Thèse de doctorat de l'université Pierre et Marie Curie - Paris VI

Spécialité Informatique

(École Doctorale Informatique, Télécommunication et Électronique)

Présentée par Joël Porquet

Pour obtenir le grade de

Docteur de l'université Pierre et Marie Curie

Architecture de sécurité dynamique pour systèmes multiprocesseurs intégrés sur puce

Soutenue le 13 décembre 2010, devant le jury composé de

M.	Albert Cohen	INRIA	Rapporteur
M.	Tanguy Risset	INSA-Lyon	Rapporteur
M.	Jean-Claude Bajard	Paris VI	Examinateur
M.	Renaud Pacalet	Télécom ParisTech	Examinateur
M.	Bernard Kasser	STMicroelectronics	Examinateur
M.	Christian Schwarz	Nagravision	Examinateur
M.	Alain Greiner	Paris VI	Directeur de thèse

Résumé

Cette thèse présente l'approche *multi-compartiment*, qui autorise un co-hébergement sécurisé et flexible de plusieurs piles logicielles autonomes au sein d'un même système multiprocesseur intégré sur puce.

Dans le marché des appareils orientés multimédia, ces piles logicielles autonomes représentent généralement les intérêts des différentes parties prenantes. Ces parties prenantes sont multiples (fabricants, fournisseurs d'accès, fournisseurs de contenu, utilisateurs, etc.) et ne se font pas forcément confiance entre elles, d'où la nécessité de trouver une manière de les exécuter ensemble mais avec une certaine garantie d'isolation. Les puces multimédia étant matériellement fortement hétérogènes – peu de processeurs généralistes sont assistés par une multitude de processeurs ou coprocesseurs spécialisés – et à mémoire partagée, il est difficile voire impossible de résoudre cette problématique uniquement avec les récentes techniques de co-hébergement (*virtualisation*).

L'approche multi-compartiment consiste en un nouveau modèle de confiance, plus flexible et générique que l'existant, qui permet à des piles logicielles variées de s'exécuter simultanément et de façon sécurisée sur des plateformes matérielles hétérogènes. Le cœur de l'approche est notamment composé d'un mécanisme global de protection, responsable du partage sécurisé de l'unique espace d'adressage et logiquement placé dans le réseau d'interconnexion afin de garantir le meilleur contrôle. Cette approche présente également des solutions pour le partage des périphériques, notamment des périphériques ayant une capacité DMA, entre ces piles logicielles. Enfin, l'approche propose des solutions pour le problème de redirection des interruptions matérielles, un aspect collatéral au partage des périphériques.

Les principaux composants des solutions matérielles et logicielles proposées sont mis en œuvre lors de la conception d'une plateforme d'expérimentation, sous la forme d'un prototype virtuel. Outre la validation de l'approche, cette plateforme permet d'en mesurer le coût, en termes de performance et de surface de silicium. Concernant ces deux aspects, les résultats obtenus montrent que le coût est négligeable.

Mots-clés : sécurité, système multiprocesseur intégré sur puce, réseau sur puce, co-hébergement logiciel, virtualisation, conception matérielle-logicielle, prototypage virtuel.

Abstract

This thesis presents the *multi-compartment* approach. This approach enables a secure and flexible co-hosting of multiple autonomous software stacks within a same multiprocessor system-on-a-chip.

In the field of multimedia oriented consumer devices, such autonomous software stacks generally represent the assets of the different stakeholders. These stakeholders, chips and set-top boxes manufacturers, network operators, content providers and customers, do not necessarily trust each other. Hence, the requirement to find a means to execute those software stacks together, while enforcing a certain degree of isolation. Multimedia chips are heavily heterogeneous – a few general purpose processors are assisted by numerous specialized processors or coprocessors – and follow a shared memory policy. These hardware specificities make it difficult, and even impossible, to solve this problematic with recent co-hosting techniques only (e.g. *virtualization*).

The multi-compartment approach consists in a new trust model, more flexible and generic than the current ones. It allows various software stacks to run securely and simultaneously on heterogeneous hardware platforms. In particular, the core of the proposed approach is composed of a global mechanism for protection. Such a mechanism is responsible for the secure sharing of the single address space and is placed within the interconnect to ensure the best control. The multi-compartment approach also presents solutions for sharing peripheral devices, and more precisely DMA capable devices, among software stacks. Finally, the approach introduces solutions for the hardware interrupts redirection problem, a collateral aspect to the peripheral devices sharing.

The main building blocks of the proposed hardware and software solutions are implemented along with the conception of an experimental platform, under the form of a virtual prototype. In addition to validating the approach, the platform is measured in terms of cost, performance and hardware surface. Considering both aspects, the obtained results show the cost is negligible.

Keywords: security, multiprocessor system-on-a-chip, network-on-chip, software co-hosting, virtualization, hardware-software co-design, virtual prototyping.

Remerciements

Les années de thèse sont des années difficiles, pendant lesquelles on alterne entre incompréhension totale et illumination géniale, fatigue et vigueur, découragement et satisfaction... Fort heureusement, j'ai eu la chance d'avoir à mes côtés des personnes de qualité qui m'ont aidé tout au long de ce parcours initiatique. Par ces quelques lignes, je voudrais leur rendre l'hommage qu'ils méritent.

Tout d'abord, je voudrais exprimer ma plus grande gratitude à mon directeur de thèse, Alain Greiner, Professeur à Paris VI, pour la confiance qu'il m'accorde depuis de nombreuses années et que je m'efforce de ne jamais trahir. Malgré l'éloignement géographique, il a su faire preuve d'une grande disponibilité à mon égard. J'espère sincèrement avoir la chance de rester dans son entourage professionnel pour les années à venir.

Je souhaite ensuite remercier l'ensemble des membres de mon jury, et particulièrement mes deux rapporteurs, Tanguy Risset, Professeur à l'INSA-Lyon, et Albert Cohen, Directeur de recherche à l'INRIA, qui ont accepté de relire ce manuscrit de thèse malgré les contraintes temporelles que je leur ai imposées, et dont les rapports me rendent aujourd'hui fier du travail accompli.

Je remercie l'entreprise STMicroelectronics, et notamment Bernard Kasser, Directeur en R&D, de m'avoir proposé cette thèse et de m'avoir accueilli dans son équipe. J'ai beaucoup aimé travailler avec Christian Schwarz, mon encadrant industriel et avant tout ami, notamment pour l'indépendance qu'il a réussi à m'obtenir. J'adresse ma sympathie à l'ensemble de l'équipe, avec une pensée spéciale pour Pierre Guillemin, Stephan Courcambeck et Albert Martinez. Je n'oublie pas les nombreux stagiaires que j'ai eu l'occasion de croiser pendant ma thèse, et que je regrette de ne plus pouvoir battre à la pétanque, sous le soleil provençal : Sébastien Cerdan, Tony Baudon, Morgan Bourré, et tous les autres.

La réalisation effective de mes travaux n'aurait pas pu être possible sans l'impressionnant travail d'Alexandre Bécoulet et de Nicolas Pouillon, qui bénéficiant déjà de mon amitié, bénéficient aujourd'hui également de mon respect professionnel le plus sincère.

La rédaction de ce manuscrit a été effectuée à Paris, au LIP6, et par conséquent, je tiens à remercier Cécile Braunstein et Étienne Faure d'avoir subi cette période douloureuse à mes côtés. Je remercie également Karine Heydermann pour sa relecture méticuleuse. J'en profite aussi pour saluer tous mes autres collègues au LIP6, notamment de m'avoir permis de réintégrer le monde universitaire après mon aventure industrielle.

Je ne peux évidemment pas oublier la famille et les amis : je témoigne donc ma profonde affection à ma mère, ma directrice de thèse de l'ombre, ainsi qu'à mon père et à mon frère. J'envoie également mes sentiments les plus tendres à Aurélie, pour son inconditionnel soutien et son dévouement quotidien. Je remercie aussi ma famille marseillaise pour les nombreuses invitations qui ont souvent égayées mes week-ends. Enfin, je renouvelle mon amitié à tous les parisiens, que je n'ai pas beaucoup vus ces trois dernières années, mais

que je suis réellement content de retrouver aujourd'hui (Fred, Mathieu, Greg, Kass, Julien, Annick&Fred, etc.)

Pour terminer, je m'excuse auprès de ceux que j'aurais pu oublier, et à qui j'adresse évidemment mes sincères remerciements.

Sommaire

Chapitre 1

Introduction

La densité d'intégration des transistors au sein d'une même puce de silicium n'a cessé d'augmenter ces dernières années. Cette intégration permet aujourd'hui de placer sur une unique puce un système de traitement complet : processeurs, coprocesseurs, périphériques d'entrée/sortie et mémoire. Ce type de *système intégré sur puce* est particulièrement utilisé dans le domaine de l'embarqué, c'est-à-dire lorsqu'un *système embarqué* est dédié à une application spécifique.

Dans le domaine des applications orientées multimédia, les appareils de type décodeur numérique pour télévision (*set-top box*) ou téléphone de dernière génération (*smartphone*) utilisent des systèmes intégrés sur puce complexes, composés de multiples processeurs ou coprocesseurs hétérogènes.

Ce type d'appareil fait actuellement face à un double enjeu de sécurité et de flexibilité : ils sont particulièrement concernés par la protection de données, puisque les flux multimédia qu'ils manipulent sont la plupart du temps protégés, et leur adoption massive par le grand public entraîne des besoins de flexibilité, souvent bien difficiles à concilier avec la sécurité.

L'hétérogénéité des systèmes intégrés sur puce, que ces appareils embarquent, rend malheureusement quasiment impossible le déploiement d'une unique pile logicielle de confiance pour contrôler l'ensemble du système. Un co-hébergement doit alors être mis en place, répartissant les différents types de processeur qui composent le système sous le contrôle de plusieurs piles logicielles indépendantes. Peu de processeurs généralistes exécutent par exemple l'interface utilisateur de l'appareil, tandis que les processeurs de faible complexité et les coprocesseurs exécutent les services que l'on souhaiterait sécuriser.

Sachant que les données à protéger sont contenues dans la mémoire du système et que ce type de système suit généralement une politique de mémoire partagée, cette sécurisation n'est pas triviale. A l'heure actuelle, aucun mécanisme n'existe pour les systèmes fortement hétérogènes qui permette un partage sécurisé, dynamique et flexible de la mémoire entre plusieurs piles logicielles indépendantes.

Le travail présenté dans cette thèse tente de résoudre cette problématique de sécurité

et de flexibilité, en proposant une architecture de sécurité dynamique pour les systèmes multiprocesseurs sur puce hétérogènes. Outre le partage de la mémoire, cette architecture s'intéresse également au partage des périphériques.

Cadre de la thèse

Cette thèse a été effectuée dans le cadre d'un partenariat, par contrat CIFRE [1], entre le **Laboratoire d'Informatique de Paris 6** (**LIP6**) et **STMicroelectronics**. Les travaux présentés ont bénéficié des compétences en conception des systèmes intégrés sur puce de l'équipe *Architecture et Logiciels pour Systèmes Embarqués sur Puce*, intégrée au département *Systèmes Embarqués sur Puce* du LIP6, et de l'expertise en sécurité informatique de l'équipe *Security Roadmap*, intégrée à la division *Advanced System Technology* de STMicroelectronics.

Ce travail a donné lieu à deux publications dans des conférences internationales avec actes et comité de lecture [1, 2], ainsi qu'à deux dépôts de brevet en Europe et aux États-Unis [3, 4].

Organisation du document

Le chapitre 2 présente progressivement le double enjeu de sécurité et de flexibilité, que les systèmes intégrés sur puce considérés doivent résoudre. Nous en déduisons les problèmes techniques que ce double enjeu provoque, et formulons les questions auxquelles nous répondrons tout au long de ce document.

Le chapitre 3 décrit, dans un premier temps, les solutions existantes pour traiter la problématique de co-hébergement de piles logicielles indépendantes. Puis, il présente les différentes approches d'architectures matérielles sécurisées.

Le chapitre 4 expose la solution présentée dans cette thèse, l'approche *multi-compartiment*, qui repose sur deux principes : l'identification et la protection. Ces principes sont expliqués, puis les différents mécanismes de sécurité, matériels et logiciels, qui permettent la mise en œuvre de ces principes. Des solutions pour le partage des périphériques sont également proposées, et les problèmes collatéraux, tels que la redirection des interruptions matérielles, sont examinés.

Le chapitre 5 introduit une plateforme d'évaluation matérielle et logicielle, compatible avec l'approche multi-compartiment. Nous expliquons les choix retenus pour la réalisation de cette plateforme, et décrivons son implémentation, matérielle et logicielle.

Le chapitre 6 évalue les différents coûts engendrés par la mise en œuvre de l'approche multi-compartiment, et plus précisément les surcoûts en performance et en surface de

1. Conventions Industrielles de Formation par la REcherche

silicium relatifs aux mécanismes matériels de sécurité.

Enfin, le chapitre 7 conclut sur le travail présenté dans cette thèse en répondant aux questions initialement posées dans le chapitre 2, et propose des perspectives à ce travail.

Chapitre 2

Problématique

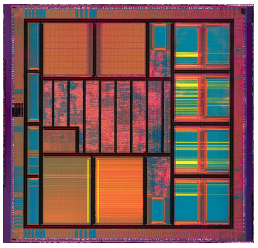
Sommaire

Ce travail de thèse s'inscrit dans le contexte de la conception de systèmes intégrés sur puce.

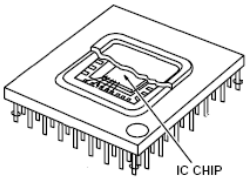
Une puce est un dispositif électronique miniaturisé composé de semi-conducteurs, principalement des transistors. D'après la « loi de Moore », le nombre de transistors qui composent une puce, ou la *densité d'intégration*, est supposé doubler tous les deux ans. Dans les années soixante, les puces ont commencé par intégrer quelques dizaines de transistors, les technologies de développement les plus récentes permettent de fabriquer des puces composées de plusieurs milliards de transistors. Concrètement, une puce est maintenant en mesure d'intégrer tous les composants d'un système de traitement complet, c'est-à-dire au moins un processeur, de la mémoire et des périphériques d'entrée/sortie. On parle alors d'un *système intégré sur puce* (ou *SoC*, pour *System-on-a-Chip*).

La consommation d'énergie est l'une des principales contraintes dans la conception de SoC. Dans de nombreuses applications, le système sera alimenté par batterie et n'incorporera aucun mécanisme de dissipation thermique. Afin de respecter cette contrainte, tout en permettant l'augmentation des performances, la densité d'intégration a généralement été mise à profit pour multiplier le nombre de processeurs dans un même système. Un tel système multiprocesseur intégré sur puce est alors appelé *MP-SoC* (*Multi-Processor System-on-a-Chip*)

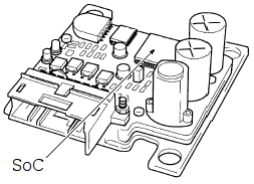
N'étant qu'une simple puce, un MP-SoC fait généralement parti d'un dispositif plus large, souvent un *système embarqué*. Les systèmes embarqués couvrent un vaste champ d'application mais ils désignent fondamentalement tous les systèmes électroniques conçus pour accomplir une ou plusieurs tâches spécifiques, et embarqués dans des appareils plus grands. Cette chaine d'inclusion est présentée sur la figure 2.1. Les systèmes embarqués sont omniprésents de nos jours. On les trouve au sein d'appareils infiniment variés : programmateurs de machine à laver, téléphones portables ou encore terminaux de géolocalisation (*GPS*), pour les usages grand public ; automobiles (les systèmes électroniques représentent aujourd'hui un tiers du prix d'un véhicule), avioniques ou encore équipement médical, pour les usages industriels.



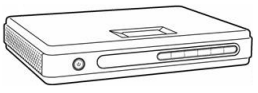
(a) Une puce



(b) Un SoC dans son boitier



(c) Un système embarqué muni d'un SoC



(d) Un appareil utilisateur équipé d'un système embarqué

FIGURE 2.1 – Chaine d'inclusion : de la puce à l'appareil.

Pour beaucoup de ces systèmes sur puce embarqués, la protection de données, et plus globalement la sécurité, se révèle être une question majeure [5]. Comme nous le mettons en évidence dans la suite, ce sont les mêmes systèmes qui sollicitent en même temps une flexibilité toujours plus grande.

L'objectif de ce chapitre est d'exprimer ce double enjeu, sécurité et flexibilité, dans un premier temps d'un point de vue de haut niveau, puis d'un point de vue technique, tout en soulevant les problèmes qu'il entraîne.

2.1 Définition des enjeux : sécurité et flexibilité

La sécurité est particulièrement présente dans les systèmes orientés multimédia, qui sont souvent amenés à manipuler des œuvres audiovisuelles protégées. Les appareils de type

set-top box, c'est-à-dire tous les décodeurs numériques capables de transformer un signal externe en contenu et de l'afficher sur un téléviseur, en sont un excellent exemple : leur besoin en termes de sécurité interne est important et complexe, tandis que les utilisateurs demandent toujours plus de flexibilité.

2.1.1 Sécurité

Dans de tels appareils, la protection de données intervient à deux niveaux.

Le premier niveau concerne globalement la gestion des droits d'accès. Comme l'explique [6], le système est conceptuellement partitionné en sous-systèmes sécurisés, chacun d'entre eux étant considéré comme une zone de confiance autonome. L'illustration de la figure 2.2 montre une set-top box qui est typiquement constituée d'une petite dizaine de sous-systèmes sécurisés, parmi lesquels :

- L'*Accès Conditionnel* (plus souvent connu sous son nom anglais : *Conditional Access*, ou *CA*) autorise l'accès à du contenu protégé, uniquement aux abonnés ou usagers accrédités. Ce système est traditionnellement appliqué au contenu télédiffusé, comme les chaînes télévisées.
- La *GDN* ou *Gestion des Droits Numériques* (également plus souvent connu sous son nom anglais : *Digital Rights Management*, ou *DRM*) est un autre système d'accès conditionnel qui concerne la protection d'œuvres numériques spécifiques, et qui peut s'appliquer à tous types de support numériques physiques (*DVD*, *Blu-ray*, etc.) ou de transmission (télédiffusion, Internet, etc).
- Le *Numériscope* (ou *PVR*, pour *Personal Video Recorder*) sécurise la copie locale du contenu, permettant ainsi à l'utilisateur de bénéficier d'un visionnage différé.
- La *Protection de sortie* gère la protection du contenu lorsque celui-ci est envoyé à l'écran, tandis que la *Protection de lien* assurent sa protection lorsqu'il est échangé avec d'autres appareils.

Ces sous-systèmes sécurisés sont généralement le reflet des différentes parties prenantes intervenant dans la composition de l'appareil, et contiennent tous des informations internes sensibles : clés ou certificats cryptographiques, propriétés intellectuelles tels des algorithmes ou du code sensible, etc. Le premier aspect de la sécurité concerne alors la protection de ces informations, propres à chaque sous-système sécurisé.

Le deuxième aspect de la sécurité des données concerne le contenu multimédia lui-même. Dans ces systèmes, le contenu est un flot de données qui doit généralement traverser plusieurs sous-systèmes pour être consommé. Par exemple, si le consommateur souhaite enregistrer une émission télévisée, alors le flot de données sera reçu par l'*Accès Conditionnel* puis envoyé au *Numériscope* ; lorsqu'il souhaitera visionner l'émission, le flot de données sera transféré du *Numériscope* à la *Protection de sortie*, pour enfin être affiché à l'écran. Tous les sous-systèmes doivent donc travailler ensemble de façon sécurisée pour éviter d'exposer le contenu protégé qu'ils font transiter.

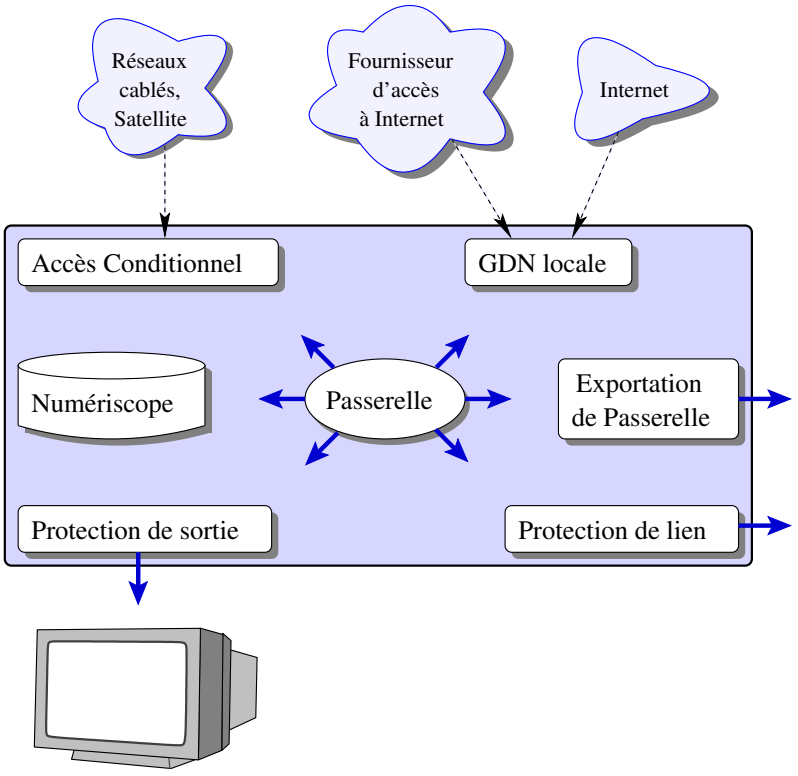


FIGURE 2.2 – Les différents sous-systèmes sécurisés d'une set-top box.

Dans le monde de la téléphonie mobile, on pouvait citer, jusqu'à présent, au moins un sous-système sécurisé : le *Baseband*, en charge du contrôle d'accès au réseau mobile. Mais de plus en plus, et notamment avec les *smartphones* de dernière génération, ces appareils sont en train de devenir une branche à part entière du marché multimédia, ce qui leur fera rencontrer les mêmes problématiques que celles que l'on développe ici.

2.1.2 Flexibilité

Les systèmes orientés multimédia sont la plupart du temps des produits grand public, donc manipulés directement par les consommateurs. Cette caractéristique importante se traduit généralement par une forte demande concernant l'augmentation du nombre de fonctionnalités, et plus globalement, une plus grande maîtrise des utilisateurs sur leurs appareils. Plus que jamais, ils attendent en effet une meilleure intégration de tous leurs équipements électroniques.

Ce besoin d'interopérabilité a, depuis longtemps, poussé les industriels à définir des standards techniques. Concernant la diffusion par exemple, des normes internationales existent, et traitent la compression, la décompression, le traitement et le codage des flots de données multimédia (*JPEG*, *MPEG*, etc). Concernant la communication, nombre de consortiums industriels se sont montés récemment avec l'objectif d'améliorer l'interopérabilité entre les équipements (par exemple, *DLNA* [7] ou l'*Open Handset Alliance* [8]).

Puisque ces standards ne cessent d'évoluer, l'utilisation de techniques de téléchargement, le plus souvent par Internet, est requise pour effectuer la mise à jour du logiciel embarqué, permettant ainsi de prolonger la durée de vie des appareils. Plus généralement, Internet est amené à jouer un rôle toujours plus important dans ce type de dispositifs. Le temps où les appareils multimédia étaient complètement indépendants est incontestablement révolu.

Du côté industriel, les besoins de flexibilité engendrent des conséquences critiques. Les parties prenantes des systèmes orientés multimédia sont confrontées à une nouvelle combinaison dans la chaîne de composition des produits. Traditionnellement, une set-top box était, par exemple, le résultat d'une coopération préétablie entre différents fabricants (ou *OEM*, pour *Original Equipment Manufacturer*), normalement un fournisseur d'accès (câble, satellite ou Internet) et des fournisseurs de contenu associés (chaines télévisées, vidéo à la demande, radio, etc). Dans ce modèle, le consommateur n'avait qu'un point d'entrée unique, souvent à travers le fournisseur d'accès. Aujourd'hui, des modèles alternatifs émergent, proposant une vente au détail des set-top box auprès des consommateurs, qui sont alors libres de choisir indépendamment leurs fournisseurs d'accès et de contenu, et même de souscrire à des services dits *OTT* (*Over-the-top*), c'est-à-dire des fournisseurs de contenu indépendants qui diffusent sur le réseau Internet sans aucune affiliation avec les fournisseurs d'accès.

2.1.3 Synthèse

La flexibilité dans les systèmes multimédia est une tendance clairement en train d'émerger. Le modèle commercial change, laissant beaucoup plus de liberté aux consommateurs. Les dispositifs sont très évolutifs, notamment grâce à l'utilisation intensive d'Internet, ce qui leur permet d'en tirer des possibilités de mise à jour et d'extension. La gestion du contenu protégé nécessite toutefois un environnement sécurisé. Des sous-systèmes indépendants doivent en effet être capables de protéger leurs secrets internes, tout en collaborant afin d'offrir du contenu protégé aux consommateurs.

2.2 Caractéristiques des systèmes orientés multimédia

Bien comprendre la difficulté d'allier les enjeux de haut niveau concernant la sécurité et la flexibilité nécessite d'abord de définir les propriétés de l'environnement dans lequel ils doivent fonctionner. Cet environnement est en fait l'association d'une plateforme matérielle et d'un logiciel embarqué, qui ensemble constituent un MP-SoC, et par extension un système embarqué.

Dans un premier temps, nous décrivons brièvement les caractéristiques générales des MP-SoC actuels, puis nous nous attardons sur les spécificités matérielles et logicielles des systèmes orientés multimédia. Enfin, nous analysons en quoi la combinaison de ces spécificités rend la conciliation de la sécurité et de la flexibilité très difficile en l'état.

2.2.1 Plateforme matérielle : généralités

Les SoC font partie d'une industrie en perpétuelle évolution, dans laquelle le temps de conception (*time-to-market*) est primordial. Dans cette optique, un SoC n'est pas conçu entièrement pour chaque nouvelle application mais utilise des blocs préexistants. Dans la conception électronique, ces blocs sont couramment appelés *IP-blocks* (pour *Intellectual Property*, car fréquemment, ils sont brevetés ou possèdent une licence d'utilisation). La méthodologie habituelle, que l'on appelle *IP reuse*, consiste alors à maximiser la réutilisation de ces IP-blocks pour la conception d'un SoC.

Les IP-blocks matériels, ou composants matériels, qui composent une plateforme matérielle peuvent généralement être catalogués en quatre groupes. La figure 2.3 montre une plateforme complète composée de ces différents composants matériels.

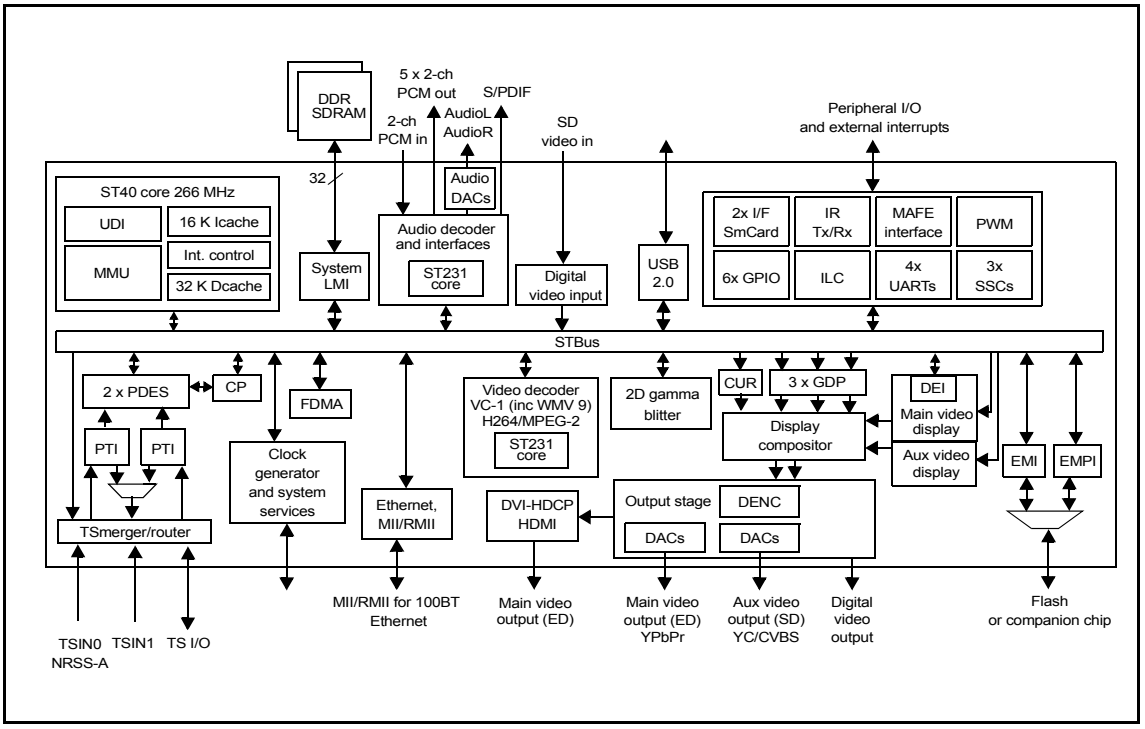


FIGURE 2.3 – Exemple de SoC orienté multimédia (STi5202), composé des quatre types de composants : processeurs, mémoires, périphériques, et réseau d'interconnexion. *Crédits : STMicroelectronics*

2.2.1.1 Processeurs

Les processeurs sont les composants qui exécutent les instructions d'un programme informatique. Ils ont l'énorme avantage d'être reprogrammables et ainsi de permettre les évolutions des applications embarquées. Les contraintes de conception des SoC amènent à privilégier des processeurs de complexité matérielle limitée, en raison de leur faible consommation d'énergie. Les processeurs qui offrent cette caractéristique suivent généralement l'ar-

chitecture *RISC* (*Reduced Instruction Set Computer*). Les principaux compétiteurs dans le marché des SoC moyens et hauts de gamme sont *ARM* (qui détient par exemple plus de 70% du marché dans la téléphonie mobile), *MIPS*, *SH*, *Power*, etc. D'autres processeurs, dits spécialisés (ou *DSP*, pour *Digital Signal Processor*), sont également souvent utilisés dans les SoC. Ils peuvent notamment proposer des jeux d'instructions optimisés pour les opérations arithmétiques complexes, ce qui les rend particulièrement attrayant dans les systèmes orientés multimédia, pour le traitement audio ou vidéo.

Dans le contexte des MP-SoC pour les systèmes embarqués, la course à la performance est principalement réalisée par l'intégration de plus de processeurs dans la plateforme. Les plateformes sont donc devenues multiprocesseurs : aujourd'hui, les plateformes industrielles peuvent embarquer une dizaine de processeurs. Les projets de recherche actuels envisagent qu'elles intègrent bientôt des centaines voire des milliers de processeurs [9].

2.2.1.2 Mémoires

Ces composants représentent le stockage du système, pour les instructions et les données des programmes. Dans un système embarqué, on trouve principalement deux niveaux de stockage, correspondant à deux usages distincts et fabriqués différemment.

Le stockage de premier niveau, ou mémoire centrale, est directement accessible par les processeurs : on dit qu'il est dans l'« espace d'adressage ». Étant surtout dévolu à héberger des données temporaires, il est surtout fait de mémoire vive (*RAM*, *Random Access Memory*), un type de mémoire volatile (c'est-à-dire dont le contenu est perdu lorsque l'alimentation électrique est éteinte).

Le second niveau de stockage ne fait pas directement parti de l'espace d'adressage, mais est accessible par l'intermédiaire d'un composant périphérique qu'il faut normalement configurer par logiciel. On appelle souvent ce niveau « stockage de masse » car il est constitué de mémoire non volatile de beaucoup plus grosse capacité que le niveau primaire. Dans les systèmes embarqués, ce niveau secondaire se matérialise souvent par de la mémoire *Flash*.

Le coût de l'espace occupé par la mémoire étant assez prohibitif, les SoC ne contiennent souvent qu'une petite partie de la mémoire centrale (de premier niveau) sur la puce elle-même. On parle alors de mémoire interne, ou embarquée, en opposition au reste de la mémoire centrale, généralement placée hors de la puce, par exemple sur la carte électronique du système embarqué, et qui est alors désigné comme externe. Le stockage secondaire est lui quasiment systématiquement externe au SoC. La figure 2.4 résume graphiquement cette organisation.

Toutefois, l'accès mémoire à la mémoire est souvent un goulot d'étranglement et la tendance actuelle consiste à maximiser la taille de la mémoire embarquée. De quelques kilooctets auparavant, les MP-SoC en contiennent maintenant plusieurs mégaoctets et les

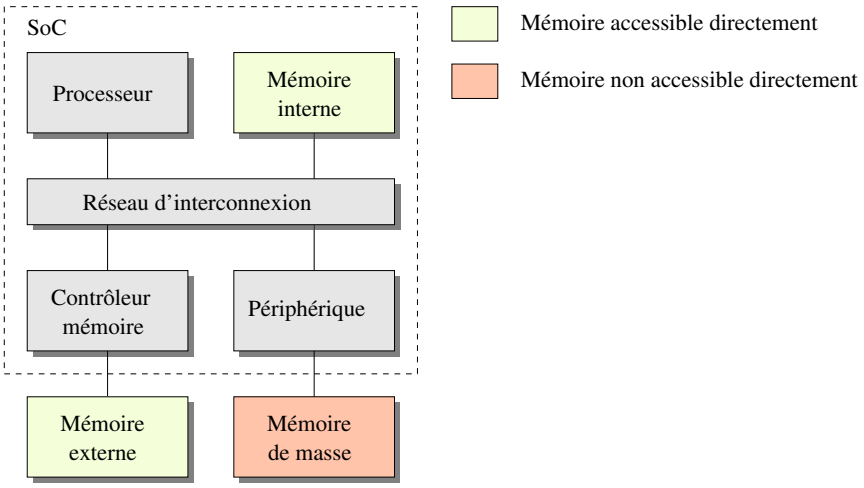


FIGURE 2.4 – Mémoire centrale (interne et externe) et stockage de masse.

technologies d'intégration futures devraient permettre d'atteindre des tailles de plusieurs centaines de méga-octets [10].

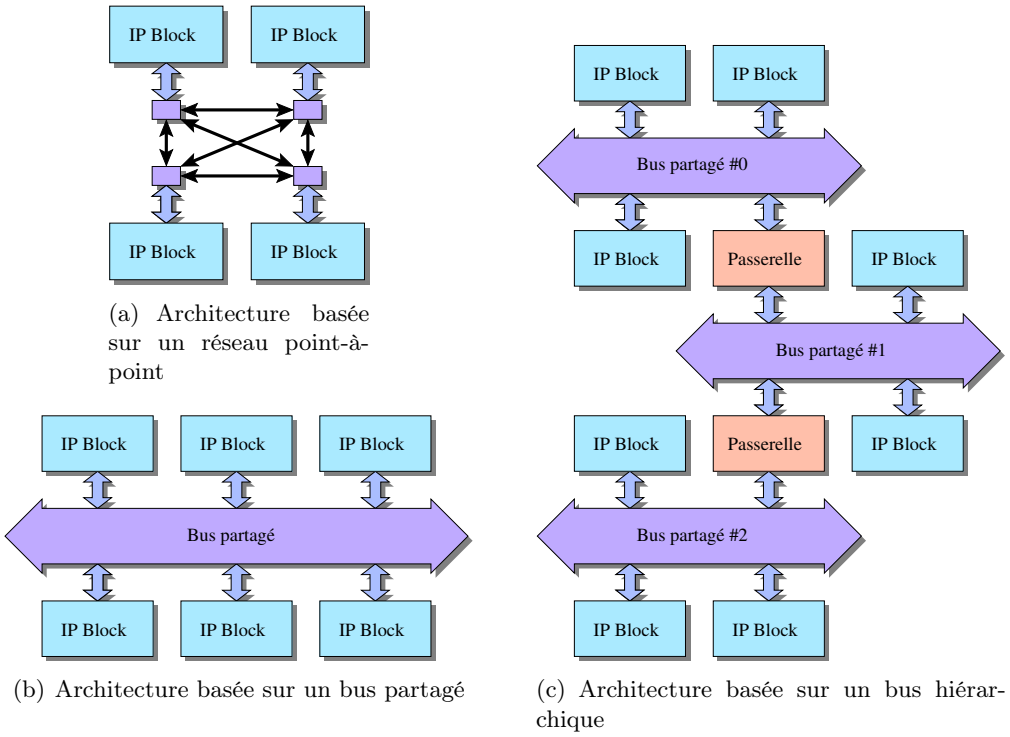
2.2.1.3 Périphériques

Ces composants servent à étendre les fonctionnalités du système. On distingue deux types de périphériques avec des objectifs bien différents. Un premier type concerne la communication avec le monde extérieur, et plus exactement avec des composants externes à la puce, tels que la mémoire externe, la gestion du temps (oscillateurs, horloges), ou des interfaces externes (ports *USB*, *Ethernet*, série, etc). Les périphériques de ce type servent alors à piloter ces dispositifs. L'autre type de périphériques concerne plutôt l'accélération de traitement de données spécifiques. Ces traitements pourraient être réalisés par des processeurs programmables, mais seraient alors excessivement lents. Dans les applications multimédia, par exemple, certaines étapes des algorithmes audio ou vidéo peuvent être accélérées par du matériel dédié.

Un périphérique est généralement accessible par une interface qu'il expose dans l'espace d'adressage de la plateforme : on dit alors que le périphérique est « adressable ». Un logiciel, par l'intermédiaire d'un processeur, peut alors le configurer et l'utiliser par de simples accès mémoire en lecture et écriture, dans la plage d'adresses du périphérique donné.

2.2.1.4 Réseaux d'interconnexion

Ces composants, un peu particuliers, servent à connecter les autres entre eux. Les processeurs, mémoires et périphériques sont alors en mesure de s'échanger des données, par un système de transactions et en respectant un certain protocole. Suivant les besoins en bande passante du système, différents modèles de réseaux d'interconnexion peuvent typiquement être utilisés, comme l'illustre la figure 2.5 :



(a) Architecture basée sur un réseau point-à-point

(b) Architecture basée sur un bus partagé

(c) Architecture basée sur un bus hiérarchique

FIGURE 2.5 – Les différents modèles de réseaux d'interconnexion traditionnels.

Réseau point-à-point Ce type de réseau d'interconnexion, aussi appelé *crossbar*, lie tous les composants du système, deux à deux. Outre la grande bande passante que peut offrir ce type de réseau, un lien étant entièrement consacré à la communication entre les deux composants qu'il connecte, le coût d'implémentation du réseau devient vite inabordable dès que l'on connecte plus qu'un petit nombre de composants.

Bus partagé Ce type de réseau d'interconnexion suit une logique inverse au crossbar. L'interconnexion des composants est réalisée par un unique ensemble de pistes. Le coût d'implémentation est donc faible mais la bande passante aussi. Puisque le bus ne supporte qu'une seule transaction à la fois, on ne peut y connecter qu'un nombre restreint de composants.

Bus hiérarchique Ce type de réseau d'interconnexion est la connexion de plusieurs bus partagés. La bande passante est donc légèrement augmentée puisqu'au même moment, une transaction par segment de bus peut être réalisée. On s'arrange généralement pour placer sur le même segment de bus les composants qui communiquent le plus entre eux.

2.2.2 Logiciel embarqué : généralités

2.2.2.1 Rôle du logiciel

Le logiciel embarqué désigne l'ensemble des programmes qui s'exécutent sur un système embarqué. On peut grossièrement distinguer deux aspects dans un logiciel embarqué, qui se révèlent néanmoins très complémentaires : la gestion du matériel, notamment des processeurs et des périphériques qui sont très dépendants de la plateforme matérielle utilisée, et l'applicatif, qui confère au système une certaine spécialisation (par exemple, du décodage audio ou vidéo pour un système orienté multimédia).

2.2.2.2 Les différents modèles de piles logicielles

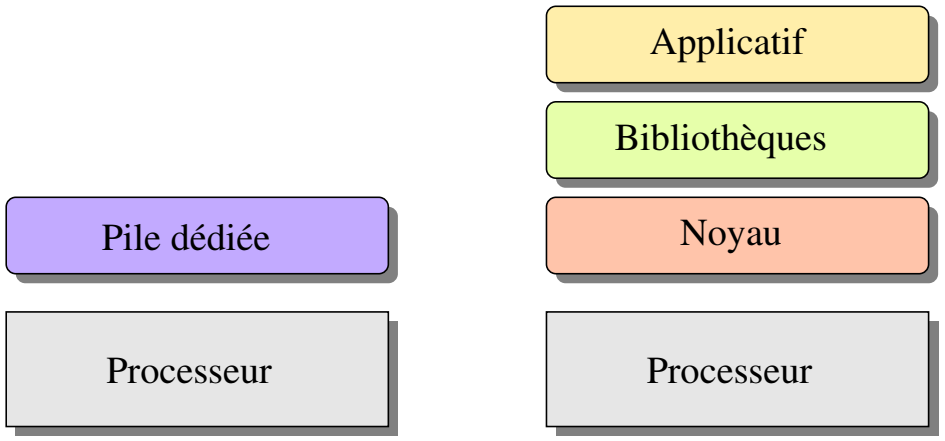
Le logiciel embarqué peut être conçu de plusieurs manières. Une première approche consiste à écrire une pile logicielle intégrée pour remplir les deux rôles évoqués ci-dessus. Cependant, à chaque nouvelle version de la plateforme matérielle, ou à chaque nouvelle spécification applicative, le concepteur devra souvent ré-implémenter le logiciel embarqué presque entièrement. On parle dans ce cas de *pile logicielle dédiée*.

Exactement comme pour le matériel, et pour les mêmes raisons de *time-to-market*, on préfère appliquer une stratégie d'*IP reuse* pour le logiciel aussi. Le logiciel embarqué suit alors un modèle en couches, qui reprend généralement les deux aspects suscités. La couche dite « noyau » représente le cœur du système logiciel, en offrant des services génériques de très bas niveau, tels que l'abstraction des processeurs et l'accès à la partie matérielle de la plateforme par les services que l'on appelle pilotes, ainsi que des services dédiés aux applications, tels que l'allocation des ressources de calcul et de mémoire, la communication inter-applications, etc. La couche dite « utilisateur » contient l'applicatif, c'est-à-dire les applications logicielles. Le noyau est complètement concerné par la stratégie de réutilisation, puisqu'il est facilement recyclable d'une plateforme matérielle à l'autre. On peut également citer une couche intermédiaire, contenant les bibliothèques utilisateur, qui est située entre le noyau et les applications et qui offre des services génériques de haut niveau aux applications, notamment en réalisant l'interface avec le noyau. C'est finalement l'association de toutes ces couches que l'on désigne communément sous le nom de *système d'exploitation* (ou *OS*, pour *Operating System*).

La figure 2.6 résume graphiquement les deux modèles de piles logicielles présentées.

2.2.3 Spécificités matérielles

Les systèmes orientés multimédia possèdent souvent des spécificités matérielles qui, comme on l'analysera par la suite, rendent les contraintes de sécurité et de flexibilité très complexes à satisfaire.



(a) Pile logicielle dédiée (b) Système d'exploitation

FIGURE 2.6 – Les deux principaux modèles de piles logicielles.

2.2.3.1 Hétérogénéité des processeurs

Dans les plateformes orientées multimédia, un système multiprocesseur est généralement hétérogène. Un petit nombre de processeurs généralistes sont assistés par des processeurs spécialisés. Ils sont souvent incompatibles entre eux, leur jeu d'instructions et la façon dont ils manipulent les données étant complètement différents. L'hétérogénéité ne concerne cependant pas seulement la spécialisation de ces différents processeurs, mais implique également d'autres dissimilitudes profondes. Les processeurs généralistes fournissent en général plusieurs niveaux de privilège, c'est-à-dire au moins un mode noyau et un mode utilisateur, et embarquent souvent une unité de mémoire virtuelle (ou *MMU*, pour *Memory Management Unit*), ou une unité de protection mémoire (ou *MPU*, pour *Memory Protection Unit*), qui proposent un mécanisme d'isolation entre les applications s'exécutant en mode utilisateur. Les processeurs spécialisés, quant à eux, ne fournissent généralement pas différents niveaux de privilège, et encore moins de protection mémoire intégrée.

2.2.3.2 Périphériques DMA

Une certaine classe de composants périphériques est équipée d'une capacité dite *DMA* (*Direct Memory Access*). Ces périphériques sont à même d'accéder la mémoire de façon autonome et peuvent donc alléger les processeurs de réaliser des transferts mémoire, coûteux en ressource de calcul. Les périphériques DMA se divisent généralement en deux classes. Les périphériques DMA de la première classe transfèrent des données d'une zone mémoire à une autre, dans le même espace d'adressage. Ce transfert est alors soit une simple copie, soit une copie modifiée, par l'intermédiaire d'un accélérateur matériel. Un tel accélérateur lit des données dans une zone mémoire, y applique une fonction, et écrit le résultat dans une autre zone mémoire. Les périphériques DMA de la deuxième classe transfèrent des données externes, non accessible dans l'espace d'adressage (par exemple, provenant d'un

périphérique de stockage externe ou d'un périphérique de réseau), vers la mémoire et vice-versa.

2.2.3.3 Interruptions

Si le principe du périphérique DMA consiste à soulager les processeurs d'effectuer eux-mêmes les transferts de mémoire, il faut cependant éviter que les processeurs soient forcés de scruter le périphérique pour savoir quand un transfert est terminé. En effet, une telle scrutation rendrait quasiment inutile la déportation du transfert sur un autre composant. On utilise alors le principe d'interruption matérielle, qui permet au périphérique de signaler un événement au processeur : typiquement, une fin de transfert. Du moment où le processeur configure le transfert jusqu'au moment où le transfert est complété, le processeur peut ainsi consacrer sa ressource de calcul à d'autres traitements.

Côté matériel, une interruption est représentée par un fil électrique, tiré entre un périphérique et un processeur. En temps normal, ce fil transporte la valeur logique 0. Lorsque le périphérique veut signaler un événement, il passe la valeur logique du fil à 1. Le processeur, qui reçoit cette ligne d'interruption sur son interface externe, part alors en exception. Le processeur offre également la possibilité de masquer les interruptions, au quel cas, à la réception d'une interruption, il ne partira pas en exception. Sachant qu'un processeur ne propose souvent qu'un nombre très restreint d'entrées d'interruption sur son interface externe, on utilise généralement un contrôleur d'interruptions (ou *ICU*, pour *Interrupt Controller Unit*). Comme illustré sur la figure 2.7, ce contrôleur prend en entrée plusieurs lignes d'interruption, provenant de plusieurs périphériques, et présente en sortie une seule ligne d'interruption, reliée au processeur. De la même façon que le processeur, l'ICU offre aussi la capacité de masquer les interruptions, globalement ou individuellement.

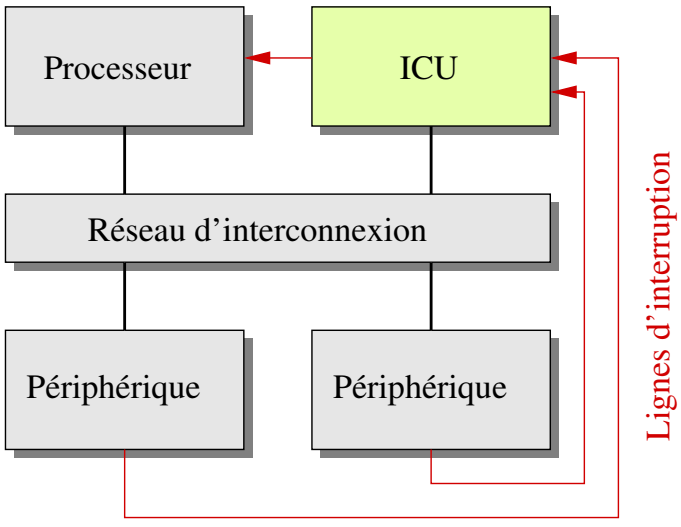


FIGURE 2.7 – Lignes d'interruption partant de périphériques et concentrées par une ICU, avant d'atteindre le processeur.

Côté logiciel, lorsqu'une interruption non masquée survient sur un processeur, ce dernier interrompt l'exécution du programme courant et commence l'exécution d'un programme particulier, le gestionnaire d'interruption. Ce gestionnaire d'interruption s'adresse à l'ICU pour déterminer quelle ligne d'interruption est active, donc quel périphérique l'a levée, et appelle en conséquence une routine d'interruption. Cette routine d'interruption est généralement fournie par le pilote logiciel du périphérique ayant provoqué l'interruption. Elle est alors chargée de traiter l'interruption, c'est-à-dire au moins de l'acquitter pour que la ligne d'interruption associée se baisse. A la fin de ce petit traitement, le processeur peut reprendre l'exécution normale du programme précédemment interrompu.

Dans la suite du document, on s'autorisera l'abus de langage suivant : on dira qu'une ligne d'interruption est reliée à un processeur, même si elle est en réalité reliée à une ICU, adjacente au processeur donné.

2.2.3.4 Mémoire partagée

Dans ce type de plateforme, toute la mémoire est exposée par un unique espace d'adressage. N'importe quel initiateur de la plateforme, c'est-à-dire n'importe quel composant matériel capable d'émettre des requêtes mémoire, tel qu'un processeur ou un périphérique DMA, est alors théoriquement en mesure d'accéder à toute la mémoire disponible ainsi qu'à tous les périphériques adressables. Ces composants mémoire ou périphériques, à qui les initiateurs peuvent s'adresser, sont ainsi appelés cibles. On notera qu'un périphérique DMA cumule souvent les rôles d'initiateur et de cible : un processeur programmable s'adressera à lui pour configurer un transfert mémoire en passant par son interface cible, et le périphérique DMA réalisera le transfert de façon indépendante grâce à son interface initiateur.

2.2.4 Caractéristiques logicielles

Le logiciel embarqué possède une importante responsabilité dans le fait de pouvoir concilier sécurité et flexibilité au sein d'un même système embarqué, puisque c'est lui qui pilote ce système et le rend fonctionnel. Toutefois, comme le montre la figure 2.8 et comme nous allons l'expliquer dans un premier temps, avant de proposer une analyse plus globale, les différents modèles de pile logicielle peinent à assurer cette conciliation.

2.2.4.1 Pile logicielle dédiée

Par sa simplicité, ce modèle intégré a souvent l'avantage de pouvoir offrir une sécurité maximum. Son fonctionnement peut effectivement être garanti, grâce, par exemple, à des techniques de preuves formelles. En revanche, pour conserver cette propriété, il est rarement envisageable qu'une pile logicielle dédiée puisse réaliser plus qu'une tâche, ce qui se révèle insuffisant par rapport à la multitude de services requis.

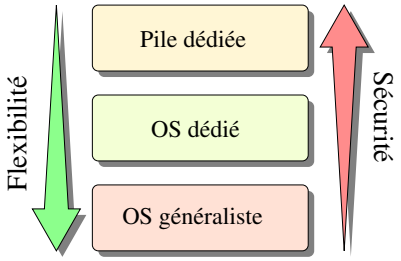


FIGURE 2.8 – Potentiels de flexibilité et de sécurité offerts par les modèles de piles logicielles standards.

2.2.4.2 Pile logicielle basée sur un système d'exploitation dédié

Il existe une classe de systèmes d'exploitation dont le support pour le matériel est réduit mais qui offrent au moins un environnement basique pour la couche applicative qui tourne au-dessus d'eux. Leur empreinte est typiquement assez limitée en termes de consommation mémoire, voire en termes de ressources de calcul si l'OS possède un support dit temps réel (ou *RT*, pour *Real-Time*), c'est-à-dire qu'il est en mesure de borner sa propre utilisation de la ressource de calcul.

La relative simplicité de ce type d'OS dédié n'autorise toutefois pas l'utilisation méthodique de techniques de preuves formelles, ou bien de manière très localisée, par exemple pour la vérification des interfaces uniquement. En outre, il arrive fréquemment que cette classe d'OS ne fournisse aucun mécanisme de protection pour la partie applicative.

La flexibilité offerte est généralement assez réduite, et ne permet pas de répondre aux besoins concernant les fonctionnalités, l'évolutivité et la haute connectivité.

2.2.4.3 Pile logicielle basée sur un système d'exploitation généraliste

Il existe enfin des OS généralistes que l'on peut également qualifier de "riches en fonctionnalités" et qui se conforment parfaitement avec les objectifs de connectivité et de flexibilité. Ils parviennent à atteindre un excellent niveau de réutilisation, ce qui leur permet d'être portables sur de nombreuses architectures matérielles et de supporter un grand nombre de périphériques (en 2008, par exemple, 50% du code source de *Linux* était consacré au pilotes de périphériques [11]). Ils fournissent généralement des environnements de développement bien répandus, et plus globalement une infrastructure complète pour l'applicatif qui expose entre autres des services de sécurité avancés, à des fins de protection et de fiabilité.

Néanmoins, cette puissance et cette versatilité s'accompagnent en contre partie d'une complexité interne significative, quasiment impossible à vérifier. De plus, même si potentiellement moins avéré dans le cadre d'OS propriétaire, ce type d'OS est généralement le résultat de collaborations massives, ce qui rend le code source très difficile à contrôler. L'évolution est de toute façon bien trop rapide pour être en mesure de maintenir des

versions parallèles sécurisées. Ce modèle de pile logicielle est donc considéré comme peu sûr.

2.2.5 Mise en perspective

Toute solution de sécurité repose sur ce que l'on appelle communément le *Trusted Computing Base* (*TCB*). Comme le définit [12], le TCB inclut l'ensemble des éléments matériels et logiciels d'un système qui sont responsables du support de la politique de sécurité et de l'isolation des objets sur lesquels la protection se base. Dans un système mono-processeur généraliste faisant fonctionner un système d'exploitation, le TCB peut typiquement être composé du noyau de l'OS, pour la partie logicielle, et des deux éléments matériels suivants : la MMU et le fait que le processeur offre plusieurs niveaux de privilège, respectivement pour permettre le partage de la mémoire et le partage local de la ressource de calcul.

Dans le cadre de systèmes multiprocesseurs, on s'attend à ce que le TCB couvre l'ensemble des processeurs, capables d'adresser la mémoire, afin d'assurer un contrôle complet et cohérent. Or, l'hétérogénéité des processeurs rend intrinsèquement impossible la distribution d'un tel TCB : les différents processeurs n'offrent pas tous une MMU, et ne disposent pas tous de plusieurs niveaux de privilège. En outre, côté logiciel, hormis quelques projets de recherche actuellement en cours [13, 14, 15], aucun OS existant ne supporte l'exécution sur une plateforme multi-processeur hétérogène. En somme, les caractéristiques matérielles de la plateforme ne permettent pas de déployer une unique pile logicielle. Pour exploiter la plateforme, il faudra forcément exécuter plusieurs piles logicielles autonomes, réparties sur les différents types de processeur disponibles.

D'autre part, aucun des modèles de piles logicielles autonomes existants n'est conçu pour concilier les contraintes – contradictoires – de sécurité et de flexibilité. Certains sont trop complexes et insuffisamment contrôlables pour servir de support de confiance aux sous-systèmes sécurisés précédemment évoqués. D'autres sont trop rigides et incapables de suivre les évolutions rapides qu'impose le marché. Pourtant, ces différents modèles trouvent tous leur utilité, dans les systèmes orientés multimédia actuels.

En effet, une pile dédiée pourra trouver place au sein d'un périphérique d'accélération intelligent, pouvant manipuler du contenu protégé. Un OS temps réel pourra être utilisé pour le décodage du contenu multimédia : la diffusion correcte d'un film par exemple, exige que des échantillons de sons et d'images soient décodés à intervalles réguliers et constants. Enfin, les OS généralistes pourront quant à eux fournir la flexibilité demandée par les utilisateurs.

Aujourd'hui, ces OS généralistes deviennent d'ailleurs un vrai argument commercial. On peut effectivement citer l'utilisation de *Mac OS* dans les produits *Apple iPhone*, ou plus récemment, l'expansion de *Android*, propulsé par *Google*, comme OS standard dans

plusieurs gammes de systèmes orientés multimédia. Dans le monde des set-top box, l'utilisation de Linux est devenue courante depuis quelques années.

La technique pour finalement concilier sécurité et flexibilité, ainsi qu'exploiter l'hétérogénéité des processeurs, consiste alors à exécuter ces différents types de piles logicielles autonomes sur la même plateforme matérielle et de façon simultanée.

2.3 Co-hébergement

Le concept de co-hébergement, ou *co-hosting*, permet de partager une plateforme matérielle entre plusieurs piles logicielles autonomes. Ces piles logicielles – OS généralistes, OS dédiés ou piles dédiées – sont considérées comme autonomes, car sur des plateformes traditionnelles – composées d'un ou plusieurs processeurs homogènes (symétriques) –, elles possèdent le contrôle complet de la plateforme. Dans ce cas, elles font généralement partie du TCB de la plateforme.

Comme expliqué dans la suite, les techniques actuelles de co-hébergement souffrent cependant de certaines limitations, en termes de sécurité ou de flexibilité.

2.3.1 Mémoire virtuelle

Le mécanisme de cloisonnement le plus répandu est incontestablement la mémoire virtuelle. Ce mécanisme permet en effet d'isoler des applications entre elles, en fournissant à chacune un espace d'adressage virtuel propre. La traduction de cet espace virtuel vers l'espace d'adressage physique de la plateforme est assurée par la MMU, qui est pilotée par un système d'exploitation. On est alors assuré qu'une application ne peut accéder illégalement aux segments mémoire appartenant à une autre application.

La MMU a la particularité importante d'être locale à un processeur, car traditionnellement intégrée physiquement à ce dernier. Cette caractéristique est un héritage fort du modèle d'architecture mono-processeur, dans lequel une unique pile logicielle autonome, généralement un OS, implémente la partie logicielle du TCB et couvre la plateforme matérielle. Cela signifie que cet OS s'exécute sur tous les processeurs de la plateforme et qu'il en contrôle alors toutes les MMU. Le partage de la mémoire physique entre les différentes applications est donc cohérent.

Si le mécanisme de mémoire virtuelle est performant pour isoler différentes applications logicielles s'exécutant au-dessus d'une unique base de confiance, il ne s'adapte pas du tout à une situation où plusieurs piles logicielles autonomes, par exemple plusieurs OS indépendants, fonctionnent simultanément sur la plateforme. En effet, un OS n'a absolument aucun pouvoir sur la MMU d'un processeur qu'il ne contrôle pas. Si plusieurs OS s'exécutent sur une plateforme, partageant les processeurs de manière exclusive, aucun d'entre eux ne peut contrôler entièrement l'espace adressable partagé en utilisant le mécanisme de MMU. En

outre, dans le cadre de plateformes hétérogènes, beaucoup de processeurs, voire la plupart, ne sont pas équipés de MMU.

La technique de mémoire virtuelle, via l'utilisation de MMU, est donc incapable de résoudre à elle seule la problématique de co-hébergement. On arrive aux mêmes conclusions avec l'utilisation d'une MPU, qui fournit un service de protection mémoire similaire à celui d'une MMU mais sans virtualisation de la mémoire (les applications fonctionnent dans l'espace d'adressage physique).

2.3.2 Partage statique des ressources

La technique la plus usitée pour résoudre le problème de co-hébergement consiste à partager de façon statique la plateforme, ou plus exactement les ressources de la plateforme.

Dans une telle approche, c'est à la conception que toutes les ressources sont attribuées définitivement aux différentes piles logicielles autonomes en présence : processeurs, zones mémoires, et périphériques adressables.

Dans le cas des processeurs, ce partage statique est intrinsèquement un partitionnement strict. C'est en effet l'hétérogénéité des processeurs qui dicte ce partitionnement entre les différentes piles logicielles autonomes. Partant de l'hypothèse que ces piles autonomes ne supportent pas l'hétérogénéité des processeurs, on ne pourra attribuer à une certaine pile autonome qu'un ou plusieurs processeurs du même type. De leur côté, les processeurs ne peuvent pas être partagés entre plusieurs piles autonomes car le mode le plus privilégié d'un processeur ne supporte toujours qu'un seul contexte d'exécution. L'attribution d'un ou plusieurs processeurs à une pile autonome est alors exclusive. En somme, le partitionnement exclusif des processeurs d'une plateforme hétérogène entre plusieurs piles logicielles autonomes est statique, dans le sens qu'il ne peut pas être modifié dynamiquement à l'exécution. Ici, et dans le reste du document, nous nous intéressons donc uniquement au partage de l'espace d'adressage, c'est-à-dire au partage de la mémoire et des périphériques adressables.

Dans cette approche, le découpage de l'espace d'adressage est uniquement spécifié théoriquement. Il n'existe notamment aucun mécanisme matériel de type MMU ou MPU, comme nous pouvons le constater par l'illustration 2.9, garantissant qu'une pile reste dans les limites fixées. Elle peut alors, par erreur ou volontairement si elle devient vérolée, accéder à des ressources qui ne lui étaient pas attribuées.

Il existe tout de même certaines manières de sécuriser un tel partage statique.

2.3.2.1 Sécurité

Une première méthode s'appuie sur l'architecture du système. Actuellement, les plateformes industrielles utilisent souvent une architecture basée sur un bus hiérarchique.

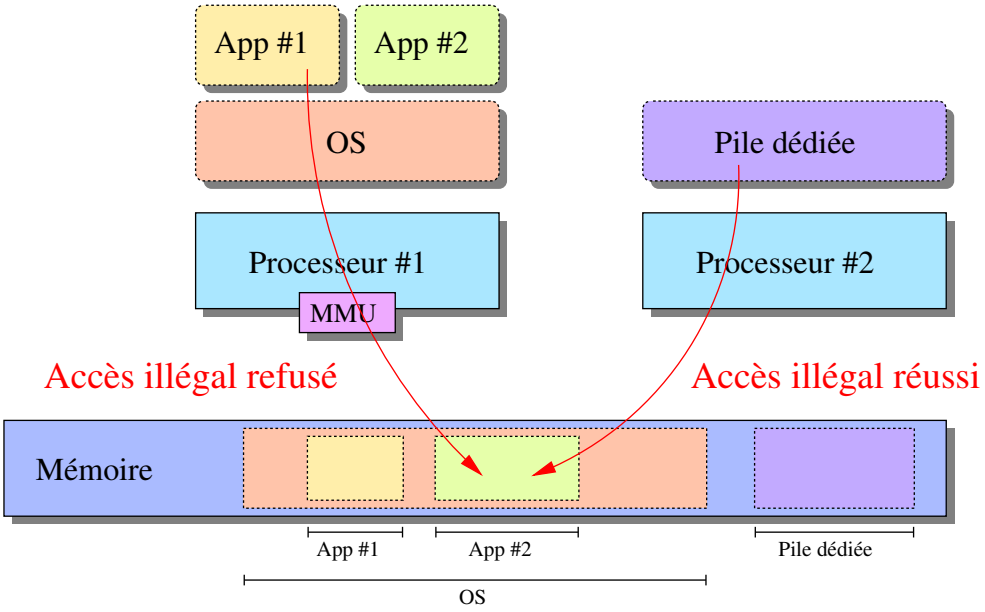


FIGURE 2.9 – Exemple de co-hébergement : une MMU peut isoler les applications s'exécutant sur le processeur dans lequel elle est intégrée, mais elle reste inopérante pour faire respecter le partitionnement des ressources aux autres processeurs.

Malgré une politique de mémoire partagée, il devient possible de restreindre l'accès à certaines ressources, c'est-à-dire certaines zones de l'espace d'adressage, et par effet de bord, d'en sécuriser l'accès. Un banc mémoire ou un périphérique par exemple peuvent n'être utilisables que par un certain processeur car ils ne sont physiquement pas accessibles par les autres processeurs, le réseau d'interconnexion ne proposant pas de chemin pour l'atteindre. Cette technique n'est cependant qu'un détournement du principe de bus hiérarchique, et ne peut pas être utilisée de façon systématique.

Une autre méthode consiste à introduire des filtres d'accès, le plus souvent intégrés au réseau d'interconnexion ou placés comme des composants autonomes devant certains bancs mémoire. De tels filtres peuvent définir des associations de composants autorisés à communiquer entre eux. On peut ainsi garantir que le logiciel s'exécutant sur un certain processeur n'accède qu'aux composants qui lui ont été ouverts. Un autre type de filtre, souvent placé uniquement devant la mémoire externe, peut définir des zones mémoires que certains composants initiateurs de la plateforme ont le droit d'adresser. Le nombre de zones reste généralement assez limité, tout comme leurs possibilités de reconfiguration. Dans le cadre de systèmes orientés multimédia, on utilise donc souvent ce mécanisme pour confiner l'OS généraliste, plus sujet à d'éventuelles corruptions.

2.3.2.2 Flexibilité

De par sa nature statique, ce type de partage manque malheureusement de flexibilité, ce qui s'observe nettement sur deux aspects.

Le système peine à s'adapter aux besoins temps réel des piles logicielles autonomes, dynamiques par essence. Il est effectivement très difficile, voire impossible, pour une pile autonome d'obtenir plus de ressources que celles qui lui ont été attribuées, ou d'en rendre si elle n'en a pas l'usage à un moment donné. Le besoin de ressources d'une pile logicielle quelconque au cours du temps est pourtant assez variable, particulièrement au niveau de la consommation mémoire, mais aussi de l'utilisation des périphériques.

Les mécanismes de sécurité existants, tels que les filtres d'accès, ont une granularité au niveau composant, ce qui a une conséquence sur les possibilités de partage des ressources. Un même processeur ne peut, par exemple, pas être partagé entre des piles logicielles (applications) pour lesquelles on voudrait différencier les droits d'accès, puisqu'il est vu par les filtres comme une seule entité. Tandis que cela peut facilement être contourné par les processeurs équipés de MMU ou MPU, puisqu'ils sont à même de pouvoir réaliser – localement – une isolation entre applications, les processeurs non équipés de MMU/MPU en sont incapables. Le problème se pose également pour le partage des périphériques initiateurs, de type DMA par exemple. S'il est partagé par plusieurs piles logicielles, un tel périphérique, puisqu'il est vu comme une seule entité par les filtres, ne peut obtenir que les droits d'accès définis pour la pile qui en a le contrôle à un instant donné. Même si les filtres peuvent éventuellement être reconfigurés entre deux utilisations du périphérique, par des piles différentes, le problème reste entier si le partage n'est pas exclusif dans le temps mais fortement entrelacé.

2.4 Conclusion

La démocratisation des systèmes embarqués auprès du grand public, et notamment des systèmes orientés multimédia, entraine des exigences contradictoires. La protection de données n'a jamais été autant d'actualité, puisque la plupart des données qui transitent dans ce type de systèmes sont sensibles. D'un autre côté, les utilisateurs sont demandeurs de plus en plus de flexibilité afin de tirer au maximum profit de leurs appareils.

Les défis techniques pour résoudre cette double contrainte sont nombreux. Les caractéristiques matérielles des plateformes actuelles empêchent l'utilisation d'une couche logicielle de confiance unique, et de surcroît, aucun des différents modèles de piles logicielles existants n'est capable de répondre seul à toutes les exigences, en termes de sécurité et de flexibilité.

Le concept de co-hébergement, c'est-à-dire l'exécution simultanée de plusieurs piles logicielles autonomes, semble être une piste pour concilier ces besoins incompatibles. Cependant, les mécanismes de sécurité traditionnels sont uniquement adaptés à des modèles mono-TCB, tandis que les mécanismes plus récents, ajoutés *a posteriori* sur ces plateformes déjà trop complexes, n'offrent qu'une sécurité rigide et souvent incomplète, bien insuffisante pour répondre aux besoins futurs.

L'objectif de cette thèse consiste donc à proposer une nouvelle architecture qui puisse

assurer un co-hébergement sécurisé et flexible, sur des systèmes multiprocesseurs intégrés sur puce hétérogènes. Cet objectif peut être formulé par les questions suivantes, auxquelles on s'efforcera de donner une réponse :

- *Quelle architecture peut être cohérente avec des plateformes multiprocesseurs hétérogènes ?*
- *Cette architecture peut-elle fournir une garantie de partage sécurisé de la mémoire ?*
- *Dans quel cadre et avec quelles restrictions, l'architecture peut-elle partager de façon sécurisée les composants initiateurs, de type processeurs ou périphériques DMA, ainsi que les périphériques cibles (autres que la mémoire) ?*
- *Quel est le coût d'une telle architecture, aussi bien au point de vue logiciel que matériel ?*
- *Est-il possible d'intégrer cette architecture de sécurité dans différents réseaux d'interconnexion de façon à la rendre pérenne pour les architectures matérielles futures ?*

Chapitre 3

État de l'art

Sommaire

Ce chapitre présente l'état de l'art en deux parties : d'une part, nous décrivons les techniques de *virtualisation*, qui proposent des solutions cherchant plutôt à traiter le problème de co-hébergement. D'autre part, nous montrons d'autres solutions, qui se focalisent sur la sécurisation globale d'architectures de systèmes intégrés sur puce. Ces deux parties abordent chacune un sous-ensemble du problème que nous cherchons à résoudre.

3.1 Virtualisation

La virtualisation, en permettant l'exécution simultanée de plusieurs piles logicielles autonomes sur la même plateforme, et le partage des ressources processeur, mémoire et périphérique, répond en partie à la problématique de co-hébergement.

3.1.1 Introduction

3.1.1.1 Historique

Inventée au début des années 70 et visant au départ le marché des ordinateurs centraux (serveurs) [16], la virtualisation cherchait notamment la consolidation de serveurs grâce à son fort potentiel de réutilisation du matériel. Un serveur virtualisé peut en effet offrir plusieurs services fonctionnant de façon concurrente et sécurisée dans différents systèmes d'exploitation (*OS*) ou dans plusieurs instances du même OS.

Plus récemment, la virtualisation est apparue dans le marché des ordinateurs de bureaux et est, par exemple, très prisée pour sa capacité à exécuter sur une même machine physique différentes applications logicielles développées pour différents systèmes d'exploitation.

Enfin, il se dessine aujourd'hui une utilisation évidente de la virtualisation dans le marché de l'embarqué pour répondre à des problématiques de sécurité. Un OS considéré comme non fiable peut fonctionner dans un système intégré sur puce sous la supervision d'un logiciel de confiance.

3.1.1.2 Concepts

La virtualisation a introduit deux concepts. Une *machine virtuelle* (ou *VM*, pour *Virtual Machine*) est le conteneur dans lequel s'exécute une pile logicielle autonome : typiquement un OS, que l'on désigne alors comme *OS invité*. Le gestionnaire des machines virtuelles (ou *VMM*, pour *Virtual Machine Manager*) est la couche logicielle de confiance qui fournit l'environnement dans lequel différentes machines virtuelles s'exécutent.

Popek et Goldberg [17] ont été les premiers à poser les conditions nécessaires formelles concernant le VMM (aujourd'hui popularisé sous le nom d'*hyperviseur*), qui doit alors respecter les trois principes suivants :

- Exécution équivalente : les programmes fonctionnant dans une machine virtuelle s'exécutent de façon identique à leur exécution native, à l'exception de différences d'ordre temporel ou de disponibilité des ressources.
- Performance : toutes les instructions « inoffensives » doivent être exécutées par le matériel directement, sans l'intervention de l'hyperviseur.
- Sûreté : l'hyperviseur contrôle toutes les ressources matérielles.

3.1.1.3 Caractéristiques de principe

La virtualisation s'appuie sur deux caractéristiques qu'un processeur doit posséder pour être *virtualisable* :

- Le processeur doit offrir au moins deux niveaux de privilège (par exemple, *utilisateur* et *noyau*). Les instructions privilégiées ne peuvent être exécutées que dans le mode le plus privilégié et sont piégées (*trapped*) si elles le sont dans un mode moins privilégié.

– Toutes les instructions dites sensibles, c'est-à-dire qui ont accès en lecture ou écriture à des ressources de configuration du système, doivent être privilégiées.

Un tel processeur virtualisable peut alors exécuter l'hyperviseur dans le mode le plus privilégié et les machines virtuelles dans un mode moins privilégié. Ces dernières sont isolées entre elles puisque l'exécution d'une instruction privilégiée, susceptible de modifier l'état global du système, est automatiquement piégée par l'hyperviseur, et le plus souvent émulée (technique de *trap-and-emulate*).

Popek et Goldberg sont moins spécifiques concernant l'accès à la mémoire. Ils font effectivement l'hypothèse que les accès mémoire des machines virtuelles sont relogés, c'est-à-dire subissent une modification, pour accéder uniquement à l'espace mémoire qui leur a été alloué. Concrètement, cette hypothèse impose l'utilisation d'une unité de mémoire virtuelle (*MMU*), à même de pouvoir effectuer cette relocation.

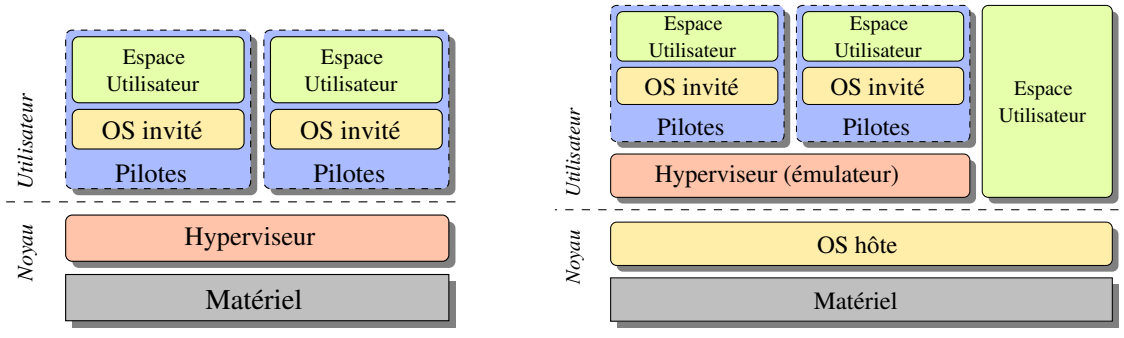
En conséquence, on peut dès à présent noter que la virtualisation ne peut fonctionner que sur des processeurs présentant plusieurs niveaux de privilège et équipés d'une MMU.

3.1.2 Stratégies de virtualisation

La virtualisation existe sous de nombreuses formes, mais on peut distinguer quatre méthodes principales, indépendamment classifiées en deux groupes : selon le type d'hyperviseur et le type d'interaction entre l'hyperviseur et les machines virtuelles.

3.1.2.1 Types d'hyperviseur

Goldberg [18] a défini deux types d'hyperviseur, dont découle la façon d'empiler les différentes couches logicielles. La figure 3.1 expose graphiquement ces types d'hyperviseur.



(a) Hyperviseur de type 1

(b) Hyperviseur de type 2

FIGURE 3.1 – Les deux principaux types d'hyperviseur.

Un *hyperviseur de type 1* est comparable à un noyau de système d'exploitation léger, et optimisé pour gérer les accès des OS invités à l'architecture matérielle sous-jacente. Dans ce schéma de virtualisation, l'hyperviseur fonctionne en mode noyau du processeur

cible, tandis que les machines virtuelles sont « poussées » entièrement en mode utilisateur. Proposée depuis quelques années dans le monde bureautique, par exemple avec *Xen* [19], cette solution est aujourd'hui la plus utilisée dans les systèmes embarqués.

Un *hyperviseur de type 2* prend la forme d'un logiciel applicatif s'exécutant sur un OS hôte. Le logiciel virtualise le matériel si les OS invités l'utilisent directement (instructions exécutées par le processeur, utilisation d'une partie de la mémoire vive, etc), ou bien l'émule si le matériel cible n'est pas le même que celui sous-jacent (émulation d'un processeur ou de périphériques différents). Dans ce schéma de virtualisation, l'hyperviseur et les machines virtuelles fonctionnent tous comme des applications au-dessus de l'OS hôte, donc en mode utilisateur du processeur. Cette solution est très flexible, puisqu'elle permet d'exécuter n'importe quel OS invité non modifié, sur n'importe quelle architecture matérielle cible, grâce à la possibilité d'émulation. En revanche, elle engendre un coût en performance très élevé (décuplé en cas d'émulation du processeur cible, puisqu'il faut émuler l'exécution de toutes les instructions), qu'il est impossible d'assumer dans les systèmes embarqués. Elle reste donc cantonnée à une utilisation bureautique, par exemple avec les solutions *QEMU* [20] ou *Bochs* [21], et ne sera pas étudiée plus profondément dans ce chapitre.

3.1.2.2 Types d'interaction

Les deux méthodes restantes, orthogonales aux deux premières, concernent la façon dont les OS invités interagissent avec l'hyperviseur.

Dans le cas de la *virtualisation complète* (plus connue sous le nom anglais de *Full Virtualization*), une machine virtuelle doit simuler suffisamment de matériel pour autoriser l'exécution d'un OS invité non modifié. Dans ce cas, l'OS invité n'a pas conscience d'être exécuté dans une machine virtuelle.

Dans le cas de la *paravirtualisation*, un OS invité a conscience d'être exécuté dans une machine virtuelle, au-dessus d'un hyperviseur. Il interagit alors directement avec l'hyperviseur, qui lui propose une panoplie de services au travers d'une interface spéciale. Cette solution permet une grande simplification de l'hyperviseur, et les machines virtuelles sont à même de fonctionner à un niveau de performance proche du natif. Cependant, cela implique une modification de l'OS invité, qui doit en effet être porté explicitement sur un hyperviseur paravirtualisé, ce qui n'est souvent pas réalisable pour les OS propriétaires dont le code source n'est pas disponible.

Cette dernière méthode, combinée avec un hyperviseur de type 1, est très appréciée dans les systèmes embarqués, en raison de ses performances. C'est d'ailleurs cette combinaison des deux méthodes qui est proposée par les principales solutions gratuites ou commerciales telles que *OKL4* [22], *VMWare MVP* [23] (anciennement *TRANGO*) ou *VLX* [24].

3.1.3 Mise en œuvre

La virtualisation d'un système complet signifie plus précisément la virtualisation des composants matériels qui le compose, c'est-à-dire des processeurs, de la mémoire et enfin des périphériques.

3.1.3.1 Processeur

Comme évoqué ci-dessus, et en cas de virtualisation complète, les instructions sensibles exécutées par une machine virtuelle sont en fait piégées et émulées par l'hyperviseur. Si une architecture comporte des instructions sensibles non privilégiées, comme c'est le cas pour l'architecture *x86*, ces dernières ne seront pas automatiquement piégées car elles peuvent être directement exécutées en mode utilisateur. L'hyperviseur doit alors réaliser une inspection préalable des instructions appartenant à une machine virtuelle avant leur exécution, grâce à la technique de *Binary Translation* [25], afin de remplacer explicitement les instructions sensibles par des pièges, qui seront ainsi émulés par l'hyperviseur.

Dans le cas de la paravirtualisation, l'utilisation d'instructions sensibles est prohibée, par la modification directe du code source des machines virtuelles, et remplacée par des appels directs à l'hyperviseur pour émulation.

Concernant le partage des ressources processeur, l'hyperviseur, fonctionnant en mode privilégié, est à même de configurer des périphériques d'horloge (*timers*) et d'en recevoir les interruptions associées. De la même façon que pour un noyau de système d'exploitation multi-tâche traditionnel, ce mécanisme lui permet de reprendre la main sur le processeur virtualisé et ainsi d'organiser l'ordonnancement des différentes machines virtuelles.

3.1.3.2 Mémoire

Lors de son exécution native sur un processeur, un OS fait souvent usage de la MMU afin de partitionner la mémoire disponible et ainsi d'isoler les applications qu'il exécute. Dans le cas où cet OS est virtualisé, l'utilisation de cette MMU est émulée, mais surtout tracée par l'hyperviseur qui maintient alors des tables de page « fantômes » (*Shadow Page Tables*) pour chaque machine virtuelle. Il peut ainsi configurer la MMU physiquement présente dans le processeur afin d'effectuer le partitionnement mémoire entre tous les OS invités, et entre les applications fonctionnant au-dessus de ces OS.

Dans le cas de la paravirtualisation, les OS invités se servent de l'interface proposée par l'hyperviseur pour demander le partitionnement mémoire local dont ils ont besoin.

3.1.3.3 Périphériques

Comme l'explique [26], la virtualisation propose généralement aux machines virtuelles trois techniques d'accès aux périphériques : l'*émulation*, la *paravirtualisation* et l'*assignation*

directe. Tandis que la seconde méthode s'intègre naturellement dans la stratégie de paravirtualisation présentée précédemment, les deux autres méthodes sont plutôt utilisées dans une stratégie de virtualisation complète.

Une quatrième méthode, combinaison de la paravirtualisation et de l'assignation directe, peut également être citée.

Virtualisation par émulation L'émulation d'un périphérique consiste à proposer un modèle de ce périphérique réalisé entièrement en logiciel. Concrètement, ce modèle s'expose dans l'espace d'adressage via une plage d'adresses factice pour laquelle tous les accès effectués par le pilote de périphérique d'un OS invité sont piégés et émulés. Le pilote de périphérique n'a pas besoin d'être modifié puisqu'il n'a pas conscience d'interagir avec un périphérique virtuel. L'hyperviseur doit néanmoins disposer d'un mécanisme pour injecter des interruptions dans l'OS invité aux moments appropriés, en se faisant passer pour le périphérique émulé. Un contrôleur d'interruptions, *ICU*[1], doit donc aussi être émulé et les accès de l'OS invité vers l'ICU doivent être piégés. La figure 3.2 illustre cette technique d'émulation.

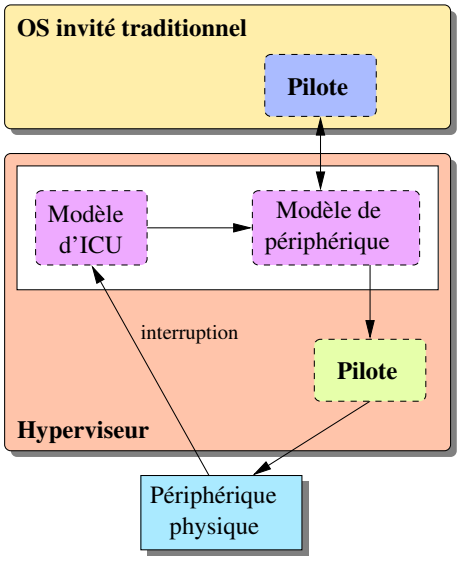


FIGURE 3.2 – Technique d'émulation de périphérique.

Cette technique d'émulation facilite le partage de périphériques car plusieurs instances du modèle de périphérique peuvent être exposées à plusieurs OS invités. Ces instances accèdent finalement à un seul périphérique matériel physique grâce à un mécanisme de partage intégré à l'hyperviseur. On peut noter que le périphérique physique est potentiellement très différent du périphérique virtuel : l'OS invité peut, par exemple, voir un périphérique de disque dur à la norme *IDE* tandis que la plateforme matérielle contient réellement un périphérique de disque dur à la norme *SATA*. L'émulation peut supporter

1. Dans le monde des ordinateurs, l'ICU est souvent connue sous le nom de *PIC* ou encore *APIC*, pour *Advanced Programmable Interrupt Controller*, mais désigne fondamentalement le même type de composant.

un grand nombre d'OS invités différents, dès lors que le modèle de périphérique choisi représente un périphérique réel connu, et donc que tous les OS invités possèdent le pilote de périphérique approprié. En revanche, le coût de l'émulation est très élevé puisqu'il faut piéger et émuler toutes les interactions entres les pilotes de périphériques des OS invités et les instances du périphérique virtuel, ce qui provoque de nombreux changements de contextes entre les OS invités et l'hyperviseur. De plus, le modèle de périphérique virtuel doit être aussi précis que sa version réelle, parfois à la révision du matériel près (ce qui inclut éventuellement de devoir émuler des bogues).

Virtualisation par paravirtualisation La paravirtualisation d'un périphérique permet à un OS invité modifié d'interagir avec l'hyperviseur au travers d'un ensemble d'interfaces d'entrée/sortie, chacune spécifique à un certain type de service haut niveau (accès réseau, système de fichier, etc.), plutôt que de s'adresser directement à un périphérique particulier. La figure 3.3 schématise ce comportement. Le mécanisme d'interruption est quant à lui généralement remplacé par un mécanisme d'événement logiciel, tel que la technique de fonction de rappel (*callback*).

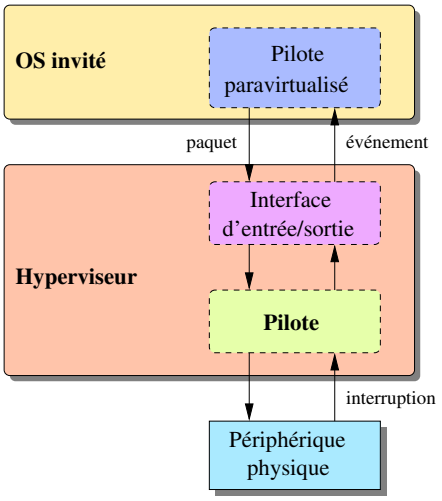


FIGURE 3.3 – Technique de paravirtualisation de périphérique.

La paravirtualisation des périphériques offre de bonnes performances puisque le nombre d'interactions (et donc de transitions) entre l'hyperviseur et l'OS invité est réduit au minimum nécessaire. Elle permet également le partage de n'importe quel périphérique physique du même type, puisque le niveau abstraction présenté aux machines virtuelles est plus élevé. En revanche, la paravirtualisation nécessite une modification des OS invités, c'est-à-dire des pilotes de périphériques mais aussi de la façon dont les OS invités reçoivent les interruptions, ce qui en limite l'utilisation.

Virtualisation par assignation directe L'assignation directe représente le cas où un périphérique physique appartient directement et exclusivement à un OS invité particulier.

L'OS peut alors interagir nativement avec le périphérique en utilisant son pilote de périphérique existant. Les interruptions doivent néanmoins être gérées par l'hyperviseur pour être redirigées vers l'OS invité destinataire.

Puisque, pour un périphérique assigné directement, l'hyperviseur n'est plus obligé de fournir lui-même le pilote de périphérique correspondant, cette technique est sensée alléger l'hyperviseur. Néanmoins, il subsiste une incohérence qu'il incombe à l'hyperviseur de résoudre. En effet, une machine virtuelle s'exécute dans un espace d'adressage virtuel qui ne correspond en rien à celui que connait le périphérique, puisque le périphérique fonctionne dans l'espace d'adressage physique. Cela pose un vrai souci si le périphérique est équipé d'une capacité DMA : premièrement car les deux espaces ne correspondent pas, ce qui rend problématique l'interaction entre l'OS invité et le périphérique, et deuxièmement, car par l'intermédiaire du périphérique DMA, l'OS invité est susceptible d'accéder à des zones mémoires qui ne lui sont pas attribuées. L'hyperviseur doit donc fournir un pilote intermédiaire, dit *pass-through*, qui est responsable d'intercepter toutes les interactions entre un OS invité et un certain périphérique. Ce pilote réalise alors la traduction entre l'espace d'adressage virtuel de l'OS invité et l'espace d'adressage physique, dès qu'il reconnait des commandes se rapportant à des adresses mémoires. Néanmoins, il doit exister un tel pilote intermédiaire par périphérique, puisque les formats de commandes sont spécifiques à chaque périphérique. Ce mécanisme ré-accroît donc la complexité de l'hyperviseur, souvent au détriment de sa fiabilité.

La technique d'assignation directe reste plus performante que l'émulation, mais de par sa nature exclusive, elle interdit le partage de périphériques entre plusieurs machines virtuelles.

Virtualisation hybride La technique de virtualisation *hybride* combine l'assignation directe et le plus souvent la paravirtualisation, comme cela est illustré dans la figure 3.4. En parallèle des machines virtuelles « standard », fonctionne une (ou plusieurs) machine virtuelle spécialement dédiée à la gestion des périphériques d'entrée/sortie. Cette machine virtuelle de service, généralement un OS traditionnel, contrôle les périphériques physiques de la plateforme grâce à la technique d'assignation directe, ce qui lui permet d'utiliser ses propres pilotes existants. En contre partie, elle fournit aux autres machines virtuelles un accès paravirtualisé à ces périphériques.

Cette décomposition de la virtualisation permet de réduire la complexité de l'hyperviseur, de rendre indépendante la gestion des périphériques, à la fois par rapport aux machines virtuelles standard mais aussi de l'hyperviseur, et enfin d'autoriser le partage des périphériques à moindre coût.

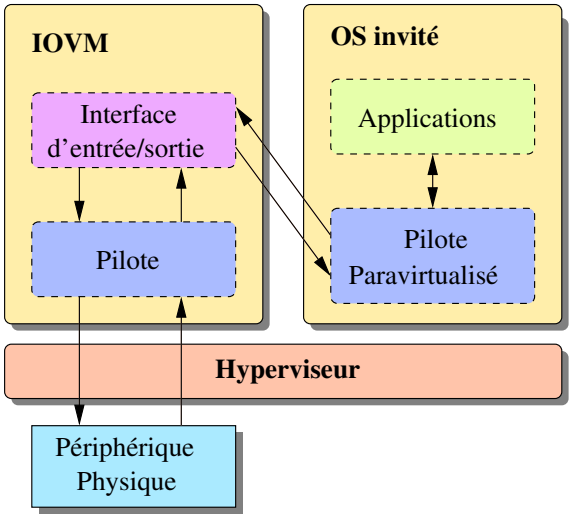


FIGURE 3.4 – Technique de virtualisation hybride de périphérique.

3.1.3.4 Assistance matérielle

Bien que la virtualisation puisse être réalisée sur des systèmes traditionnels, de bien meilleures performances peuvent être atteintes si le système est matériellement modifié à cet escient.

L'assistance matérielle à la virtualisation a été – largement – explorée il y a quelques décennies par *IBM* [27], mais ce n'est qu'assez récemment que les principaux fabricants de processeur pour systèmes bureautiques (*Intel* et *AMD*) ont commencé à s'y intéresser [28, 25]. Ils ont alors, par étapes successives, proposé des modifications matérielles pour permettre une virtualisation plus efficace des processeurs, de la mémoire, et enfin des périphériques.

Processeur Par l'intermédiaire des technologies *VT-x* [29] et *AMD-V* [30], Intel et AMD ont tous les deux ajouté à leurs processeurs un nouveau mode d'exécution, plus privilégié que le mode noyau. Les machines virtuelles n'ont alors plus besoin d'être « poussées » entièrement en mode utilisateur, mais peuvent occuper normalement les traditionnels modes utilisateur et noyau. Ce niveau de privilège supplémentaire, qui permet finalement l'exécution de plusieurs contextes d'exécution en mode noyau, évite à l'hyperviseur de devoir recourir à la technique de *binary translation* précédemment citée. Enfin, ce support matériel aide également l'hyperviseur à piéger facilement et systématiquement les instructions sensibles exécutées par les machines virtuelles.

Mémoire Concernant la virtualisation de la mémoire, Intel et AMD ont chacun étendu leur technique de virtualisation avec les technologies respectives d'*Extended Page Table* [31] et de *Nested Paging* [32]. Dans les deux cas, les manipulations des machines virtuelles sur leurs propres tables de pages ou sur la MMU n'ont plus besoin d'être interceptées à la

volée par l'hyperviseur et émulées par des tables de page fantômes. La MMU peut en effet réaliser la double traduction « adresse virtuelle vers adresse physique » (propre à chaque machine virtuelle) et « adresse physique vers adresse machine » (gérée par l'hyperviseur) de façon autonome et automatique. Ce mécanisme réduit grandement le nombre de transitions entre les machines virtuelles et l'hyperviseur.

Périphériques Enfin concernant les périphériques, notamment équipés de capacité DMA, Intel et AMD ont respectivement proposé les technologies *VT-d* [26] et *IOMMU* [33]. Dans les deux cas, ces technologies se concrétisent par l'introduction d'un module matériel de traduction dans le contrôleur d'accès à la mémoire centrale. Ce module permet de résoudre l'incohérence, évoquée précédemment, entre l'espace d'adressage virtuel d'une machine virtuelle et l'espace d'adressage vu par les périphériques. Un périphérique appartenant à une machine virtuelle (par la technique d'assignation directe) voit effectivement tous ses accès mémoire modifiés par le module matériel de traduction, qui utilise alors la même page des table que celle utilisée pour la machine virtuelle qui l'utilise. En conséquence, la machine virtuelle et son périphérique DMA fonctionnent dans le même espace d'adressage virtuel.

Moyennant une assistance matérielle intégrée aux périphériques DMA, les deux technologies permettent également à ces derniers d'être partagés entre plusieurs machines virtuelles. Cependant, en raison des coûts élevés de mise en œuvre, très peu de produits proposent ce service à l'heure actuelle.

3.1.4 Technologie ARM/TrustZone

Les mécanismes d'assistance matérielle dédiés à la virtualisation ne sont, à ce jour, pas encore apparus dans le monde des systèmes embarqués. En 2004, *ARM*, un des leaders du marché de l'embarqué, a proposé la technologie *TrustZone* [34], que l'on peut considérer comme dérivée de la virtualisation.

Le concept de TrustZone consiste à distinguer deux mondes au sein d'une même plateforme matérielle, un sécurisé et un non sécurisé. Typiquement, le monde non sécurisé héberge un OS traditionnel, tandis que le monde sécurisé héberge une pile logicielle de confiance, en charge du partitionnement global de la plateforme (allocation de la mémoire et des périphériques), mais aussi des opérations sensibles (gestion des droits numériques, authentification sécurisée, etc.), notamment requises pour les systèmes orientés multimédia.

Comme il est expliqué dans suite, cette technologie de co-hébergement « binaire » nécessite une assistance matérielle au niveau processeur, ainsi que des modules matériels dans la plateforme.

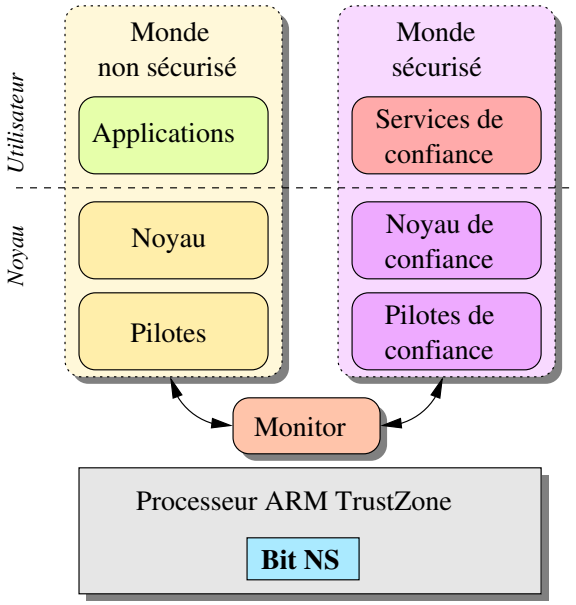


FIGURE 3.5 – Monde parallèle sécurisé proposé par la technologie TrustZone.

3.1.4.1 Processeur TrustZone

Contrairement à la virtualisation assistée par matériel, dans laquelle un nouveau niveau de privilège a été ajouté verticalement sous les autres, l'approche TrustZone duplique horizontalement les modes de privilège traditionnels. La figure 3.5 illustre ce concept. Un bit, nommé *NS* (pour *Non-secure Bit*), situé dans un registre de coprocesseur indique alors lequel des deux mondes occupe le processeur à un instant donné.

À la grande différence de la virtualisation, l'unique « machine virtuelle », appartenant au monde non sécurisé, s'exécute dans l'espace d'adressage physique. De plus, grâce à la duplication de certains registres sensibles, aucune instruction sensible n'a besoin d'être piégée (et émulée). Par exemple, le bit d'activation de la MMU interne au processeur est dupliqué, ce qui permet à la machine virtuelle de régler sa propre utilisation de la MMU, sans altérer l'utilisation de la MMU faite du côté sécurisé. La MMU est effectivement partagée entre les deux mondes, ses entrées étant étiquetées pour distinguer les deux utilisations.

TrustZone autorise donc ce que l'on pourrait apparenter à une virtualisation complète, c'est-à-dire qu'un OS invité non modifié s'exécute dans le monde non sécurisé de façon presque autonome, du moins tant qu'il respecte les limites qui lui ont été fixées. Dans ce cas, le monde sécurisé peut quand même reprendre la main périodiquement, en se faisant réveiller par un périphérique d'horloge sécurisé. TrustZone permet aussi une certaine communication entre les deux mondes, ce qui permet une forme de paravirtualisation. Au travers d'un nouveau mode intermédiaire, nommé *Monitor*, commun aux deux mondes, et d'un mécanisme d'interruption logicielle, le monde non sécurisé peut solliciter des services, offerts par le monde sécurisé.

3.1.4.2 Plateforme TrustZone

Afin que la distinction des deux mondes puisse être effectuée au niveau de la plateforme, TrustZone définit l'ajout d'un bit supplémentaire au protocole *AMBA AXI* [35]. Ce champ reprend en réalité le bit NS situé dans le processeur. Toutes les transactions réalisées par un processeur TrustZone sont alors étiquetées avec la valeur du bit NS, ce qui permet de savoir pour chaque transaction, si elle a été initiée par le monde sécurisé ou non sécurisé.

TrustZone propose alors trois méthodes pour partitionner les ressources matérielles de la plateforme, toutes sous le contrôle du monde sécurisé. Premièrement, la plateforme peut utiliser un bus partagé AMBA AXI compatible TrustZone qui permet de définir des droits d'accès entre processeurs et périphériques. Certains périphériques peuvent alors n'être accessibles que par le monde sécurisé (comme, un périphérique d'horloge sécurisé), tandis que d'autres sont ouverts au monde non sécurisé. Cette granularité au niveau composant physique n'est cependant pas idéale pour partager des bancs mémoires physiques. Les deux dernières méthodes se matérialisent par l'introduction de deux modules matériels, de complexité différente, placés entre le bus partagé et les bancs mémoires à partager. Le premier module, nommé *TZMA* (pour *TrustZone Memory Adapter*), permet de diviser un banc mémoire en deux zones, une pour chacun des deux mondes, et est souvent utilisé devant les bancs mémoire embarqués. L'autre module, nommé *TZASC* (pour *TrustZone Address Space Controller*) permet de segmenter un banc mémoire en plusieurs zones, ce qui est plutôt utilisé pour partitionner une mémoire externe. La figure 3.6 montre un exemple de plateforme TrustZone complète.

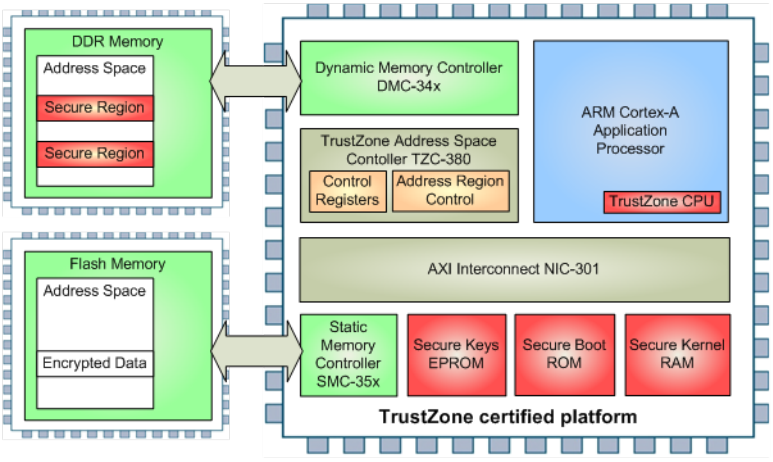


FIGURE 3.6 – Exemple de plateforme compatible TrustZone. *Crédits : ARM*

3.1.5 Conclusion sur la virtualisation

Comme nous l'avons expliqué dans cette section, la virtualisation permet une technique performante de co-hébergement de plusieurs piles logicielles autonomes sur la même plateforme matérielle, en partageant les processeurs, la mémoire et les périphériques.

Même s'il est fort probable qu'elles apparaissent d'ici peu, les techniques d'assistance matérielle les plus avancées ne sont pas encore apparues dans le monde de l'embarqué. Pour le moment, seules des solutions purement logicielles existent. Ces solutions nécessitent tout de même des processeurs homogènes, offrant plusieurs niveaux de privilège, pour exécuter l'hyperviseur dans un mode plus privilégié que les machines virtuelles, et équipés de MMU, pour réaliser le partitionnement de l'espace d'adressage.

La virtualisation permet donc un partage efficace des ressources processeur et mémoire sur des architectures multiprocesseurs homogènes (symétriques), qui peuvent être sous le contrôle d'une base logicielle de confiance unique (en l'occurrence, un hyperviseur). En revanche, dans des systèmes multiprocesseurs hétérogènes à mémoire partagée, puisque la propagation d'une couche de logicielle unique est impossible, cette approche n'est pas adaptée. Une pile logicielle s'exécutant indépendamment sur un processeur spécialisé, qui ne posséderait ni MMU, ni différents niveaux de privilège et qui ne supporterait pas d'être virtualisé, pourrait effectivement accéder à l'espace d'adressage entier, en court-circuitant complètement n'importe quelle couche de virtualisation.

Le mécanisme TrustZone proposé par ARM permet la séparation d'un système en deux mondes, sécurisé et non sécurisé. Pour être pleinement fonctionnel, tous les processeurs de la plateforme doivent également être homogènes dans le sens où ils doivent bénéficier du support TrustZone. Ce co-hébergement binaire n'est cependant pas adapté pour répondre aux besoins de flexibilité exposés dans le chapitre précédent, où de multiples piles logicielles autonomes doivent se partager la même plateforme matérielle.

3.2 Architectures sécurisées

Bien que la virtualisation vise plutôt la problématique de co-hébergement, nous avons vu qu'elle permettait, par l'utilisation de composants matériels supplémentaires, la mise en place d'un mécanisme de sécurité au niveau de la plateforme. Dans le cas de TrustZone, ce mécanisme, qui autorise le monde non sécurisé à définir des droits d'accès sur la plateforme, introduit donc une première technique de protection de données.

Les projets que nous allons développer dans la suite de ce chapitre, se focalisent uniquement sur la protection de données, dans le domaine des systèmes multiprocesseurs intégrés sur puce. Comme préconisé par TrustZone, cette protection de données intervient toujours au cœur de la plateforme, c'est-à-dire dans le réseau d'interconnexion.

3.2.1 Architectures basées sur un bus partagé

Le projet qui semble s'être intéressé le premier à la protection de données intégrée au réseau d'interconnexion est *SECA* [36], développé par NEC.

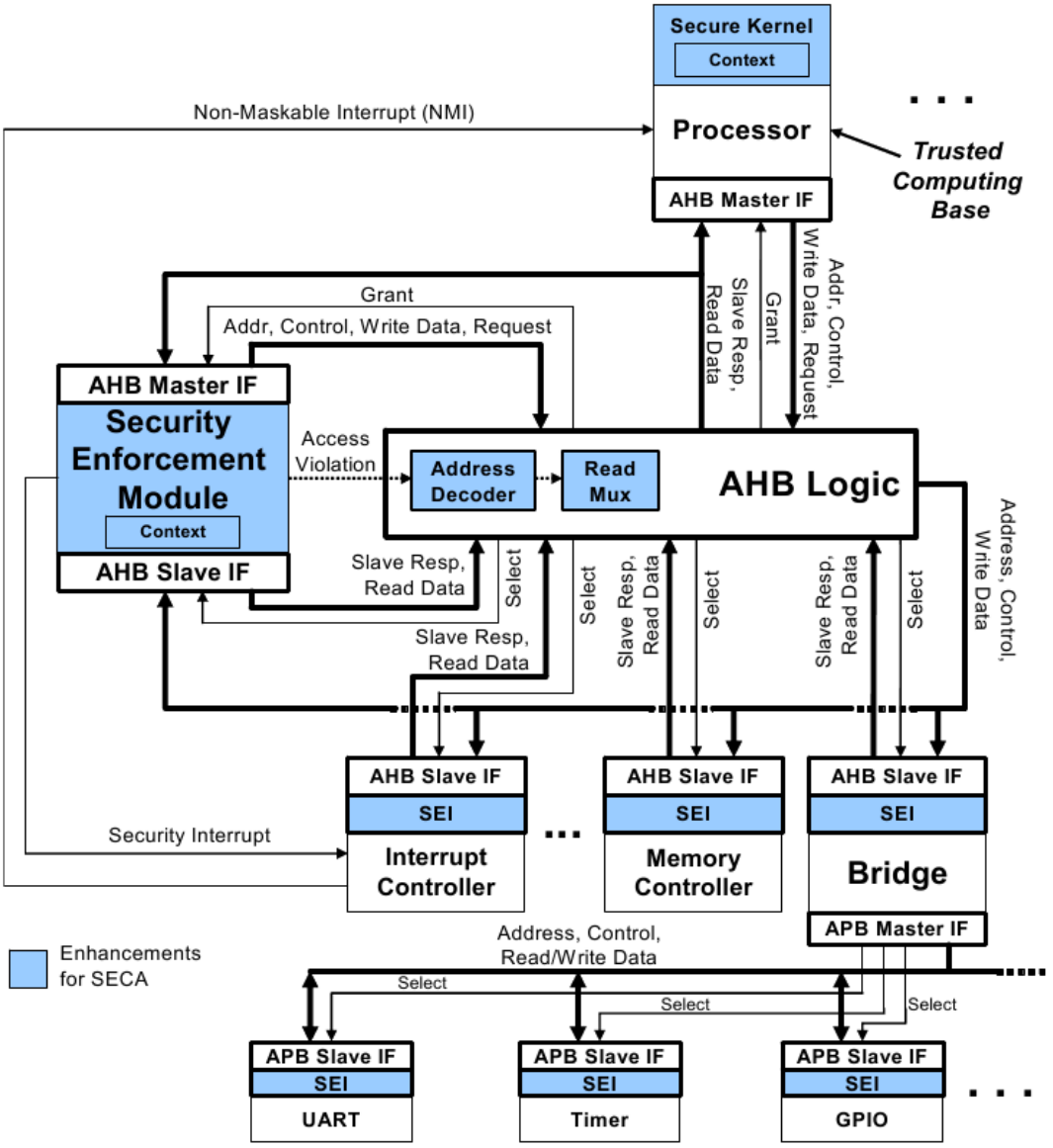


FIGURE 3.7 – Mécanisme de protection SECA pour bus de type ARM AMBA AHB. *Crédits : [36]*

SECA (pour *Security-Enhanced Communication Architecture*) permet de surveiller les interactions entre les différents composants matériels d'un système sur puce. Comme illustré dans la figure 3.7, ce mécanisme est essentiellement centré autour d'un module matériel, nommé *SEM* (pour *Security Enforcement Module*), qui contrôle le trafic d'un bus partagé de type *ARM AMBA AHB* [37]. Le SEM propose trois fonctionnalités de sécurité, dont la plus pertinente concernant la problématique de protection de données s'intitule *Address Protection Unit* (*APU*). Ce sous-module permet de définir des règles de droits d'accès, spécifiant comment un composant matériel (initiateur) peut accéder à une plage d'adresses (correspondant à un périphérique) dans un contexte particulier. Le contexte, qui s'applique au système entier, est défini par un noyau sécurisé fonctionnant sur l'un des processeurs de

la plateforme. Au sein d'un contexte donné, les processeurs et autres composants initiateurs peuvent alors être autorisés à accéder à des segments dans l'espace d'adressage. Ces droits d'accès (lecture seule, écriture seule, lecture-écriture, aucun) sont stockés dans une table de correspondance (*lookup table*), embarquée dans le SEM. En somme, dès qu'une transaction se produit sur le bus partagé, le SEM peut l'analyser en fonction du composant qui l'a initié, le contexte courant et l'adresse mémoire de destination, et l'autoriser ou non suivant les règles d'accès contenues dans la table de correspondance.

Bien que ce projet semble proposer une solution intéressante, il est basé sur l'utilisation de bus partagé. Hors, durant la dernière décennie, la technologie des réseaux d'interconnexion a bénéficié d'avancées significatives. Les bus partagés sont effectivement en passe d'être remplacés, au sein des MP-SoC industriels de nouvelle génération, par des *réseaux sur puce*. Comme l'avait évoqué [38] et comme le résume [39], il est aujourd'hui crucial de s'intéresser à la sécurité dans les réseaux sur puce. La section suivante présente donc les architectures de sécurité existantes qui sont basées sur l'utilisation d'un réseau sur puce.

3.2.2 Architectures basées sur un réseau sur puce

3.2.2.1 Introduction

La technologie de bus partagé, utilisée de façon standard depuis de nombreuses années, souffre d'un important défaut : la bande passante d'un bus ne supporte pas le passage à l'échelle (*scalabilité*). Le bus partagé ne permettant qu'une seule transaction entre deux composants du système à la fois, plus le nombre de composants augmente, moins chacun peut disposer du bus. A l'heure où la capacité d'intégration autorise des systèmes sur puce embarquant plusieurs dizaines, voire plusieurs centaines, de composants, le bus partagé n'est plus suffisant pour assurer une communication efficace.

Pour éviter que le réseau d'interconnexion ne devienne un goulot d'étranglement, un nouveau paradigme a été inventé : le « réseau sur puce »[40, 41] (ou *NoC*, pour *Network-on-Chip*). Un réseau sur puce est basé sur de multiples liens points à points, interconnectés par des routeurs. Les composants du système sont eux connectés au routeur par des contrôleurs d'interface (ou *NIC*, pour *Network Interface Controller*), notamment en charge de la conversion de protocole entre les composants et le réseau. Le plus souvent, un réseau sur puce utilise effectivement son propre protocole d'échange interne (par exemple, par commutation de paquets). Les transactions peuvent être transmises à partir de n'importe quelle source jusqu'à n'importe quelle destination, en utilisant plusieurs liens et en étant routées par les routeurs situés entre elles. La figure 3.8 montre un exemple de plateforme matérielle basée sur l'utilisation d'un réseau sur puce, respectant une topologie en grille.

Bien qu'un réseau sur puce soit intrinsèquement plus gros qu'un bus partagé, ce coût est généralement considéré comme acceptable, notamment grâce à la surface de silicium gagnée par l'augmentation de la finesse de gravure des puces. De plus, là où le bus partagé

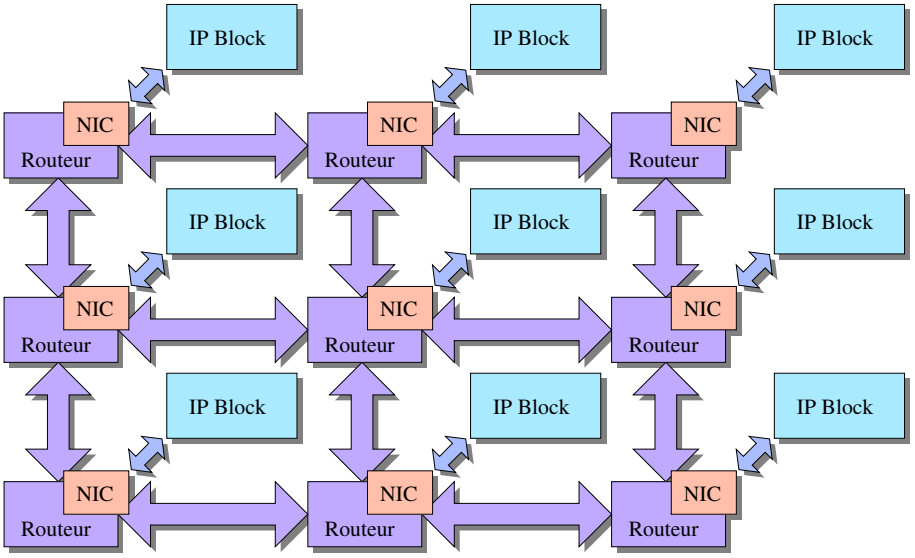


FIGURE 3.8 – Architecture basée sur un réseau sur puce ayant une topologie de type grille.

n'autorise qu'une seule transaction à la fois, tous les liens d'un réseau sur puce peuvent opérer simultanément pour transmettre différentes transactions, permettant ainsi un fort niveau de parallélisme et une bonne distribution de la bande passante. L'ajout de composants dans le système entraînant l'ajout de nouveaux routeurs, et donc de nouveaux liens de connexion, la bande passante augmente proportionnellement à la taille du système : la bande passante devient donc *scalable*.

Un peu à la manière des bus hiérarchiques, l'empilement de plusieurs niveaux d'interconnexion est également rendu possible avec les réseaux sur puce. À chaque routeur, peut être attachée une grappe de composants (*cluster*), tels que des processeurs, bancs mémoires ou périphériques, tous interconnectés par un bus local (typiquement un crossbar ou un bus partagé). Ce type d'architecture est appelé *multi-cluster*, tandis qu'une architecture ne comportant qu'un composant par routeur est dite *à plat*.

3.2.2.2 Sonics SMART Interconnect

Le *SMART Interconnect* [42], commercialisé par *Sonics*, est un réseau d'interconnexion commercial complet (qui semble être un NoC), malheureusement très peu documenté, fournissant une fonctionnalité de protection d'accès. Il est possible de définir des régions de protection dans l'espace d'adressage de certains composants cibles. Le mécanisme peut être dynamique, avec des définitions à la volée de régions de taille variable. Le mécanisme peut également être fonction du « rôle » des initiateurs, ce qui signifie qu'un processeur peut bénéficier de droits d'accès différents suivant le contexte logique qu'il exécute à un moment donné (par exemple, selon qu'il exécute le noyau d'un OS ou des applications utilisateurs).

3.2.2.3 μSpider

Dans le domaine académique, Diguet et al. [43, 44] semblent avoir été les premiers à proposer un mécanisme de sécurité intégré à un réseau sur puce à plat, en l'occurrence le réseau *μSpider*.

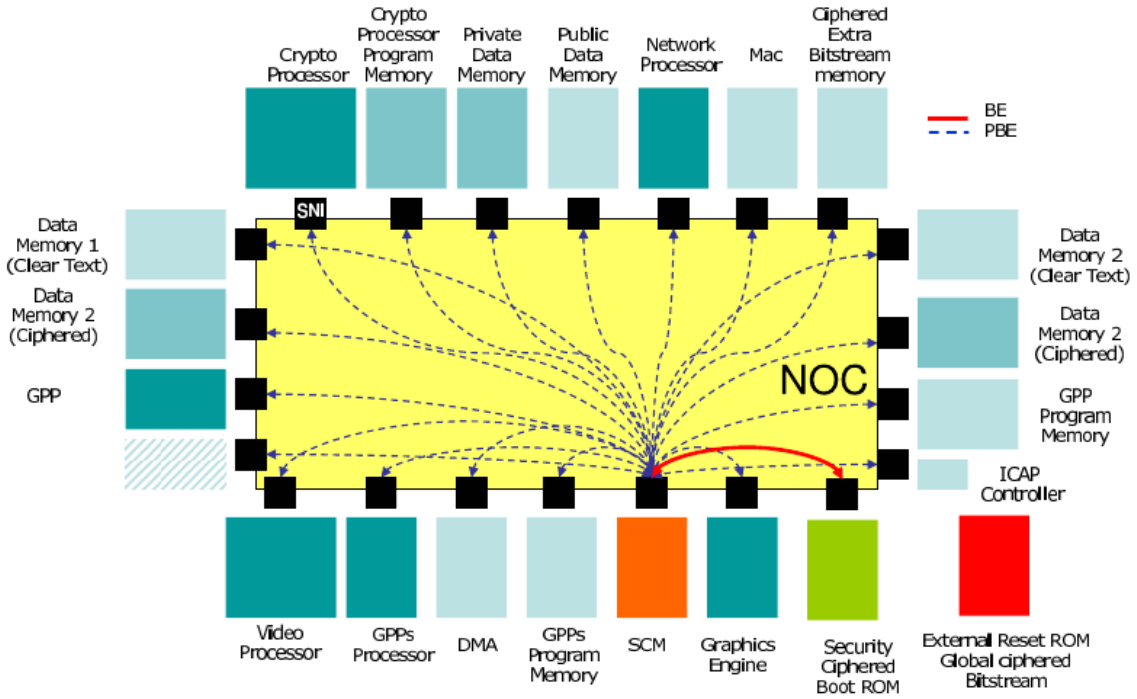


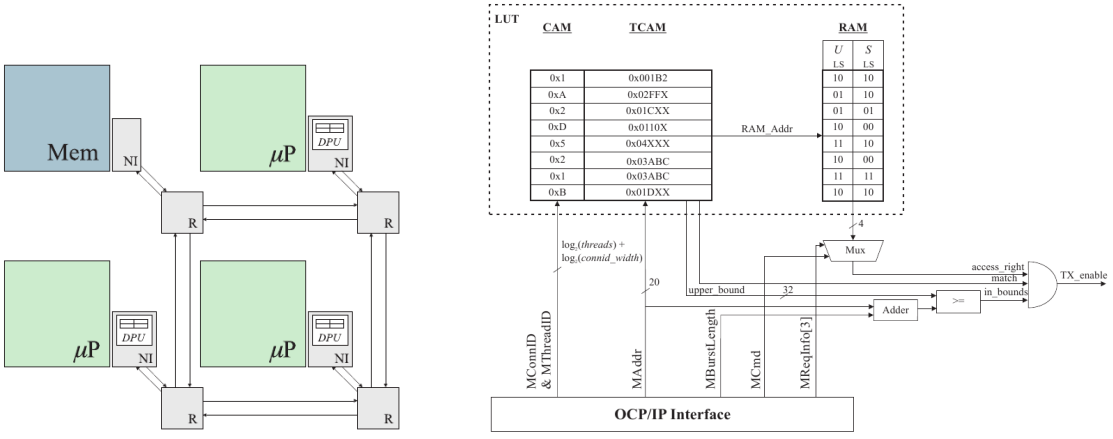
FIGURE 3.9 – Configuration des SNI du réseau μSpider par le SCM. *Crédits : [43]*

Le mécanisme est composé de NIC sécurisés, appelés ici *Secure Network Interface* (*SNI*), qui sont à l'interface entre les composants et le réseau sur puce, et d'un composant particulier, nommé *Secure Configuration Manager* (*SCM*). Comme l'illustre la figure 3.9, le SCM sert essentiellement à effectuer un démarrage sécurisé de la plateforme, notamment par la configuration des SNI. Pour éviter les perturbations de composants non contrôlables, qui peuvent tenter de bloquer le réseau par des transactions intempestives, cette configuration est réalisée au moyen de canaux de communication virtuels prioritaires. Une fois les SNI configurés, ils sont alors en mesure d'assurer des fonctions de contrôle d'accès pendant l'exécution du système. Premièrement, un SNI attaché à un composant initiateur peut vérifier, en amont, c'est-à-dire avant l'entrée sur le réseau sur puce, qu'une transaction initiée par le composant donné ne dépasse pas une longueur prédéfinie. Deuxièmement, un SNI attaché à un composant cible peut vérifier qu'une transaction reçue appartient à un composant initiateur possédant les droits requis pour accéder au composant cible donné. La granularité du contrôle d'accès (lecture seule, écriture seule, lecture-écriture, aucun) est dans ce cas au niveau composant matériel : par exemple, tel périphérique peut être accédé par tel processeur, mais pas par tel autre. Grâce à l'architecture interne des SNI, cette vérification peut être réalisée en parallèle du traitement des paquets, ce qui diminue voire

annule le surcoût en performance engendré par cette opération. Un SNI cible peut également vérifier qu'une transaction reçue reste bornée au segment d'espace d'adressage offert par le composant cible sous-jacent. Une réplication de ce mécanisme est nécessaire, dans le système proposé, pour segmenter un composant mémoire en plusieurs régions pour lesquelles les droits d'accès sont différents : le composant mémoire est dans ce cas accessible par plusieurs SNI, dont chacun protège un bout du segment d'espace d'adressage sous-jacent. Enfin, les SNI sont capables de signaler toute anomalie au SCM, qui fait office de surveillant du système. Ce dernier peut alors prendre les décisions adéquates, par exemple reconfigurer dynamiquement les SNI.

3.2.2.4 DPU

Fiorin et al. [45, 46, 47] proposent une solution de protection de données pour des architectures de type MP-SoC à mémoire partagée, basées sur des réseaux sur puce.



(a) DPU intégrés au NIC d'un réseau sur puce

(b) Architecture interne d'un DPU

FIGURE 3.10 – Solution DPU. *Crédits : [45, 46]*

Comme l'illustre la figure 3.10, leur mécanisme est composé de Data Protection Unit (*DPU*), des modules matériels spécifiques implémentés à l'intérieur des NIC d'un réseau sur puce. Un DPU peut vérifier et limiter les droits d'accès d'un composant initiateur, c'est-à-dire autoriser l'accès à diverses régions au sein de l'espace mémoire partagé : par exemple, tel processeur est autorisé à accéder à telle plage de l'espace d'adressage. Les droits d'accès sur l'espace d'adressage (lecture seule, écriture seule, lecture-écriture, aucun) sont définis à l'échelle de chaque composant initiateur, avec cependant une granularité plus fine pour les processeurs. Les DPU peuvent en effet distinguer les différents modes d'exécution des processeurs (utilisateur ou noyau, ou même sécurisé ou non sécurisé dans un environnement TrustZone). Cette solution partage deux similarités avec le mécanisme proposé par μSpider. Premièrement, le surcoût en performance de la procédure de vérification est nul, car cette dernière est réalisée en parallèle de la conversion de protocole entre le réseau et les

composants matériels. Enfin, un agent de confiance particulier, nommé *Network Security Manager* (*NSM*), est chargé de la configuration dynamique des DPU de la plateforme.

3.2.3 Conclusion sur les architectures sécurisées

Les projets présentés dans cette section s'appliquent à proposer une solution de protection de données globale à un système intégré sur puce, en utilisant le réseau d'interconnexion.

SECA fut le premier à s'intéresser à cette problématique, et propose une politique de segmentation de l'espace d'adressage pour chaque composant initiateur, avec une notion de contexte global. Cependant, ce projet s'appuie sur l'utilisation d'un bus partagé, technologie en passe d'être remplacée par celle des réseaux sur puce.

La solution de protection proposée par Sonics n'est malheureusement pas assez documentée pour permettre une quelconque analyse.

Le réseau sur puce μSpider propose une protection au niveau des composants physiques, qui supporte mal le partage des composants mémoire, puisqu'il est très difficile de définir des droits d'accès différents à diverses régions mémoires au sein du même banc mémoire physique. En outre, cette méthode de filtrage n'est pas suffisamment flexible dans le sens où elle ne supporte pas que plusieurs piles logicielles (par exemple des applications), pour lesquelles on voudrait définir différents droits d'accès, soient amenées à partager les mêmes processeurs, ou éventuellement les mêmes périphériques DMA.

En étant capable de différencier les différents modes d'exécution des processeurs, DPU est la solution de protection de donnée la plus avancée. Cependant, cette capacité n'est toujours pas suffisante pour supporter un partage flexible des composants initiateurs, processeurs ou périphériques DMA. Concernant le partage de l'espace d'adressage, la politique de segmentation implémentée par DPU n'est également pas assez flexible pour répondre aux besoins. À la manière de SECA, les tables de droits d'accès sont directement embarquées dans les modules de sécurité, sans mécanisme de cache. Il semble alors difficile pour cette stratégie de supporter une exécution flexible de plusieurs piles logicielles demandant potentiellement l'accès à de multiples régions mémoires, puisque les tables de droits d'accès embarquées seraient trop grosses, et difficiles à maintenir.

3.3 Conclusion

Dans les systèmes embarqués, la technologie industrielle TrustZone de ARM reste aujourd'hui la plus complète : elle traite aussi bien le partage des processeurs, que le partage du reste de la plateforme, le tout de façon sécurisée. Néanmoins, elle n'accepte pas les plateformes hétérogènes, et dû à son aspect binaire, n'est pas adaptée aux prochaines générations de MP-SoC, surtout dans le domaine du multimédia.

De la même façon, les solutions actuelles de protection de données dans les réseaux sur puce ne sont pas prévues pour répondre aux futures exigences de flexibilité, n'ayant pas encore vraiment tenu compte de la problématique de co-hébergement dans son intégralité.

Chapitre 4

Approche multi-compartiment

Sommaire

Ce chapitre expose les principes de la solution que nous proposons pour réaliser un co-hébergement sécurisé, flexible et efficace de plusieurs piles logicielles autonomes sur une même plateforme matérielle multiprocesseur hétérogène à mémoire partagée.

4.1 Introduction

Dans un système traditionnel, c'est-à-dire composé d'un processeur généraliste et exécutant un OS, le *Trusted Computing Base* (*TCB*), expliqué précédemment dans la section 2.2.5, réalise finalement deux tâches. Le partage de la ressource processeur entre les applications est la première tâche du TCB. Elle est rendue possible par deux éléments, respectivement matériel et logiciel : l'existence matérielle d'un mode plus privilégié dans le

processeur permet au noyau de l'OS d'être maître de la ressource processeur et d'en orchestrer le partage. La deuxième tâche du TCB est le partage de la mémoire, qui est également rendue possible par deux éléments, respectivement matériel et logiciel : le processeur offre localement une MPU ou MMU, qui est sous la supervision logicielle du noyau de l'OS.

Toutes les solutions de co-hébergement présentées jusqu'à maintenant, suivent toujours la logique de répliquer ce TCB sur les différents processeurs, ou groupe de processeurs, d'une plateforme hétérogène. Pour la tâche de partage de processeur, cela n'est pas problématique, car c'est une tâche finalement locale à chaque processeur. En revanche, la tâche de partage de la mémoire ne peut pas rester un mécanisme local à chaque processeur. Dans le cadre d'une plateforme à mémoire partagée, le mécanisme de partage de la mémoire doit être global, et donc être géré par un TCB global : autant pour le mécanisme matériel qui le permettra, que pour la partie logicielle qui l'organisera.

Pour les raisons juste évoquées, nous proposons en premier lieu un nouveau modèle de TCB, hiérarchique, qui distingue le partage des processeurs et le partage de la mémoire. Ce modèle est illustré dans la figure 4.1. Les processeurs qui offrent plusieurs niveaux de privilège, peuvent localement organiser leur propre partage, en association avec un agent logiciel de confiance local, appelé *LTA* pour *Local Trusted Agent* (concrètement, le LTA sera représenté par un noyau d'OS ou un hyperviseur). Les processeurs non partageables ne peuvent héberger qu'une pile logicielle dédiée, que l'on peut conceptuellement considérer comme un LTA. Le partage de l'espace d'adressage est lui globalement géré par un TCB global et indépendant, composé de deux éléments, respectivement matériel et logiciel : un mécanisme matériel de protection de l'espace d'adressage, supervisé par un agent de confiance global, que l'on appelle alors *GTA* pour *Global Trusted Agent*.

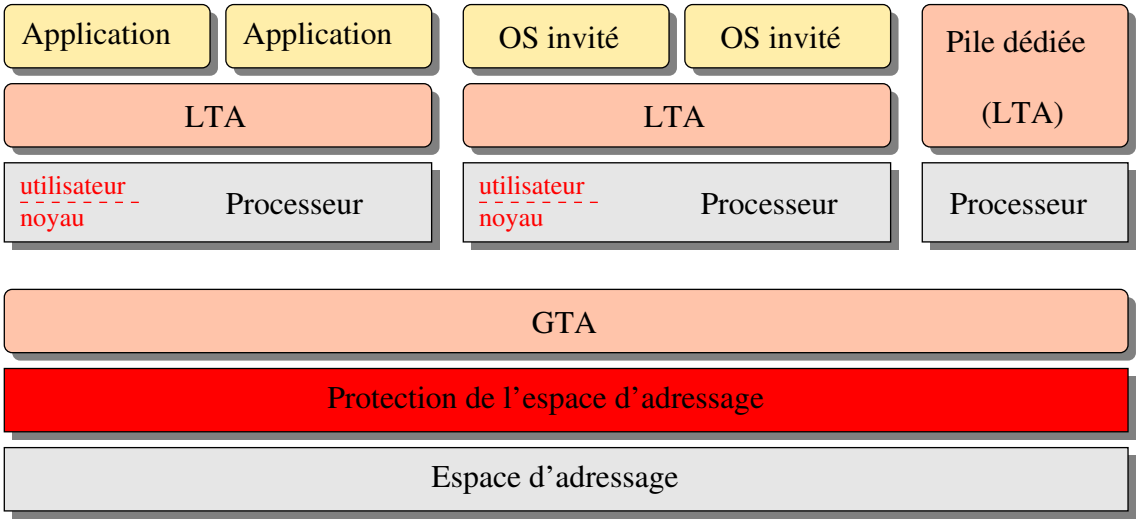


FIGURE 4.1 – TCB hiérarchique : partage local des processeurs et partage global de l'espace d'adressage.

Avant de détailler ce modèle de TCB hiérarchique, définissons la terminologie qui sera utilisée dans le reste de ce document. Un *domaine de protection* est une collection de

droits d'accès spécifiques sur l'espace d'adressage. Chaque pile logicielle, pour laquelle on veut pouvoir différencier les droits d'accès dans l'espace d'adressage, fonctionne dans un domaine de protection distinct. Les domaines de protection peuvent potentiellement se recouvrir entre eux, c'est-à-dire que plusieurs domaines de protection peuvent avoir accès à des zones identiques dans l'espace d'adressage – pas nécessairement avec les mêmes droits d'accès. On appelle *compartiment* l'entité logique représentant une pile logicielle et son domaine de protection associé.

Dans cette optique, le concept de *multi-compartiment* désigne une architecture globale matérielle et logicielle, permettant le co-hébergement de plusieurs compartiments, c'est-à-dire de plusieurs piles logicielles fonctionnant chacune dans un domaine de protection spécifique. Pour remettre ce concept en parallèle avec l'idée de TCB hiérarchique sur plateforme multi-processeur hétérogène, un compartiment pourra être une pile dédiée, un LTA, ou une application/machine virtuelle fonctionnant au-dessus d'un LTA, ou encore, le GTA. Le concept de multi-compartiment est multi-niveau, dans le sens où il supporte dans un premier temps le co-hébergement de plusieurs LTA, c'est-à-dire plusieurs piles logicielles autonomes, et dans un deuxième temps, avec le même mécanisme de protection, le co-hébergement des applications/machines virtuelles fonctionnant au-dessus de ces LTA.

Dans la suite de cette section, nous expliquons les deux principes fondateurs de l'approche multi-compartiment, proposée dans cette thèse. Les sections suivantes détaillent la procédure complète pour rendre un MP-SoC compatible avec ces deux principes, et par conséquent avec l'approche multi-compartiment.

4.1.1 Identification

Accomplir un co-hébergement sécurisé de plusieurs compartiments sur une même plateforme matérielle nécessite dans un premier temps de pouvoir les distinguer correctement. Le mécanisme de partage de l'espace d'adressage devant être global à la plateforme, il faut intégrer un mécanisme d'identification des compartiments directement au niveau matériel. Pour toute transaction, c'est-à-dire tout échange de requête/réponse entre un composant initiateur et un composant cible au travers du réseau d'interconnexion, l'identification doit permettre de retrouver le compartiment émetteur. À cet escient, on introduit un identifiant de compartiment, nommé *Compartment IDentifier* ou *CID*, qui permet de lier toute transaction avec le compartiment associé. Concrètement, seuls les composants initiateurs sont concernés par l'identification, puisqu'ils sont les seuls à pouvoir émettre des transactions. Tout initiateur doit alors être capable de fournir l'identité exacte du compartiment pour lequel il exécute des transactions à un moment donné. Le CID doit être disponible directement en sortie du composant initiateur, c'est-à-dire sur le lien entre le composant et le réseau d'interconnexion.

Une telle identification permet de mettre en place un contrôle d'accès pour chaque transaction.

4.1.2 Protection

Protéger les compartiments les uns des autres signifie en réalité protéger leur ressources (code, données, utilisations exclusives de périphériques, etc.) contre de mauvaises utilisations. Dans un espace d'adressage partagé, ces ressources sont des plages d'adresses, accessibles par des transactions de lecture ou d'écriture dans cet espace d'adressage partagé. Le contrôle d'accès doit en conséquence définir, pour chaque compartiment, une série de droits d'accès sur diverses plages d'adresses. Puisque, grâce à l'identification, les transactions sont toutes associées à un CID, l'isolation est réalisée en confrontant les transactions aux séries de droits d'accès définis pour les CID associés. Contrairement à la plupart des approches de l'état de l'art, la granularité du contrôle d'accès devient alors le compartiment logique, au lieu du composant physique uniquement.

4.2 Mise en œuvre de l'identification

L'application du principe d'identification dans un MP-SoC nécessite trois niveaux de compatibilité. Le premier niveau de compatibilité consiste à véhiculer le CID au sein de la plateforme. Les deux autres niveaux de compatibilité concernent respectivement les processeurs et les périphériques possédant une capacité DMA, et plus exactement comment ces composants initiateurs peuvent rattacher toutes les transactions qu'ils émettent dans l'espace d'adressage avec le compartiment associé.

4.2.1 Propagation de l'identifiant

Au cœur de l'approche multi-compartiment se trouve l'identifiant de compartiment, qui étiquette toutes les transactions, au niveau matériel. Comme le montre la figure 4.2, notre solution consiste à étendre le protocole d'interconnexion avec un nouveau champ, afin de transmettre le CID dans la plateforme.

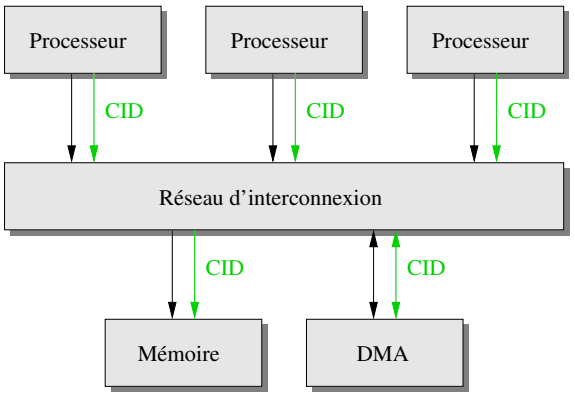


FIGURE 4.2 – Propagation du CID dans la plateforme matérielle.

En réalité, un tel champ existe déjà dans la plupart des protocoles d'interconnexion standards pour les systèmes embarqués : *VCI* [48] et *OCP* [49] définissent respectivement les champs `TRDID` et `MThreadID`.

Bien que ces champs semblent d'abord avoir été spécifiés pour supporter des transactions simultanées – transactions dans le désordre – émises par des composants possédant la propriété de « multi-threading », ils peuvent aussi servir pour étiqueter les transactions avec un CID. On remarquera d'ailleurs que la définition de ces champs peut être interprétée dans ce sens. VCI définit par exemple le champ `TRDID` de la façon suivante :

> « [Le champ `TRDID`] peut être utilisé comme une extension du champ `SRCID` [identifiant physique d'un composant initiateur] pour créer des composants logiques ou virtuels. » [1]

Cette définition peut tout à fait correspondre à notre problème, où les composants initiateurs physiques doivent diffuser des identités logiques différentes, suivant le compartiment qu'ils sont en train d'exécuter à un moment donné.

4.2.2 Processeurs

Rendre les processeurs compatibles avec l'approche multi-compartiment peut être réalisé de plusieurs manières selon le niveau de flexibilité souhaité, mais aussi selon le type de processeur. Comme expliqué plus haut, le but de cette réalisation est de permettre aux processeurs de respecter le principe d'identification, c'est-à-dire d'étiqueter toutes les transactions émises par les processeurs avec le CID du compartiment que ces derniers exécutent à un instant donné. Deux cas de figure peuvent alors être distingués, selon que le processeur est partagé et partageable entre plusieurs compartiments ou non. Dans tous les cas, le TCB local mis en œuvre est composé d'une partie matérielle et d'une partie logicielle.

4.2.2.1 Utilisation en multi-compartiment

Matériel La condition préalable essentielle, pour qu'un processeur puisse être partagé entre plusieurs compartiments, est qu'il doit fournir au moins deux niveaux de privilège (par exemple, mode noyau et mode utilisateur). Dans un contexte global de sécurité et de protection, la coopération directe inter-compartiment n'est effectivement pas envisageable : il faut une entité de confiance supérieure pour garantir cette coopération et ainsi autoriser le partage.

La compabilité d'un processeur partageable avec l'approche multi-compartiment repose principalement sur l'introduction d'un nouveau registre de contrôle. Ce registre de contrôle, uniquement accessible par le niveau de privilège le plus élevé, contient le CID du

1. En anglais dans le texte : « [`TRDID`] can be used as an extension to the `SRCID` to create logical, or virtual devices ».

compartiment en cours d'exécution. Sa valeur sert alors à étiqueter toutes les transactions, d'instructions et de données, que le processeur émet dans la plateforme pour le compte du compartiment courant. La figure 4.3 illustre cette solution matérielle.

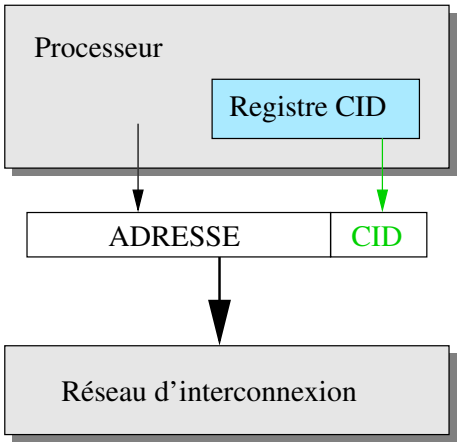


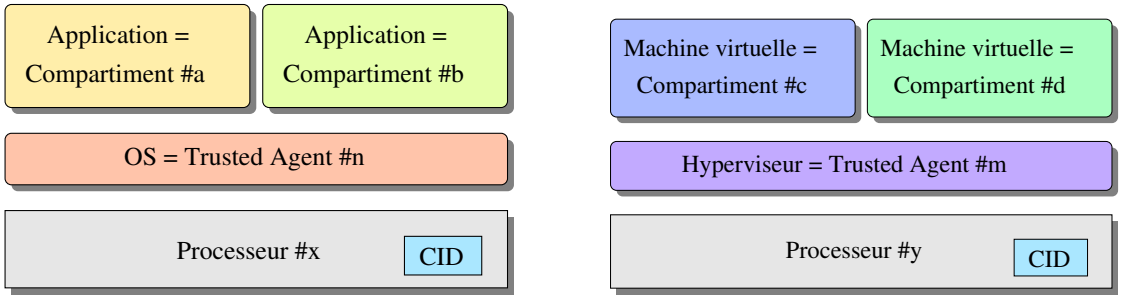
FIGURE 4.3 – Transformation d'un processeur pour un partage multi-compartiment.

On notera qu'il est conseillé d'étiqueter également les accélérateurs d'accès à la mémoire, c'est-à-dire les caches processeur (instructions et données) et les tampons d'écriture, avec le CID. Cette modification évite de devoir les purger quand on change de compartiment courant sur le processeur, et minimise ainsi une éventuelle dégradation des performances.

Logiciel Au niveau logiciel, nous avons introduit la notion d'agent de confiance local, le *LTA*. Cet agent logiciel s'exécute dans le niveau le plus privilégié du processeur, et est responsable d'au minimum deux tâches : premièrement, l'ordonnancement des compartiments, qui s'exécutent dans un niveau de privilège moindre, pour le partage de la ressource processeur et deuxièmement, la mise à jour du registre de contrôle CID en conséquence.

Ce concept de TCB matériel et logiciel s'appuie essentiellement sur des mécanismes existants, ce qui permet de gérer facilement la plupart des situations. Si les compartiments sont des applications, alors le LTA peut être représenté par un noyau d'OS traditionnel. Si les compartiments sont plus complexes, tels que des OS invités, alors le LTA peut être représenté par un hyperviseur. La figure 4.4 illustre ces deux possibilités.

Coût de la compatibilité Au niveau matériel, le coût d'introduction du registre de contrôle de CID s'avère faible. Il existe généralement des registres de contrôle disponibles dans les processeurs, pour permettre ce type d'utilisation. Typiquement, ce registre ressemble fortement à un registre *ASID* (*Address Space IDentifier*), utilisé localement par beaucoup de processeurs équipés de MMU, pour distinguer automatiquement les accès à cette dernière. La seule modification est alors de faire sortir la valeur du registre hors du processeur pour étiqueter les transactions. Si des modèles de processeur ayant une assistance matérielle pour la virtualisation sont un jour amenés à exister dans le marché de



(a) Le LTA est un noyau d'OS, les compartiments des applications

(b) Le LTA est un hyperviseur, les compartiments des machines virtuelles

FIGURE 4.4 – Compatibilité du multi-compartiment avec différentes stratégies logicielles.

l'embarqué, ils posséderont certainement un registre *VMID* (*Virtual Machine IDentifier*) comparable à l'ASID, pour différencier les machines virtuelles, et qui pourra être utilisé exactement de la même façon.

Au niveau logiciel, la tâche d'ordonnancement qui doit être assurée par le LTA est en fait quasiment toujours proposée par n'importe quel OS ou hyperviseur. La seule tâche « nouvelle » à effectuer est alors la mise à jour du registre de CID avec une valeur adéquate, ce qui a un coût négligeable. On notera également que l'implémentation du LTA est significativement simplifiée puisque ce dernier n'est dorénavant plus obligé de gérer la protection mémoire : la MMU ou MPU d'un processeur qui en est équipé peut être désactivée, au profit du contrôle d'accès global offert par l'approche multi-compartiment.

4.2.2.2 Utilisation en mono-compartiment

L'utilisation d'un processeur par un seul compartiment concerne plutôt les processeurs spécialisés, qui ne possèdent pas différents niveaux de privilège et qui, par conséquent, ne peuvent pas exécuter plus d'une seule pile logicielle.

Dans ce cas d'utilisation, où il n'y a qu'une valeur de CID pour tout le logiciel fonctionnant sur le processeur, le registre de CID n'est pas embarqué dans le processeur. Il peut être placé à l'extérieur, entre le processeur et le réseau d'interconnexion. Comme précédemment, la valeur de ce registre est utilisée pour étiqueter *a posteriori* toutes les transactions, d'instructions et de données, émises par le processeur. La figure 4.5 montre graphiquement ce cas d'utilisation.

Sachant que le processeur n'exécute qu'un unique compartiment, la valeur du registre de CID externe peut être statique. Cette valeur peut être fixée au moment de la conception, ou mieux, configurée au moment du démarrage par l'agent de confiance global, le *GTA*. Dans ce dernier cas, cela implique que le registre apparait dans l'espace d'adressage, mais reste protégé par le mécanisme global de protection.

Dans certains cas, si un OS s'exécute sur un processeur équipé d'une MMU, on peut trouver judicieux que l'OS entier soit vu comme un unique compartiment vis à vis de

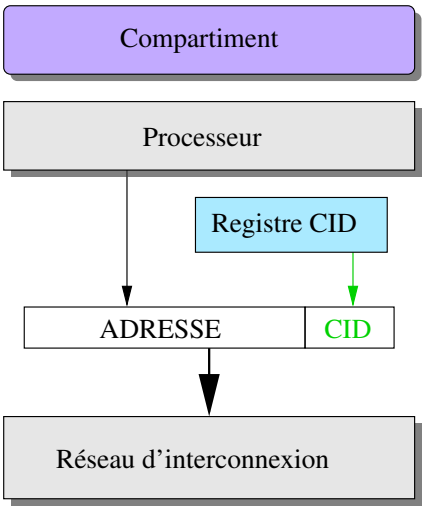


FIGURE 4.5 – Utilisation d'un processeur en mono-compartiment.

la plateforme. Les ressources mémoire qui lui sont attribuées, sont alors partagées à sa discrétion entre les applications qu'il exécute, typiquement grâce à l'utilisation de la MMU mais en dehors du mécanisme global de protection proposé. Ce comportement est ramené à un simple cas d'utilisation mono-compartiment.

4.2.2.3 Structure hiérarchique du CID

De manière générale, et comme le montre la figure 4.6, le CID véhiculé dans la plateforme et qui sert au mécanisme global de protection pour identifier les différents compartiments, est en réalité structuré en deux parties : un préfixe et un suffixe. Tandis que la taille du champ de CID est fixe pour toute la plateforme, les tailles de préfixe et de suffixe peuvent être différentes selon le cas d'utilisation.

FIGURE 4.6 – Structuration du CID.

Cette structuration s'inscrit dans la logique de TCB hiérarchique, et se révèle nécessaire pour assurer une sécurité complète. Dans le cas d'utilisation d'un processeur en multi-compartiment, la valeur du registre de CID contrôlé par le LTA ne peut pas être le CID entier utilisé par le mécanisme de protection. En effet, le LTA pourrait sinon prendre l'identité de n'importe quel compartiment de la plateforme, notamment d'autres LTA, voire du GTA ce qui annulerait la hiérarchisation de confiance proposée par notre modèle.

Le registre de CID du ou des processeurs sous le contrôle d'un même LTA ne représente alors qu'une sous-partie du CID entier, et plus exactement son suffixe. Le reste du CID, c'est-à-dire son préfixe, contient lui une valeur fixe. Pour cela, on peut reprendre le mécanisme proposé pour l'utilisation mono-compartiment, en sortant cette partie fixe du

registre de CID hors du processeur. Cette partie fixe peut ainsi être définie à la conception, ou dynamiquement par le GTA. La figure 4.7 présente graphiquement ce mécanisme.

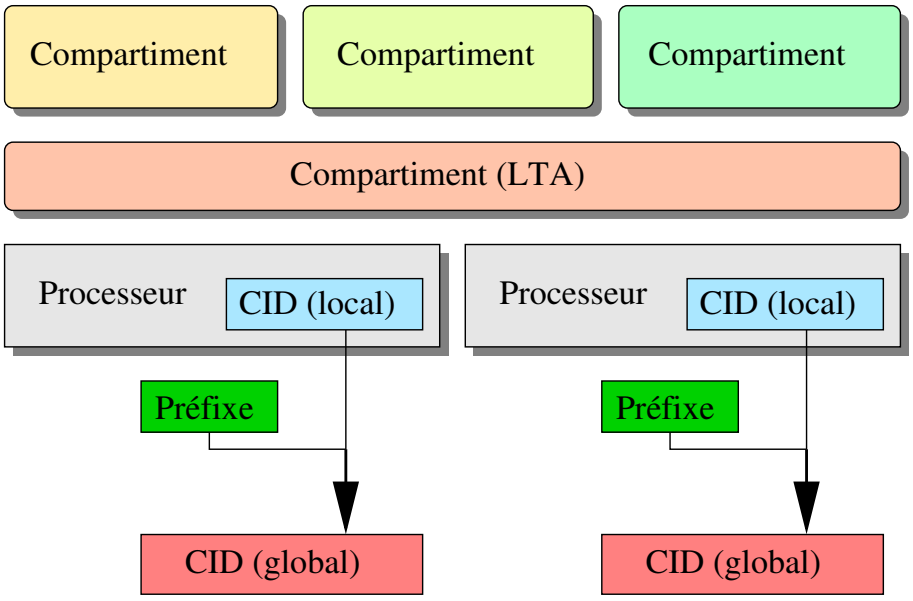


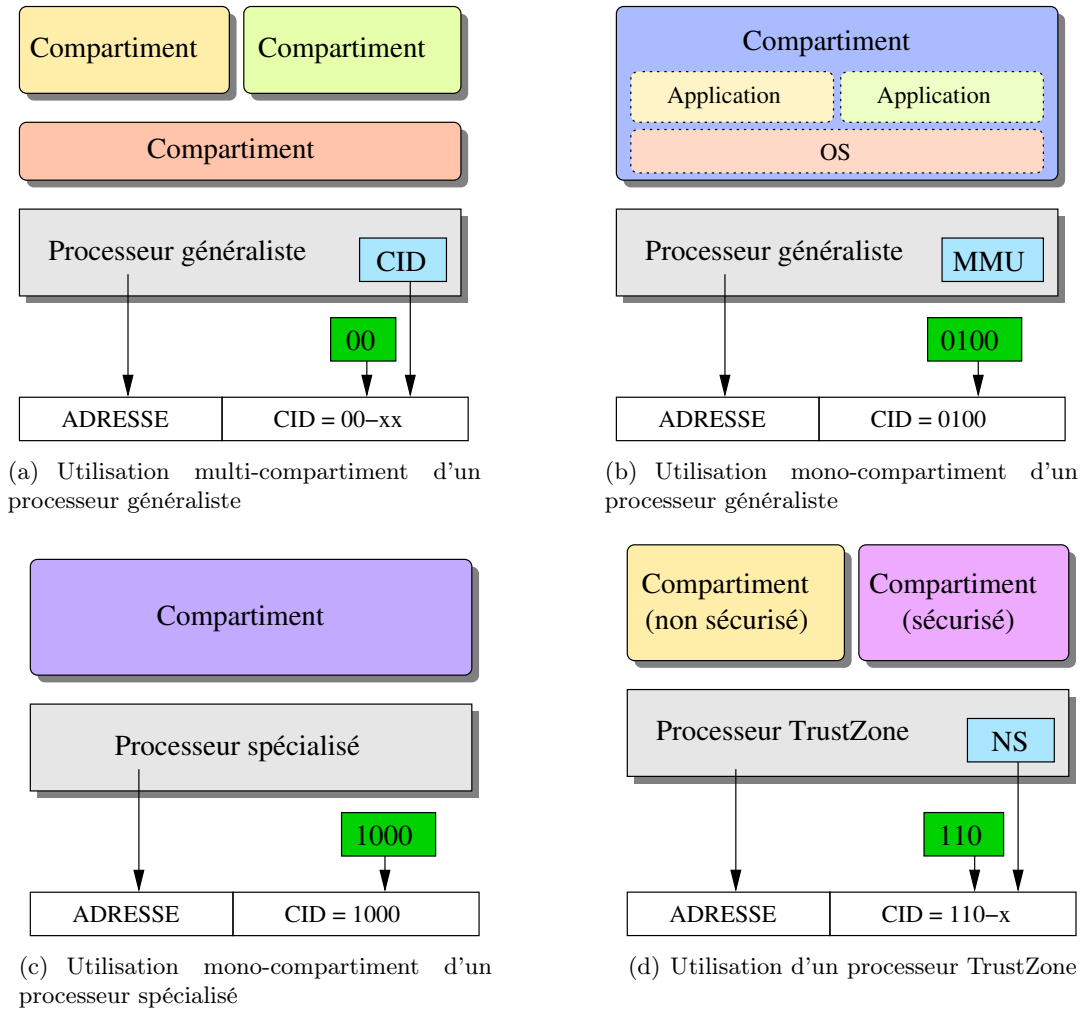
FIGURE 4.7 – Découpage logique du CID en deux parties : le suffixe est contrôlé localement par le LTA, le préfixe est contrôlé globalement et de façon externe par le GTA.

Cette structuration hiérarchique implique que les LTA sont finalement considérés comme des compartiments, puisque possédant chacun un préfixe de CID distinct. Les LTA deviennent donc soumis au mécanisme global de protection. Le suffixe du CID, affecté par la valeur du registre de CID interne à un processeur, reste quant à lui sous le contrôle du LTA, qui ne dispose alors plus que d'une plage de CID afin de pouvoir s'identifier lui-même, ainsi que les compartiments qu'il exécute.

On notera que dans le cas d'utilisation d'un processeur en mono-compartiment, le préfixe du CID, extérieur au processeur et contrôlé par le GTA, représente déjà le CID entier.

Enfin, grâce à cette technique de structuration, l'approche *TrustZone* proposée par *ARM* devient compatible avec l'approche multi-compartiment. Un processeur TrustZone est effectivement un cas particulier de processeur multi-compartiment, puisqu'il peut exécuter deux compartiments, dont un LTA. Le bit *NS*, fourni automatiquement par le processeur sur son interface externe, sert alors de bit de suffixe à un registre de CID externe.

La figure 4.8 résume graphiquement la compatibilité de tous les types de processeurs pouvant composer un MPSoC avec l'approche multi-compartiment, ainsi que la structuration du CID en conséquence.



(a) Utilisation multi-compartiment d'un processeur généraliste

(b) Utilisation mono-compartiment d'un processeur généraliste

(c) Utilisation mono-compartiment d'un processeur spécialisé

(d) Utilisation d'un processeur TrustZone

FIGURE 4.8 – Compatibilité de différents types de processeur avec l'approche multi-compartiment.

4.2.3 Périphériques DMA

Puisque les périphériques initiateurs, ayant une capacité DMA, sont capables d'accéder directement à l'espace d'adressage, de la même façon que les processeurs, ces périphériques sont également concernés par la problématique d'identification définie par l'approche multi-compartiment.

Dans un premier temps, on présente un moyen de rendre compatible un périphérique DMA avec l'approche multi-compartiment, dans le cadre d'une utilisation mono-compartiment. Ensuite, on s'intéresse à une utilisation multi-compartiment des périphériques DMA, c'est-à-dire à leur partage entre plusieurs compartiments, en distinguant deux cas, selon que le périphérique DMA est de type mémoire à mémoire, ou mémoire à interface externe.

4.2.3.1 Utilisation mono-compartiment

Les périphériques DMA ont la particularité de ne pas être complètement autonome, dans le sens où les transferts qu'ils réalisent dans l'espace d'adressage sont toujours commandés par une entité logique extérieure, en l'occurrence un compartiment.

Pour assurer la compatibilité d'un tel périphérique avec une utilisation mono-compartiment, nous appliquons un principe d'héritage. Lorsqu'un compartiment configure un périphérique DMA, c'est-à-dire qu'il définit le transfert mémoire à effectuer, il réalise cette configuration par une suite d'une ou plusieurs transactions à destination des registres de configuration du périphérique. L'identité du compartiment commanditaire est inclus dans ces transactions de configuration, grâce à la propagation du CID. Comme le montre la figure 4.9 et au coût d'une modification mineure, le périphérique DMA peut donc réutiliser le CID du compartiment qui l'a configuré pour réaliser le transfert mémoire effectif. Les accès dans l'espace d'adressage effectués par le périphérique DMA, mais pour le compte d'un compartiment donné, héritent alors des mêmes droits d'accès que ceux initialement définis pour le compartiment.

Ici, on considère que les périphériques DMA sont des composants entièrement matériels, ce qui nous assure que ce principe d'héritage est fiable et non corruptible.

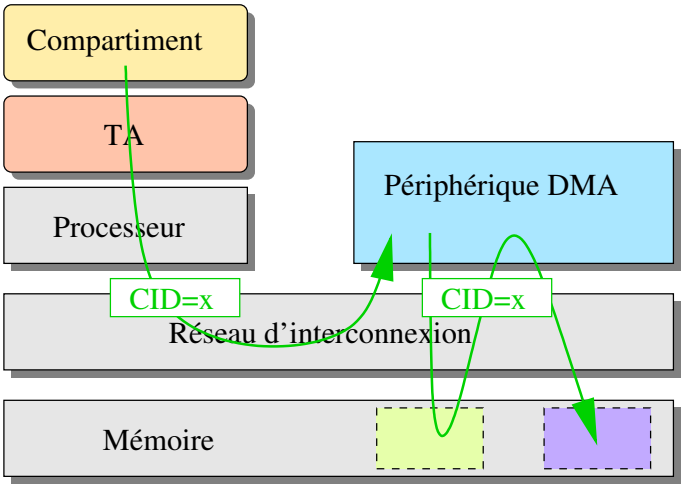


FIGURE 4.9 – Exemple d'héritage du CID pour un transfert de périphérique DMA de type mémoire à mémoire.

Aussi mineure soit-elle, si la modification du périphérique DMA s'avère impossible, le mécanisme peut être réalisé de façon externe, par l'intermédiaire d'une mince couche matérielle de sécurisation entre le périphérique DMA et le réseau d'interconnexion. Ce module, qui se branche sur les deux interfaces du périphérique, respectivement l'interface cible pour la configuration et l'interface initiateur pour le transfert, récupère le CID du compartiment commanditaire sur la première interface et peut ainsi étiqueter toutes les transactions de la deuxième interface, avec ce même CID, avant leur introduction sur le réseau d'interconnexion.

Les avantages de la compatibilité des périphériques DMA avec l'approche multi-compartiment sont importants. Avec la technique de virtualisation des périphériques par d'assignation directe, présentée dans la section 3.1.3.3, les instructions de transfert émises par les machines virtuelles doivent d'abord être validées, c'est-à-dire filtrées par l'hyperviseur. Ici, mêmes les compartiments qui ne sont pas des LTA (donc qui sont potentiellement des applications ou des machines virtuelles) sont capables d'utiliser un périphérique DMA directement. Enfin, ce mécanisme d'héritage montre que la notion de compartiment représente en réalité une entité logique transversale, ayant une capacité migratoire au sein de la plateforme.

4.2.3.2 Utilisation multi-compartiment

Comme nous l'avons expliqué dans la section 2.2.3.2, il existe deux types de périphériques DMA : les périphériques DMA de type mémoire vers mémoire et de type mémoire vers interface externe. Le rôle fondamental d'un DMA est de transférer le contenu d'une région mémoire vers une autre, respectivement par des lectures et des écritures. Dans le cas d'un périphérique DMA de type mémoire à mémoire, les transactions de lectures et les écritures s'exécutent toutes dans l'espace d'adressage de la plateforme. La globalité du transfert est donc automatiquement soumise au contrôle d'accès mis en place par l'approche multi-compartiment. Pour les périphériques DMA de type mémoire vers interface externe (comme un contrôle disque ou réseau), seul l'accès à l'espace d'adressage subit le contrôle d'accès. En revanche, l'accès à l'interface externe (espace disque ou réseau) n'appartient pas à l'espace d'adressage de la plateforme, il est donc hors de tout contrôle. Cette caractéristique rend très difficile la mise en place d'un mécanisme systématique et générique pour gérer le partage de ce type de périphérique DMA. Il existe alors deux solutions : soit le périphérique doit être transformé de façon *ad hoc* pour supporter nativement l'approche multi-compartiment, souvent au prix de modifications profondes, voire d'une refonte complète du composant ; soit on peut réutiliser des techniques de virtualisation, telles que l'émulation, la paravirtualisation ou encore l'approche hybride, pour lesquelles un seul intervenant utilise vraiment le périphérique, ce qui est alors couvert par le cas d'utilisation mono-compartiment.

En reprenant le concept d'héritage présenté précédemment, et moyennant des modifications matérielles supplémentaires, nous proposons une solution pour qu'un périphérique DMA de type mémoire vers mémoire puisse être partagé, c'est-à-dire supporter des requêtes de configuration concurrentes, provenant de plusieurs compartiments distincts [4].

On considère un cas de périphérique DMA minimal, présentant trois registres de configuration pour effectuer un transfert mémoire : un registre pour indiquer l'adresse source, un registre pour indiquer l'adresse destination et enfin, un registre pour indiquer la taille du transfert (en octets). Si cette série de registres est répliquée, chaque compartiment qui a l'utilité du périphérique DMA peut configurer indépendamment sa propre série de registres. Comme le représente la figure 4.10, la valeur du CID qui accompagne les transactions de

configuration est utilisée comme un démultiplexeur pour qu'un compartiment puisse accéder à sa série de registres de façon totalement transparente. Enfin, grâce à une politique d'arbitrage interne (par exemple, via une stratégie de tourniquet ou une stratégie par priorité fixe), le périphérique DMA peut décider quel transfert effectuer parmi les différentes requêtes. Comme pour l'utilisation mono-compartiment, le moteur de transfert hérite alors du CID de la série choisie pour réaliser les transactions, qui bénéficient ainsi du contrôle d'accès.

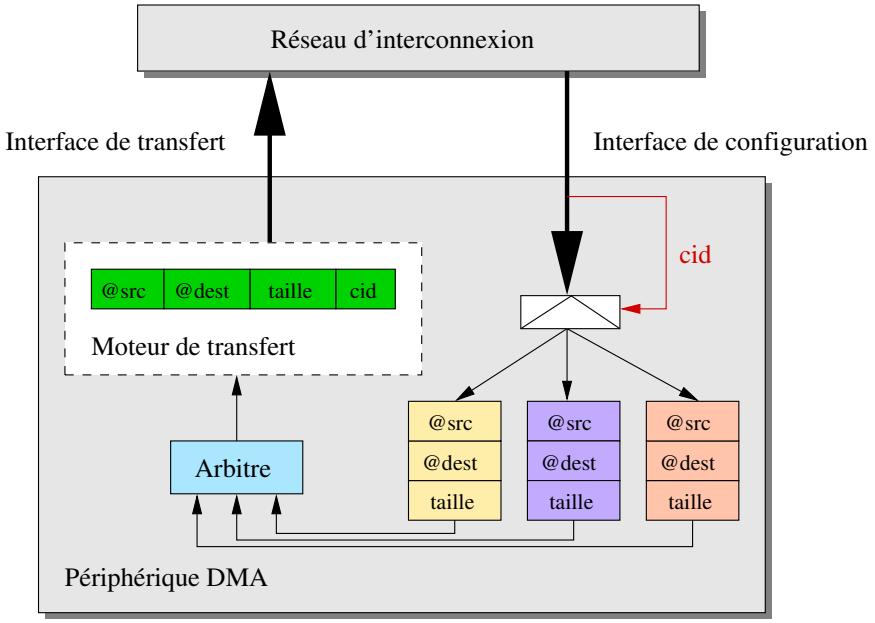


FIGURE 4.10 – Périphérique DMA supportant l'accès concurrent de plusieurs compartiments.

4.3 Mise en œuvre de la protection

Une fois que tous les composants initiateurs de la plateforme sont compatibles avec le principe d'identification, c'est-à-dire que toutes les transactions qu'ils émettent peuvent être étiquetées avec un identifiant de compartiment, on peut mettre en œuvre un mécanisme de protection ou contrôle d'accès. Afin d'assurer une protection complète, ce mécanisme prend naturellement place au cœur de la plateforme, à l'endroit où toutes les transactions passent, et peuvent donc être contrôlées : au sein du réseau d'interconnexion.

La prochaine sous-section discute des différentes stratégies que l'on peut adopter pour réaliser le mécanisme de protection. La sous-section suivante présente la gestion globale de la protection au niveau de la plateforme.

4.3.1 Mécanisme de filtrage

De manière générale et comme la figure 4.11 le représente, le mécanisme de protection est matérialisé par des modules matériels de filtrage, qualifiés de « pare-feu » car ils offrent des services de filtrage similaires à ceux fournis dans le monde des réseaux locaux traditionnels. Ces modules reprennent l'idée développée par les solutions d'architecture sécurisée, que nous avons présentées dans l'état de l'art.

Dans l'exemple, ces modules sont placés « côté initiateur », c'est-à-dire entre les composants initiateurs et les entrées du réseau d'interconnexion. Ils peuvent également être placés « côté cible », entre les composants cibles et les sorties du réseau d'interconnexion. Le choix du placement des modules est important : une localisation côté initiateur peut permettre d'éviter d'éventuelles attaques par saturation du réseau d'interconnexion (attaques par déni-de-service) provoquées par un nombre massif d'accès illégaux lancés par des composants initiateurs corrompus. L'approche côté initiateur semble donc toujours plus judicieuse, même si nous étudions les deux approches dans la suite.

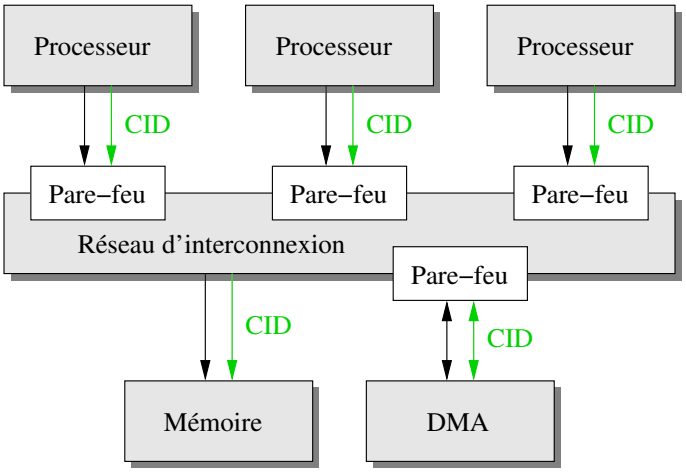


FIGURE 4.11 – Modules pare-feu placés aux entrées du réseau d'interconnexion.

L'algorithme de filtrage réalisé par les modules pare-feu est conceptuellement représenté par une *table de permission*, doublement indexée par l'adresse mémoire d'une transaction et par le CID associé, et contenant les droits d'accès pour chaque combinaison de ces deux entrées. Une telle combinaison associée à un droit d'accès est appelée *règle de droits d'accès*. Lorsqu'une transaction passe dans le réseau d'interconnexion, la table de permission est consultée en fonction de l'adresse mémoire que vise la transaction et de son compartiment émetteur. Les droits d'accès que la table de permission renvoie servent à accepter ou à refuser la transaction.

Cette table de permission peut être implémentée en suivant différentes stratégies, que nous détaillons et discutons dans la suite.

4.3.1.1 Tables distribuées et embarquées, et approche segmentée

Présentation Dans cette stratégie, adoptée par la plupart des solutions d'architecture sécurisée et présentée graphiquement dans la figure 4.12, les règles de droits d'accès de la table de permission sont représentées par des segments. Ceux-ci sont définis par une adresse de base et possiblement une taille (des solutions telles que *DPU* [45] définissent statiquement une taille de segment unique, à la conception). À chaque compartiment correspondent alors un ensemble de segments, associés à des droits d'accès. Cet ensemble de règles de droits d'accès dessinent ainsi la cartographie des zones de la mémoire auxquelles le compartiment a le droit d'accéder. Enfin, cette table de permission est découpée en sous-tables, réparties dans les différents modules pare-feu de la plateforme. Dans le cas d'un filtrage côté initiateur, ce découpage suit la logique suivante : toutes les règles d'accès concernant un compartiment donné sont embarquées dans le module pare-feu adjacent au composant initiateur qui exécute ce compartiment. Dans le cas d'un filtrage côté cible, un module pare-feu contient toutes les règles se rapportant aux segments contenus dans le composant cible adjacent.

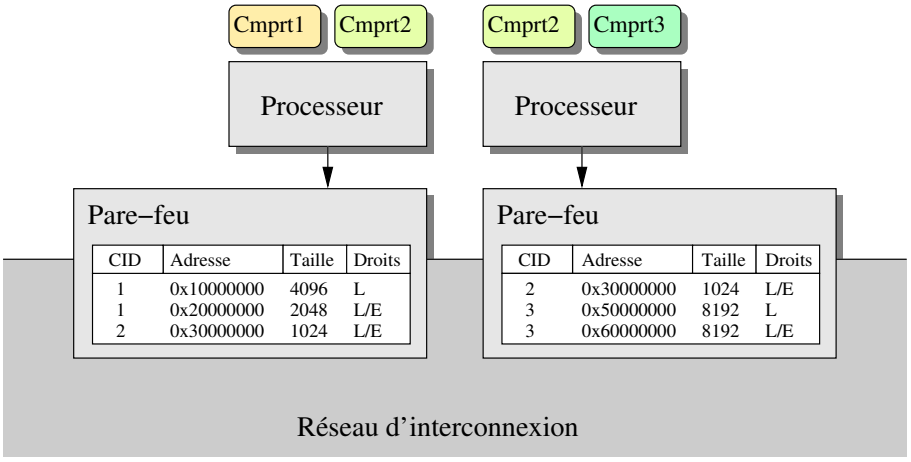


FIGURE 4.12 – Stratégie de segmentation, embarquée dans les modules pare-feu côté initiateur.

Critique Bien que cette stratégie ait l'avantage de pouvoir supporter la définition de segments de taille variable, elle possède de nombreux inconvénients.

Une première difficulté existe dans le cas d'un filtrage côté initiateur, et concerne la cohérence des sous-tables de permission. Pour un compartiment s'exécutant sur plusieurs composants initiateurs, par exemple plusieurs processeurs, les règles de droit d'accès qui le concernent doivent être répliquées dans tous les modules pare-feu adjacents aux composants initiateurs utilisés. Outre le fait que cette réplication occupe de la place dans le peu de mémoire disponible au sein des modules, cela signifie aussi que la moindre mise à jour dans ces règles doit être répercutée dans tous les modules pare-feu qui contiennent effectivement ces règles. La gestion des modules pare-feu s'en retrouve compliquée. Dans le cas d'un

filtrage côté cible, ce problème de cohérence ne se pose pas, puisque les règles embarquées dans un module pare-feu concernent uniquement les segments attachés au composant cible concerné : il n'y a pas de réplication. En revanche, si le composant cible sous-jacent est un composant mémoire, donc *a priori* partagé entre de multiples compartiments, il est probable que la sous-table de permission doive être assez large. Or, pour des raisons de surface matérielle, le nombre de règles qu'il est possible d'embarquer dans un module pare-feu reste très limité [2].

Dans la perspective où plusieurs compartiments peuvent partager la plateforme, et possiblement les mêmes composants physiques (processeurs et périphériques DMA), et requérir l'accès à plusieurs régions dans l'espace d'adressage, le nombre de règles par module pare-feu représente dans tous les cas un goulot d'étranglement.

4.3.1.2 Table en mémoire et approche segmentée

Présentation Afin de contourner les problèmes de cohérence et de scalabilité, cette stratégie préconise que la table de permission soit située dans la mémoire centrale – donc dans l'espace d'adressage partagé – comme le montre la figure 4.13. La table de permission contient des règles de droits d'accès représentées par des segments, définis de la même façon que dans la stratégie précédente. Pour des raisons de sécurité et de performance, il est conseillé de placer la région mémoire contenant la table de permission dans la mémoire embarquée sur la puce. Le peu de mémoire que peuvent embarquer les modules pare-feu, pour un coût en surface matérielle raisonnable, sert alors de mémoire cache pour contenir des copies de règles récemment utilisées. Lorsqu'une transaction arrive dans un module pare-feu, la règle de droit d'accès concernant l'adresse mémoire à laquelle la transaction essaie d'accéder est recherchée. Si cette règle n'est pas déjà dans le cache du module, elle y est rapatriée à partir de la table de permission globale.

Afin de réduire la latence additionnelle introduite par cette stratégie, le mécanisme de rapatriement d'une règle à partir de la table de permission doit être réalisé en matériel. Les modules pare-feu intègrent donc, en plus d'un registre contenant l'adresse mémoire de la table de permission, un automate qui peut effectuer la recherche dans la table de permission d'une règle de droits d'accès en mémoire, ainsi que son ajout dans le cache interne. Cet automate, déclenché lors de défauts de cache, est appelé « automate de traitement des MISS ».

Critique Cette stratégie de centralisation résout entièrement le problème de cohérence de la stratégie précédente. Les règles de droits d'accès ne sont définies qu'une seule fois, dans la table de permission globale. Lors d'une modification de règles dans la table globale,

2. DPU a conduit des expériences dans lesquelles leurs modules pare-feu embarquaient au maximum 16 entrées. Du côté de SECA, leur module pare-feu unique embarquait au maximum 80 entrées, mais dans ce cas, le module était de 10% supérieur en surface au processeur de référence de leur plateforme d'expérimentation.

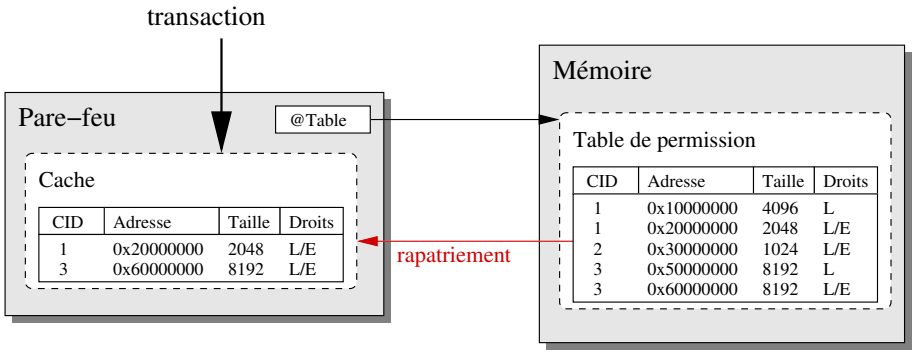


FIGURE 4.13 – Architecture d'un module pare-feu associé avec une table de permission segmentée stockée en mémoire.

il suffit d'invalider par un mécanisme global les éventuelles copies présentes dans les caches des modules pare-feu.

Ne contenant plus que des caches de la table de permission globale, le choix du placement des modules pare-feu (côté initiateur ou côté cible) est également réglé : théoriquement, les modules peuvent être indifféremment placés de n'importe quel côté sans problème de cohérence.

Le problème de scalabilité n'est quant à lui que partiellement résolu. L'utilisation de la mémoire centrale pour contenir la table de permission permet d'éliminer le goulot d'étranglement constitué par la capacité de stockage limitée des modules pare-feu. Le nouveau goulot d'étranglement réside maintenant dans l'approche segmentée. Pour un compartiment donné, les règles de droits d'accès inclues dans la table de permission sont généralement ordonnées en fonction de l'adresse de base des segments auxquels elles se rapportent. Pour chercher la règle qui concerne une adresse donnée, l'automate de traitement des MISS doit alors parcourir l'ensemble des règles, du début à la fin. Quand le nombre de règles définies pour un compartiment augmente, le coût de la recherche peut devenir important. En outre, la mise à jour d'une table de segmentation ainsi triée peut se révéler assez lourde : l'insertion d'une nouvelle règle oblige en effet à décaler toutes les règles suivantes pour respecter l'ordre des adresses de base des segments.

4.3.1.3 Table en mémoire et approche paginée

La stratégie que nous proposons consiste à partitionner l'espace d'adressage en pages de taille fixe. La table de permission ne contient plus que les droits d'accès définis pour chaque page. Lorsqu'une transaction arrive, les bits de poids fort de l'adresse mémoire qu'elle tente d'accéder servent à l'automate de traitement des MISS, d'index dans la table de permission, que l'on appelle alors « table des pages », pour en extraire directement et en temps borné les droits d'accès associés.

Dans une première approche, une unique table des pages peut régir l'accès à l'ensemble de l'espace d'adressage. Pour une page donnée, la table contient alors les droits d'accès pour

l'ensemble des compartiments de la plateforme. Cette approche entraine potentiellement un énorme gâchis de mémoire, dans la table des pages mais surtout dans les caches embarqués par les modules pare-feu : il est effectivement fort improbable qu'une page soit partagée par tous les compartiments de la plateforme, et donc que l'espace réservé pour contenir les droits de tous les compartiments de la plateforme soit réellement utilisé.

Pour résoudre ce problème, la deuxième approche consiste à construire une table des pages par compartiment. Comme dans les techniques de pagination traditionnelles, on peut minimiser l'encombrement de ces tables, grâce à des tables des pages multi-niveaux (au moins deux). Ce schéma est présenté dans la figure 4.14. Ce mécanisme de pagination multi-niveau existe depuis quelques décennies, ce qui permet de profiter du fait que les systèmes se sont largement adaptés au concept de page, d'un point de vue matériel et logiciel. Par exemple, le découpage de l'espace d'adressage d'une plateforme suit généralement cette granularité de page : l'agencement des espaces d'adressage de chaque périphérique de la plateforme dans l'espace d'adressage global s'aligne sur la taille d'une page. Du côté logiciel, la plupart des systèmes d'exploitation intègrent également la granularité au niveau page, et la construction des tables des pages, ainsi que leur maintien, sont deux concepts bien connus et maitrisés.

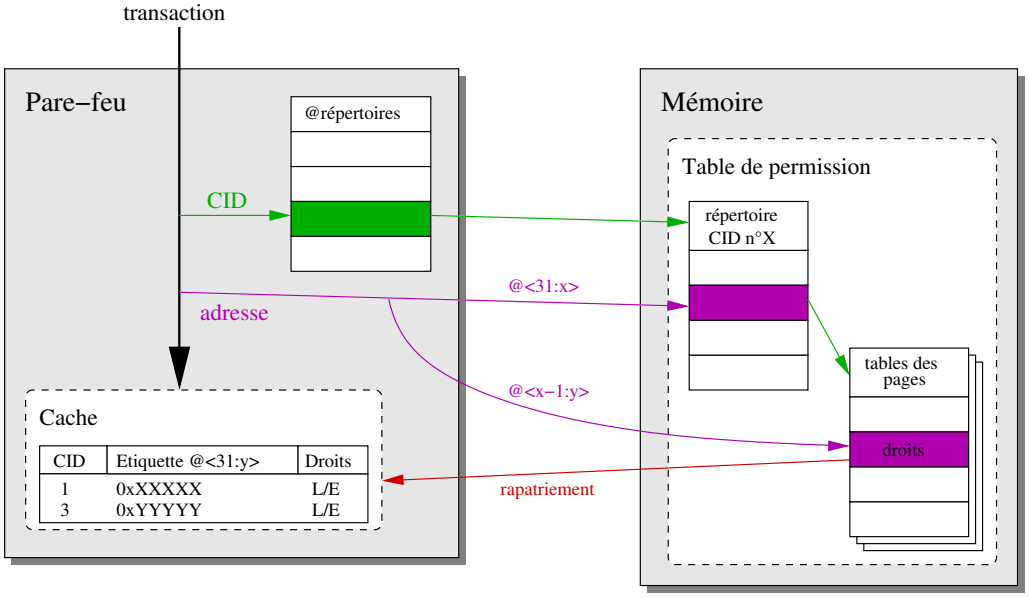


FIGURE 4.14 – Architecture d'un module pare-feu associé avec une table de permission paginée stockée en mémoire.

Un inconvénient qui semble subsister dans cette approche réside dans l'espace qu'occupent les tables des pages dans la mémoire centrale. Cependant, si l'on considère que l'augmentation de la densité d'intégration permet d'embarquer de plus en plus de mémoire, cet inconvénient est largement atténué. Il l'est encore plus si l'on prend en compte le gain en flexibilité apporté par cette stratégie de protection. En outre, des projets tels que *Mondrian Memory Protection* [50] ou *Guarded Page Table* [51], ont déjà étudié ce problème

de taille de tables des pages (dans le cadre de la mémoire virtuelle paginée) et proposent des architectures de tables de pages efficaces qui pourraient tout à fait être utilisables dans notre cas.

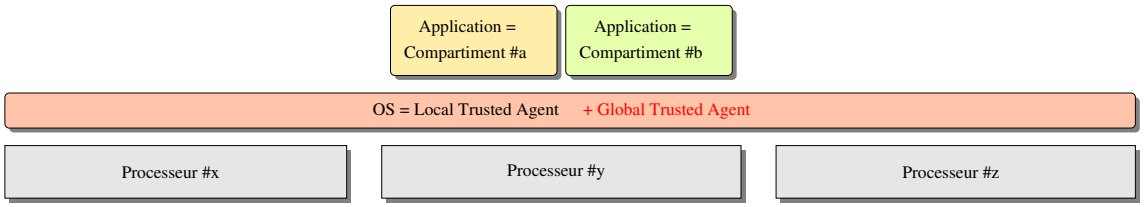
4.3.2 Gestion globale

Dans l'approche que nous présentons, chaque processeur partageable est sous le contrôle d'un agent de confiance local (*LTA*), essentiellement pour organiser l'ordonnancement des compartiments utilisateurs et la gestion des plages locales de CID. La définition des préfixes des CID, pour ces processeurs multi-compartiments mais aussi pour les processeurs – spécialisés – mono-compartiments, est quant à elle laissée sous le contrôle de l'agent de confiance globale (*GTA*), qui comme son nom le laisse entendre, agit au niveau de la plateforme. Le GTA gère donc les piles logicielles autonomes (LTA), par l'attribution de valeurs ou de plages de CID, mais surtout supervise la définition – dynamique – des droits d'accès à l'espace d'adressage partagé. L'attribution de la mémoire et des périphériques adressables est réalisée par la définition, pour chaque compartiment, d'une table des pages de protection. Ces tables de pages sont donc créées et maintenues par le GTA, qui reçoit et traite également les événements de violations des règles de droits d'accès. On peut considérer le GTA comme un compartiment racine, ayant tous les droits sur la plateforme.

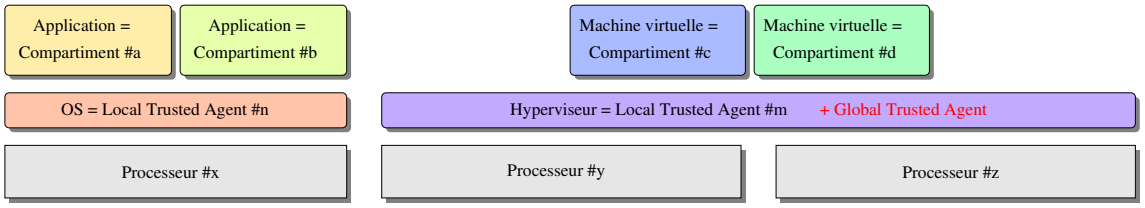
Ce modèle global de TCB hiérarchique peut en réalité être implémenté de différentes façons. Comme l'illustre la figure 4.15, si une plateforme peut être recouverte par une base de confiance unique, par exemple un OS, car la plateforme est homogène, alors l'OS est capable d'agir en tant que LTA sur tous les processeurs mais aussi en tant que GTA pour la gestion de la plateforme. On s'aperçoit dans ce cas que notre modèle est finalement un sur-ensemble du modèle mono-TCB traditionnel. Sur une plateforme, partagée par deux OS, chacun joue le rôle de LTA sur les processeurs qu'il contrôle, tandis que seulement un des deux agit en tant que GTA. Le modèle peut également satisfaire des cas beaucoup plus complexes, mais fréquents dans les systèmes orientés multimédia. Un hyperviseur contrôlant plusieurs compartiments complexes (OS invités) sur un processeur généraliste, et un OS exécutant des compartiments simples (applications) sur un processeur spécialisé, représentent tous deux des LTA sur leur processeur. Une pile dédiée s'exécutant sur un processeur indépendant joue le rôle de GTA et administre la plateforme.

4.4 Partage des périphériques

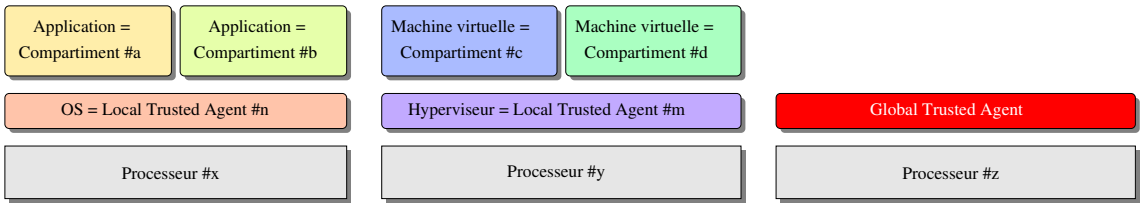
Jusqu'à présent, nous avons présenté la compatibilité d'une plateforme typique avec l'approche multi-compartiment. Cela concernait principalement les composants initiateurs, qui doivent respecter le principe d'identification, et la mise en place dans le réseau d'interconnexion d'un mécanisme de protection de l'espace d'adressage partagé. Il reste une classe de composants qui n'a pas été abordée : les périphériques cibles.



(a) Unique pile logicielle autonome qui joue le rôle de LTA et GTA



(b) Multiples piles logicielles autonomes : toutes sont des LTA, mais une seule est à la fois le GTA



(c) Paranoïaque : le GTA est complètement séparé des LTA

FIGURE 4.15 – Différentes applications du modèle de TCB hiérarchique dans l'approche multi-compartiment.

Cette section se propose dans un premier temps d'évoquer brièvement la compatibilité des périphériques cibles avec l'approche multi-compartiment. Dans un deuxième temps, le problème des interruptions matérielles est abordé.

4.4.1 Périphériques cibles

Ne pouvant pas initier de transactions dans le réseau d'interconnexion, les périphériques cibles ont un statut un peu particulier dans l'approche multi-compartiment. En effet, la seule « compatibilité » qu'ils peuvent offrir avec cette approche est d'être partageables entre plusieurs compartiments.

Tout d'abord, il faut remarquer que les périphériques cibles peuvent être utilisés de façon exclusive par un seul compartiment. D'ailleurs, dans le cadre de systèmes orientés multimédia, on s'aperçoit que la plupart des périphériques cibles sont généralement utilisés par un seul acteur logiciel.

Autrement, puisque les périphériques cibles sont par définition en contact avec le monde extérieur, donc non contrôlables par le mécanisme de protection, le même raisonnement que pour les périphériques DMA de type mémoire à interface externe s'applique. Il est très difficile de mettre en place un mécanisme systématique et générique pour assurer le partage de ces périphériques. On peut alors utiliser les mêmes techniques que celles proposées par

la virtualisation, telles que l'émulation, la paravirtualisation ou l'approche hybride, qui reviennent toutes à un partitionnement strict de l'ensemble des périphériques cibles entre les compartiments. Autrement, on peut modifier de façon *ad hoc* chaque périphérique cible, si cela est possible, pour le rendre partageable de façon native.

Même si ce travail ne s'attarde pas d'avantage dans la résolution de ce problème, on pourra noter que les futures solutions, destinées à la virtualisation, et concernant le partage entre plusieurs machines virtuelles de périphériques cibles ou DMA de type mémoire vers interface externe, seront *a priori* utilisables presque immédiatement au sein de notre approche.

4.4.2 Gestion des interruptions matérielles

Les interruptions matérielles sont généralement utilisées par les périphériques DMA, ce qui rend leur étude particulièrement importante dans l'approche multi-compartiment.

4.4.2.1 Introduction

Souvent, le réseau de lignes d'interruption d'une plateforme est fixe, dans le sens où les fils reliant les périphériques aux processeurs sont câblés statiquement à la conception. Dans les systèmes traditionnels, lorsque la plateforme est couverte d'une unique couche logicielle de confiance, cette caractéristique n'est pas problématique. Par exemple, sur une plateforme mono-processeur exécutant un OS, toutes les lignes d'interruption sont connectées au processeur, et gérées par l'OS. Sur une plateforme multi-processeur exécutant un OS ou un hyperviseur, les lignes d'interruption peuvent être attachées à n'importe quel processeur, car ensuite gérées par l'OS ou l'hyperviseur qui couvre tous les processeurs.

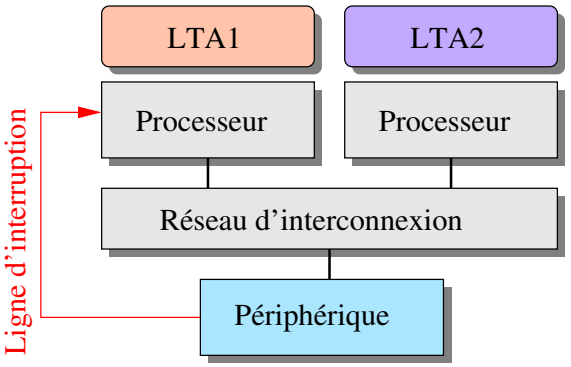


FIGURE 4.16 – Ligne d'interruption d'un périphérique attachée à un seul processeur de la plateforme.

Dans l'approche multi-compartiment, le co-hébergement de plusieurs piles logicielles autonomes, utilisant chacune un sous-ensemble exclusif des processeurs de la plateforme, rend un tel réseau de lignes d'interruption statique problématique. Avec l'architecture actuelle, un LTA, que l'on nomme LTA1 pour l'exemple, recevrait en effet toutes les interruptions

d'un périphérique rattaché à un des processeurs que LTA1 contrôle. Cet exemple est schématisé dans la figure 4.16. Notons que c'est LTA1 qui reçoit les interruptions, même si c'est un compartiment qui s'exécute au-dessus de LTA1 qui a programmé le périphérique, car ce type d'événement est toujours géré par le logiciel qui fonctionne dans le mode le plus privilégié du processeur destinataire. Maintenant, s'il s'avère que LTA1 partage ce périphérique avec d'autres LTA de la plateforme, par exemple LTA2, alors ces interruptions concerneront parfois LTA2. Il faudrait alors mettre en place un mécanisme pour que LTA1 puisse signaler à LTA2, qui est destinataire de l'interruption mais qui s'exécute sur un autre processeur de la plateforme, que ce dernier a reçu une interruption afin qu'il puisse la traiter correctement. Cette approche est dangereuse car elle peut permettre des attaques de type déni-de-service. Si LTA2 est corrompu, il peut sciemment provoquer des interruptions intempestives, qui gêneront l'exécution de LTA1, récepteur de ces interruptions. Dans le sens inverse, si LTA1 est corrompu, il peut ne jamais signaler des interruptions qu'il a reçues pour LTA2, et ainsi en gêner le bon fonctionnement.

Le premier problème à résoudre consiste donc à rediriger automatiquement et dynamiquement les interruptions vers le bon processeur, et donc le bon LTA, en fonction du compartiment destinataire de l'interruption.

4.4.2.2 Redirection processeur

ICU partagée Afin de permettre à un périphérique d'envoyer des interruptions sur différents processeurs, une première solution consiste à utiliser une ICU partagée et configurable au niveau plateforme. Cette ICU reçoit en entrée toutes les lignes d'interruption de la plateforme, et possède en sortie des lignes d'interruption vers tous les processeurs de la plateforme. Cette approche pose de multiples problèmes. Premièrement, cela engendre une concentration des fils d'interruption dans la plateforme, ce qui n'est pas forcément évident à gérer au moment de la conception (notamment au moment du routage du circuit). Mais surtout, la configuration de l'ICU partagée implique nécessairement l'agent de confiance global pour associer, à un moment donné, un périphérique à un processeur. Cette association, bien que non problématique pour l'utilisation exclusive d'un périphérique par un seul compartiment, interdit par contre l'utilisation entrelacée d'un même périphérique par plusieurs compartiments puisque le périphérique n'est attaché qu'à un seul processeur à un instant donné.

Interruptions par transactions implicites Récemment, le protocole *OCP* [49] a intégré les signaux de contrôle, tels que les interruptions ou encore le signal de redémarrage (reset), à côté des signaux dédiés au trafic transactionnel de données. Même si dans la plupart des plateformes actuelles, le réseau d'interruption continue à être implémenté indépendamment par des fils *ad hoc*, les architectures futures devraient permettre la transmission des interruptions par le réseau d'interconnexion. Les réseaux sur puce *ANoC* [52]

et *MANGO* [53] proposent d'ailleurs cette fonctionnalité. En configurant le *Network Interface Controller* (*NIC*) adjacent à un périphérique, qui transforme le signal d'interruption en transaction, l'interruption peut être transmise à n'importe quel autre NIC, adjacent à un processeur, qui retransforme alors la transaction en signal d'interruption. Cependant, comme le montre la figure 4.17, on voit que l'interface côté périphérique, et côté processeur, ne change pas : même si elle est transmise par le réseau, la ligne d'interruption commence par un fil, et finit par un fil. Cette solution possède les mêmes caractéristiques que la technique d'ICU partagée au niveau plateforme, et ne résout que le problème de routage des lignes d'interruption.

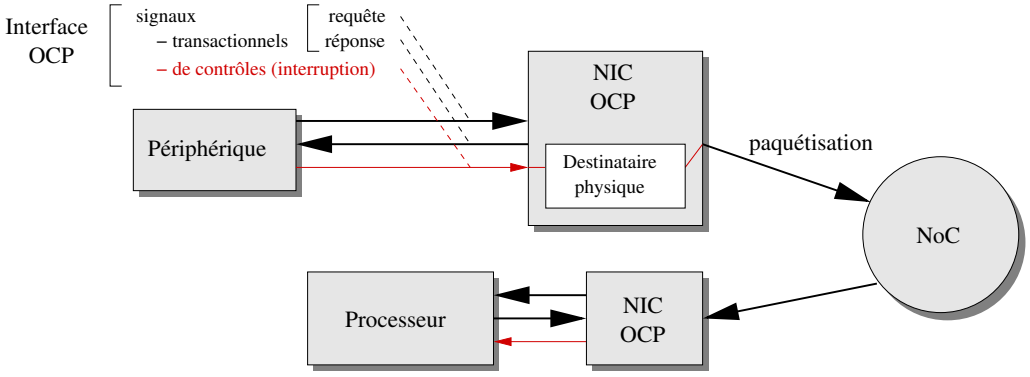


FIGURE 4.17 – Transmission des interruptions par le réseau sur puce.

Interruptions par transactions explicites Pour des raisons de flexibilité et de sécurité, on se propose d'aller plus loin que la solution précédente, toujours en transmettant les interruptions par le réseau sur puce, mais de bout en bout, un peu comme le proposent les standards *PCI* [54] et *PCI-Express* [55] pour les ordinateurs de bureaux avec les *Message Signaled Interrupts* (*MSI*).

Les ICU récentes savent généralement recevoir les interruptions inter-processeur (ou *IPI*, pour *Inter-Processor Interrupt*), qui sont déjà transmises par le réseau : ces interruptions sont provoquées par logiciel et entièrement transmises par des transactions. Ce comportement peut donc être généralisé à tous les types d'interruptions, et contribue ainsi à faire disparaitre les signaux d'interruption traditionnels.

Côté périphériques, les périphériques DMA possèdent déjà une interface initiateur, leur permettant d'émettre des transactions pour effectuer les transferts mémoire. Le signal d'interruption classique peut donc disparaitre, au profit de cette interface, alors réutilisée pour réaliser la transmission des interruptions par transactions.

Le coût global de la transmission des interruptions par transactions reste assez faible au vu de la flexibilité gagnée. Le seul inconvénient potentiel de cette approche réside dans la latence du réseau d'interconnexion pour transmettre les transactions d'interruption, mais il peut être discuté. Du point de vue bande passante, les réseaux sur puce peuvent supporter l'introduction de ce nouveau trafic. Ensuite, les réseaux sur puce fournissent la plupart du

temps des fonctionnalités de qualité de service (QoS) permettant de garantir une latence maximale. Enfin, la gestion d'une interruption, au niveau du processeur, entraine inévitablement de nombreuses transactions sur le réseau : le code du gestionnaire d'interruption n'est pas forcément présent dans le cache d'instructions, et il faut ensuite quelques transactions pour traiter effectivement l'interruption, au moins pour l'acquitter. Dans cette perspective, on peut admettre que l'introduction d'une unique transaction supplémentaire reste négligeable.

Pour qu'un périphérique DMA sache à quelle ICU il doit envoyer la transaction d'interruption, nous proposons la méthode suivante. Le compartiment, qui programme le périphérique DMA, spécifie simplement au périphérique la plage d'adresses de l'ICU à laquelle le périphérique devra envoyer l'interruption à la fin du transfert. La plage d'adresses d'une ICU doit alors être ouverte dans les droits d'accès du compartiment initiateur du transfert, puisque la transaction d'interruption sera envoyée avec le même CID que ce compartiment.

Les avantages de cette approche par transactions explicites sont la flexibilité et la sécurité. Il n'y a effectivement plus aucun obstacle pour que le périphérique DMA puisse être utilisé de façon entrelacée par plusieurs compartiments : l'interruption envoyée à la fin d'un transfert n'empêche pas le périphérique de commencer un nouveau transfert, pour le compte d'un autre compartiment, voire d'envoyer une nouvelle interruption à la fin de ce nouveau transfert, et ce, sans avoir reçu l'acquittement relatif au premier transfert. En ce qui concerne la sécurité, les interruptions, qui sont maintenant des transactions, sont soumises au mécanisme général de contrôle d'accès de l'approche multi-compartiment.

4.4.2.3 Redirection compartiment

L'approche de transmission des interruptions par transactions explicites permet à un LTA de recevoir directement les interruptions d'un périphérique que ce LTA, ou qu'un des compartiments s'exécutant au-dessus de ce LTA, a programmé. Dans le cas où le LTA est un noyau de système d'exploitation, et les compartiments qu'il exécute sont des applications, il est fort probable que les interruptions reçues soit toujours destinées au LTA lui-même. Dans les architectures d'OS traditionnels, les applications ne programment jamais directement les périphériques ; elles font appel au noyau, qui contient les pilotes de périphériques. Cependant, ce comportement est différent dans le cas où le LTA est un hyperviseur, et que les compartiments – complexes – que le LTA exécute sont des OS invités. Ces OS invités peuvent programmer directement les périphériques, puisqu'ils contiennent des pilotes de périphériques. Cela a d'ailleurs été rendu possible, dans notre approche, grâce au mécanisme d'héritage de CID. Pourtant, étant un événement privilégié, la réception d'une interruption, relative à la programmation d'un transfert par un OS invité, sera toujours reçue par le hyperviseur sous-jacent.

Nous nous intéressons donc aux possibles inconvénients de cette situation, d'abord en l'étudiant dans le cadre de la virtualisation, puis en proposant des améliorations.

Comportement de la virtualisation Dans le cadre de la virtualisation, et comme l'explique Robert Rose [56], lorsqu'une interruption arrive sur un processeur exécutant un hyperviseur et des machines virtuelles, le comportement général est le suivant : l'hyperviseur prend la main et exécute la machine virtuelle destinataire de l'interruption. La probabilité que l'interruption soit destinée à la machine virtuelle en cours d'exécution s'amenuise au fur et à mesure que le nombre de machines virtuelles augmente. À cause du surcoût de temps nécessaire pour changer de contexte et exécuter la bonne machine virtuelle, l'hyperviseur Denali [57, 58] opère un autre fonctionnement, nommé *interrupt queueing*, qui consiste à mettre l'interruption en file d'attente. Elle sera alors distribuée à la machine virtuelle destinataire lorsque celle ci sera exécutée la prochaine fois.

On peut relever deux caractéristiques intéressantes du comportement mis en place par Denali. La première est que l'interruption n'est pas traitée immédiatement, ce qui justifie indirectement que la latence additionnelle causée par le concept de transmission explicite des interruptions par le réseau d'interconnexion proposé précédemment est négligeable. Ensuite, on voit que l'hyperviseur est toujours obligé d'intervenir lorsqu'une interruption survient. Ce comportement est problématique pour plusieurs raisons. Premièrement, une machine virtuelle corrompue peut provoquer beaucoup d'interruptions, ce qui gênera le fonctionnement global : l'hyperviseur étant réveillé trop souvent, la disponibilité des ressources processeur pour l'exécution des machines virtuelles est réduite. Deuxièmement, la machine virtuelle destinataire de l'interruption, qu'elle soit la machine virtuelle en cours d'exécution ou non, peut avoir masqué l'interruption, généralement par l'intermédiaire de son modèle virtuel d'ICU. Pourtant, l'interruption physique correspondante n'est pas nécessairement masquée, ce qui oblige l'hyperviseur à être réveillé même si l'interruption est finalement écartée.

Approche multi-compartiment Dans le cadre de l'approche multi-compartiment, on voudrait que les compartiments complexes – OS invités – ne soient interrompus par le LTA – hyperviseur – que pour les interruptions les concernant directement, afin de réduire le coût relatif aux changements de contexte, ceci pour des raisons de performance mais aussi de sécurité (déni-de-service). Pour parvenir à ce résultat, nous proposons quelques modifications, principalement au niveau des ICU [3].

Côté périphérique, puisque l'interruption est maintenant transmise comme une transaction, elle est étiquetée avec un identifiant de compartiment, celui pour qui est destinée l'interruption.

Côté ICU, les deux champs de bits habituels, *statut*, qui contient l'état courant des lignes d'interruption, et *masque*, qui permet de masquer séparément chacune de ces lignes d'interruption, sont répliqués autant de fois que le nombre de compartiments que le processeur attaché à l'ICU peut accueillir. Lorsqu'une transaction d'interruption est reçue par l'ICU, le CID qui étiquette la transaction sert de démultiplexeur pour adresser le champ de bits *statut* correspondant. De la même façon, lorsqu'un compartiment accède à l'ICU

pour vérifier ou modifier les champs de bits, il n'a accès qu'à la série de champs qui lui correspondent.

En ce qui concerne le signal d'interruption de sortie, liant une ICU et le processeur auquel elle est attachée, on suppose que l'ICU a connaissance du CID du compartiment en cours d'exécution sur le processeur. Cela permet ainsi à l'ICU de relayer au processeur uniquement les interruptions concernant le compartiment en cours d'exécution, tandis que les interruptions concernant les autres compartiments sont automatiquement mises en attente. Elles seront relayées quand les compartiments en question seront à leur tour exécutés sur le processeur. Cette solution évite ainsi que le processeur reçoive les interruptions concernant les autres compartiments que celui en cours d'exécution. Tout ce mécanisme est détaillé graphiquement dans la figure 4.18.

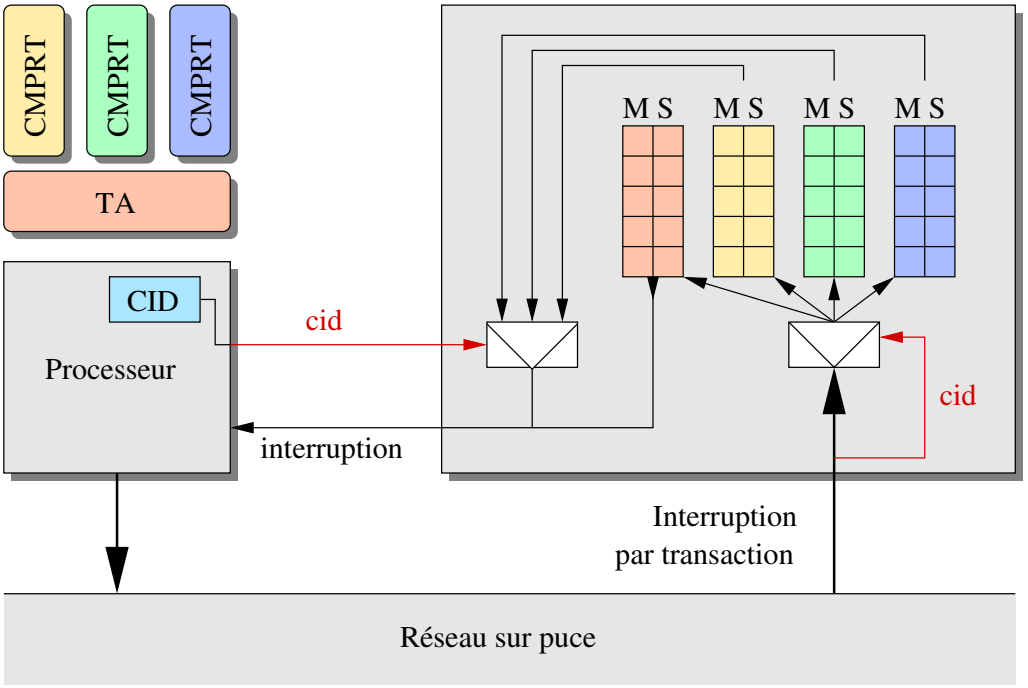


FIGURE 4.18 – ICU modifiée pour l'approche multi-compartiment.

L'ICU peut toutefois être configurée, par le LTA, pour délivrer certaines interruptions même si le compartiment en cours d'exécution n'est pas le destinataire. Les interruptions destinées aux LTA en sont un exemple, puisqu'elles doivent généralement être traitées immédiatement : on peut typiquement citer l'interruption d'horloge qui permet l'ordonnancement des compartiments sur le processeur.

On pourra souligner que cette solution est réutilisable dans le cadre de la virtualisation. On pourra également noter que l'hyperviseur n'est plus obligé de fournir un modèle virtuel d'ICU, les OS invités pouvant accéder directement à l'ICU physique disponible.

4.5 Conclusion

Le co-hébergement sécurisé et flexible de plusieurs piles logicielles autonomes (OS, hyperviseur, pile dédiée) sur une même plateforme multiprocesseur hétérogène à mémoire partagée est possible. Ce co-hébergement est même multi-niveau, puisqu'il supporte de façon cohérente et homogène les applications ou OS invités s'exécutant au-dessus des piles logicielles autonomes.

Le bénéfice apporté par un tel co-hébergement est réel, puisqu'il permet de s'adapter parfaitement aux exigences des systèmes embarqués actuels et futurs, notamment ceux orientés multimédia.

Dans ce chapitre, nous avons donc exposé les deux principes fondateurs d'un nouveau modèle de confiance, l'approche multi-compartiment. L'identification de toutes les transactions matérielles du système permet la mise en place d'un mécanisme de protection efficace et flexible de type paginé.

Les composants initiateurs (processeurs, périphériques à capacité DMA), concernés par le principe d'identification, peuvent être rendus compatibles avec l'approche multi-compartiment grâce à quelques modifications très peu coûteuses. Hormis les processeurs spécialisés, intrinsèquement non partageables, les processeurs généralistes et les périphériques DMA de type mémoire à mémoire peuvent être partagés entre plusieurs compartiments.

Le mécanisme de protection est placé au cœur de la plateforme, à l'endroit où transitent toutes les transactions, dans le réseau d'interconnexion. Des modules pare-feu, placés à l'entrée de ce réseau, sont chargés de l'analyse des transactions, afin de vérifier leur légalité. L'algorithme de filtrage repose sur l'utilisation de tables des pages, une par compartiment, qui définissent un partitionnement de l'espace d'adressage ainsi que des droits d'accès associés. Ces tables des pages sont placées en mémoire centrale, et les modules pare-feu ne contiennent que des caches de ces tables. Cette caractéristique importante permet de résoudre tous les inconvénients notés dans les solutions proposées dans l'état de l'art.

La gestion de confiance de la plateforme, qui organise le partage de toutes les ressources, calcul et mémoire, suit un modèle hiérarchique dont l'architecture peut être adaptée à toutes les situations typiquement rencontrées dans les systèmes sur puce traditionnels.

Les périphériques cibles ne rentrent pas à proprement parler dans l'approche multi-compartiment, mais leur partage, qui peut uniquement augmenter la flexibilité de l'approche, a été brièvement abordé. Enfin, le problème de la redirection des interruptions matérielles, qui s'avère critique dans l'approche multi-compartiment, à cause du partitionnement des processeurs entre plusieurs bases de confiance locales, a été étudié et des solutions conceptuelles innovantes ont été proposées.

La seule incertitude concernant cette approche réside dans le coût du mécanisme de protection. En effet, étant placé sur le chemin entre les composants initiateurs et la mémoire, il

est nécessaire d'étudier l'éventuelle dégradation de performance d'un tel mécanisme. Pour cette raison, dans les chapitres suivants, nous nous intéressons particulièrement à la réalisation du mécanisme de protection, ainsi qu'à son coût associé, en termes de performance, mais aussi en termes de surface de silicium.

Chapitre 5

Plateforme d'évaluation

Sommaire

Dans ce chapitre, nous détaillons la réalisation d'une plateforme d'évaluation matérielle et logicielle, illustrant une manière d'implémenter le modèle de confiance, *multi-compartiment*, présenté au chapitre précédent. Nous commençons par justifier les choix retenus pour cette plateforme. Ensuite, nous décrivons concrètement l'implémentation de l'architecture matérielle puis celle de l'architecture logicielle.

5.1 Introduction

5.1.1 Choix d'implémentation

La figure 5.1 résume grossièrement la composition d'un système embarqué orienté multimédia. Un ou deux processeurs généralistes exécutent un système d'exploitation généraliste,

qui fournit l'interface utilisateur de l'appareil multimédia, quelques processeurs de faible complexité exécutent un système d'exploitation spécialisé, qui fournit entre autres les *codecs* (fonctions de codage et de décodage) multimédia, tandis que quelques coprocesseurs spécialisés exécutent des piles logicielles dédiées pour des traitements très spécifiques.

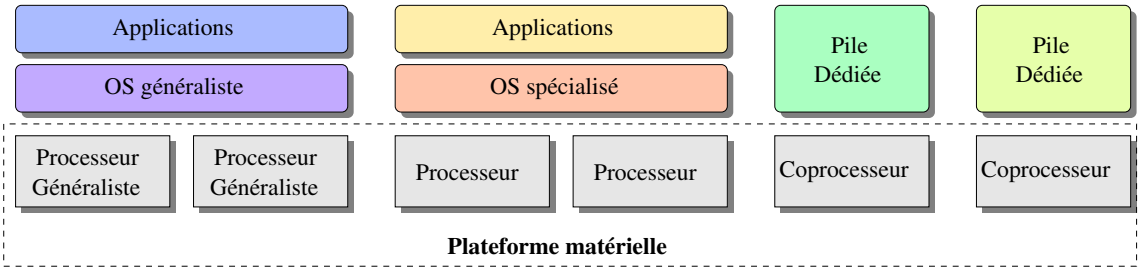


FIGURE 5.1 – Système embarqué orienté multimédia traditionnel.

Parmi toutes ces piles logicielles autonomes, on peut supposer que le système d'exploitation généraliste utilise les MMU disponibles dans les processeurs généralistes pour isoler ses applications, ce qui correspond à un cas d'utilisation mono-compartiment ; les piles logicielles dédiées sont également des cas d'utilisation mono-compartiment ; la pile logicielle constituée par un système d'exploitation spécialisé, n'utilisant pas de MMU, s'avère alors le candidat idéal pour illustrer notre modèle de confiance multi-niveau.

Les services multimédia qui fonctionnent sur le système d'exploitation spécialisé ne sont pas toujours isolés les uns des autres. Souvent, ces services fonctionnent directement en mode noyau, donc au même niveau que le système d'exploitation spécialisé. Pourtant, ces services représentent des sous-systèmes indépendants, que l'on voudrait isoler pour protéger leurs intérêts propres, et les flux de données multimédia qu'ils manipulent.

Nous avons donc décidé de focaliser l'implémentation de notre modèle de confiance sur cette sous-partie matérielle et logicielle spécifique, puisqu'elle présente le plus grand intérêt. Comme l'annonce le scénario d'évaluation présenté dans la figure 5.2, nous détaillerons toutes les modifications matérielles et logicielles pour mettre en conformité cette sous-partie avec l'approche multi-compartiment.

L'implémentation de ce scénario a pour premier objectif de valider la faisabilité du modèle, c'est-à-dire les principales solutions matérielles et logicielles présentées au chapitre précédent. De cette implémentation, on peut obtenir des estimations concernant le coût matériel. Le deuxième objectif est d'effectuer des mesures de performances, particulièrement sur les points sensibles de l'approche, et plus exactement sur le mécanisme de protection au cœur de notre proposition.

5.1.2 Partie matérielle

La mise en œuvre d'une plateforme matérielle compatible avec l'approche multi-compartiment nécessite trois interventions dans l'architecture matérielle. Pour être validé, le principe

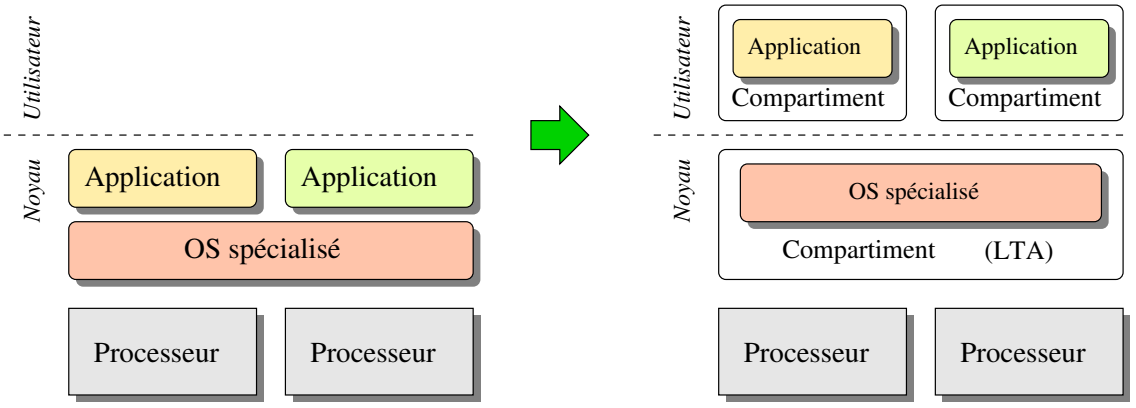


FIGURE 5.2 – Scénario d'évaluation : des applications noyau fonctionnant au côté d'un système d'exploitation, vers des compartiments utilisant s'exécutant au-dessus d'un LTA.

d'identification requiert la modification du groupe de processeurs – homogènes – sur lequel le système d'exploitation s'exécute, afin que les processeurs étiquettent les transactions qu'ils émettent avec un identifiant de compartiment (CID). Ce principe implique également que cet identifiant soit véhiculé dans la plateforme. Enfin, le principe de protection est validé par la mise en place d'un mécanisme matériel, en entrée du réseau d'interconnexion, afin d'effectuer le contrôle d'accès sur les transactions de la plateforme. Ces trois éléments sont mis en avant graphiquement sur la figure 5.3.

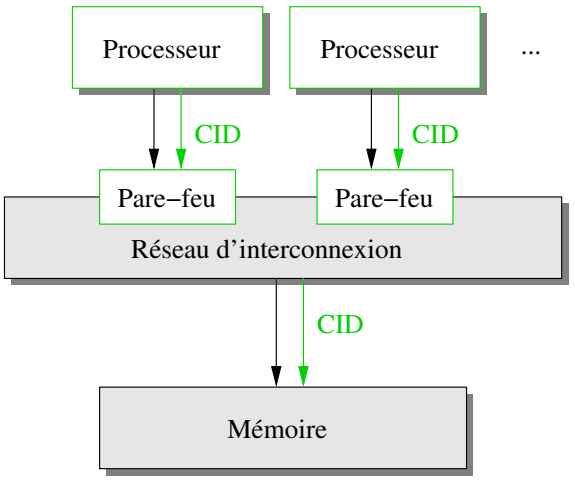


FIGURE 5.3 – Modifications matérielles pour implémenter l'approche multi-compartiment.

Pour la partie matérielle, nous avons choisi de nous appuyer sur l'environnement de prototypage virtuel *SoCLib* [59]. Avec SoCLib, le comportement de chacun des composants de la plateforme matérielle est décrit dans le langage de programmation de haut niveau *SystemC* [60]. Ces modèles de simulation sont disponibles sous une licence libre, c'est-à-dire dont le code source est disponible, ce qui est une caractéristique importante dans un contexte d'exploration et donc de possibles modifications de modèles existants. Dans la bibliothèque SoCLib, on trouve, entre autres, les types de modèles de composants suivants,

respectant tous l'interface *VCI* [48] :

- Réseaux d'interconnexion : réseaux virtuels reproduisant un comportement typique mais sans implémentations physiques équivalentes possibles (*Virtual Generic Micro Network*, *Virtual Generic System Bus*), bus partagés (*PiBus*, *Avalon*, *WishBone*), réseaux point-à-point (*LocalCrossbar*), réseaux sur puce de topologie grille (*DSPIN* [61], *ANoC* [52], etc.) et réseaux en anneau (*LocalRing*, etc.).
- Processeurs : *Mips-32* [62], *Ppc-405*, *Arm-V6T*, *Sparc-V8*, *Lattice Mico 32*, *NiosII*, etc.
- Caches processeur [1] : *XCache* (cache associatif par ensemble à N-voies) et *VCache* (cache associatif par ensemble à N-voies embarquant une MMU paramétrable).
- Mémoires : mémoire non-volatile en lecture seule (*ROM*) et mémoire volatile (*RAM*).
- Périphériques d'entrées/sorties : *TTY*, *BlockDevice*, *FrameBuffer*, etc.
- Périphériques internes : horloge (*timer*), concentrateur d'interruptions matérielles (*ICU*), contrôleur *DMA*, etc.

Pour chacun de ces composants, SoCLib propose deux types de modèles de simulation, représentant deux abstractions comportementales différentes. La première est nommée *CABA* pour *Cycle-Accurate Bit-Accurate* (parfois connue sous le nom *BCA*, pour *Bit/Cycle Accurate*), et se définit par le fait que les modèles simulent un comportement précis des composants au cycle d'horloge près (par rapport à une version physique réelle de ces mêmes composants), ainsi qu'une interface externe précise au bit près. La deuxième abstraction, dérivée d'un concept plus récent, est nommée *TLM* (pour *Transaction-Level Modeling* [63]) et se distingue principalement par le fait que les modèles simulent uniquement la fonctionnalité des composants, sans nécessairement garantir une précision d'horloge précise. Bien que cette deuxième approche permette d'atteindre une vitesse de simulation très rapide (de l'ordre du mégahertz), le fait que nous voulons évaluer précisément l'impact en performance du mécanisme de protection de l'espace d'adressage nous contraint à utiliser l'approche précise au cycle d'horloge. En outre, il faut noter qu'une ré-implémentation du moteur de simulation SystemC réalisée en 2006, *SystemCASS* [64], à laquelle les modèles de simulation CABA proposés par SoCLib sont tous compatibles, offre également une vitesse de simulation relativement élevée (jusqu'à plusieurs centaines de kilohertz).

5.1.3 Partie logicielle

La mise en œuvre d'un système logiciel compatible avec l'approche multi-compartiment nécessite la réalisation d'un modèle de confiance logiciel hiérarchique, c'est-à-dire la réalisation d'un agent de confiance local (*LTA*) et d'un agent de confiance global (*GTA*). Comme décrit par le scénario d'évaluation juste évoqué, et comme le représente graphiquement la figure 5.4, un système d'exploitation exécute des compartiments, qui sont représentés

1. Pour des raisons de généricité, SoCLib a fait le choix de séparer les modèles de processeurs des modèles de caches processeur, ce qui permet ainsi de réutiliser le même modèle de cache processeur avec différents cœurs de processeur.

par des applications s'exécutant en mode utilisateur. Dans cette implémentation logicielle, l'OS cumule les fonctions de LTA et de GTA, et nous verrons en détail l'implémentation de ces responsabilités. En tant que LTA, il doit définir un identifiant de compartiment pour chaque compartiment qu'il exécute, il est responsable de l'ordonnancement de ces compartiments et doit mettre à jour le registre de CID du processeur en conséquence. En tant que GTA, il gère le partage sécurisé et dynamique de l'espace d'adressage, ce qui correspond essentiellement à la construction et au maintien des tables des pages de protection.

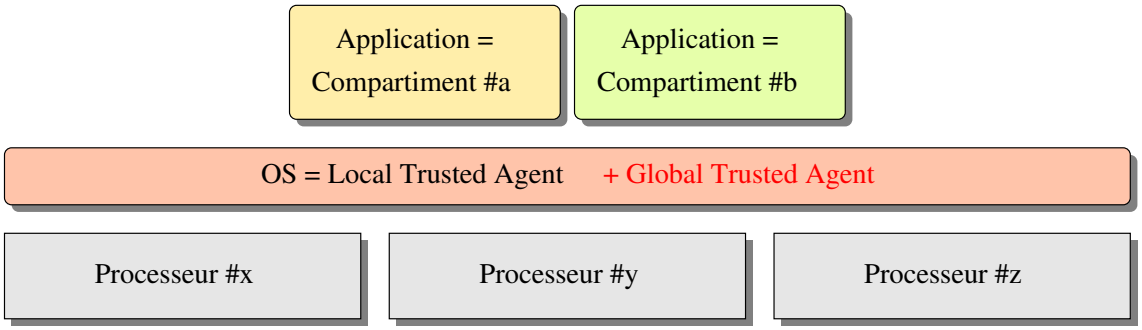


FIGURE 5.4 – Un unique système d'exploitation, exécutant des applications – compartiments –, cumule les rôles de LTA et de GTA.

De la même façon que pour l'environnement de développement de la plateforme matérielle, le système d'exploitation doit autoriser une exploration architecturale logicielle facile et rapide. En ce sens, nous avons délaissé les OS traditionnels de type *Unix* (*Linux*, *NetBSD*), car trop complexes pour telle exploration, ainsi que les OS propriétaires, qui ne facilitent généralement pas les modifications internes. En outre, l'OS doit pouvoir fonctionner sur une plateforme matérielle construite à base de modèles de composants SoCLib.

Pour toutes ces raisons, nous avons choisi le système d'exploitation *Mutek/H* [65, 14], initialement développé à but académique pour résoudre des problématiques d'hétérogénéité des processeurs. Mutek/H est structuré en suivant le modèle d'exo-noyau [66]. Une couche logicielle d'abstraction matérielle (ou *HAL*, pour *Hardware Abstraction Layer*) forme la base minimale du noyau de l'OS, et permet aux couches logicielles supérieures, c'est-à-dire les services noyau tels que les pilotes de périphériques, l'allocation mémoire, l'ordonnancement des contextes d'exécution, la gestion du système de fichier virtuel (ou *VFS*, pour *Virtual File System*), etc., d'être totalement indépendantes des ressources matérielles sous-jacentes. Ce comportement colle parfaitement avec notre besoin d'exploration, les responsabilités de LTA et GTA que le noyau doit proposer pouvant être réalisées comme des services noyau indépendants.

5.2 Mise en œuvre matérielle

Dans cette section, nous détaillons l'implémentation des deux principes fondateurs de l'approche multi-compartiment : l'identification qui, dans notre plateforme matérielle,

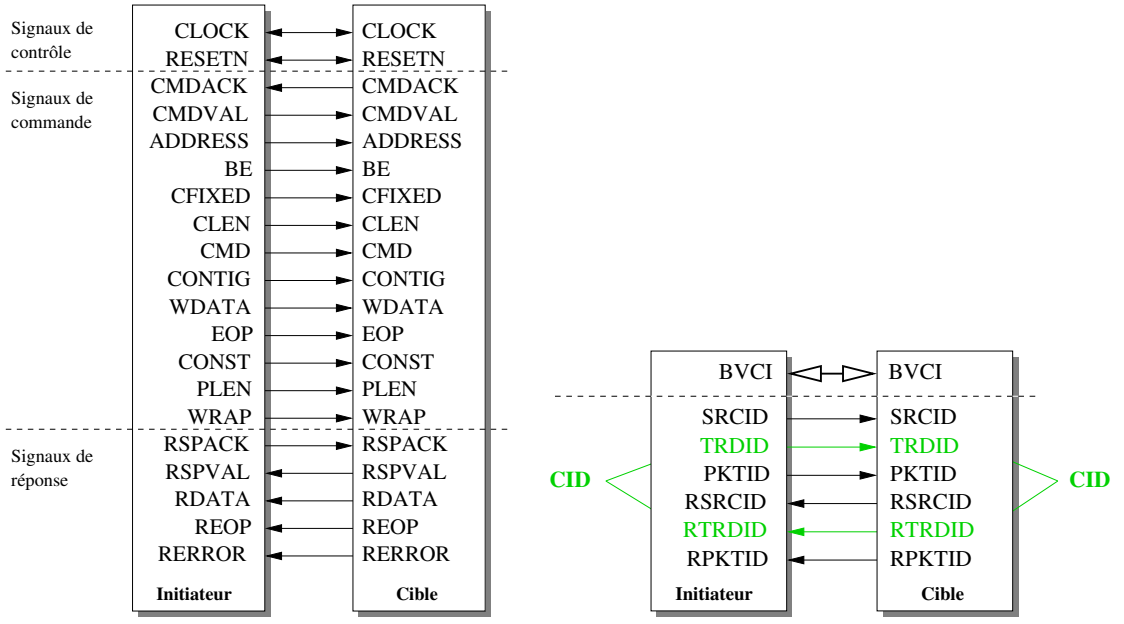
concerne le transport du CID et les processeurs, et la protection, qui concerne le réseau d'interconnexion.

5.2.1 Identification

5.2.1.1 Propagation du CID

Le protocole d'interconnexion VCI, utilisé dans SoCLib, définit plusieurs niveaux de service. L'interface *Basic VCI* (*BVCI*), illustrée dans la figure 5.5(a), est généralement suffisante pour remplir les besoins de la plupart des réseaux d'interconnexions. SoCLib propose un sous-ensemble de l'interface *Advanced VCI* (*AVCI*) qui composée d'extensions optionnelles au BVCI, comme le montre la figure 5.5(b), .

L'AVCI définit notamment le champ `TRDID`, qui comme évoqué au chapitre précédent est utilisé pour véhiculer le CID, attaché à toutes les transactions matérielles de la plateforme.



(a) Interface Basic VCI

(b) Interface Advanced VCI : le champ `TRDID` est utilisé pour véhiculer le CID

FIGURE 5.5 – Les deux types d'interface VCI.

5.2.1.2 Processeur

Pour partager un même processeur entre plusieurs compartiments utilisateur, il faut qu'il offre au moins deux niveaux de privilège. Puisqu'il respecte cette condition, et puisque son architecture *RISC* est réputée pour sa simplicité, nous avons choisi le processeur *Mips32* [62].

L'adaptation du Mips32 pour respecter le principe d'identification implique plusieurs modifications. Premièrement, un registre devant contenir le CID du compartiment en cours

d'exécution sur le processeur est ajouté. Ensuite, la valeur de ce registre doit sortir du processeur pour étiqueter les requêtes, aussi bien d'instructions que de données (« architecture harvard »). Enfin, pour des raisons de performances, le cache processeur doit étiqueter sa chaîne mémoire interne, c'est-à-dire ses cases de cache, ainsi que son tampon d'écriture.

Processeur Mips32 L'architecture générale du processeur Mips32 définit de nombreux registres protégés dont certains sont dépendants de l'implémentation. Nous avons donc ajouté le registre destiné à contenir la valeur du CID courant, `r_cid`, dans le coprocesseur n° 0, ce qui permet d'y accéder en lecture et en écriture par des instructions Mips32 standard, respectivement `mfc0` et `mtc0`. La valeur de sortie de ce registre sur l'interface externe du processeur est toutefois conditionnée par le mode d'exécution courant, comme le montre la figure 5.6. Quand le processeur est en mode noyau, cette valeur est forcée à 0, ce qui signifie que le LTA d'un processeur possède toujours le suffixe de CID 0. En mode utilisateur, la valeur que le processeur délivre reflète bien le contenu du registre CID. Ce comportement câblé matériellement permet de changer de valeur de CID automatiquement lors d'une exception, pour passer de l'identité d'un compartiment en mode utilisateur, à l'identité spéciale du LTA.

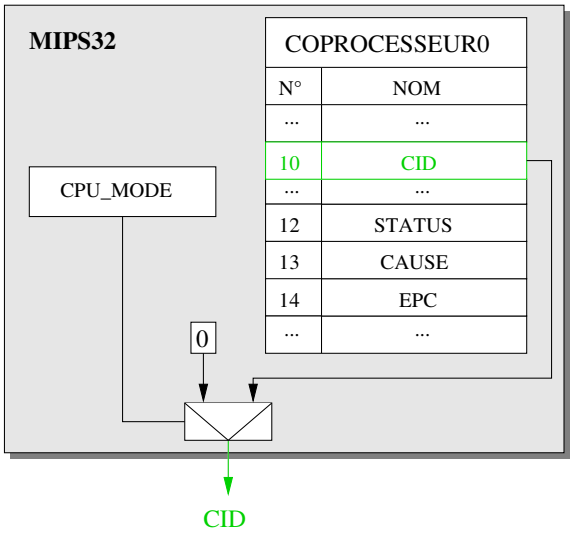


FIGURE 5.6 – Affectation de son interface externe par le Mips32.

Interface processeur/cache SoCLib permet à plusieurs types de processeur d'utiliser les mêmes modèles de cache processeur. Pour rendre ce mécanisme possible, SoCLib définit une interface générique, que nous avons modifiée comme montré dans la figure 5.7 pour y ajouter le CID. Cette interface, nommée *ISS* pour *Instruction Set Simulator* [67], englobe entre autres les interfaces d'instructions et de données. Concernant l'interface des instructions, une requête est présente sur l'interface si le champ `valid` est à 1. Une requête d'instruction est alors décrite par son adresse de destination (`addr`) et son identifiant de compartiment (`cid`). Concernant l'interface des données, le champ `valid` sert également

à spécifier qu'une requête est présente sur l'interface, et les champs `addr` (addresse de destination), `wdata` (mot à écrire en mémoire si la requête est une écriture), `type` (type d'opération mémoire : lecture, écriture, etc.), `be` (octets du mot à lire ou écrire) et enfin `cid` (identifiant de compartiment), servent à décrire effectivement la requête de donnée.

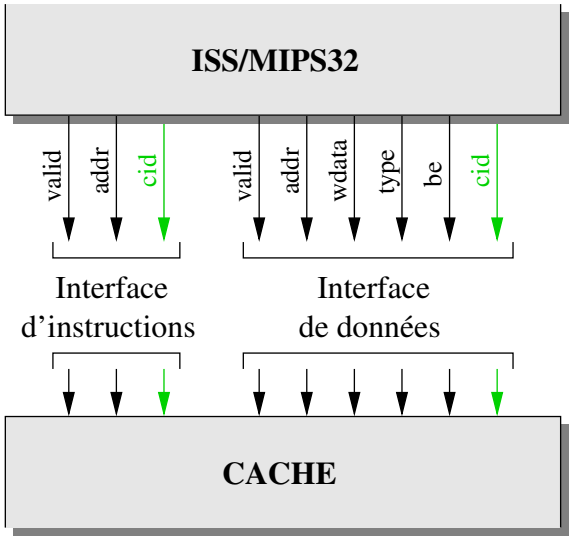


FIGURE 5.7 – Interface processeur/cache modifiée.

Cache processeur La modification du cache processeur, instructions et données, peut être réalisée de manière très simple. En effet, le CID peut être représenté comme une extension de l'étiquette des cases de cache dans le répertoire, et comme une extension de l'adresse dans le tampon d'écriture. Au sein d'une plateforme matérielle basée sur des adresses mémoires de 32 bits, avec 8 bits de CID, il suffit de déclarer un cache processeur manipulant des adresses de 40 bits. Les champs d'adresse et de CID des requêtes processeur qui entrent dans le cache doivent alors être fusionnés ; en sortie du cache, en cas de défaut de cache sur une lecture ou pour une écriture, ils doivent être démultiplexés pour affecter les 32 bits du champ `ADDRESS` de la transaction VCI, ainsi que les 8 bits du champ `TRDID` associé. Cette modification sur un exemple de cache processeur totalement associatif est illustrée sur la figure 5.8.

5.2.2 Protection

Comme nous l'avons expliqué au chapitre précédent, la protection de l'espace d'adressage est rendue possible par l'introduction modules « pare-feu », placés aux entrées du réseau d'interconnexion et contrôlant toutes les transactions matérielles émises par les composants initiateurs et étiquetées avec un identifiant de compartiment. L'espace d'adressage est partitionné en pages de taille fixe, et pour chaque compartiment, les droits d'accès sont définis par une table des pages distincte. Tandis que toutes les tables des pages sont stockées en mémoire, les modules pare-feu embarquent essentiellement un cache, contenant

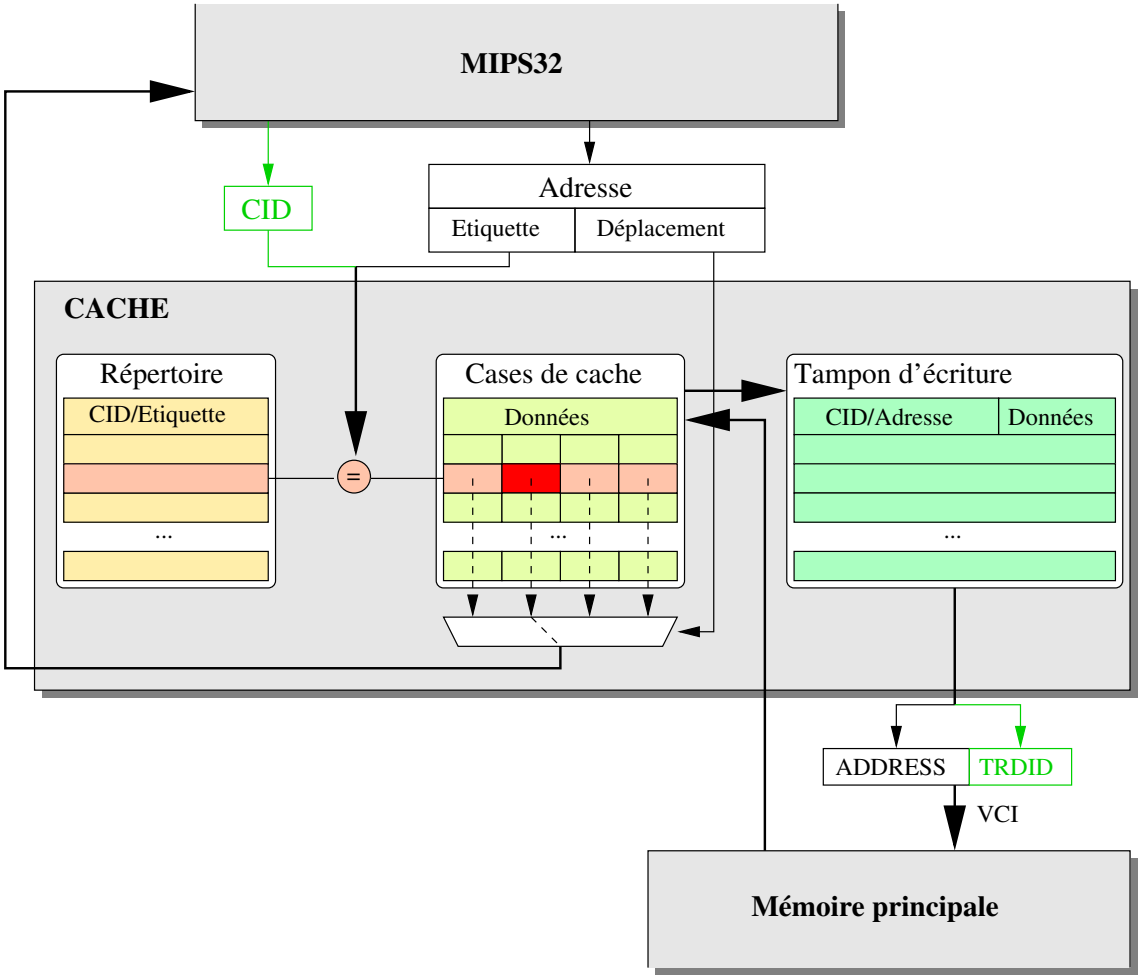


FIGURE 5.8 – Intégration du CID dans un cache processeur totalement associatif.

les règles de droits d'accès les plus récemment utilisées, et un automate de traitement des MISS (en cas de défaut de cache).

Ici, nous considérons un espace d'adressage défini sur 32 bits, partitionné en pages de taille fixée à 4 KiB, ce qui correspond à la granularité de pagination traditionnellement utilisée par les mécanismes de mémoire virtuelle. L'accès au contenu d'une page est donc déterminé par les 12 bits de poids faible d'une adresse mémoire, tandis que les 20 bits de poids fort servent à indexer les deux niveaux d'une table des pages : 10 bits par niveau.

De la même façon que certaines des solutions d'architecture sécurisée présentées dans l'état de l'art, les modules pare-feu, que nous avons nommés *MPU* pour *Memory Protection Unit*, sont intégrés au sein d'un réseau sur puce. Dans la suite, nous commençons donc à détailler cette intégration, puis nous nous intéressons au fonctionnement interne des MPU. Pour cela, nous étudions successivement les deux parties logiques principales qui composent une MPU : le cache de règles de droits d'accès, et l'automate de traitement des MISS. Enfin, nous précisons la configuration des MPU par l'agent de confiance global (GTA).

5.2.2.1 Intégration dans un réseau sur puce

Le réseau sur puce que nous avons choisi pour accueillir les MPU est le réseau *DSPIN* [61], un réseau sur puce à commutation de paquets dédié aux architectures multiprocesseurs à mémoire partagée. Ce mécanisme de protection pourrait toutefois être implémenté dans n'importe quel réseau sur puce possédant les mêmes caractéristiques, c'est-à-dire supportant des transactions de lecture et d'écriture dans un espace d'adressage partagé.

Comme l'illustre la figure 5.9, notre architecture matérielle suit une topologie en grille à plat : un seul composant physique, initiateur ou cible, est connecté à chaque routeur du réseau sur puce, par l'intermédiaire d'un contrôleur d'interface (*NIC*). Dans le réseau DSPIN, un NIC est principalement composé d'un adaptateur en charge de convertir le format VCI au format DSPIN. Cette conversion de format est essentiellement une sérialisation, puisque la longueur d'un paquet DSPIN est plus petite que celle d'un paquet VCI. Nous distinguons deux types de NIC, selon qu'il connecte un composant initiateur (*NIC/I*) ou un composant cible (*NIC/T*).

Pour les raisons de performance que nous expliquerons ci-après, les MPU sont embarquées dans les NIC/I et fonctionnent en parallèle de la conversion de protocole.

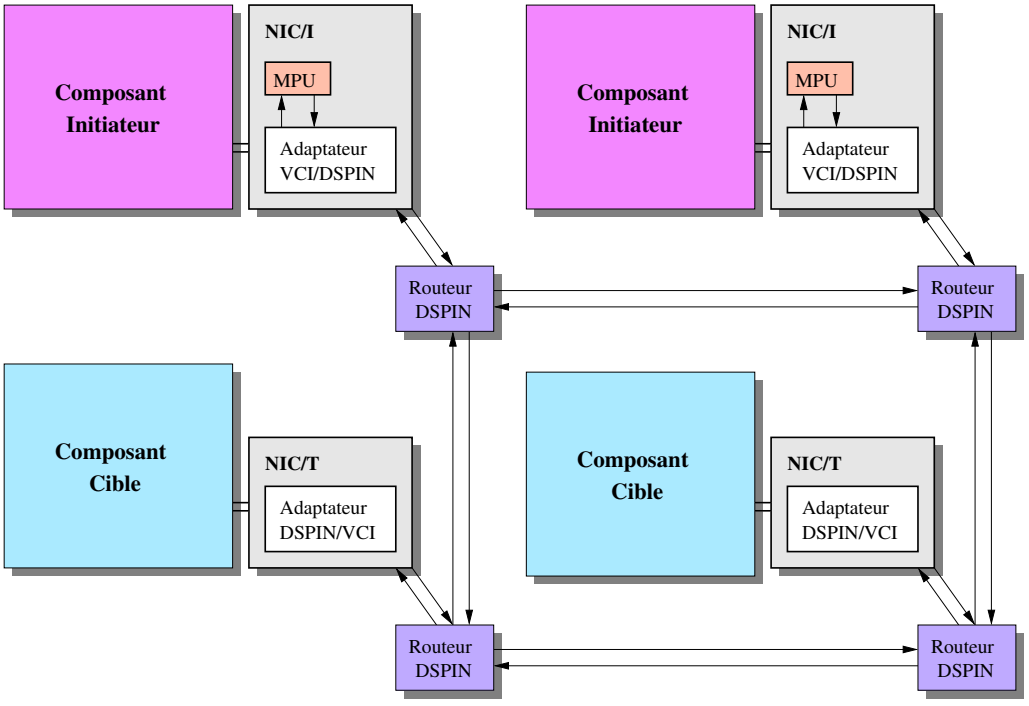


FIGURE 5.9 – Architecture matérielle basée sur le réseau sur puce DSPIN, et embarquant des MPU dans les NIC/I.

5.2.2.2 Cache de règles de droits d'accès

Lorsqu'une transaction se présente à l'entrée d'un NIC/I, toutes les informations la caractérisant sont comprises dans les signaux VCI. L'automate qui implémente la fonctionnalité de conversion du NIC/I prend plusieurs cycles pour analyser ces informations et construire les paquets DSPIN qui seront émis dans le réseau sur puce pour atteindre la cible de la transaction. C'est seulement au dernier état de cet automate que la transaction VCI présente sur l'interface d'entrée est acquittée.

Le premier état de l'automate de conversion consiste à consulter une table de correspondance, indexée par les bits de poids fort de l'adresse de la transaction VCI. Cette table renvoie les coordonnées physiques du routeur de destination. Ces coordonnées permettent alors de router les paquets dans le réseau.

C'est en parallèle de ce premier état d'automate que se positionne le cache des règles de droits d'accès, ou plus exactement sa consultation. Comme illustré dans la figure 5.10, le cache utilise trois des signaux VCI de la transaction pour en vérifier les droits d'accès. Le CID (contenu dans le champ `VCI/TRDID`) et les 20 premiers bits du champ `VCI/ADDRESS` servent à indexer le cache, qui renvoie, s'il contient l'information, les droits d'accès en lecture et en écriture. Enfin, le champ `VCI/CMD`, qui contient le type d'opération (lecture ou écriture), permet de vérifier effectivement la transaction, et de la laisser passer sur le réseau ou pas.

Si le cache contient la règle de droits d'accès, sa consultation a un coût nul, car elle est réalisée en parallèle de la consultation de la table de correspondance dans le premier état de l'automate. La vérification des droits d'accès dans ce cas est donc transparente.

Pour une transaction émise par l'identifiant de compartiment n° 0, considéré comme étant le GTA (compartiment racine ayant tous les droits sur la plateforme), aucune règle n'est consultée : la transaction est automatiquement autorisée.

5.2.2.3 Automate de traitement des MISS

Si le cache ne contient pas la règle de droits d'accès permettant de vérifier la transaction VCI en cours de traitement, alors l'automate de conversion reste bloqué dans le premier état. Puisque la transaction VCI n'est acquittée qu'au dernier état de cet automate, ce comportement n'est pas problématique car le contrôle de flux est géré par le protocole VCI.

En attendant et comme schématisé sur la figure 5.11, l'automate de traitement des MISS prend la main pour récupérer les informations de droits d'accès en mémoire. Un tableau interne de pointeurs sur les répertoires de pages (*Page Directory*) est indexé par le champ CID pour trouver l'adresse de base du répertoire de page associé au compartiment émetteur de la transaction. Les 10 premiers bits de l'adresse de la transaction sont utilisés comme déplacement pour trouver, dans ce premier niveau de table des pages, l'adresse de

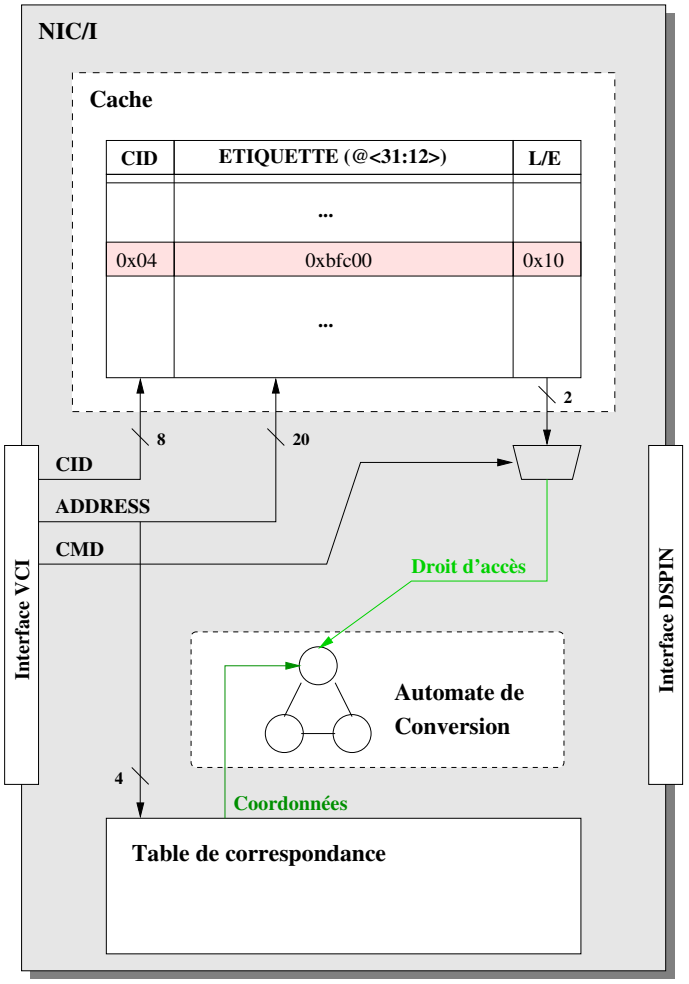


FIGURE 5.10 – Architecture interne du cache de règles de droits d'accès et interaction avec le NIC/I.

base de la table de page de second niveau (*Page Table*). Cette table de page est ensuite accédée, en utilisant les 10 bits suivant de l'adresse de la transaction comme déplacement pour trouver les droits d'accès en lecture et écriture (2 bits).

Ces droits d'accès sont enfin ajoutés au cache, débloquant par la même occasion l'automate de conversion qui peut alors reconsulter le cache pour trouver la règle de droits d'accès concernant la transaction en cours de traitement. Le traitement d'un MISS nécessite donc deux transactions vers la mémoire, à travers le réseau. Autrement, les caches de règles de droits d'accès ne modifiant pas les règles, aucun mécanisme particulier n'est requis lors de l'éviction d'une règle pour en stocker une autre.

En principe, les deux lectures effectuées par l'automate de traitement des MISS sont injectées directement dans le réseau sur puce, en réutilisant le même identifiant de source physique (champ *VCI/SRCID*) que celui du composant initiateur qui a émis la transaction, c'est-à-dire le composant initiateur attaché au NIC/I donné. Cet identifiant physique unique à chaque composant initiateur, permet en effet à la réponse, associée à une transaction

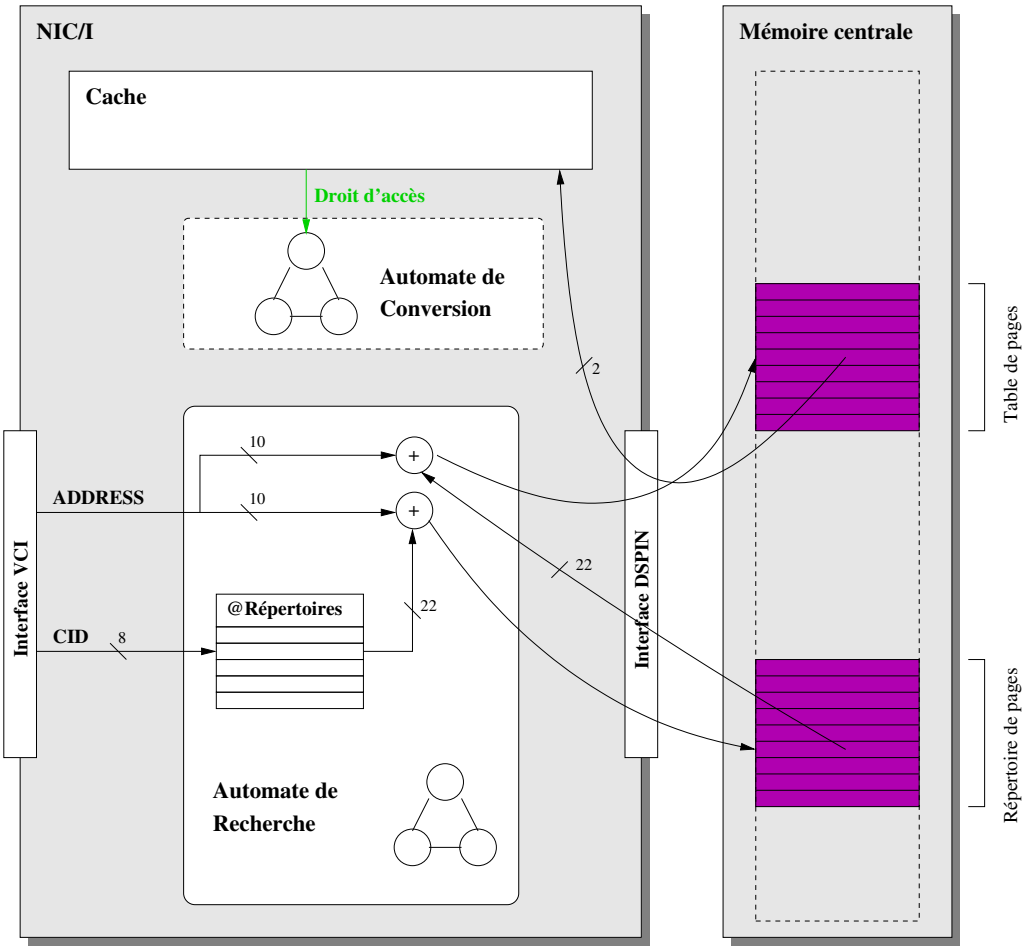


FIGURE 5.11 – Schématisation de l'automate de traitement des MISS et interaction avec le NIC/I.

initiale, d'être routée par le réseau sur puce vers le composant initiateur qui a émis la commande. Lorsque les réponses aux deux lectures de l'automate de traitement des MISS reviennent dans le NIC/I, il peut simplement les intercepter. Ces lectures sont étiquetées avec l'identifiant de compartiment n° 0, correspondant au GTA.

Configuration La configuration des MPU par le GTA se réduit à définir le contenu du tableau de pointeurs sur les répertoires de pages, afin que les automates de traitement des MISS puissent effectuer correctement le parcours des tables des pages.

Cette configuration peut être implémentée de plusieurs manières. Comme le propose l'approche μSpider (section 3.2.2.3), le réseau sur puce peut par exemple offrir un canal de configuration spécifique uniquement accessible par le GTA. Dans notre cas, par mesure de simplicité, les MPU sont accessibles par l'intermédiaire d'interfaces VCI cible, ouvertes uniquement pour le GTA.

5.2.3 Conclusion

Les modifications à apporter aux processeurs pour les rendre compatibles avec le principe d'identification sont minimes. La mise en place du mécanisme de protection, par l'intermédiaire des modules pare-feu, MPU, est lui plus complexe mais trouve facilement sa place dans un réseau sur puce standard. La consultation des caches de règles de droits d'accès embarqués dans les MPU est transparente, puisque elle est effectuée en parallèle avec la conversion de format VCI vers DSPIN. L'éventuelle dégradation de performance peut alors provenir du mécanisme matériel de traitement des MISS de cache, et c'est un des points principaux que nous cherchons à évaluer au chapitre suivant.

5.3 Mise en œuvre logicielle

L'implémentation du système logiciel, en utilisant le système d'exploitation Mutek/H, a nécessité le développement de trois éléments que nous allons détailler. Premièrement, pour respecter le scénario logiciel proposé, les compartiments (applications) doivent s'exécuter en mode utilisateur ; ensuite, le système d'exploitation doit être agrémenté des rôles de LTA et de GTA, afin de respecter l'approche multi-compartiment.

5.3.1 Applications utilisateur

La figure 5.12 résume l'architecture globale de l'OS Mutek/H. Le cœur de l'exo-noyau est composé du sous-système *Hexo*, responsable de l'abstraction matérielle, ce qui comprend le support des différents processeurs et des différents plateformes, et de la bibliothèque de base *Mutek*, qui contient différentes stratégies d'allocation mémoire, différentes stratégies d'ordonnancement, etc. Des bibliothèques indépendantes (et optionnelles) de plus haut niveau, contiennent les pilotes de périphériques, des bibliothèques de services (`libc`, `VFS`), et des bibliothèques d'interface avec les applications.

Aujourd'hui dans Mutek/H, la bibliothèque d'interface avec les applications la plus développée et utilisée est la bibliothèque implémentant le standard des *threads POSIX* [68]. L'utilisation de cette bibliothèque implique que les applications sont compilées statiquement avec le noyau, et fonctionnent en mode noyau.

L'implémentation d'une stratégie dans laquelle nos compartiments s'exécutent en mode utilisateur a donc entraîné le développement de plusieurs bibliothèques supplémentaires.

Lorsqu'une application fonctionne dans le mode utilisateur d'un processeur, elle peut faire appel à des services offerts par le noyau grâce à l'instruction spéciale `syscall`. Cette instruction permet essentiellement de passer le processeur en mode noyau pour exécuter du code fourni par le noyau du système d'exploitation. Une application utilisateur, en ayant configuré certains registres généraux du processeur, peut rendre paramétrables ses appels au noyau et ainsi utiliser les différents services offerts. Dans notre cas, ce mécanisme d'*appels*

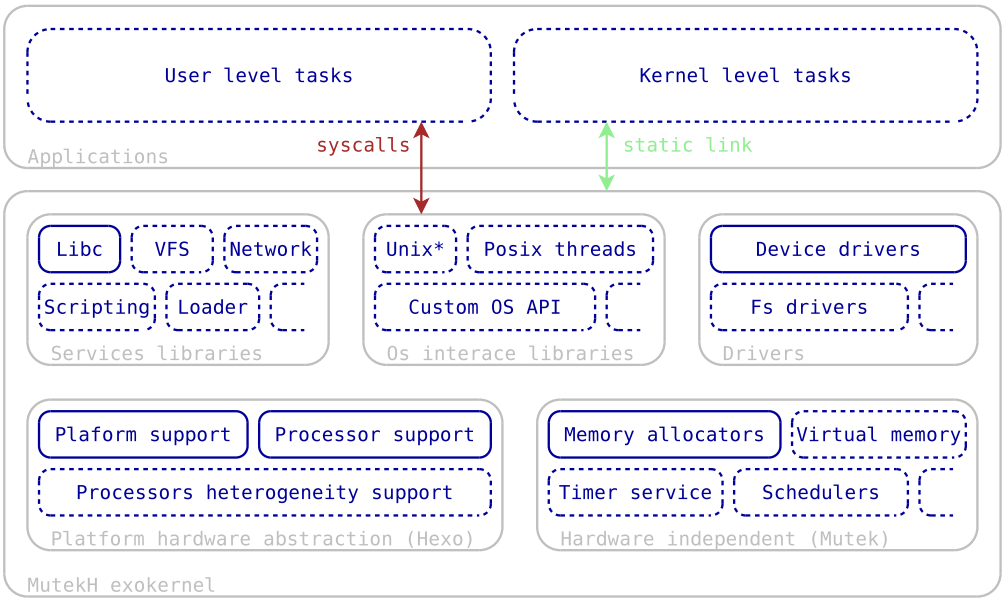


FIGURE 5.12 – Vue globale de l'architecture logicielle de Mutek/H. *Crédits : [14]*

systèmes a été réalisé par le développement de deux bibliothèques, une fonctionnant côté noyau, l'autre côté utilisateur.

5.3.1.1 Bibliothèque d'appels systèmes (côté noyau)

La première bibliothèque noyau que nous avons développée permet à Mutek/H de proposer quelques services à nos compartiments utilisateur. Cette bibliothèque, nommée `libsyscall`, offre une gamme de services minimale, et plus précisément des services d'ordonnancement uniquement. Si un compartiment n'a plus besoin de la ressource processeur car il est en attente d'un événement (prise d'un verrou, par exemple), il peut s'adresser au noyau pour être mis en attente et ainsi laisser l'ordonnanceur du noyau exécuter un autre compartiment.

5.3.1.2 Bibliothèque d'appels systèmes (côté utilisateur)

Sauf cas rare, une application utilisateur a toujours recours à une bibliothèque de services standard, avec laquelle elle est compilée. Pour les applications développées en langage C, cette boite à outil logicielle est par exemple communément appelée `libc`. Elle définit une collection de fonctions utilisées pour implémenter les opérations les plus fondamentales, telles que les appels systèmes mais également la gestion des entrées/sorties (fichiers, affichage, mémoire, etc) ou la manipulation de chaîne de caractères.

Nous avons donc développé une bibliothèque de fonctions utilisables par un compartiment s'exécutant en mode utilisateur, que l'on a appelée `libuser`. Cette bibliothèque propose un sous-ensemble minimal d'opérations dont :

– Appels aux services offerts par le noyau (`syscall`).
– Utilisation de fonctions mathématiques (`math`).
– Manipulation de chaines de caractères (`string`).
– Affichage de caractères et de chaines de caractères (`stdio`).
– Utilisation de barrières de synchronisation (`barrier`).

Un autre rôle de cette bibliothèque est le démarrage des compartiments. En effet, au moment de la création et du lancement d'un compartiment par le noyau, celui-ci finit par passer en mode utilisateur, dans un code de démarrage (traditionnellement défini par la fonction nommée `_start`). Ce code de démarrage, fourni par la bibliothèque standard, est notamment censé préparer la pile d'exécution du compartiment, et peut ensuite appeler le code du compartiment, traditionnellement situé dans une fonction nommée `main`.

La figure 5.13 présente graphiquement la `libuser`, et ses interactions avec le compartiment utilisateur avec lequel elle est compilée, ainsi qu'avec le noyau par l'intermédiaire de la `libsyscall`.

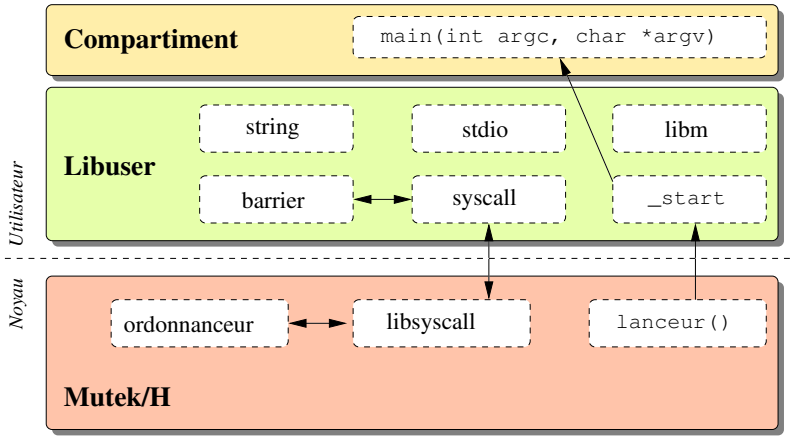


FIGURE 5.13 – Architecture logicielle de la `libuser` et ses interactions avec les autres composants logiciels.

5.3.1.3 Bibliothèque de chargement dynamique

Dans le cas d'applications s'exécutant en mode noyau, ces dernières sont souvent compilées et liées statiquement avec le noyau du système d'exploitation. Il résulte de cette opération un unique exécutable binaire chargé au démarrage de la plateforme matérielle. Même s'il est possible de répliquer ce fonctionnement avec des applications utilisateur, ce n'est généralement pas le comportement recherché. En effet, ce mécanisme implique que le noyau et toutes les applications sont chargés en même temps en mémoire, car appartenant au même exécutable, alors même que les applications ne sont pas toutes requises à un moment donné. Dans un contexte de plateforme matérielle embarquée, où la mémoire est une ressource considérée comme précieuse, on cherche plutôt à en minimiser l'utilisation.

Dans le cas de systèmes orientés multimédia par exemple, une application de décodage vidéo n'est utile qu'au moment où l'utilisateur décide de visionner un flux vidéo.

Dans notre cas, les compartiments sont donc compilés avec la bibliothèque standard (`libuser`) de façon indépendante. Le noyau est également obtenu indépendamment, ce qui permet à terme de charger les compartiments utilisateur au moment où ils sont requis, et ainsi gagner en flexibilité.

Les exécutables des compartiments utilisateur étant créés au format *ELF* [69] (*Executable and Linkable Format*), nous avons développé une bibliothèque de chargement dynamique, nommée `libelf` et fonctionnant côté noyau. Cette bibliothèque permet alors de charger dynamiquement des compartiments en mémoire centrale à partir d'un périphérique de stockage de masse. En outre, ces compartiments sont compilées en *PIC* (*Position Independant Code*), ce qui implique que leur code ne repose sur aucune adresse mémoire préétablie (contrairement à une compilation statique) : ils peuvent alors être placés n'importe où en mémoire, permettant ainsi de s'adapter au mieux à la disponibilité de la ressource mémoire au moment du chargement.

La flexibilité apportée par ce chargement dynamique est en réalité plus profonde car les compartiments ne sont pas liés statiquement avec la `libuser`. Cela signifie qu'un compartiment et la `libuser` sont deux exécutables binaires séparés. Ce n'est qu'au moment du chargement de l'exécutable d'un compartiment que la `libelf` prend connaissance de la dépendance qui lie le compartiment avec la `libuser`. La `libelf` charge la `libuser` en conséquence et réalise une édition des liens dynamiques, pour que le compartiment en question puisse effectivement utiliser les fonctions proposées par la `libuser`. De surcroît, grâce à la compilation de tous ces exécutables en PIC, les compartiments sont capables de partager le code et les données des mêmes bibliothèques. Concrètement, cela veut dire que la `libuser`, dont dépendent tous les compartiments de notre scénario d'évaluation, n'a besoin que d'être chargée une seule fois en mémoire pour être partagée par les compartiments.

Outre le respect du scénario proposé, on notera qu'une raison importante pour justifier l'implémentation d'une telle bibliothèque est de montrer la flexibilité intrinsèque de notre modèle et sa capacité à supporter une politique de sécurité dynamique. Nous voulons en effet montrer que les compartiments peuvent être créés dynamiquement, ce qui implique la définition de nouveaux domaines de protection associés, et qu'ils peuvent également partager des zones mémoires de manière flexible.

5.3.2 Agent de confiance local

Le rôle d'agent de confiance local (LTA) que Mutek/H doit endosser pour respecter l'approche multi-compartiment, est normalement responsable de deux tâches : l'ordonnancement des compartiments, et la mise à jour du registre de CID intégré au processeur en conséquence. Dans un système d'exploitation traditionnel, ces deux tâches sont un peu diffuses, dans le sens où elles se réduisent généralement à de légères modifications éparses

dans des bibliothèques noyau existantes. Par exemple, l'ordonnancement est déjà un service offert par Mutek/H. La mise à jour du registre de CID s'intègre quant à elle principalement dans la routine de changement de contexte d'exécution.

La bibliothèque, `liblta`, que nous avons développée et que nous détaillons ci-après, joue plutôt le rôle d'un gestionnaire de compartiments complet : elle se charge de leur création, de leur chargement et de leur lancement. Elle gère même les compartiments pendant leur exécution, puisque en réalité c'est elle qui intègre la `libsyscall`, présentée précédemment.

5.3.2.1 Création d'un compartiment

Au moment du chargement d'un compartiment utilisateur, la bibliothèque `liblta` est chargée de créer un nouvel identifiant de compartiment. Si le CID au niveau plateforme est codé sur 8 bits, cette étape se résume conceptuellement au maintien d'un tableau de booléen de 255 cases (le compartiment n° 0 est détenu par le GTA) qui indique si les identifiants sont disponibles ou non. Lorsque la bibliothèque doit attribuer un nouvel identifiant, elle doit alors chercher dans la table quel est le premier identifiant disponible. Elle doit ensuite informer le GTA de la création de ce nouveau compartiment, afin que le GTA débute la création d'une nouvelle table des pages, qui contiendra les droits d'accès du nouveau compartiment sur l'espace d'adressage.

5.3.2.2 Chargement d'un compartiment

Pendant le chargement du compartiment, et notamment lors des allocations mémoires qui le concerne (zone de code, de données, de pile d'exécution, etc.), la `liblta` fait appel au GTA pour mettre à jour la table des pages de protection. Pour le chargement effectif du compartiment, elle fait appel à la `libelf`.

Dans notre scénario d'évaluation, l'allocation des pages de mémoire est en réalité gérée de façon indépendante par un allocateur mémoire de Mutek/H. Cette allocation est cohérente dans le sens où la plateforme matérielle est couverte par une seule pile logicielle autonome. Dans une implémentation où le GTA est physiquement séparé des LTA, donc à partir du moment où la plateforme matérielle héberge plusieurs piles logicielles autonomes, cette allocation peut être hiérarchique. Le GTA, en plus de la gestion des tables des pages de protection, peut gérer l'attribution dynamique et à gros gain de la ressource mémoire aux différents LTA de la plateforme. Chaque LTA peut ainsi localement distribuer le ou les gros blocs de mémoire qui lui ont été alloués aux compartiments qu'il crée et exécute. En plus d'alléger le protocole de communication entre les LTA et le GTA, cette stratégie permet de conserver les allocateurs mémoires – performants – existant dans la plupart des systèmes d'exploitation.

La figure 5.14 montre les interactions entre les différentes bibliothèques noyau de notre système logiciel. La `liblta` joue le rôle de gestionnaire de compartiments, faisant appel à la

`libelf` pour le chargement des compartiments, et s'adressant à la `libgta` pour la protection de l'espace d'adressage mais également pour l'allocation de nouvelles pages de mémoire (bibliothèque `palloc`). La `libelf` fait appel au système de fichier virtuel (bibliothèque `libvfs`) pour charger les exécutables des compartiments à partir d'un périphérique de stockage de masse.

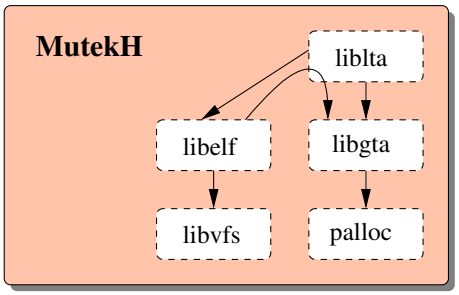


FIGURE 5.14 – Interactions des différents bibliothèques pour gérer les compartiments.

5.3.2.3 Gestion du registre de CID

La dernière tâche accomplie par la bibliothèque `liblta` est la gestion du registre de CID présent dans chaque processeur. Avant le lancement effectif du compartiment, c'est-à-dire juste avant le passage en mode utilisateur pour donner la main à la fonction `_start`, elle écrit l'identifiant de compartiment dans le registre CID. Au moment du passage en mode utilisateur, le processeur effectue ainsi automatiquement les transactions mémoires pour le compte du compartiment en question.

Lorsque le processeur repasse en mode noyau en raison d'une exception provoquée par un appel système provenant du compartiment en cours d'exécution ou d'une interruption matérielle, la première mission du système d'exploitation est de sauver l'état exact du compartiment qui vient d'être interrompu. Cela permet, au moment où la ressource processeur peut être rendue au compartiment, de pouvoir le recharger à l'identique afin qu'il puisse continuer son exécution. Cette sauvegarde consiste généralement à garder la valeur de tous les registres généraux du processeur, que le compartiment est susceptible d'utiliser. Dans notre cas, il faut également sauver la valeur du registre de CID puisqu'elle fait partie de l'état du compartiment.

Le code d'exception, spécifique au processeur Mips32, a donc été très légèrement modifié pour effectuer la sauvegarde du registre de CID lors de l'entrée en mode noyau, et son rétablissement lors de la sortie. La figure 5.15 montre les principales instructions Mips32 qui ont été ajoutées au code d'exception pour implémenter ce comportement.

```
...
/*
 * When entering the context switch routine:
 * - Read CID control register (10th register of coprocessor0)
 * - Save CID value in compartiment's state (pointed by $sp register)
 */
mfc0    $7,     $10          /* mfc0: move from coprocessor0 */
sw      $7,     36*4($sp)    /* sw: store word */
...

...
/*
 * When leaving the context switch routine:
 * - Load CID value from compartiment's state (pointed by $sp register)
 * - Write CID control register (10th register of coprocessor0)
 */
lw      $31,    36*4($sp)    /* lw: load word */
mtc0    $31,    $10          /* mtc0 : move to coprocessor0 */
...
```

FIGURE 5.15 – Les principales instructions Mips32 supplémentaires pour la gestion du registre CID, lors d'un changement de contexte d'exécution.

5.3.3 Agent de confiance global

Le rôle d'agent de confiance global (GTA) fourni par Mutek/H a été réalisé par l'implémentation d'une bibliothèque noyau, nommée `libgta`. Cette bibliothèque est sollicitée pour une unique tâche, la gestion des tables des pages de protection.

Puisque Mutek/H offre déjà une bibliothèque de gestion de mémoire virtuelle, l'implémentation de la `libgta` a principalement consisté à reprendre cette bibliothèque et à n'en garder que les parties responsables de la construction des tables des pages.

5.3.3.1 Création d'un compartiment

Comme expliqué précédemment, lors du lancement d'un compartiment utilisateur, la `liblta` fait appel à la `libgta` pour la création d'une nouvelle table des pages de protection. Cette opération se résume en réalité à la création de la page de premier niveau en mémoire, le répertoire des pages. La bibliothèque configure alors tous les modules pare-feu, MPU, de la plateforme matérielle : elle définit dans le tableau interne des MPU le pointeur vers le répertoire de pages pour le compartiment donné.

5.3.3.2 Chargement d'un compartiment

Lors du chargement d'un compartiment, et plus particulièrement lors des allocations de pages mémoires qu'il requière, la `libgta` est sollicitée pour maintenir à jour la tables des pages. Concrètement, cette opération signifie de créer des tables de pages de second niveau, pour contenir les droits d'accès associés au pages mémoire allouées au compartiment.

5.3.4 Conclusion

Le développement du système logiciel s'est traduit par la réalisation de plusieurs bibliothèques indépendantes. Les plus complexes – la `libsyscall` et la `libelf` côté noyau, et la `libuser` côté utilisateur – servent uniquement à la création de compartiments utilisateur, chargés dynamiquement. Le reste des bibliothèques, la `liblta` et la `libgta`, qui représentent le cœur du développement logiciel relatif à la compatibilité avec l'approche multi-compartiment, restent quant à elles assez simples. Les tâches qu'elles fournissent véritablement sont effectivement soit très légères, soit dérivent ou utilisent des bibliothèques noyau existantes.

5.4 Conclusion

Dans ce chapitre, nous avons défini et justifié un scénario d'évaluation de l'approche multi-compartiment : un unique système d'exploitation assurant les fonctions d'agents de confiance local (LTA) et global (GTA), et des compartiments, sous la forme d'applications utilisateurs. La réalisation de ce scénario a impliqué de rendre compatible une plateforme matérielle modélisée avec SoCLib, c'est-à-dire de modifier le processeur Mips32 pour accueillir le registre de CID, et le réseau sur puce DSPIN pour accueillir le mécanisme de protection. Au niveau logiciel, nous avons utilisé le système d'exploitation Mutek/H, dans lequel nous avons développé plusieurs bibliothèques. Ces bibliothèques permettent de respecter le scénario d'évaluation proposé, c'est-à-dire de pouvoir exécuter des compartiments utilisateur, mais aussi de transformer Mutek/H en LTA et en GTA.

Dans le chapitre suivant, nous étudions et discutons l'expérimentation réelle de ce scénario d'évaluation. Nous en tirons alors des résultats, notamment en ce qui concerne la dégradation de performance due aux éventuels défauts de cache dans les modules pare-feu de protection.

Chapitre 6

Résultats expérimentaux

Sommaire

Dans ce chapitre, nous présentons les résultats expérimentaux obtenus sur la plateforme d'expérimentation, matérielle et logicielle, définie au chapitre précédent. Nous nous intéressons essentiellement aux coûts engendrés par le mécanisme de protection de l'espace d'adressage que nous avons proposé. Ces coûts concernent la dégradation en performance, ainsi que la surface de silicium occupée par les modifications nécessaires à la mise en œuvre de l'approche *multi-compartiment*.

6.1 Plateforme d'expérimentation

Dans la perspective de tester le modèle de confiance hiérarchique proposé et d'en mesurer les coûts associés, notre plateforme d'expérimentation se compose d'une partie matérielle et d'une partie logicielle, que nous présentons dans la suite de cette section.

6.1.1 Partie matérielle

La plateforme matérielle contient deux à six processeurs, et utilise un réseau sur puce. Tel que nous l'avions évoqué au chapitre précédent, l'architecture est à plat : un seul composant physique est connecté à chaque routeur du réseau sur puce. Cette hypothèse permet en effet de garantir que le trafic de la plateforme, c'est-à-dire toutes les transactions, transite par le réseau sur puce, et donc par les modules pare-feu (MPU). Pour des raisons de vitesse de simulation, nous avons toutefois infléchi cette hypothèse, et construit une architecture en grappe (ou *clusterisée*), dans laquelle chaque grappe peut contenir plusieurs composants matériels. Mais la répartition des composants est telle que toute transaction passe obligatoirement par le réseau.

Puisque notre plateforme matérielle comporte finalement assez peu de composants, on utilise un seul routeur. Le routeur *DSPIN* dispose de cinq connexions (Local, Est, Ouest, Nord et Sud) que nous avons utilisées pour attacher des grappes de composants. Chaque grappe est alors composée d'un ou plusieurs composants physiques, interconnectés localement par un réseau point-à-point (crossbar).

Comme le montre la figure 6.1, la plateforme est organisée comme suit :

- Les grappes Est et Ouest contiennent les processeurs Mips32.
- La grappe Sud embarque la mémoire centrale.
- la grappe Nord inclue un contrôleur d'interruption générique (*XICU*) qui intègre de manière interne un périphérique d'horloge (*timer*), un terminal d'affichage de caractères (*TTY*) et enfin, les interfaces VCI utilisées pour la configuration des MPU.
- La grappe Local comporte un périphérique de bloc (*BD*), qui donne accès à un stockage de masse.

On souligne que les composants initiateurs, et plus particulièrement les processeurs, sont isolés des composants cibles : le trafic qu'ils génèrent passe forcément par le réseau sur puce, et est donc vérifié. En outre, si le trafic sortant du périphérique de bloc (BD) n'est pas vérifié (il n'y a pas de MPU à l'entrée du réseau pour la grappe Local), c'est dû uniquement au caractère spécifique de notre scénario d'évaluation : ce périphérique n'est utilisé que pour charger les compartiments utilisateurs en mémoire, et n'est alors contrôlé que par l'unique système d'exploitation de la plateforme, qui a le rôle de GTA. Comme nous l'évoquions dans la section 4.2.3.2, il est difficile de rendre ce type de périphériques compatible avec l'approche multi-compartiment, car leur interface externe (ici, le stockage de masse) ne fait pas partie de l'espace d'adressage, rendant alors leur accès hors de tout contrôle. Comme nous n'avons pas traité plus en détail cette question, les compartiments utilisateur n'ont pas accès à ce périphérique dans cette plateforme d'expérimentation.

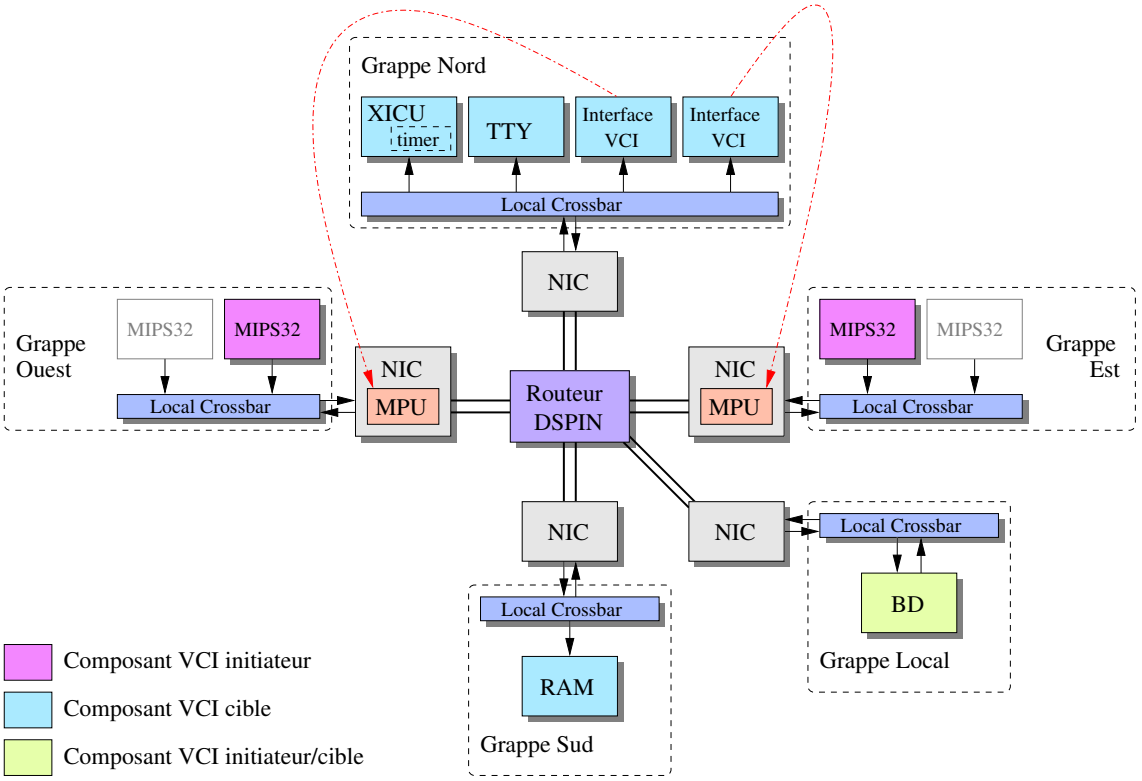


FIGURE 6.1 – Architecture de la plateforme matérielle basée sur un unique routeur DSPIN et composée de plusieurs grappes de composants.

6.1.2 Partie logicielle

Le scénario que nous cherchons à exécuter comporte plusieurs compartiments utilisateur. Puisque ce travail se positionne dans le monde des produits multimédia, nous avons utilisé essentiellement des applications logicielles orientées multimédia.

Dans cette optique, nous avons porté six applications existantes, dont la liste est la suivante :

- `cjpeg` et `djpeg` : un encodeur et un décodeur d'images au format *JPEG*, extraits de la collection d'applications de test de performance *MediaBench* [70].
- `xvid_enc` et `xvid_dec` : un encodeur et un décodeur de vidéos au format *MPEG-4*, extraits de l'application *XviD* [71].
- `minimad` : un lecteur de fichiers musicaux au format *MP3*, extrait de l'application *MAD* [72].
- `aes` : une application de test de performance appliquée au standard de chiffrement *AES* [73].

Outre la bibliothèque `libuser`, la plupart de ces compartiments dépendent de bibliothèques externes : `libjpeg` pour `cjpeg` et `djpeg`, `libxvid` pour `xvid_enc` et `xvid_dec`, et `libmad` pour `minimad`. Comme nous l'avons évoqué au chapitre précédent, notre bibliothèque de chargement, `libelf`, est capable de gérer ces dépendances, et les bibliothèques

en question ne sont alors chargées qu'une seule fois en mémoire, puis partagées entre les compartiments qui le requièrent.

Autrement, tous ces compartiments suivent quasiment le même schéma concernant leur occupation mémoire, et donc leur occupation dans l'espace d'adressage partagé. Ce schéma est présenté progressivement ci-après.

Mis à part le compartiment `aes`, tous les compartiments orientés multimédia ont un fonctionnement presque similaire. Ils récupèrent un flux de données en entrée, opèrent une transformation sur ce flux, et génère un nouveau flux de données en sortie. Typiquement, le compartiment `djpeg` reçoit une image au format JPEG, la décode, et renvoie une image décodée brute (format BMP). Afin d'en valider l'exactitude, le flux de données de sortie est passé dans une fonction de hachage de type `CRC` (ce service est fournie par la `libuser`). Le CRC obtenu est alors comparé avec un CRC de référence.

Concrètement, le flux de données d'entrée et le CRC de référence sont concaténés dans un unique tampon mémoire qui est transmis aux compartiments. Ce tampon est chargé à partir de fichiers présents sur le périphérique de stockage de masse : typiquement, le compartiment reçoit dans un tampon mémoire le contenu du fichier `djpeg.jpg`, qui contient le flux de données d'entrée, et le contenu du fichier `djpeg.crc`, qui contient le CRC de référence. Cette opération de chargement n'est toutefois pas effectuée directement par les compartiments, puisqu'ils n'ont pas accès au périphérique de stockage de masse : elle est alors prise en charge par la bibliothèque noyau `liblta` au moment du chargement des compartiments, donc avant leur exécution. Au moment du lancement effectif des compartiments, les données sont déjà chargées en mémoire et deux pointeurs, respectivement vers le flux de données d'entrée et vers le CRC, leur sont passés explicitement en tant que paramètres.

Toujours à l'exclusion du compartiment `aes`, les autres compartiments nécessitent un tas d'exécution (*heap*). Pour cela, un pointeur vers une zone mémoire pré-allouée leur est également fourni en tant que paramètre. C'est alors au compartiment de gérer l'allocation « locale » de sa zone de tas, grâce à aux services offerts par la `libuser`. Sachant que, pour ce type de compartiment multimédia, on peut généralement caractériser *a priori* leur consommation mémoire, l'hypothèse d'un tel comportement semble raisonnable, et nous a surtout permis d'alléger considérablement le nombre d'appels systèmes que le noyau doit fournir aux compartiments.

Pour tous les compartiments, une zone de mémoire contenant la pile d'exécution (*stack*) est créée.

Les compartiments étant chargés séquentiellement, les compartiments attendent sur une barrière de synchronisation partagée, afin qu'ils débutent leur exécution simultanément. Comme pour les tampons mémoire, un pointeur vers cette barrière est transmis aux compartiments en tant que paramètre.

Enfin, en plus de l'accès à toutes les zones de la mémoire dont les compartiments ont

besoin pour s'exécuter, ils peuvent accéder au segment de l'espace d'adressage défini pour le périphérique d'affichage de caractères (TTY), ceci à des fins de débogage, mais également pour qu'ils puissent afficher les informations de temps d'exécution nécessaires aux mesures de performance.

La figure 6.2 résume graphiquement ce schéma d'occupation de l'espace d'adressage, en illustrant un exemple concret qui concerne le compartiment `djpeg`. On notera que dans l'optique d'une protection de l'espace d'adressage par pagination, toutes les zones créées dans la mémoire sont des ensembles d'une ou plusieurs pages.

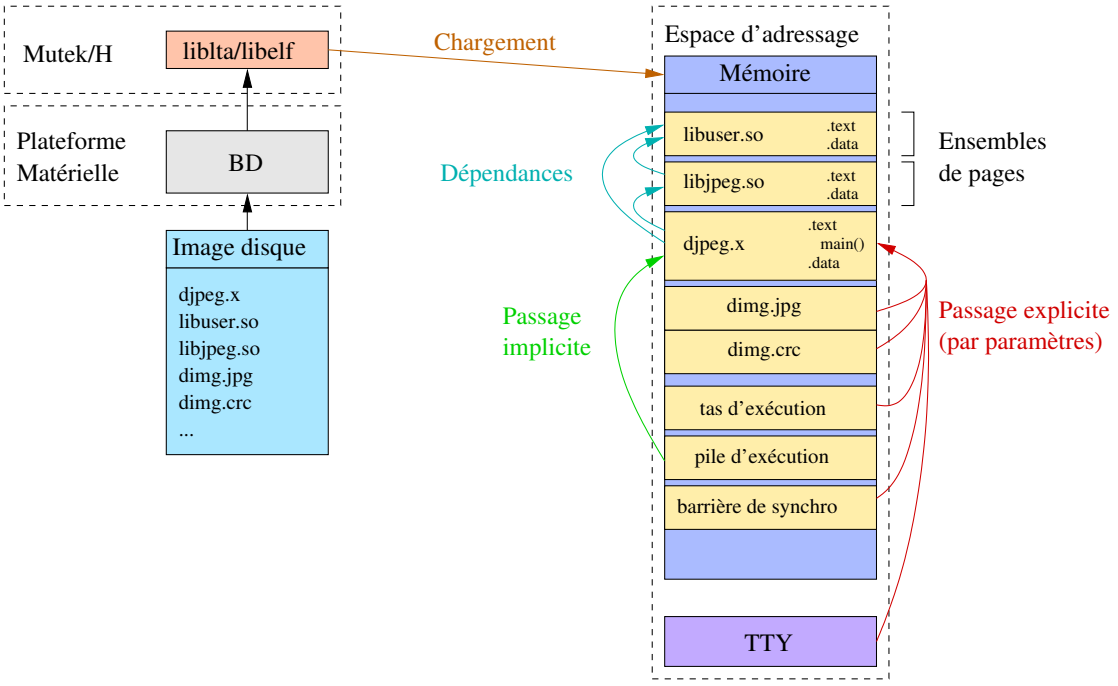


FIGURE 6.2 – Exemple d'occupation de l'espace d'adressage pour le compartiment `djpeg`.

Le tableau 6.1 résume l'occupation de l'espace d'adressage de chacun de nos compartiments de test, ce qui donne ainsi une idée de la taille des tables des pages de protection que l'agent de confiance global, par l'intermédiaire de la `libgta`, doit créer. Un compartiment consomme donc de 50 KiB à plus de 2 MiB de mémoire.

`Compartiment`	Éxecutable	Bibliothèques partagées	Flux (données d'entrée et CRC)	Tas d'exécution	Pile d'exécution	TTY	Total
`aes`	26 KiB	15 KiB	–	–	8 KiB	4 Kib	53 Kib
`cjpeg`	36 KiB	30 KiB	4 KiB	60 KiB	8 KiB	4 Kib	142 KiB
`djpeg`	39 KiB	30 KiB	7 KiB	120 KiB	8 KiB	4 Kib	208 KiB
`xvid_enc`	3 KiB	905 KiB	186 KiB	1,5 MiB	8 KiB	4 Kib	2,6 MiB
`xvid_dec`	3 KiB	905 KiB	19 KiB	1 MiB	8 KiB	4 Kib	1,9 MiB
`minimad`	3 KiB	100 KiB	50 KiB	50 KiB	16 KiB	4 Kib	223 KiB

TABLE 6.1 – Occupation de l'espace d'adressage de chacun des compartiments de test.

6.2 Coût en performance

L'objectif principal de cette section est de mesurer le coût de la protection de l'espace d'adressage, effectué par les modules pare-feu (MPU) à l'entrée du réseau sur puce. Dans cette optique, nous avons comparé les temps d'exécution de nos six compartiments de test sur même la plateforme matérielle, avec et sans les MPU. Tous les graphiques présentés dans cette section montrent alors le surcoût en performance du temps d'exécution avec MPU, sous forme de pourcentage par rapport au temps d'exécution sans MPU.

Puisque les compartiments de test n'ont pas tous le même temps d'exécution intrinsèque, nous avons intégré un mécanisme permettant la ré-exécution paramétrable d'un compartiment. Au lancement du compartiment par le noyau, un compteur d'exécution est alors passé en paramètre aux compartiments, et ces derniers recommencent leur exécution complète autant de fois que spécifié par le compteur. Durant la simulation complète, l'encodeur MPEG-4 (`xvid_enc`) ne s'exécute par exemple qu'une seule fois, tandis que le codeur JPEG (`cjpeg`) s'exécute lui treize fois. Ce comportement permet de garder une activité logicielle conséquente, et plus exactement un trafic transactionnel permanent dans le réseau sur puce, pendant toute la durée de la simulation, les compartiments finissant tous leur exécution complète quasiment simultanément.

6.2.1 Plateforme à deux processeurs

Dans la première expérience, les six compartiments de test s'exécutent sur la plateforme matérielle à deux processeurs, un processeur dans la grappe Ouest et un processeur dans la grappe Est. Le système d'exploitation, Mutek/H, est configuré pour permettre la migration de tâches, donc les compartiments peuvent migrer d'un processeur à l'autre, au moment des changements de contexte. Ces changements de contexte sont programmés pour avoir lieu tous les 2 millions de cycles d'horloge, représentant un quantum de temps de 10 millisecondes si l'on fait l'hypothèse que la plateforme matérielle fonctionne à 200 mégahertz. Ce quantum de temps est similaire à ceux qui sont utilisés dans les systèmes d'exploitation ou hyperviseurs traditionnels. Enfin, les caches processeur, instructions et données, ont chacun une taille de 4 KiB.

Comme le montre le graphique de la figure 6.3, nous avons testé différentes stratégies pour les caches de règles de droits d'accès embarqués dans les MPU. Les quatre premières stratégies sont à correspondance directe, de 1 entrée jusqu'à 8 entrées ; les trois stratégies suivantes sont associatives par ensemble à 2 voies, de 4 entrées jusqu'à 16 entrées ; enfin, les deux dernières stratégies sont totalement associatives, avec 4 ou 8 entrées.

Lorsque le nombre d'entrées des caches de MPU est supérieure à 4, on observe que le surcoût en performance est quasiment nul : 1% ou moins. Quelques gains en performance peuvent même être aperçus, mais cela est uniquement dû à des effets de cache processeur. Dans l'exécution de la plateforme sans MPU, un allocateur mémoire non paginé a été utilisé

pour charger les données appartenant aux compartiments, et l'alignement mémoire de ces données doit simplement être un peu moins favorable.

Comme nous pouvons également le constater, différentes tailles de caches processeurs ont également été testées, respectivement 1 KiB et 16 KiB, avec des caches de MPU typiques, c'est-à-dire associatif par ensemble à 2 voies de 8 entrées : on ne distingue aucun impact particulier.

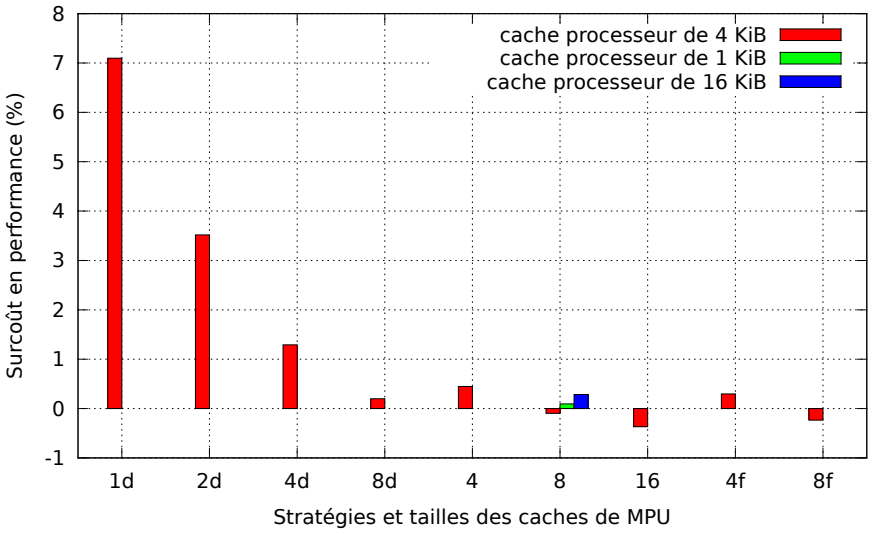


FIGURE 6.3 – Surcoût en performance de différentes stratégies de caches de MPU.

Outre le surcoût en performance qui concerne le temps d'exécution total, nous avons également mesuré le temps de chargement dynamique des compartiments – avant exécution. Dans une plateforme matérielle équipée de MPU, avec des caches processeur de 4 KiB, ce temps de chargement est augmenté de 7 millions de cycles, ce qui est presque entièrement dû à la construction des tables des pages de protection. En gardant l'hypothèse d'une fréquence de fonctionnement de l'ordre de 200 mégahertz, cela représente un surcoût négligeable de 35 millisecondes.

6.2.2 Plateforme à six processeurs

Afin de stresser le mécanisme de protection, et plus précisément d'essayer d'augmenter le nombre de défauts de cache affectant les caches de MPU, la seconde expérience s'intéresse à une architecture matérielle composée de six processeurs. La même plateforme matérielle que précédemment est réutilisée, mais les grappes de composants Est et Ouest embarquent maintenant trois processeurs chacune. Cela signifie que les deux MPU intégrées aux entrées du réseau, c'est-à-dire devant les grappes qui contiennent les processeurs, doivent maintenant traiter chacune les transactions matérielles de trois processeurs, donc de trois compartiments différents, simultanément.

Dans cette expérience, chacun des six compartiments est assigné statiquement à chacun

des six processeurs, selon la répartition suivante : `aes`, `cjpeg` et `djpeg` s'exécutent dans la grappe Ouest, tandis que `minimad`, `xvid_enc` et `xvid_dec` s'exécutent dans la grappe Est.

Les résultats de cette expérience, en utilisant les mêmes stratégies que précédemment, sont présentés dans le graphique de la figure 6.4. Comme on pouvait s'y attendre, les caches de MPU de petite taille implémentant la stratégie de correspondance directe ne peuvent définitivement pas supporter une telle charge. En revanche, les caches de MPU de taille raisonnable, comme le cache totalement associatif à 8 entrées, permettent de réduire le surcoût en performance à moins de 1%.

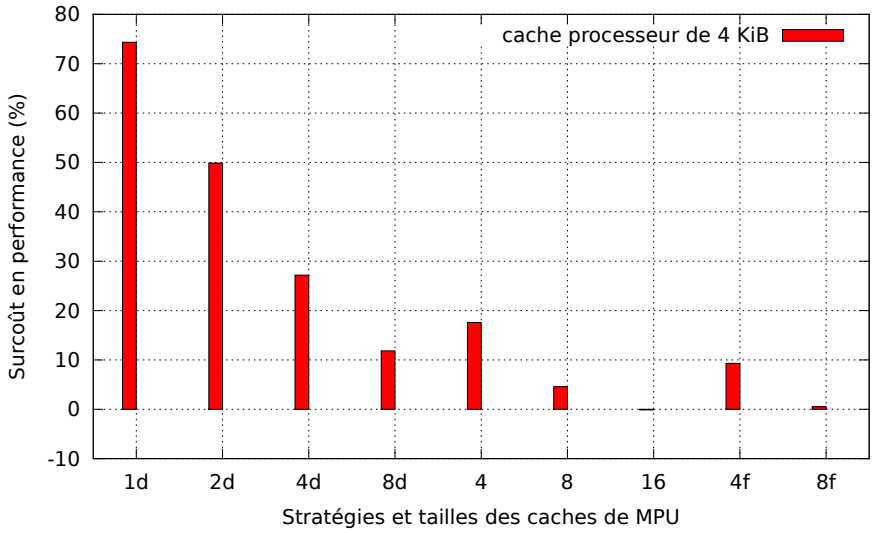


FIGURE 6.4 – Surcoût en performance de différentes stratégies de caches de MPU, avec trois processeurs face à chaque MPU.

6.3 Coût en surface de silicium

Les modifications matérielles engendrées par l'approche multi-compartiment ont touchées plusieurs composants physiques. Dans cette section, nous tentons de les évaluer, en termes de surcoût en surface de silicium.

6.3.1 Processeurs et caches processeur

Concernant la modification des processeurs, l'ajout du registre de CID « supplémentaire » peut être considéré comme négligeable, puisque dans le Mips32, un registre de contrôle similaire existe déjà. La transformation de l'interface entre les processeurs et leurs caches implique essentiellement l'ajout de fils électriques, et se révèle donc difficilement évaluable.

Concernant les caches processeur, si l'on fait l'hypothèse que le champ de CID occupe 8 bits, soit 1 octet, alors 1 octet supplémentaire est nécessaire par cases de caches pour étendre

les étiquettes. 1 octet supplémentaire est également requis pour étendre les adresses de chacune des entrées du tampon d'écriture, dont la profondeur est généralement équivalente au nombre de mots que contient une ligne de cache. En considérant un cache typique de 4 KiB comportant des lignes de caches de 16 mots, donc composé de 64 cases de caches, alors le surcoût en surface s'établit à $64 + 16 = 80 octets$, soit moins de 2% d'augmentation par rapport à la taille du cache ($80/4096 * 100 = 1,95\%$).

6.3.2 MPU

Les modules pare-feu, MPU, ont été uniquement décrits comme des modèles de simulation SystemC précis au cycle. Même si cette description est assez proche d'une description physique, telle qu'une description en langage *VHDL* le serait, elle n'est toutefois pas synthétisable, c'est-à-dire qu'il n'existe pas de chemin vers l'implémentation physique du composant. Nous ne sommes donc pas en mesure de fournir une valeur exacte du surcoût en surface en silicium causé par l'introduction des MPU dans le réseau sur puce. Cependant, nous pouvons en obtenir une bonne estimation en comptant le nombre de bits mémorisants internes des MPU, et en comparant cette valeur au nombre de bits mémorisants internes dans l'infrastructure de réseau sur puce initiale.

Pour cette estimation, nous nous replaçons dans le cas d'une architecture à plat, c'est-à-dire qu'un routeur du réseau sur puce ne connecte qu'une seul composant physique.

L'adaptateur de protocole (NIC) du réseau DSPIN contient 2 files d'attente matérielles, de type *FIFO*, de respectivement 160 et 132 bits mémorisants. Un NIC inclut donc $(160 + 132) = 292$ bits mémorisants. Un routeur DSPIN est composé d'un sous-routeur de commande et de réponse (les deux sous-réseaux sont séparés), qui contiennent tous les deux 10 FIFO de respectivement 160 et 132 bits mémorisants. Donc un routeur contient $((10 * 160) + (10 * 132)) = 2920$ bits mémorisants. L'infrastructure de réseau sur puce attachée à un composant physique compte alors $(292 + 2920) = 3212$ bits mémorisants.

Chaque entrée d'un cache de MPU contient 31 bits : 8 bits pour le CID, 20 bits pour l'étiquette d'adresse, 2 bits pour les droits d'accès et 1 bit pour le statut de la case de cache. Si l'on considère un cache de MPU totalement associatif de 8 entrées, il faut également ajouter un champ de 8 bits pour l'implémentation d'une stratégie de pseudo-LRU (afin de gérer le remplacement des cases de caches), ce qui porte le nombre de bits mémorisants du cache de MPU à $((8 * 31) + 8) = 256$ bits.

L'évaluation du surcoût en surface de silicium engendré par une MPU par rapport à l'infrastructure de réseau sur puce attachée un composant physique est donc de $(256/3212) = 8\%$. Si l'on prend en considération que les MPU n'équipent que les NIC reliant des composants initiateurs, le surcoût des MPU dans le cas de notre première plateforme d'expérimentation (deux initiateurs pour quatre cibles) se réduit alors à $((256 * 2)/(3212 * 6)) = 2,6\%$.

6.4 Conclusion

Concernant le surcoût en performance, nous avons mené deux expériences, une avec un processeur devant chaque MPU et une avec trois processeurs devant chaque MPU. D'après les résultats obtenus par ces deux expériences, nous pouvons conclure que le mécanisme de MPU est en mesure de protéger de multiples compartiments de taille variable (c'est-à-dire avec des occupations de l'espace d'adressage variées), d'une manière flexible (les pages peuvent être partagées entre les compartiments), tout en introduisant un surcoût en performance négligeable, même avec plusieurs composants initiateurs adressant la même MPU.

Concernant le surcoût en surface de silicium, nous nous sommes intéressés à la modification des caches processeur, qui cause un surcoût inférieur à 2%. Nous nous sommes également intéressés à la surface occupée par les MPU. Dans le cas de notre plateforme d'expérimentation, et avec une architecture de réseau sur puce à plat, les MPU ne représentent une augmentation que d'environ 2% par rapport à la surface initialement occupée par le réseau sur puce. On peut donc en conclure que ce surcoût en surface de silicium est négligeable.

Chapitre 7

Conclusion

Dans cette thèse, nous avons présenté une architecture de sécurité dynamique pour systèmes multiprocesseurs intégrés sur puce. Cette architecture, dénommée *multi-compartiment*, permet le partage sécurisé, dynamique et flexible de l'espace d'adressage entre plusieurs piles logicielles autonomes, s'exécutant sur une plateforme multiprocesseur potentiellement hétérogène.

Nous sommes donc en mesure de répondre précisément aux questions formulées lors de l'énoncé de la problématique :

- **Quelle architecture peut être cohérente avec des plateformes multiprocesseurs hétérogènes ?**
  La principale difficulté provoquée par l'utilisation d'une plateforme multiprocesseur hétérogène réside dans le fait que les mécanismes de protection mémoire existants sont locaux aux processeurs. Dans une optique de co-hébergement de plusieurs piles logicielles autonomes, cela signifie que ces mécanismes ne peuvent pas être contrôlés par une unique base de confiance et il en résulte alors une incohérence dans le partage et surtout la protection de l'espace d'adressage.
  L'approche multi-compartiment, que nous avons proposée, met en place un mécanisme de protection et de partage de l'espace d'adressage, global à la plateforme. Il est composé d'une partie logicielle et d'une partie matérielle. La partie logicielle, nommée *GTA* pour *Global Trusted Agent*, est chargée de définir les droits d'accès à l'espace d'adressage pour chacune des piles logicielles co-hébergées sur la plateforme. La partie matérielle est concrétisée par l'introduction de modules pare-feu matériels au cœur de la plateforme, c'est-à-dire au sein du réseau d'interconnexion. Ces modules contrôlent toutes les transactions matérielles transitant dans le réseau d'interconnexion, et vérifient qu'elles correspondent aux droits d'accès définis pour les piles logicielles qui les ont émises.
  Puisque la protection de l'espace d'adressage n'est plus dépendante des processeurs, mais s'appuie sur un mécanisme global à la plateforme, le problème de l'hétérogénéité des processeurs disparaît. On peut donc conclure que ce modèle de confiance permet

d'être cohérent avec des plateformes multiprocesseurs hétérogènes.

– **Cette architecture peut-elle fournir une garantie de partage sécurisé de la mémoire ?**

Nous avons appelé *compartiment*, l'association d'une pile logicielle avec un domaine de protection, c'est-à-dire une collection de droits d'accès spécifiques sur l'espace d'adressage. Pour réaliser le confinement des compartiments s'exécutant sur une même plateforme, le GTA affecte à chaque compartiment un identifiant logique distinct, que l'on a nommé *CID*. Grâce à des modifications mineures dans les composants initiateurs, c'est-à-dire les composants capables d'émettre des transactions dans le réseau d'interconnexion, on s'assure que chaque transaction qui transite dans la plateforme est étiquetée avec un CID. Le CID d'une transaction est alors utilisé par le mécanisme global de protection pour distinguer à quel domaine de protection la transaction doit être confrontée.

En rendant le mécanisme de protection, global à la plateforme, et surtout contrôlé par un agent de confiance extérieur aux compartiments que l'on confine, l'approche multi-compartiment garantit que le partage de l'espace d'adressage est sécurisé. Outre cette garantie de sécurité, le mécanisme est également très flexible puisque les domaines de protection peuvent se recouvrir, c'est-à-dire que plusieurs compartiments peuvent être autorisés à partager les mêmes zones dans l'espace d'adressage. Enfin, les domaines de protection étant définis par des tables des pages en mémoire, ce mécanisme est aussi très dynamique puisque le GTA peut modifier ces tables des pages au cours du temps.

– **Dans quel cadre et avec quelles restrictions, l'architecture peut-elle partager de façon sécurisée les composants initiateurs, de type processeurs ou périphériques DMA, ainsi que les périphériques cibles (autres que la mémoire) ?**

Les processeurs spécialisés ou coprocesseurs ne peuvent intrinsèquement pas être partagés entre différents compartiments puisqu'ils ne disposent pas de plusieurs niveaux de privilège. Dans ce cas, ils n'exécutent qu'un seul compartiment et ne nécessitent aucune modification matérielle particulière.

Les processeurs généralistes qui disposent de plusieurs niveaux de privilège peuvent être partagés entre plusieurs compartiments. Dans ce cas, la pile logicielle qui s'exécute dans le mode le plus privilégié d'un processeur généraliste est appelée *LTA* pour *Local Trusted Agent*. Le LTA peut alors organiser le partage des processeurs qu'il contrôle entre plusieurs compartiments de moindre privilège.

Concrètement, ce mécanisme signifie que l'approche multi-compartiment peut s'adapter à une plateforme complexe dans laquelle un groupe de processeurs de faible complexité est contrôlé par un noyau de système d'exploitation qui exécute des applications, un groupe de processeurs généralistes est contrôlé par un hyperviseur qui exécute des machines virtuelles, et enfin, un processeur spécialisé est contrôlé par une pile logicielle dédiée. En outre, ce mécanisme permet à l'approche multi-compartiment

d'être compatible avec les processeurs *TrustZone* proposés par *ARM*, que l'on peut considérer comme un cas particulier de processeurs multi-compartiments.
Grâce à un mécanisme d'héritage simple, l'utilisation exclusive de n'importe quel périphérique DMA par un compartiment est supportée. Le transfert effectué par le périphérique DMA bénéficie alors des mêmes droits d'accès que ceux du compartiment qui l'a configuré.
En complétant le mécanisme d'héritage d'autres modifications matérielles, les périphériques DMA de type mémoire vers mémoire peuvent être partagés entre plusieurs compartiments. Dans ce cas, le problème de la redirection des interruptions a été traité. Lorsqu'un transfert pour le compte d'un certain compartiment est terminé, l'interruption est envoyée automatiquement au processeur sur lequel le compartiment en question s'exécute, rendant le traitement des interruptions le plus sécurisé et efficace possible.
Ne pouvant pas émettre de transaction, les périphériques cibles peuvent être utilisés de façon exclusive par un compartiment. En revanche, nous n'avons pas traité leur partage entre plusieurs compartiments, puisque cela impliquait la mise en place de mécanismes *ad hoc* à chaque type de périphériques. Pour la même raison, nous n'avons pas traité les périphériques DMA de type mémoire vers interface externe.

– **Quel est le coût d'une telle architecture, aussi bien du point de vue logiciel que matériel ?**
Afin d'évaluer le coût de l'approche multi-compartiment, nous avons construit une plateforme d'expérimentation, sous la forme d'un prototype virtuel. La plateforme est composée d'une partie matérielle et d'une partie logicielle. La partie matérielle est composée de plusieurs processeurs, supportant une utilisation multi-compartiment, et est basée sur un réseau sur puce dans lequel a été intégré le mécanisme de protection de l'espace d'adressage. La partie logicielle est composée d'une unique pile logicielle de confiance, qui cumule donc les rôles de LTA et de GTA, et qui exécute plusieurs compartiments.
Au niveau logiciel, nous avons évalué que l'implémentation des fonctions de LTA et de GTA dans un système d'exploitation existant était simple.
Au niveau matériel, plusieurs simulations ont confirmé que le mécanisme de protection intégré au réseau sur puce provoque un surcoût en performance négligeable, même lorsque le mécanisme est testé au-delà des hypothèses définies initialement.
Le surcoût en surface de silicium concerne les modifications matérielles nécessaires à la mise en œuvre de l'approche multi-compartiment. Nos évaluations montrent que le surcoût dû aux modifications des processeurs généralistes est négligeable, de même que le surcoût dû à l'introduction du mécanisme de protection dans le réseau sur puce.
On peut donc conclure que l'ajout du mécanisme de protection, qui assure le partage de l'espace d'adressage de manière globale, est transparent.

– **Est-il possible d'intégrer cette architecture de sécurité dans différents ré-**

seaux d'interconnexion de façon à la rendre pérenne pour les architectures matérielles futures ?

Les anciens réseaux d'interconnexion, tels que les bus partagés, sont en fort déclin dans les systèmes intégrés sur puce récents. Les concepteurs lui préfèrent aujourd'hui le réseau sur puce, qui garantit notamment une scalabilité de la bande passante, caractéristique nécessaire pour les systèmes multiprocesseurs complexes, tels que les systèmes orientés multimédia.

Ayant démontré dans ce document qu'un mécanisme de sécurité peut apporter la flexibilité et la dynamicité que les systèmes complexes de prochaine génération requièrent, tout en s'intégrant de façon transparente dans un réseau sur puce, on peut en conclure que ce type de mécanisme devrait s'imposer comme incontournable dans le futur.

Perspectives

La prochaine étape du travail présenté dans cette thèse est d'étudier la faisabilité et le coût d'un mécanisme global qui effectuerait la protection de l'espace d'adressage, mais également sa virtualisation. Ce nouveau mécanisme matériel et logiciel introduirait une nouvelle couche de mémoire virtuelle paginée, embarquée dans le réseau sur puce. Le mécanisme matériel utiliserait sa propre collection de pages des tables, définie par un agent global de confiance. Ce dernier agirait alors comme une sorte d'hyperviseur de la plateforme. Il semble quasiment certain qu'une telle unité de mémoire virtuelle pour réseau sur puce (*NoC-MMU*) fournirait une grande flexibilité concernant le partage de l'espace d'adressage. Mais elle introduirait également de nouveaux problèmes à résoudre, notamment concernant le support de certaines piles logicielles « héritées » programmées pour fonctionner dans l'espace d'adressage physique, concernant les communications entre les différents compartiments (par exemple, pour les échanges de pointeurs d'adresse mémoire), et concernant le surcoût, en surface de silicium (un cache de traduction d'adresse est plus large qu'un cache de protection) et en performance (la traduction d'adresse n'est pas forcément parallélisable avec la conversion de protocole du réseau sur puce).

Références bibliographiques

[1] Joël Porquet, Christian Schwarz, and Alain Greiner. Multi-compartment: a new architecture for secure co-hosting on SoC. In *SoC'09: Proceedings of the 11th International Symposium on System-on-Chip*, pages 124–127, 2009. 2

[2] Joël Porquet, Christian Schwarz, and Alain Greiner. NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs. In *Accepted at DATE'11: the Conference on Design, Automation and Test in Europe*, 2011. 2

[3] Joël Porquet and Christian Schwarz. Method of routing an interrupt signal directly to a virtual processing unit in a system with one or more physical processing units. European Patent: EP-2-157-511-A1, US Patent: US-20100049892, 2010. 2, 69

[4] Joël Porquet and Christian Schwarz. Method for enabling several virtual processing units to directly and concurrently access a peripheral unit. European Patent: EP-2-221-730-A2, US Patent: US-20100161854, 2010. 2, 56

[5] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. Security as a new dimension in embedded system design. In *DAC'04: Proceedings of the 41st annual Design Automation Conference*, pages 753–760, 2004. 6

[6] Cloakware Inc. Security Impacts of Next-Generation Set-Top Boxes. White paper: `http://www.csimagazine.com/pdf/Cloakware-STB.pdf`, 2009. 7

[7] The Digital Living Network Alliance. `http://www.dlna.org`. 8

[8] The Open Handset Alliance. `http://www.openhandsetalliance.com`. 8

[9] TSAR. `https://www-soc.lip6.fr/trac/tsar`. 11

[10] Kaustav Banerjee, Shukri J. Souri, Pawan Kapur, and Krishna C. Saraswat. 3-D ICs: A Novel Chip Design for Improving Deep-Submicrometer Interconnect Performance and Systems-on-Chip Integration. *Proceedings of the IEEE*, 89(5):602–633, May 2001. 12

[11] Yoann Padioleau, Julia Lawall, René Rydhof Hansen, and Gilles Muller. Documenting and automating collateral evolutions in Linux device drivers. In *Eurosys'08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, pages 247–260, 2008. 18

[12] US Department of Defense. Department of Defense Trusted Computer System Evaluation Criteria. DOD 5200.28-STD, December 1985. 19

[13] Alexandre Becoulet, Nicolas Pouillon, and Joël Porquet. MutekH: An operating system with native support for multiprocessor heterogeneity. 2011. 19

[14] Alexandre Becoulet. *Conception d'un système d'exploitation supportant nativement les architectures multiprocesseurs hétérogènes à mémoire partagée*. PhD thesis, Université Pierre et Marie Curie, Paris, France, 2010. 19, 77, 87

[15] Xavier Guerin and Frédéric Pétrot. A System Framework for the Design of Embedded Software Targeting Heterogeneous Multi-core SoCs. In *ASAP'09: Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 153–160, 2009. 19

[16] Robert J. Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25(5):483–490, 1981. 26

[17] Gerald J. Popek and Robert P. Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974. 26

[18] Robert P. Goldberg. *Architectural Principles for Virtual Computer Systems*. PhD thesis, Harvard University, Cambridge, Massachusetts, USA, 1973. 27

[19] Ian Pratt, Keir Fraser, Steven Hand, Christian Limpach, Andrew Warfield, Dan Magenheimer, Jun Nakajima, and Asit Mallick. Xen 3.0 and the Art of Virtualization. In *Proceedings of the Linux Symposium*, July 2005. 28

[20] Fabrice Bellard. QEMU, a fast and portable dynamic translator. In *ATEC'05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 41–46, 2005. 28

[21] Kevin P. Lawton. Bochs: A Portable PC Emulator for Unix/X. *Linux Journal*, page 7, 1996. 28

[22] Open Kernel Labs. `http://www.ok-labs.com`. 28

[23] VMWare. `http://www.vmware.com/products/mobile/index.html`. 28

[24] VirtualLogix. `http://www.virtuallogix.com`. 28

[25] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–13, 2006. 29, 33

[26] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel Virtualization Technology for Directed I/O. *Intel Technology Journal*, 10(3), August 2006. 29, 34

[27] Peter H. Gum. System/370 extended architecture: facilities for virtual machines. *IBM Journal of Research and Development*, 27(6):530–544, 1983. 33

[28] John Fisher-Ogden. Hardware Support for Efficient Virtualization. Technical report, University of California, San Diego, USA, 2006. 33

[29] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L. Santoni, Fernando C. M. Martins, Andrew V. Anderson, Steven M. Bennett, Alain Kagi, Felix H. Leung, and Larry Smith. Intel Virtualization Technology. *Computer*, 38(5):48–56, 2005. 33

[30] Advanced Micro Devices, Inc. *AMD64 Virtualization Codenamed "Pacifica" Technology: Secure Virtual Machine Architecture Reference Manual*, May 2005. 33

[31] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel Virtualization Technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), August 2006. 33

[32] Advanced Micro Devices, Inc. AMD-V Nested Paging. Whitepaper: http://developer.amd.com/assets/NPT-WP-11-final-TM.pdf, July 2008. Revision 1.0. 33

[33] Advanced Micro Devices, Inc. AMD I/O Virtualization Technology (IOMMU) Specification. Whitepaper: http://support.amd.com/fr/Processor_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf, February 2009. PID 34434 Rev 1.26. 34

[34] Tiago Alves and Don Felton. ARM TrustZone: Integrated Hardware and Software Security. Whitepaper: http://www.arm.com, July 2004. 34

[35] ARM Ltd. *AMBA AXI Protocol Specification*, v1.0b edition, 2004. 36

[36] Joel Coburn, Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. SECA: security-enhanced communication architecture. In *CASES'05: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 78–89, 2005. 37, 38

[37] ARM Ltd. *AMBA Specification*, Rev 2.0 edition, 1999. 38

[38] Arteris. A comparison of Network-on-Chip and Busses. Whitepaper, 2005. 39

[39] Matthieu Texier. Network-on-Chip security: overview of existing solutions. Technical report, South Britanny University, France, 2009. http://pub.mtexier.com/m2imars/research_methodology_day/article_NoC_Security_final_v2.pdf. 39

[40] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE'00: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 250–256, 2000. 39

[41] Luca Benini and Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. *Computer*, 35:70–78, 2002. 39

[42] Sonics, Inc. SonicsMX SMART Interconnect Solution. Whitepaper: http://www.sonicsinc.com/uploads/pdfs/sonicsmx_DS_021610.pdf, 2005. 40

[43] Jean-Philippe Diguet, Samuel Evain, Romain Vaslin, Guy Gogniat, and Emmanuel Juin. NOC-centric Security of Reconfigurable SoC. In *NOCS'07: Proceedings of the 1st International Symposium on Networks-on-Chip*, pages 223–232, 2007. 41

[44] Samuel Evain. *μSpider : Environnement de Conception de Réseau sur Puce*. PhD thesis, Institut Nationale des Sciences Appliquées de Rennes, France, 2006. 41

[45] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, and Cristina Silvano. A data protection unit for NoC-based architectures. In *CODES+ISSS'07: Proceedings of the 5th IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, pages 167–172, 2007. 42, 59

[46] Leandro Fiorin, Gianluca Palermo, and Cristina Silvano. A security monitoring service for NoCs. In *CODES+ISSS'08: Proceedings of the 6th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 197–202, 2008. 42

[47] Leandro Fiorin, Gianluca Palermo, Slobodan Lukovic, Valerio Catalano, and Cristina Silvano. Secure Memory Accesses on Networks-on-Chip. *IEEE Transactions on Computers*, 57(9):1216–1229, 2008. 42

[48] VSI Alliance. *Virtual Component Interface Standard*, Version 2, OCB 2 2.0 edition, April 2001. 49, 76

[49] OCP-IP Alliance. *Open Core Protocol Specification*, Release 2.1 edition, 2005. 49, 66

[50] Emmett Witchel, Josh Cates, and Krste Asanović. Mondrian Memory Protection. In *ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 304–316, 2002. 62

[51] Jochen Liedtke. Address space sparsity and fine granularity. In *EW'6: Proceedings of the 6th Workshop on ACM SIGOPS European Workshop*, pages 78–81, 1994. 62

[52] Edith Beigne, Fabien Clermidy, Pascal Vivet, Alain Clouard, and Marc Renaudin. An asynchronous NOC architecture providing low latency service and its multi-level design framework. In *ASYNC'2005: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 54–63, March 2005. 66, 76

[53] Tobias Bjerregaard, Shankar Mahadevan, Rasmus G. Olsen, and Jens Sparsoe. An OCP Compliant Network Adapter for GALS-based SoC Design Using the MANGO Network-on-Chip. In *SoC'05: Proceedings of the 7th International Symposium on System-on-Chip*, pages 171–174, November 2005. 67

[54] PCI-SIG. *PCI Local Bus Specification*, Rev 2.2 and later edition, 1998–2004. 67

[55] PCI-SIG. *PCI Express Base Specification*, Rev 1.1 edition, 2005. 67

[56] Robert Rose. Survey of system virtualization techniques. Technical report, 2004. 69

[57] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: a scalable isolation kernel. In *EW'10: Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, pages 10–15, 2002. 69

[58] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical report, University of Washington, USA, February 2002. 69

[59] SoCLib. `http://www.soclib.fr`. 75

[60] O.S.C. Initiative. SystemC. `http://www.systemc.org`, 1999. 75

[61] Ivan Miro Panades. *Conception et implantation d'un micro-réseau sur puce avec garantie de service*. PhD thesis, Université Pierre et Marie Curie, Paris, France, 2008. 76, 82

[62] MIPS Technologies. *MIPS32 Architecture For Programmers*, Revision 0.95 edition, March 2001. Volume I: Introduction to the MIPS32 Architecture, Volume II: The MIPS32 Instruction Set, Volume III: The MIPS32 Privileged Resource Architecture. 76, 78

[63] Adam Rose, Stuart Swan, John Pierce, and Jean-Michel Fernandez. Transaction Level Modeling in SystemC. http://www.systemc.org, 2005. OSCI TLM Working Group. 76

[64] Richard Buchmann. *Modélisation et simulation rapide au niveau cycle pour l'exploration architecturale de systèmes intégrés sur puce*. PhD thesis, Université Pierre et Marie Curie, Paris, France, 2010. 76

[65] Alexandre Becoulet et al. MutekH. http://www.mutekh.org. 77

[66] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. Exokernel: an operating system architecture for application-level resource management. In *SOSP'95: Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 251–266, December 1995. 77

[67] Nicolas Pouillon, Alexandre Becoulet, Aline Vieira de Mello, Francois Pecheux, and Alain Greiner. A Generic Instruction Set Simulator API for Timed and Untimed Simulation and Debug of MP2-SoCs. In *RSP'09: Proceedings of the IEEE/IFIP International Symposium on Rapid System Prototyping*, pages 116–122, 2009. 79

[68] IEEE. *POSIX 1003.1c Threads Extension*, 1995. 86

[69] The Santa Cruz Operation, Inc. *System V Application Binary Interface : MIPS RISC Processor Supplement*, 3rd edition edition, February 1996. 89

[70] Jason E. Fritts, Frederick W. Steiling, Joseph A. Tucek, and Wayne Wolf. MediaBench II video: Expediting the next generation of video systems research. *Microprocessors and Microsystems*, 33(4):301–318, 2009. 97

[71] Christoph Lampert, Michael Militzer, Peter Ross, et al. Xvid MPEG-4 core library. http://www.xvid.org. 97

[72] Underbit Technologies, Inc. MAD: MPEG Audio Decoder. http://www.underbit.com/products/mad/. 97

[73] Joan Daemen and Vincent Rijmen. AES Proposal: Rijndael, 1999. 97