_i et les éléments (adresses) des tables de recherches (T_0, T_1, T_2 et T_3) utilisés durant l'opération, il peut déduire facilement l'octet k_i de la clé. Cette section utilise ce scénario d'attaque de cache pour démontrer la faisabilité d'une attaque par analyse de la mémoire cache depuis la partie reconfigurable du SoC.

3.1.2. Modèle de menaces

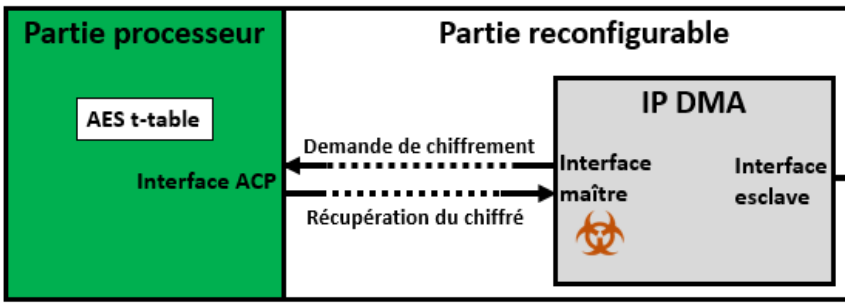


Figure 42: Schéma de l'attaque par canal auxiliaire

La figure 43 présente le modèle de menaces utilisé pour l'attaque Flush+Reload et l'attaque Evict+Time. Dans la partie reconfigurable du SoC Xilinx Zynq-7000, une IP DMA récupère des données à chiffrer d'un périphérique d'E/S et les transmet à la partie processeur pour les chiffrer à l'aide d'une implémentation de l'AES-128 T-table vulnérable aux attaques par analyse temporelle de la mémoire cache. Par la suite, l'IP DMA récupère les données chiffrées à la sortie de l'algorithme de chiffrement pour le transmettre au périphérique d'E/S. L'IP DMA n'a aucune information sur la clé secrète utilisée dans la partie processeur. L'interface maître de l'IP DMA inclus un cheval de Troie matériel qui exploite la cohérence de cache pour récupérer la clé de chiffrement puis la transmettre à l'extérieur du SoC. Le cheval de Troie a la possibilité d'initialiser des chiffrements depuis la partie reconfigurable en envoyant un texte clair.

Parmi les challenges rencontrés pour mettre en œuvre une attaque par l'analyse de la mémoire cache depuis la partie reconfigurable du SoC, il y a la recherche de l'emplacement des quatre tables de recherches dans la mémoire principale et l'alignement des tables $T_0, T_1, T_2\ et\ T_3$dans la mémoire. Dans les attaques présentées dans cette section les adresses des tables sont à fournir à l'interface maître au début de l'attaque. Mais si les adresses ne sont pas fournies, l'interface maître doit parcourir la totalité de la mémoire principale à la recherche des 4 premiers éléments de chaque table pour les localiser avant de lancer l'attaque.

3.1.3. Attaque Flush+Reload

L'attaque Flush+Reload présentée vise le premier tour de l'AES-128 T-table exécuté dans le monde sécurisé de la partie processeur. L'interface maître utilise l'algorithme 4 pour définir le seuil qui permet de distinguer les échecs des succès des requêtes à la mémoire cache avant chaque attaque.

Depuis la partie reconfigurable, l'interface maître déclenche l'exécution dans la partie processeur du chiffrement de 256 textes clairs aléatoires avec l'octet p_i fixe Avant chaque chiffrement, l'interface maître expulse la ligne de cache contenant un élément parmi les 256 éléments de la table T_0 si cet élément est dans la mémoire cache. Il mesure alors le temps d'accès à cet élément après la réception du texte chiffré. A l'aide du seuil et du temps d'accès aux éléments de la table T_0, l'interface maître peut en déduire quels sont les éléments de la table T_0 utilisés par le chiffrement et donc l'octet k_i de la clé [57].

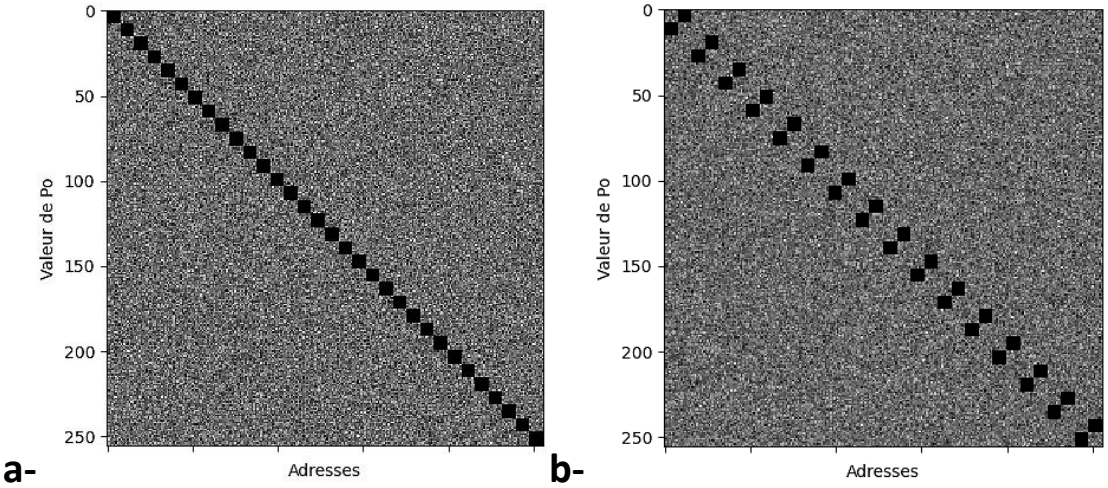


Figure 43: Attaque Flush+Reload exploitant la cohérence de cache,
a- fréquence = 250Mhz et $k_0 = 0x00$, b- fréquence = 100Mhz et $k_0 = 0x0f$

La figure 44-a présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x00$ et une fréquence de fonctionnement de 250Mhz dans la partie reconfigurable. Cette matrice est obtenue en ciblant les adresses de la table T_0 avec l'attaque Flush+Reload pour les 256 valeurs de p_i. La matrice de la figure 44-a présente une diagonale car l'état intermédiaire S_1 de l'algorithme de chiffrement accède à la valeur $T_0[p_i = 0x00 \oplus p_\mathrm{i}]$. Dans le cas d'une taille de ligne de cache égale à 32 octets, la matrice révèle que les 5 bits supérieurs de l'octet k_0 [57] et cela réduit l'espace de recherche de la clé à 48 bits. Pour récupérer la totalité de la clé, il faut suivre l'une des méthodes présentées dans [71], [75]–[77].

La figure 44-b présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x0f$ avec une fréquence d'horloge de 100Mhz pour la partie reconfigurable du SoC. Cette matrice nous montre qu'il est possible de réaliser l'attaque de la mémoire cache à l'aide de la méthode présentée dans la section Chap4-2.1.1, ceci même avec une différence temporelle de seulement un cycle d'horloge entre un échec et un succès de la mémoire cache.

3.1.4. Evict+Time

L'attaque Evict+Time que nous avons réalisée vise aussi le premier tour de l'AES-128 T-table exécuté dans le monde sécurisé de la partie processeur. Les étapes principales de l'attaque Evict+Time réalisée sont :

Etape 1 : Depuis la partie reconfigurable, l'interface maître déclenche l'exécution dans la partie processeur du chiffrement d'un texte clair. Cette première opération charge dans la mémoire cache les éléments nécessaires pour la réalisée notamment les T-tables.

Etape 2 : Toujours depuis la partie reconfigurable, l'interface maître expulse la ligne de cache contenant un élément de la table T_0 si celui-ci réside dans la mémoire cache.

Etape 3 : L'interface maître déclenche à nouveau l'exécution dans la partie processeur du chiffrement du même texte clair. Maintenant, cette opération charge la mémoire cache avec seulement les éléments manquants pour la réaliser. La table T_0 qui a été retirée dans l'étape précédente sera donc chargée uniquement si elle est utilisée.

Etape 4 : L'interface maître mesure alors le temps écoulé entre le déclenchement du chiffrement et la réception du texte chiffré. Si le temps de chiffrement est supérieur à un certain seuil, l'interface maître peut en déduire que l'élément de la table T_0 a été utilisé puisque l'allongement du temps tient au chargement de la table dans la T_0 mémoire cache et ainsi trouver l'octet k_i de la clé.

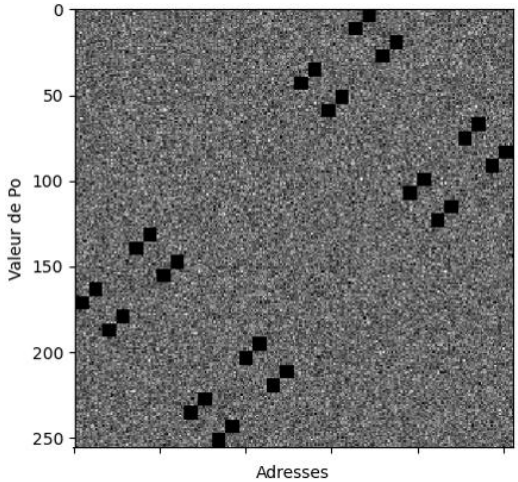


Figure 44: Attaque Evict+Time depuis la partie reconfigurable $k_0 = 0x51$

La figure 45 présente la matrice de paterne d'accès aux adresses de la table T_0 pour $k_0 = 0x51$ avec une fréquence de fonctionnement de 250Mhz pour la partie reconfigurable du SoC.

Cette matrice est obtenue en utilisant l'attaque Evict+Time. Elle valide la possibilité de cette attaque depuis la partie reconfigurable du SoC.

3.2. Attaque par canal dérobé

Dans cette attaque, le processus espion est une application logicielle malicieuse exécutée dans le monde sécurisé de la partie processeur du SoC. Le processus récepteur est un cheval de Troie matériel inséré dans la partie reconfigurable du SoC au niveau de l'interface maître de l'IP DMA. Le processus espion et le processus récepteur n'ont normalement pas le droit de communiquer directement. Cependant, ils utilisent tous les deux une adresse mémoire située dans une région sécurisée de la mémoire principale pour communiquer. Cette adresse est accessible uniquement en lecture par le processus récepteur et elle est accessible à la fois en lecture et en écriture par le processus espion.

Le processus espion transmet de l'information en chargeant ou en expulsant l'adresse partagée par les deux processus de la mémoire cache. Ainsi il peut coder l'information au travers de cette action. L'algorithme 5 présente l'encodage des données par le processus espion. Pour transmettre un 1 logique, le processus espion expulse l'adresse de la mémoire cache suivi d'une temporisation *Tempo_1*. Pour transmettre un 0 logique, le processus espion expulse l'adresse de la mémoire cache suivi d'une temporisation *Tempo_2*. La temporisation *Tempo_1* est plus lente que la *Tempo_2.* Entre deux bits successifs envoyés, le processus espion charge la donnée dans la mémoire cache suivi d'une temporisation *Tempo_3*. Le choix des temporisations *Tempo_1*, *Tempo_2 et Tempo_3* joue sur la bande passante et le taux d'erreur. La méthode d'encodage présentée dans cette section permet de réduire le taux d'erreur, même si la bande passante est très faible (1Mbs).

Le processus récepteur effectue une opération de lecture en continue à l'adresse partagée par les deux processus pour détecter les échecs et les succès de sa requête à la mémoire cache et les décoder pour récupérer l'information transmise par le processus espion. Le processus récepteur utilise l'algorithme 6 pour décoder les informations dérobées. A la détection d'un échec, le processus récepteur enregistre la valeur 1 dans le tableau de données. S'il détecte un succès, il enregistre la valeur 0. La figure 46 présente un exemple de décodage du message ''hi'', les symboles « larges » dans la figure sont décodés par un 1 logique et les symboles « étroits » sont décodés par un 0 logique.

Algorithme 5: Encodage des données dérobées par le processus espion

```
Input: adresse, donnée_à_transférer
  For i = donnée_à_transférer_size To 0 Do
    If (donnée_à_transférer[i] = 1) Then
      Expulser_ligne_cache (adresse)
      loop for Tempo_1
    Else
      Expulser_ligne_cache (adresse)
      loop for Tempo_2
    End If
    Accès(adresse)
    loop for Tempo_3
  End For
```

Algorithme 6 : Décodage des données dérobées par le processus récepteurs

```
Input: adresse
Output: donnée
  While(1)
    Temps = Measure_clock_cycles (Accès(adresse))
    i++
    Exploitation des données dérobées
    If Temps > seuil Then
        donnée[i] = 1
    Else
        donnée[i] = 0
    End If
```

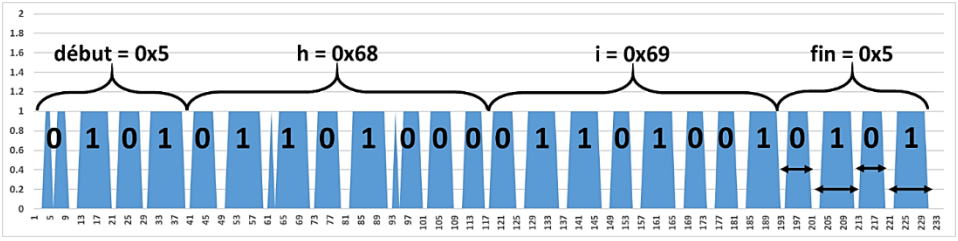


Figure 45: le décodage de l'information dérobée

L'attaque présentée montre la possibilité d'une attaque par canal dérobé par exploitation malicieuse de la cohérence de cache entre une application logicielle malicieuse exécutée dans le monde sécurisé de la partie processeur du SoC et une IP matérielle corrompue (par un cheval de Troie) de la partie reconfigurable du SoC. Pour transmettre une information de taille importante, un protocole simple est utilisé : l'envoie de quatre bits spécifiques pour indiquer le début de la transmission et de quatre bits spécifiques pour indiquer la fin de la transmission. Ce protocole

peut être développé par exemple, par l'ajout d'information telle que la taille de la trame échangée par exemple.

Enfin, la figure 47 montre l'effet de la politique de remplacement de la ligne de cache sur la bande passante du canal dérobé. La politique de remplacement round-robin prend plus de temps pour remplacer une ligne de la mémoire cache ce qui augmente le temps de transmission d'une trame de données et diminue ainsi la bande passante.

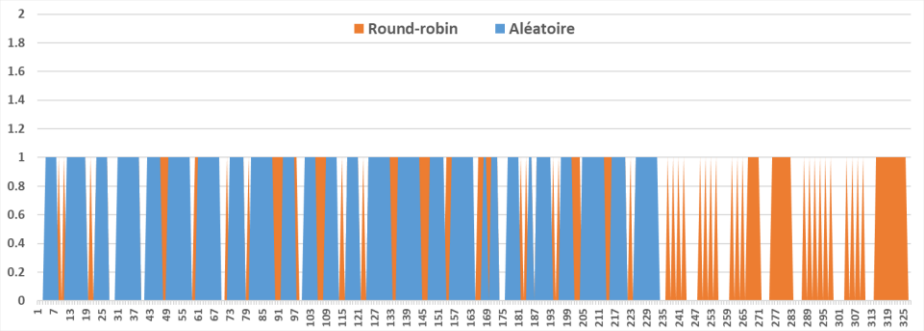


Figure 46: L'impact de la politique de remplacement sur la bande passante

4. Conclusion :

Ce chapitre a apporté la preuve que le mécanisme de cohérence de la mémoire cache dans les SoC complexes hétérogènes constitue une grande menace pour un système sécurisé. Il a notamment montré que les attaques classiques par analyse temporelle de la mémoire cache (telles que les attaques Flush+Reload et Evict+Time) étaient possibles à réaliser depuis la partie reconfigurable. Ces attaques pouvant, comme cela a était montré dans ce chapitre, cibler l'exécution d'un algorithme de chiffrement symétrique dans le monde sécurisé de la partie processeur du SoC. De plus, ce chapitre a montré que le mécanisme de cohérence de cache pouvait être exploité malicieusement pour mettre en place un canal dérobé dans le SoC entre la partie processeur et la partie reconfigurable et échanger ainsi secrètement et en dehors de toutes les règles de sécurité des informations sensibles entre des processus évoluant dans des mondes différents.

Chapitre 5 : Contre-mesures

Les chapitres précédents ont présenté des chemins d'attaques efficaces qui exploitent différents vecteurs d'attaque tels qu'un cheval de Troie matériel, une IP tiers malveillante, un outil CAO corrompu ou un concepteur malveillant. Ces attaques permettent de réaliser une élévation de privilège, un déni de service, un vol de clés cryptographiques ou de données confidentielles.

Ce chapitre présente des solutions pour améliorer la sécurité des SoC FPGA embarquant la technologie ARM TrustZone en se protégeant des attaques présentées précédemment. Il présente une méthodologie d'isolation des IP sécurisées, mais aussi un système de vérification des règles de conception et une étude de l'utilisation d'un chiffrement de flux pour sécuriser les transactions sur le bus AXI.

1. Protection #1 : Isolation des IP sécurisées de la partie reconfigurable du SoC

Cette section présente une méthodologie d'isolation matérielle et logicielle des IP sécurisées de la partie reconfigurable d'un SoC complexe hétérogène embarquant la technologie ARM TrustZone. Cette méthodologie est composée de trois étapes et utilise une technique d'isolation matérielle et des configurations logicielles pour protéger les IP sécurisées et les régions sécurisées de la mémoire externe. La figure 48 donne un aperçu de l'isolation des IP sécurisées (encadrées en violet) dans la partie reconfigurable du SoC.

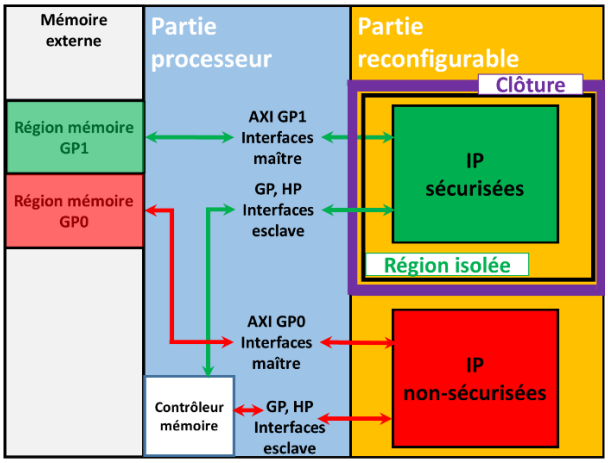


Figure 47: L'isolation des IP sécurisées dans la partie reconfigurable du SoC

Dans cette méthodologie, les IP sécurisées à isoler doivent être des IP de confiance, c'est-à-dire que l'intégrateur doit avoir l'assurance qu'elles ne contiennent pas de vulnérabilité. Des certifications (à l'image de celles proposées pour les Trust Applications que l'on peut exécuter dans une TEE) pourraient être envisagées dans ce but. Cependant, aujourd'hui il n'existe pas de certifications pour cela et c'est avant tout un lien de confiance entre l'intégrateur et le concepteur de l'IP qui doit permettre de réduire les risques (c'est le cas si par exemple l'intégrateur et le concepteur de l'IP font partie de la même société).

Dans le cas d'un SoC du type Xilinx Zynq, la méthode que nous proposons pour l'isolation des IP sécurisées de la partie reconfigurable est constituée des trois étapes suivantes :

Etape 1 : **Isolation physique et logique des IP sécurisées dans la partie reconfigurable du SoC.** Dans cette étape, le concepteur doit utiliser la méthodologie d'isolation du flux de conception (Isolation Design Flow [78], IDF) de Xilinx qui permet à des IP indépendantes de fonctionner sur un seul FPGA avec une séparation logique et physique. L'IDF implémente et route

les IP dans des zones isolées et entourées d'une « clôture » composée d'un ensemble d'éléments logiques configurables non configurés (sans fonction logique) dans lesquels aucun routage n'est présent (pas de connexions internes ni externes, pas de ligne de routage traversant les éléments logiques configurables en dehors des connexions aux entrées-sorties de l'IP isolée). L'IDF utilise également un outil indépendant de vérification des règles de conception (*Design Rule Check*, DRC en anglais), pour vérifier la clôture et les connexions avec la zone isolée.

Etape 2 : Utilisation de deux interfaces maîtres de la partie processeur du SoC. Comme cela est représenté au centre de la figure 48, dans la partie processeur du SoC, deux interfaces vont contrôler les interfaces des IP sécurisées (connexions vertes sur la figure 48) et pour contrôler celles des IP non-sécurisées (connexions rouges sur la figure 48). Dans le cas du SoC Xilinx Zynq-7000, la partie processeur possède deux interfaces maîtres à usage général : l'interface AXI GP0 et l'interface AXI GP1. Ces deux interfaces peuvent être utilisées pour contrôler les interfaces esclaves des IP sécurisées et non sécurisées comme cela est présenté sur la figure 48. Sur cette figure, l'interface AXI GP1 est utilisé pour contrôler les IP sécurisées et l'interface AXI GP0 est utilisé pour contrôler les IP non sécurisées. Chacun de ces deux interfaces a une région de la mémoire externe qui lui est dédiée (la région mémoire GP1 pour l'interface AXI GP1 et la région mémoire GP0 pour l'interface AXI GP0 comme cela apparait sur la figure 48). De plus, le SoC Xilinx Zynq-7000 propose des registres de configuration liés à chacune des deux interfaces maîtres AXI GP1 et AXI GP0. Ces registres configurent la propagation de l'état de sécurité des requêtes de la partie processeur vers la partie reconfigurable du SoC. Par exemple, dans la figure 48, comme il contrôle les IP non sécurisées, l'interface AXI GP0 est configuré pour interdire la propagation de l'état de sécurité des requêtes de la partie processeur vers les IP sécurisées. Toute requête non-sécurisée destinée aux IP sécurisées (accès à la région mémoire GP1 (en vert dans la figure 48) de la mémoire externe dédiée à l'interface AXI GP1) est rejetée au niveau de l'interface AXI GP0, et une exception est alors déclenchée dans la partie processeur du SoC.

Etape 3 : Limitation des accès aux régions sécurisées de la mémoire externe. Pour cette étape, le concepteur doit interdire aux interfaces maîtres des IP non-sécurisées d'accéder aux régions sécurisées de la mémoire externe. Dans le cas du SoC Xilinx Zynq-7000, la partie processeur possède deux interfaces esclaves AXI GP (interface esclave GP0 et interface esclave GP1), quatre interfaces haute performance (HP) et une interface ACP pour la cohérence de la mémoire cache (dont nous avons montré l'exploitation malicieuse dans le chapitre précédant).

Ces interfaces permettent théoriquement aux IP de la partie reconfigurable du SoC un accès direct à l'intégralité de la mémoire externe. Cependant, il est possible de limiter cet accès à des régions spécifiques. Pour cela, les interfaces esclaves AXI GP et HP disponibles doivent être scindées en deux groupes distinct pour être connectés aux interfaces maîtres des IP sécurisées (connexions vertes sur la figure 48) et non-sécurisées (connexions rouges sur la figure 48) de la partie reconfigurable du SoC. Pour limiter la possibilité de réaliser les attaques ciblant la mémoire cache (voir chapitre 4), l'utilisation de l'interface ACP doit être strictement limitée aux seules IP sécurisées de la partie reconfigurable du SoC. Pour le SoC Xilinx Zynq-7000 il existe des registres de configuration liés à chacune des interfaces esclaves AXI GP et HP. Ces registres contrôlent les accès aux régions sécurisées de la mémoire externe depuis la partie reconfigurable du SoC. Par exemple, dans le cas de la figure 48, les registres de contrôle des interfaces esclaves AXI GP et HP connectées aux IP non-sécurisées (connexions rouges dans la figure 48) doivent être configurés pour interdire aux interfaces maîtres des IP non-sécurisées d'accéder à la région mémoire sécurisée (région mémoire GP1 en vert dans la figure 48) de la mémoire externe. Cette configuration réduit les menaces que représente des attaques par accès directes DMA depuis la partie reconfigurable que nous avons présenté dans le chapitre 3 de ce manuscrit.

Cette méthodologie en trois étapes permet aux concepteurs de renforcer la sécurité d'un SoC hétérogène complexe contre la majorité des attaques présentées dans ce manuscrit. Cependant, cela n'est pas suffisant, et des protections supplémentaires doivent être utilisées pour protéger le système contre les chevaux de Troie matériels insérés dans la partie reconfigurable du SoC. La suite de ce chapitre présente un système de vérification des règles de conception qui permet de vérifier les connexions entre les différents blocs du design.

2. Protection #2 : Vérification des règles de conception

Comme nous l'avons montré, la plupart des attaques présentées dans le deuxième chapitre de ce manuscrit peuvent être mise en place en exploitant un outil de CAO corrompu par un logiciel malveillant ou directement par un concepteur malveillant. Pour protéger la conception d'un SoC contre de telles menaces, l'outil de CAO et son utilisation nécessitent une protection logicielle dédiée. Le partage des fichiers entre développeurs doit être sécurisé à l'aide de fonctions cryptographiques (chiffrement, signature, etc.) et le concepteur doit obligatoirement vérifier les règles de conception. Cette vérification assure que le système conçu est conforme aux

spécifications fonctionnelles déclarées en amont de la conception. Elle peut aussi limiter l'insertion de chevaux de Troie matériel visant la AXI corruption des signaux du bus AXI.

La non-corruption des connexions entre les interfaces du système font partie des règles que doit vérifier le concepteur. Celui-ci doit s'assurer que tous les signaux des canaux AXI ne sont pas corrompus. Pour cela, nous proposons un programme python qui vérifie la connexion des signaux de sécurité entre les interfaces. Le code source est disponible en ligne [79]. Malgré le fait qu'il s'agit d'un code dédié à l'outil de CAO Vivado, ce code python peut facilement être adapté à d'autres outils de CAO.

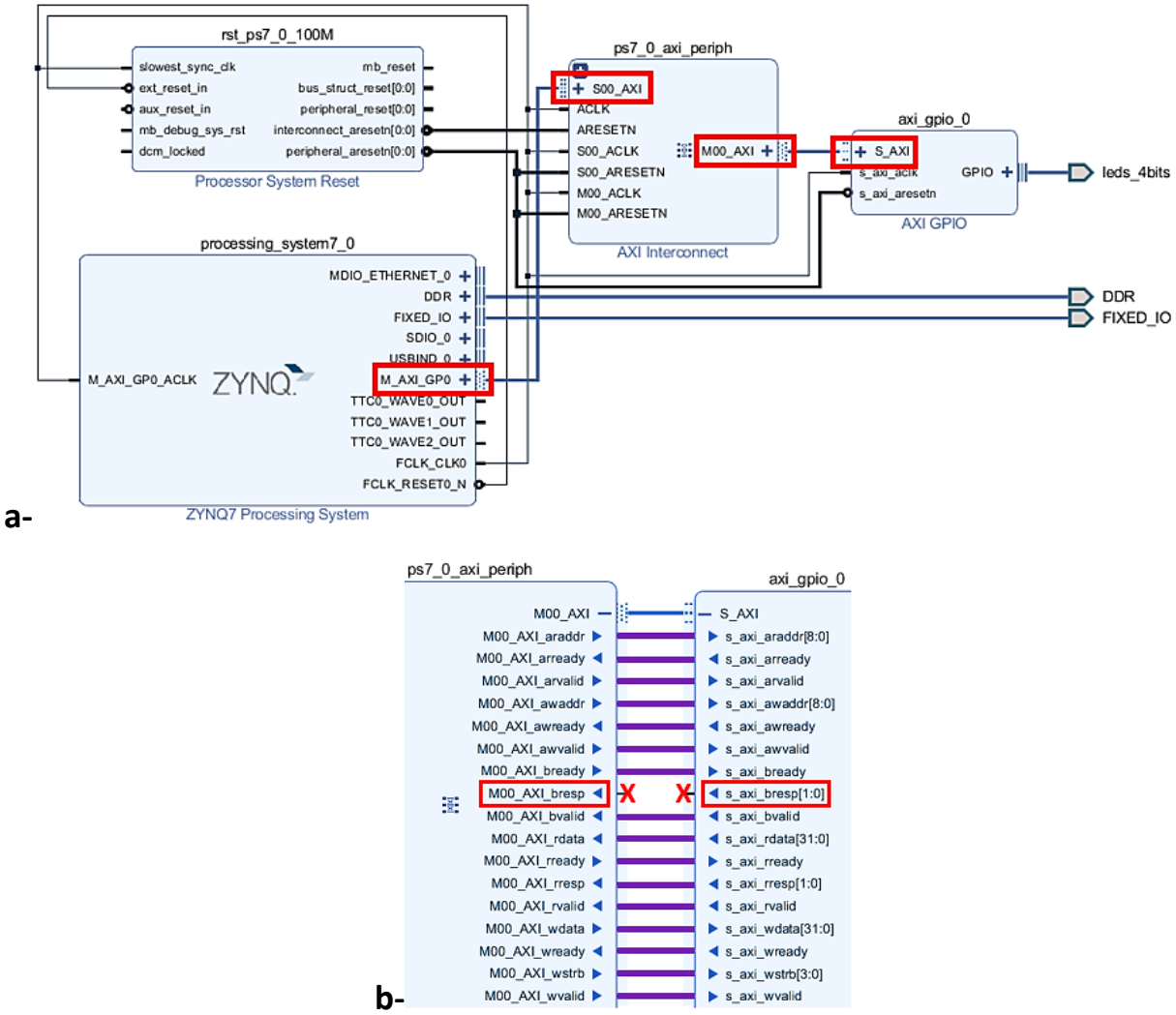


Figure 48: Vérification des règles de conception, a- design à vérifier , b- connexion corrompu entre l'interface M00_AXI de l'IP ps7_0_axi_periph et l'interfaces S_AXI de l'IP axi_gpio_0

Le programme python prend en entrée le code source VHDL du design et fournit en sortie la liste des connexions corrompues s'ils en existent. La liste contient les noms des signaux

corrompus avec le nom des deux interfaces affectées. Par exemple pour vérifier le design de la figure 49-a, Le programme python commence par recenser les différentes IP (entités) du design en utilisant le code source du top level puis il répertorie les interfaces AXI (rectangles rouge, figure 49-a), maîtres et esclaves, avec les signaux connectés à ses interfaces pour chaque IP. Ensuite, le programme prend deux par deux les interfaces connectées et vérifie les signaux qui les connectent. Dans notre exemple, la figure 49-b montre que la connexion entre le signal AXI_bresp (rectangle rouge, figure 49-b) de l'interface M00_AXI de l'IP ps7_0_axi_periph et celui de l'interfaces S_AXI de l'IP axi_gpio_0 est corrompu. Dans ce cas, le programme python renvoi le nom du signal corrompu et le nom des deux interfaces AXI affectées, ps7_0_axi_periph/M00_AXI et axi_gpio_0/S_AXI.

Cette solution doit être systématiquement ajoutée à tous les outils de CAO pour assurer l'intégralité du code source RTL et éviter toute modification malveillante du code source. Mais elle ne permet pas de régler tous les problèmes de sécurité et des protections complémentaires doivent être utilisées comme le chiffrement du bus AXI tel que la section suivant le présente.

3. Protection #3 : Chiffrement des transactions du bus AXI

Cette section présente une étude de faisabilité sur l'utilisation d'un système de chiffrement des données pour sécuriser les transactions sur le bus AXI et pour protéger ainsi l'échange de données entre deux entités dans un SoC hétérogène complexe. Dans cette section, l'algorithme de chiffrement de flux (appelé aussi chiffrement par flot) Trivium [80] est utilisé pour sécuriser les transactions entre une IP sécurisée de la partie reconfigurable du SoC et une ou des applications logicielles sécurisées exécutées dans la partie processeur du SoC. Les performances de l'implémentation logicielle et matérielle sur le SoC Xilinx Zynq-7000 seront étudiées.

Remarque : Cette section ne traite pas de la problématique de la gestion des clés cryptographiques et de l'échange de ces clés entre les parties processeur et reconfigurable du SoC.

3.1. Architecture logicielle et matérielle du système de chiffrement du bus AXI

Cette section utilise le design présenté dans la section chap3-1. Les différentes ressources du système hétérogène, notamment la partie processeur, la partie reconfigurable et la mémoire externe, sont partitionnées en ressources sécurisées et ressources non-sécurisées. Les IP

sécurisées de la partie reconfigurable sont implantées en utilisant la méthodologie en trois étapes présentée le chapitre 5 de ce manuscrit.

Pour chiffrer les transactions sur le bus AXI entre les parties processeur et reconfigurable du SoC hétérogène complexe, une application sécurisée intégrant un algorithme de chiffrement de flux est implémentée dans la partie processeur pour effectuer les opérations cryptographiques. Cette application est utilisée chaque fois qu'une transaction est destinée à une IP matérielle sécurisée. La clé cryptographique utilisée par l'application est enregistrée dans une région sécurisée de la mémoire externe. Dans la partie reconfigurable, une crypto-IP matérielle réalisant le même algorithme de chiffrement par flux est implantée dans la partie reconfigurable du SoC entre l'AXI Interconnect et l'IP sécurisée. Cette crypto-IP matérielle est composée de quatre blocs comme cela est présenté sur la figure 50. La crypto-IP matérielle dispose d'une interface maître pour communiquer avec l'IP sécurisée, d'une interface esclave pour échanger avec l'AXI Interconnect, d'un contrôleur pour le traitement des données et la planification des opérations cryptographiques, et d'un algorithme léger de chiffrement de flux pour chiffrer et déchiffrer les transactions.

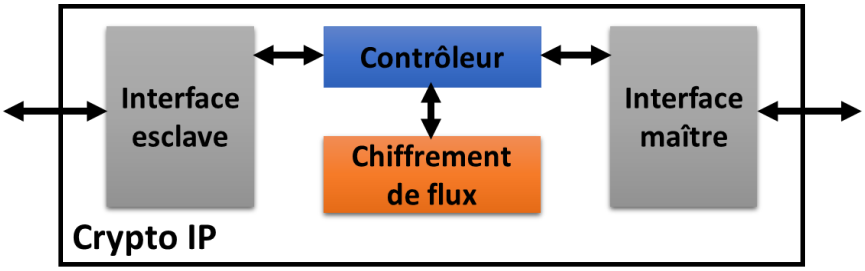


Figure 49: Architecture de la crypto IP matérielle implantée dans la partie reconfigurable du SoC

3.2. Le chiffrement léger de flux (Trivium)

Un chiffrement de flux, ou chiffrement en continue, est un algorithme de chiffrement à clé symétrique qui prend en entrée une clé et un vecteur d'initialisation, et fournit en sortie une chaîne de bits pseudo aléatoires (flux). Cette chaîne est ajoutée à un texte clair en utilisant l'opération logique ou-exclusif (xor) pour donner le texte chiffré. L'algorithme de chiffrement est généralement composé de registres à décalage et de fonctions de rétroaction linéaires et non-linéaires. Le processus de déchiffrement est analogue au chiffrement, il utilise la même clé et le même vecteur d'initialisation pour générer un flux pseudo-aléatoire identique à celui utilisé pour le chiffrement afin de récupérer le message original.

Trivium [80] est un algorithme de chiffrement de flux synchrone qui utilise une clé de 80 bits et un vecteur d'initialisation de 80 bits aussi. Son état interne est de 288 bits divisée sur trois registres à décalage de tailles différentes. Pour l'initialisation de Trivium, les trois registres à décalage sont initialisés à l'aide de la clé, le vecteur d'initialisation et des vecteurs de motif fixe. Ensuite, l'état interne est mis à jour 1152 fois (après 1152 périodes de l'horloge) de sorte que chacun de ses bits dépend de chaque bit de la clé et du vecteur d'initialisation d'une manière complexe non-linéaire.

Pour le chiffrement du bus AXI, nous utilisons Trivium car il est conçu pour fournir un compromis souple entre la vitesse et le nombre de portes logiques nécessaires à sa réalisation matérielle [80]. Un deuxième avantage est que la taille de sortie de Trivium peut être adapter à la taille du bus AXI (32 bits et 64 bits) utilisée dans le SoC Xilinx Zynq-7000, Trivium propose une taille de sortie allant de 1 à 64 bits. Dans ce chapitre, nous étudions aussi les implémentations de Trivium avec une taille de sortie de 8 bits et 16 bits pour présenter les performances de l'utilisation du chiffrement du bus de communication dans le cadre des microcontrôleurs de 8 bits et de 16 bits.

La suite de ce chapitre présente les performances des implémentations logicielles et matérielles de Trivium, une avec une taille de sortie de 8 bits, une avec 16 bits, une avec 32 bits et une avec 64bits.

3.3. Performances des implémentations logicielles et matérielles de Trivium

Cette section présente les performances des implémentations logicielles et matérielles de Trivium utilisant le SoC Xilinx Zynq-7000. Pour les expérimentations réalisées, la partie processeur du SoC Xilinx Zynq-7000 fonctionne à une fréquence de 650Mhz et la partie reconfigurable à une fréquence de 100 Mhz.

3.3.1. Performances des implémentations logicielles de Trivium

Dans cette partie, le code source de Trivium utilisé est basé sur l'implémentation logicielle de référence soumises au projet eSTREAM [81]. Elle est modifiée uniquement pour son adaptation au SoC Xilinx Zynq-7000 et aussi pour offrir différentes tailles de sortie : 8 bits, 16 bits et 32 bits, 64 bits. Les différentes implémentations logicielles de l'algorithme Trivium occupent un espace de 12 kilo-octets de la mémoire externe du SoC. La figure 51 donne le temps nécessaire

pour chiffrer 1 Méga-octet de données pour chacune des quatre implémentations logicielles (8 bits, 16 bits, 32 bits et 64 bits).

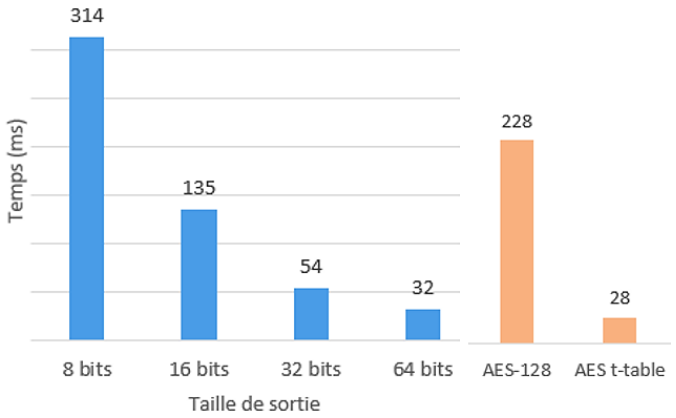


Figure 50: Temps nécessaire pour chiffrer 1 Méga-octets de données avec les quatre implémentations de Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits), l'AES-128 et l'AES t-table exécutées sur la partie processeur du SoC cadencée à 650 MHz.

La figure 51 montre l'effet de la taille de sortie de Trivium sur le temps de génération du flux. Si nous utilisons l'implémentation 8 bits comme référence, la génération du flux est accéléré d'un facteur de 2.3 pour l'implémentation 16 bits, d'un facteur de 5.4 pour l'implémentation 32 bits et d'un facteur de 9.8 pour l'implémentation 64 bits. La figure 51 montre aussi que les implémentations, 16bits, 32bits, et 64 bits de Trivium sont rapides par rapport une implémentation logicielle de l'AES-128 mais lentes par rapport à une implémentation logicielle de l'AES t-table. La partie suivante présente les performances de l'implémentation matérielle de Trivium et les performances de l'implémentation matérielle de l'AES-128.

3.3.2. Performances des implémentations matérielles de Trivium

Dans cette section, les implémentations matérielles de l'algorithme de chiffrement de flux Trivium sont basées sur les implémentations publiées dans [80]. L'algorithme de chiffrement a été implémenté avec différente tailles de sortie (8 bits, 16 bits, et 32 bits, 64 bits). Les quatre implémentations matérielles de l'algorithmes Trivium sont au cœur de la Crypto-IP matérielle présentée à la figure 49. Le surcoût matériel ajouté par les deux interfaces maître et esclave, de la crypto-IP est de 105 LUT. Ce surcoût est fixe car il ne dépend pas de la taille de la sortie de l'algorithme Trivium. Le surcoût matériel ajouté par le contrôleur et par l'algorithme de chiffrement dépend lui de la taille de la sortie du chiffreur de flux.

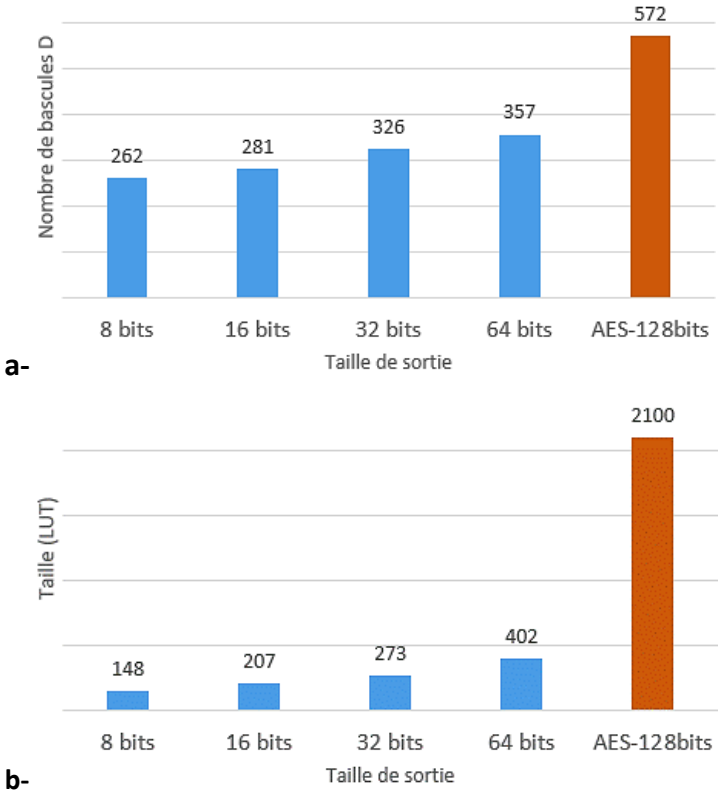


Figure 51: Performance des quatre implémentations matérielles de l'algorithme de chiffrement de flux Trivium (avec la taille de la sortie sur 8 bits, 16 bits, 32 bits et 64 bits) dans la partie reconfigurable du SoC, a- nombre de bascules D nécessaires à l'implémentation matérielle, b- nombre de LUT nécessaires à l'implémentation matérielle

Les figures 52-a et 52-b montrent que le nombre de bascules D et le nombre de LUT occupées par chacune des quatre implémentations matérielles de Trivium est systématiquement inférieure à celui nécessaire à une implémentation AES-128bits. Nous ajoutons aussi que chacune des quatre implémentations de Trivium n'utilisent qu'un très faible nombre des ressources disponibles dans la partie reconfigurable du SoC Xilinx Zynq-7000 qui contient 17 600 LUT et 35 200 bascules D.

3.4. Discussion

Les résultats précédents montrent que le surcoût global apporté par l'utilisation d'un chiffrement de flux pour chiffrer les transactions sur le bus AXI n'est pas très élevé en comparaison de la taille et des performances du SoC Xilinx Zynq-7000. Pour cette étude de faisabilité nous avons choisi d'utiliser l'algorithme de chiffrement par flux Trivium pour le chiffrement du bus AXI, mais le concepteur peut utiliser d'autres algorithmes de chiffrement de flux, comme par exemple Grain-128 [82]. Le concepteur peut aussi limiter l'utilisation du

chiffrement aux applications du monde sécurisé car ces applications n'effectuent pas un grand nombre de transactions et le système exécute les applications dans le monde non-sécurisé la majorité du temps. Le chiffrement des transactions du bus AXI est donc une solution efficace pour sécuriser les échanges entre les applications critiques du système et protéger les données privées.

4. Conclusion

Ce chapitre a présenté une méthodologie d'isolation des IP sécurisées dans un SoC FPGA embarquant la technologie ARM TrustZone. Le chapitre présente aussi un système de vérification des règles de conception notamment au niveau des connexions du bus AXI. Avec l'étude de chiffrement de flux et de leurs implémentation logicielles et matérielles, ce chapitre a montré la faisabilité du chiffrement du bus AXI avec un surcoût limité pour l'implémentation qu'elle soit logicielle ou bien matérielle.

Si nous faisons une synthèse en termes de sécurité apporté par les protections présentées dans ce chapitre nous pouvons dire que :

- Les protections #1 et #2, présentées dans ce chapitre, protègent contre les différentes attaques visant les signaux du bus AXI (attaques présentées dans le chapitre 2).
- Pour assurer l'efficacité des protections #1 et #2, il faut que les IP sécurisées soient de confiance comme cela est indiqué dans la section chap5-1. Effectivement, ces deux protections ne protègent pas contre les attaques basées sur des IP corrompus ou bien malveillants qui incluent des chevaux de Troie.
- La protection #3 protège contre l'attaque par FIFO présentée dans la section chap2-3.3.3, en général cette protection protège contre les attaques du type man-in-the-middle.
- Les protections proposées dans ce chapitre ne protègent pas contre les attaques DMA, les attaques par manipulation malicieuse du système DVFS et les attaques par exploitation malicieuse de la cohérence de cache. Proposer des protections contre ces attaques fait partie des perspectives de cette thèse.

Conclusion et perspectives

Ce manuscrit a présenté une évaluation en profondeur encore jamais réalisée de la sécurité de la technologie ARM TrustZone dans un SoC complexe hétérogène. Il a présenté de nombreuses attaques matérielles qui touchent différent éléments d'un SoC, comme le bus de communication AXI, la hiérarchie mémoire, le système de gestion d'énergie et le système de gestion de la cohérence de la mémoire cache. Différents scénarios d'attaques ont été présenté et certains implique la corruption du SoC lors de sa conception. Les attaques présentées exploitant un cheval de Troie matériel, une IP tiers malicieuse, un script malicieux ou un outil de CAO compromis par un logiciel malicieux ont une très grande portée. Elles doivent être absolument prises en compte par les concepteurs car elles peuvent conduire à une élévation de privilège, un déni de service, ou encore au vol de clés cryptographiques ou de données confidentielles. Les travaux réalisés dans le cadre de cette thèse de doctorat et présentés dans ce manuscrit prouvent que l'extension de la technologie TrustZone de ARM dans un SoC hétérogène complexe n'est pas sécurisée à l'heure actuelle. De plus, l'architecture des SoC hétérogène complexe disponible dans le commerce ne prend pas en compte les problématiques de sécurité et ouvre de nombreuses vulnérabilités alors même que le concepteur ayant mise en œuvre la technologie TrustZone de ARM au sein du SoC se pense protégé par celle-ci. C'est pourquoi, et les travaux développés durant cette thèse le montrent clairement, le déploiement de la technologie TrustZone de ARM dans des SoC hétérogènes complexes doit inévitablement s'accompagner de protections matérielles dédiées et de méthodologies de conception prenant en compte la sécurité du système et des données.

Pour répondre aux problématiques de sécurité du système et des données, nous avons présenté dans ce manuscrit des protections efficaces contre la majorité des attaques que nous avons étudié durant cette thèse. Nous avons proposé une méthodologie d'isolation matérielle renforcée pour sécurité la partie reconfigurable des SoC complexes et hétérogènes embarquant la technologie ARM TrustZone. Nous avons aussi apporté une preuve de faisabilité du chiffrement par flots du bus AXI.

Perspectives :

Les travaux présentées dans ce manuscrit ont introduit plus d'attaques que de contremesures. Pour cette raison, la continuité de ces travaux pourrait s'inscrire principalement dans la recherche de nouvelles contremesures qui puissent permettre de limiter les attaques DMA, les attaques par manipulation malicieuse du système DVFS et les attaques par exploitation malicieuse de la cohérence de cache. Nous pensons qu'il serait préférable de travailler sur le développement d'une nouvelle version de la technologie TrustZone dédiée aux SoC complexes hétérogènes qui intègre des protections matérielles et qui soit efficace à la fois contre les attaques matérielles et à la fois contre les attaques logicielles.

Nos travaux ont également mis en exergue l'augmentation de la complexité des chemins d'attaques (qui mixent l'exploitation de vulnérabilités logicielles et matérielles) conjointement à l'augmentation de la complexité des SoC. Il convient donc de continuer l'étude des chemins complexes d'attaque tout en proposant des solutions de sécurité qui protègent ces système complexes. L'utilisation d'une modélisation plus formelle des chemins d'attaque, par exemple en exploitant des arbres d'attaque et de défense, pourrait être une piste intéressante à explorer. Conjointement, la modélisation et la simulation complète d'une architecture complexe, par exemple avec un outil du type gem5, est aussi une piste prometteuse.

Un dernier aspect de notre travail est celui ayant trait aux vulnérabilités apportées par les outils de CAO pour les SoC hétérogènes et à leur sécurité. Le travail de thèse présenté dans ce manuscrit a démontré le danger que peut constituer un outil de CAO corrompu par un logiciel malveillant. Il convient de prendre en compte cette possibilité, d'étudier la possibilité de réaliser des logiciels malveillants ciblant les outils de CAO et de proposer des solutions efficaces pour limiter les vulnérabilités de ces derniers.

Publications et communications

Journal international avec comité de lecture

- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''The Security of ARM TrustZone in a FPGA-Based SoC''**. IEEE Transactions on Computers 68(8): 1238-1248 (2019)

Conférence international avec comité de lecture

- El Mehdi Benhani, Cédric Marchand, Alain Aubert, Lilian Bossuet. **''On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC''**. SoCC 2017: 108-113
- El Mehdi Benhani, Lilian Bossuet. **''DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC''**. ICECS 2018: 489-492

Posters

- El Mehdi Benhani, Cuauhtemoc Mancillas-López, Lilian Bossuet. **'Secure Internal Communication of a Trustzone-Enabled Heterogeneous SoC Lightweight Encryption''**. FPT 2019: 239-242
- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''L'évaluation de la sécurité de la technologie ARM TrustZone dans système sur puce hétérogène''**. Journée de la recherche de l'école doctorale EDSIS, Saint-Etienne, France, Juin 2019
- El Mehdi Benhani, Lilian Bossuet, Alain Aubert. **''L'évaluation de la sécurité de la technologie ARM TrustZone dans système sur puce hétérogène''**. Colloque national du GDR SoC2, Paris, France, juin 2018

Références bibliographies

[1] P. Kocher *et al.*, « Spectre attacks: Exploiting speculative execution », *arXiv preprint arXiv:1801.01203*, 2018.

[2] M. Lipp *et al.*, « Meltdown », *arXiv preprint arXiv:1801.01207*, 2018.

[3] S. Nie, L. Liu, et Y. Du, « Free-fall: hacking tesla from wireless to CAN bus », *Briefing, Black Hat USA*, p. 1–16, 2017.

[4] P. Kocher, J. Jaffe, B. Jun, et others, *Introduction to differential power analysis and related attacks*. 1998.

[5] S. Micali et L. Reyzin, « Physically observable cryptography », in *Theory of Cryptography Conference*, 2004, p. 278–296.

[6] N. Timmers, « Bypassing Secure Boot using Fault Injection », in *black hat*, 2016.

[7] B. Colombier, A. Menu, J.-M. Dutertre, P.-A. Moëllic, J.-B. Rigaud, et J.-L. Danger, « Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller. »

[8] A. Vasselle, H. Thiebeauld, Q. Maouhoub, A. Morisset, et S. Ermeneux, « Laser-induced fault injection on smartphone bypassing the secure boot », *IEEE Transactions on Computers*, 2018.

[9] S. Tajik, H. Lohrke, J.-P. Seifert, et C. Boit, « On the power of optical contactless probing: Attacking bitstream encryption of FPGAs », in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, p. 1661–1674.

[10] A. Cui et R. Housley, « BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection », in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[11] N. Timmers, A. Spruyt, et M. Witteman, « Controlling PC on ARM using fault injection », in *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016, p. 25–35.

[12] R. Muresan et C. Gebotys, « Current flattening in software and hardware for security applications », in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2004, p. 218–223.

[13] K. Tiri, M. Akmal, et I. Verbauwhede, « A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards », in *Proceedings of the 28th European solid-state circuits conference*, 2002, p. 403–406.

[14] J. Irwin, D. Page, et N. P. Smart, « Instruction stream mutation for non-deterministic processors », in *Proceedings IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, 2002, p. 286–295.

[15] J. A. Ambrose, R. G. Ragel, et S. Parameswaran, « RIJID: random code injection to mask power analysis based side channel attacks », in *Proceedings of the 44th annual Design Automation Conference*, 2007, p. 489–492.

[16] S. Skorobogatov, « Physical attacks and tamper resistance », in *Introduction to Hardware Security and Trust*, Springer, 2012, p. 143–173.

[17] M. A. Wahab, P. Cotret, M. N. Allah, G. Hiet, V. Lapotre, et G. Gogniat, « ARMHEx: A hardware extension for DIFT on ARM-based SoCs », in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, p. 1–7.

[18] GlobalPlatform, « TEE Protection Profile, GlobalPlatform Device Committee TEE Protection Profile Version 1.2.1 », *https://www.globalpla-tform.org/specificationsdevice.asp*.
[19] GlobalPlatform, « TEE client API specification version 1.0 », *http://globalpla-tform.org*, 2010.
[20] A. ARM, « Security technology building a secure system using trustzone technology (white paper) », *ARM Limited*, 2009.
[21] L. Bossuet, « Sécurité des systèmes embarqués », 2018.
[22] U. Kanonov et A. Wool, « Secure containers in Android: the Samsung KNOX case study », in *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2016, p. 3–12.
[23] QSEE, « QSEE, Retrieved from https://www.qualcomm.com/snapdragon/security », PhD Thesis.
[24] TRUSTONIC, « Security Target Kinibi v311A Security Target », PhD Thesis, 2017.
[25] SierraTEE, « Retrieved from http://www.openvirtualization.org/ », PhD Thesis, 2012.
[26] OP-TEE, « Project OP-TEE, GitHub repository », *https://github.com/OP-TEE*, 2017.
[27] B. McGillion T. Dettenborn, T. Nyman et N. Asokan, « Open-TEE - an open virtual trusted execution environment », PhD Thesis, 2015.
[28] TOPPERS, « SafeG, Retrieved from https://www.toppers.jp/en/safeg.html », PhD Thesis.
[29] S. Pinto et N. Santos, « Demystifying Arm TrustZone: A Comprehensive Survey », *ACM Computing Surveys (CSUR)*, vol. 51, n° 6, p. 130, 2019.
[30] S. Chow, P. Eisen, H. Johnson, et P. C. Van Oorschot, « White-box cryptography and an AES implementation », in *International Workshop on Selected Areas in Cryptography*, 2002, p. 250–270.
[31] Xilinx, « Zynq UltraScale+ MPSoC Technical Reference Manual UG1085 (v1.0) November 24, 2015 », 2015.
[32] Intel, « Intel Stratix 10 Hard ProcessorSystem Technical Reference Manual », 2019.
[33] V. Kindratenko et D. Pointer, « A case study in porting a production scientific supercomputing application to a reconfigurable computer », in *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2006, p. 13–22.
[34] A. Holdings, « AMBA AXI and ACE Protocol Specification », Tech. rep. 2011. url: http://infocenter. arm. com/help/topic/com. arm. doc …, 2013.
[35] A. Holdings, *ARM system memory management unit architecture specification—SMMU architecture version 2.0*. 2013.
[36] I. AMD et O. Virtualization, « Technology (IOMMU) Specification », 2007.
[37] Xilinx, « Zynq-7000 All Programmable SoC Technical Reference Manual, UG585 v1.11 », 2016.
[38] A. ARM, « Cortex-A9 TrustZone example », *http://infocenter.arm.co-m/help/topic/com.arm.doc.faqs/ka15417.html*, 2013.
[39] A. Moradi, A. Barenghi, T. Kasper, et C. Paar, « On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from xilinx Virtex-II FPGAs », in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, p. 111–124.
[40] D. Guillaume et A. Iván, « Vulnerabilities in High Assurance Boot of NXP i.MX microprocessors », *https://blog.quarkslab.com/vulnerabilities-in-high-assurance-boot-of-nxp-imx-microprocessors.html*, 2017.
[41] O. Savry, T. Hiscock, et M. El Majihi, *Sécurité matérielle des systèmes: Vulnérabilité des processeurs et techniques d'exploitation*. Dunod, 2019.
[42] F. Majeric, B. Gonzalvo, et L. Bossuet, « Jtag combined attack-another approach for fault injection », in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2016, p. 1–5.
[43] F. Majeric, « Etude d'attaques matérielles et combinées sur les "System-on-chip" », PhD Thesis, 2018.

[44] A. Moradi, D. Oswald, C. Paar, et P. Swierczynski, « Side-channel attacks on the bitstream encryption mechanism of Altera Stratix II: facilitating black-box analysis using software reverse-engineering », in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013, p. 91–100.
[45] A. Moradi et T. Schneider, « Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series », in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2016, p. 71–87.
[46] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, et G. Sigl, « How to break secure boot on fpga socs through malicious hardware », in *International Conference on Cryptographic Hardware and Embedded Systems*, 2017, p. 425–442.
[47] W. Li, Y. Xia, L. Lu, H. Chen, et B. Zang, « TEEv: virtualizing trusted execution environments on mobile platforms », in *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2019, p. 2–16.
[48] P. Carru, « Attack TrustZone with Rowhammer », *https://firmwares-ecurity.com/tag/rowhammer/*, 2017.
[49] V. Van Der Veen *et al.*, « Drammer: Deterministic rowhammer attacks on mobile platforms », in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, p. 1675–1689.
[50] J. A. Halderman *et al.*, « Lest we remember: cold-boot attacks on encryption keys », *Communications of the ACM*, vol. 52, n° 5, p. 91–98, 2009.
[51] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, et S. Mangard, « DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks », in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, p. 565–581.
[52] S. Chaudhuri, « A security vulnerability analysis of SoCFPGA architectures », in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, p. 1–6.
[53] B. Gras, K. Razavi, E. Bosman, H. Bos, et C. Giuffrida, « ASLR on the Line: Practical Cache Attacks on the MMU. », in *NDSS*, 2017, vol. 17, p. 26.
[54] D. Gruss, C. Maurice, K. Wagner, et S. Mangard, « Flush+ Flush: a fast and stealthy cache attack », in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, p. 279–299.
[55] Y. Yarom et K. Falkner, « FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack », in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, p. 719–732.
[56] B. Gras, K. Razavi, H. Bos, et C. Giuffrida, « Translation leak-aside buffer: Defeating cache side-channel protections with ${$TLB$}$ attacks », in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, p. 955–972.
[57] D. A. Osvik, A. Shamir, et E. Tromer, « Cache attacks and countermeasures: the case of AES », in *Cryptographers' track at the RSA conference*, 2006, p. 1–20.
[58] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, et S. Mangard, « Armageddon: Cache attacks on mobile devices », in *25th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 16)*, 2016, p. 549–564.
[59] A. Tang, S. Sethumadhavan, et S. Stolfo, « CLKscrew: Exposing the Perils of Security-Oblivious Energy Management, Usenix 2018 (Distinguished Paper Award) », 2018.
[60] J. Krautter, D. R. Gnad, et M. B. Tahoori, « FPGAhammer: remote voltage fault attacks on shared FPGAs, suitable for DFA on AES », *IACR Transactions on Cryptographic Hardware and Embedded Systems*, p. 44–68, 2018.
[61] M. Zhao et G. E. Suh, « FPGA-based remote power side-channel attacks », in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, p. 229–244.
[62] L. Bossuet, P. Bayon, et V. Fischer, « Electromagnetic transmission of intellectual property data to protect FPGA designs », in *IFIP/IEEE International Conference on Very Large Scale Integration-System on a Chip*, 2015, p. 150–169.

[63] E. Benhani et L. Bossuet, « Design a TrustZone-enalble SoC usign Xilinx VIVADO CAD tool », Technical Report, University of Lyon, 2017. https://perso. univ-stetienne …, 2017.
[64] C. Benhani El mehdi, Marchand, A. Aubert, et L. Bossuet, « On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC », in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, 2017, p. 108–113.
[65] E. Benhani, L. Bossuet, et A. Aubert, « The Security of ARM TrustZone in a FPGA-based SoC », *IEEE Transactions on Computers*, vol. 68, n° 8, p. 1238–1248, 2019.
[66] « (UG835) Vivado Design Suite Tcl Command Reference Guide », *Xilinx Inc*, 2013.
[67] N. Fern, I. San, C. K. Koç, et K.-T. T. Cheng, « Hardware trojans in incompletely specified on-chip bus systems », in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, 2016, p. 527–530.
[68] R. J. Masti, D. Rai, A. Ranganathan, C. Müller, L. Thiele, et S. Capkun, « Thermal covert channels on multi-core platforms », in *24th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 15)*, 2015, p. 865–880.
[69] M. Alagappan, J. Rajendran, M. Doroslovački, et G. Venkataramani, « DFS covert channels on multi-core platforms », in *2017 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, p. 1–6.
[70] L. Bossuet, P. Bayon, et V. Fischer, « An Ultra-Lightweight Transmitter for Contactless Rapid Identification of Embedded IP in FPGA », *IEEE Embedded Systems Letters*, vol. 7, p. 1-1, 2015, doi: 10.1109/LES.2015.2454236.
[71] L. Bossuet et others, « DVFS as a Security Failure of TrustZone-enabled Heterogeneous SoC », in *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2018, p. 489–492.
[72] M. Kim, S. Kong, B. Hong, L. Xu, W. Shi, et T. Suh, « Evaluating coherence-exploiting hardware trojan », in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, p. 157–162.
[73] J. Daemen et V. Rijmen, *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.
[74] D. J. Bernstein, « Cache-timing attacks on AES », 2005.
[75] B. Gülmezoğlu, M. S. Inci, G. Irazoqui, T. Eisenbarth, et B. Sunar, « A faster and more realistic flush+ reload attack on AES », in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, 2015, p. 111–126.
[76] E. Savaş et C. Yılmaz, « A generic method for the analysis of a class of cache attacks: a case study for AES », *The Computer Journal*, vol. 58, n° 10, p. 2716–2737, 2015.
[77] E. Tromer, D. A. Osvik, et A. Shamir, « Efficient cache attacks on AES, and countermeasures », *Journal of Cryptology*, vol. 23, n° 1, p. 37–71, 2010.
[78] E. Hallett, « Isolation Design Flow for Xilinx 7 Series FPGAs or Zynq-7000 AP SoCs (Vivado Tools) ».
[79] L. B. Benhani El Mehdi, [En ligne]. Disponible sur: http://labh-curien.univ-st-etienne.fr/~bossuet/Security_design-rules_checker.zip.
[80] C. De Canniere et B. Preneel, « Trivium specifications », in *eSTREAM, ECRYPT Stream Cipher Project*, 2005.
[81] S. Babbage *et al.*, « The eSTREAM portfolio », *eSTREAM, ECRYPT Stream Cipher Project*, p. 1–6, 2008.
[82] M. Hell, T. Johansson, A. Maximov, et W. Meier, « A stream cipher proposal: Grain-128 », in *2006 IEEE International Symposium on Information Theory*, 2006, p. 1614–1618.