Les SoCs complexes hétérogènes et leur sécurité

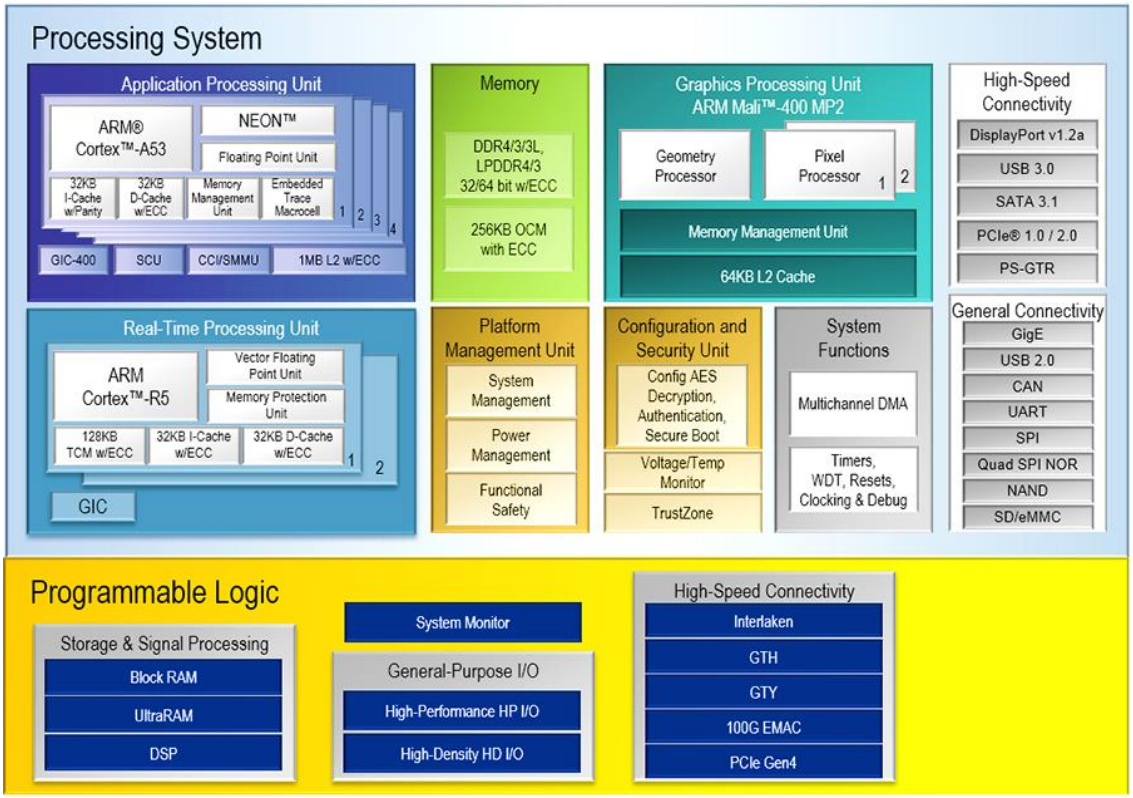
Ce chapitre présente le contexte des travaux présentés dans ce manuscrit. Au travers de celui-ci, la complexité des SoCs hétérogènes modernes, leurs architectures de bus et celles de leurs mémoires sont présentées. La technique du DVFS (Dynamic Voltage and Frequency Scaling) utilisée pour réduire la 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-

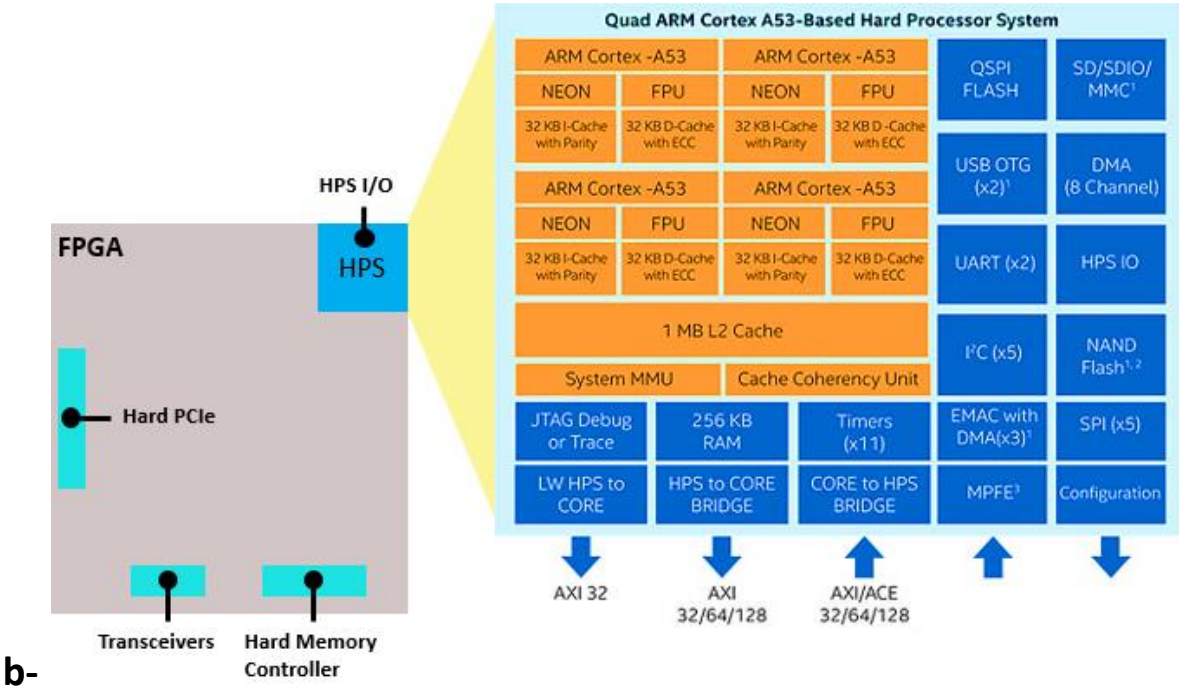


Figure 2 : Architecture d'un SoC complexe hétérogène, a- Architecture du SoC Xilinx Zynq Ultrascale+, b- Architecture du SoC Intel Stratix 10 SX

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.

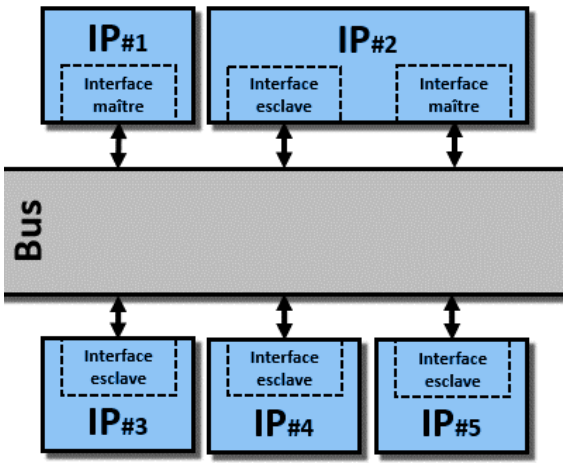


Figure 3: Structure d'un bus

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 :

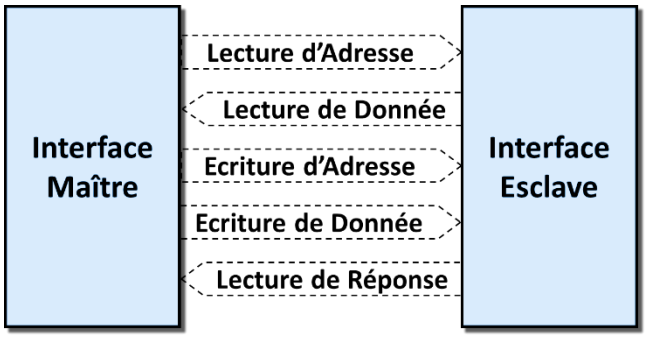


Figure 4: Les canaux de communication du bus AXI entre une interface maître et une interface esclave

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

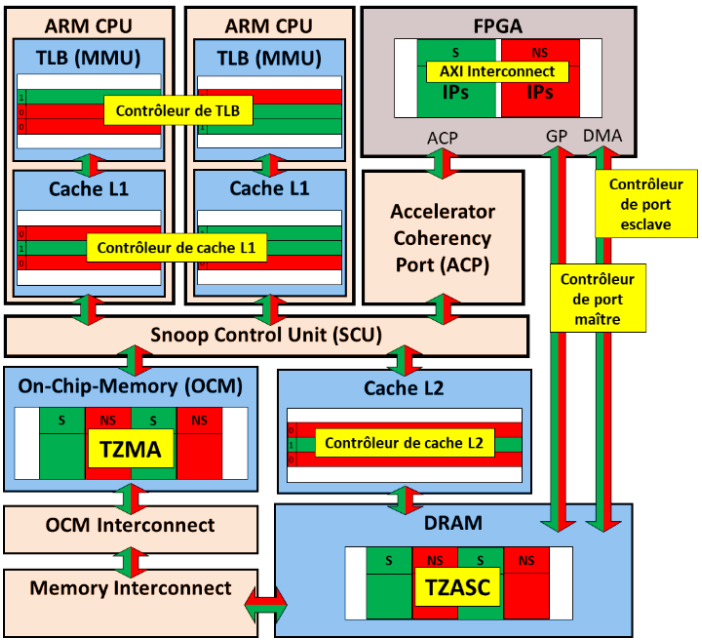


Figure 5 : Représentation schématique de la hiérarchie mémoire dans un SoC complexe hétérogène, tels que les SoC Xilinx Zynq-7000 et Intel Cyclone V.

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.

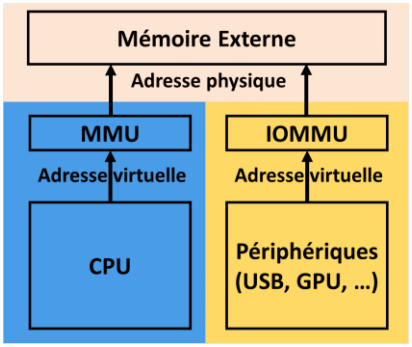


Figure 6 : Les unités de gestion de mémoire dans un SoC FPGA

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).

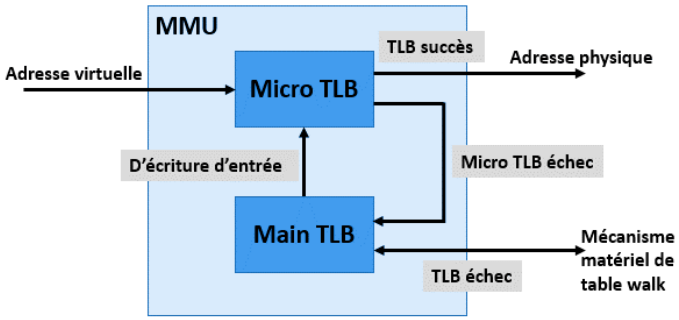


Figure 7 : MMU, processus de traduction d'une adresse virtuelle en adresse physique

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.

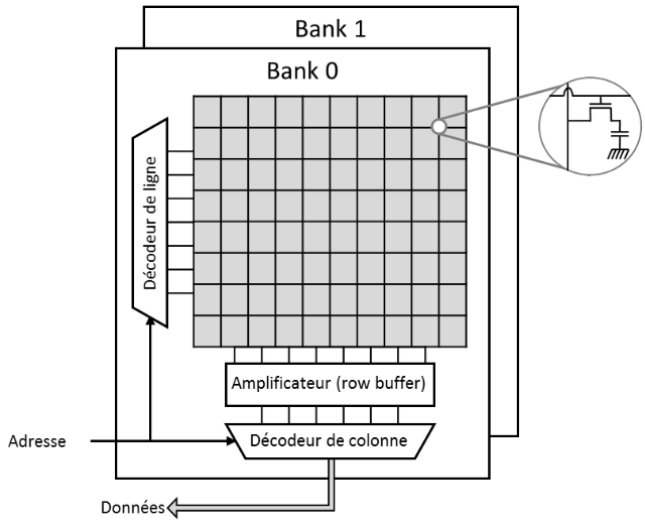


Figure 8 : Structure d'une matrice de cellules mémoire (Bank) DRAM

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.

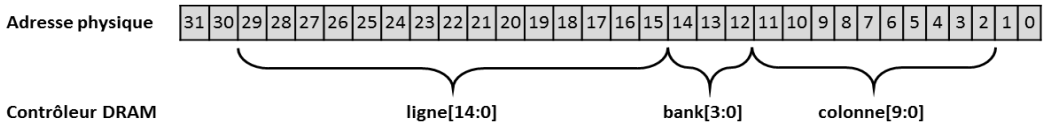


Figure 9 : Décodage d'une adresse physique dans le SoC Zynq-7000

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.

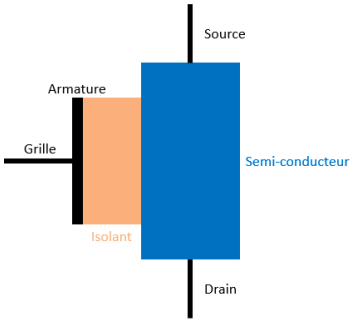


Figure 10 : Anatomie d'un CMOS

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

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.

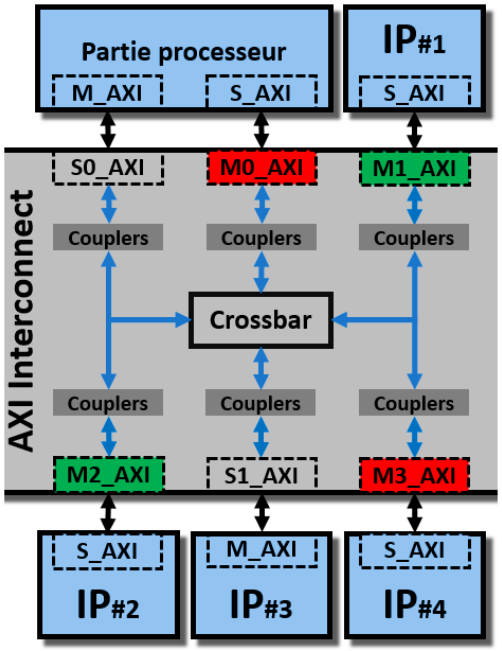


Figure 11: Illustration d'un AXI Interconnect couplant des interfaces maître et esclave

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.