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.

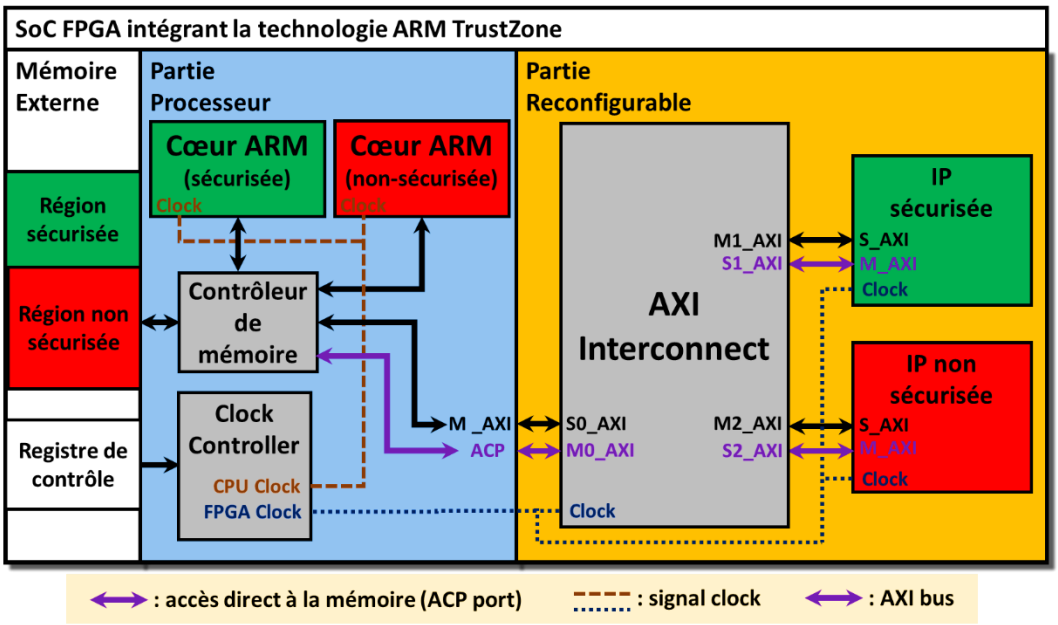


Figure 28: Design expérimental

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.

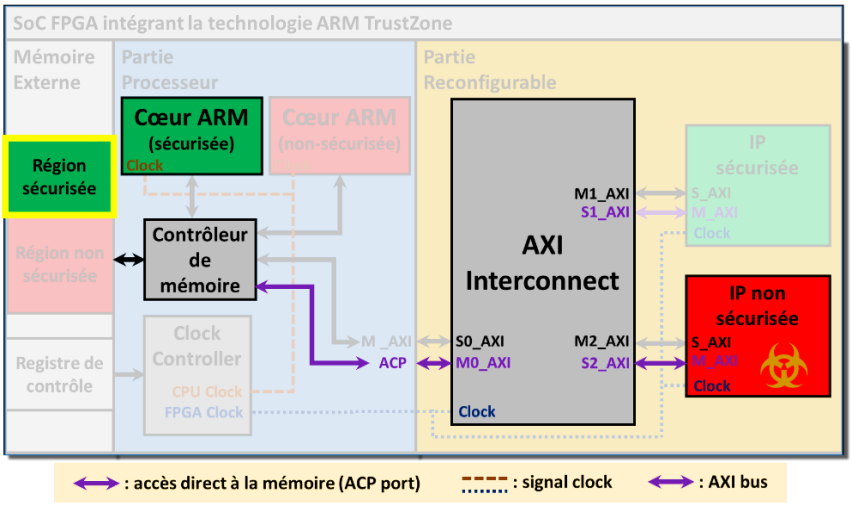


Figure 29: Attaque DMA

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.

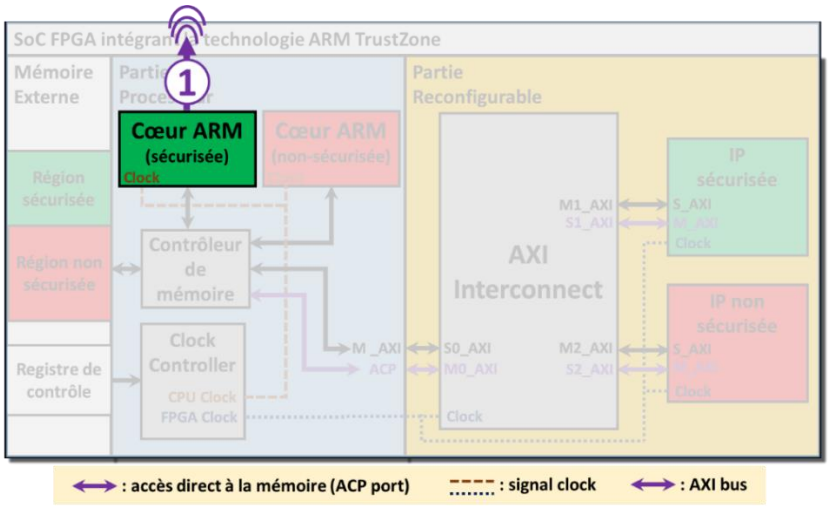


Figure 30: Transfert des données sensibles vers l'extérieur du SoC FPGA

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.

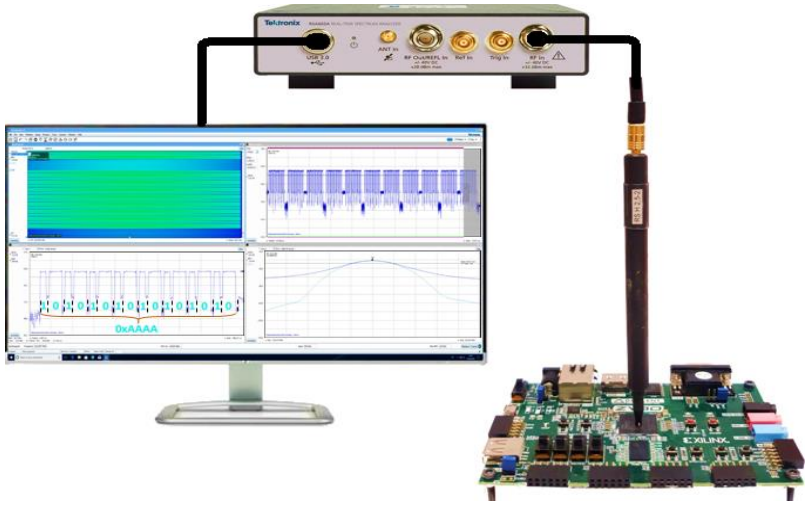


Figure 31: Analyse spectrale en temps réel de la fuite électromagnétique de données secrètes

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.

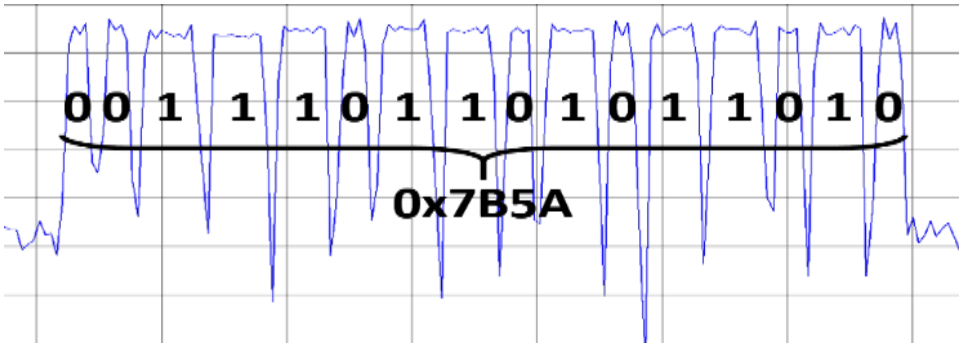
Pour le processus espion, le pilote de la technique DFS du système d'exploitation de confiance inclus le code malveillant présenté dans l'algorithme 1. Le code malveillant utilise la modulation de fréquence pour transférer les données sensibles. Pour transférer un 1 logique, l'algorithme 1 force le système 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

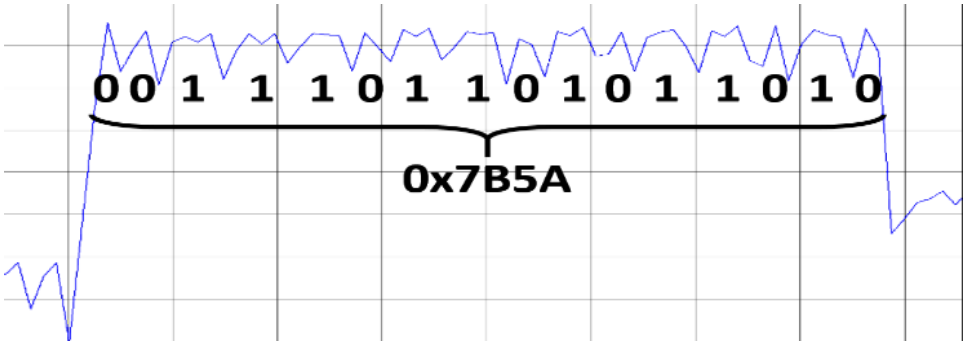
```
Input: donnée_à_transférer
    For i = donnée_à_transférer_size To 0 Do
        If (donnée_à_transférer[i] = 1) Then
          freq_actuelle = freq_1
          loop for Tempo_1
        Else
          freq_actuelle = freq_1
          loop for Tempo_2
          End If
        freq_actuelle = freq_2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

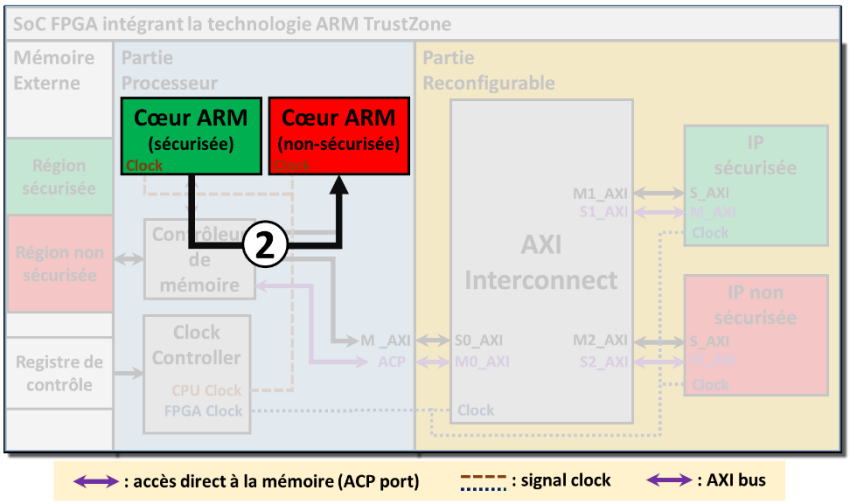


Figure 33: Transfert des données sensibles du cœur ARM sécurisé vers le cœur ARM non-sécurisé dans SoC FPGA

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.

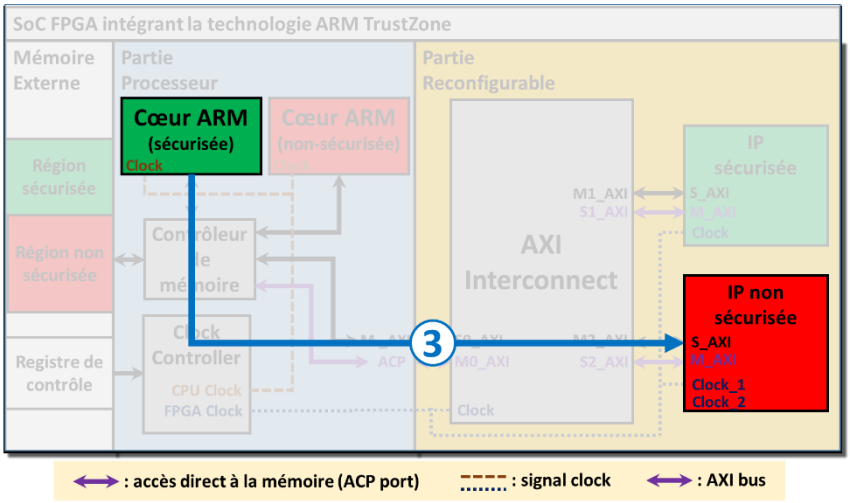


Figure 34: Transfert des données sensibles du cœur ARM sécurisé vers l'IP non-sécurisés dans un SoC FPGA

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.

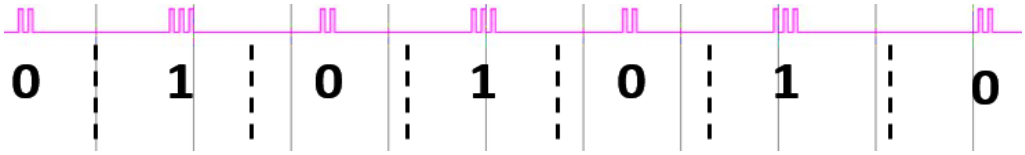


Figure 35: Décodage des données reçues

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

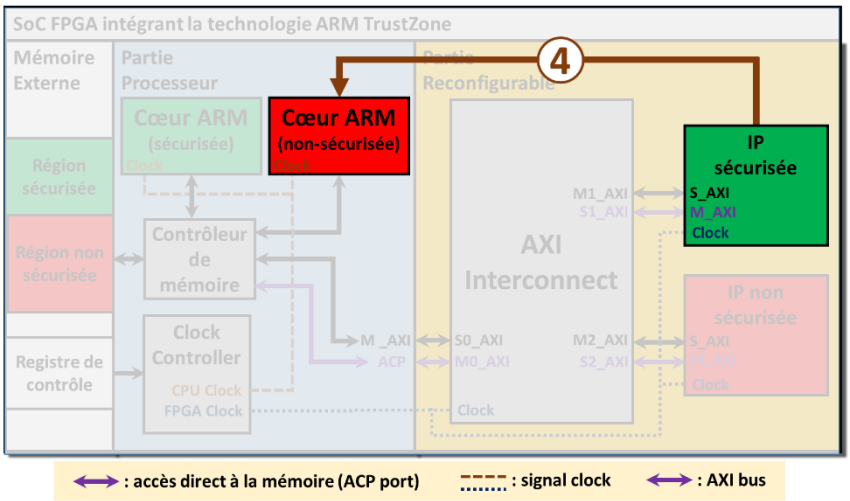


Figure 36: Transfert des données sensibles de l'IP sécurisée vers le cœur ARM non-sécurisé dans un SoC FPGA

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.

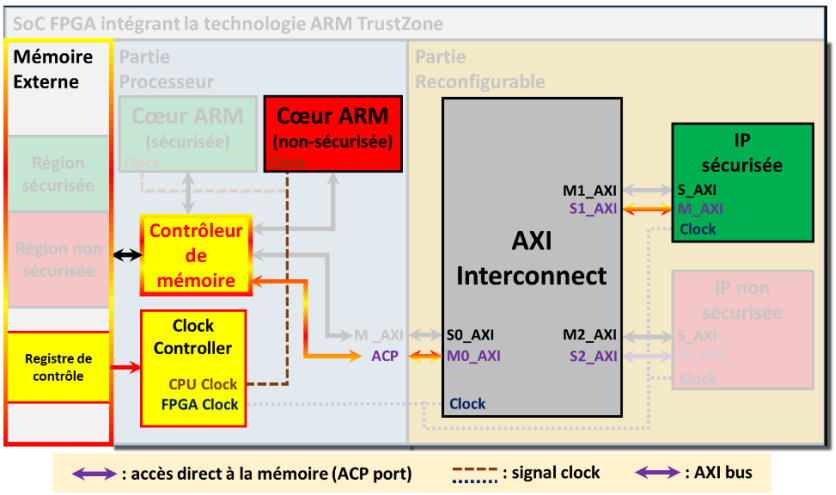


Figure 37: Manipulation de l'horloge par accès direct à la mémoire externe via l'interface ACP

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.