Exploration d'architectures basée sur la génération automatique de plates-formes matérielles et le portage rapide du logiciel

M. Fiandino

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N°attribué par la bibliothèque
|_|_|_|_|_|_|_|_|_|_|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : *« Micro et Nano Electronique »*
préparée au laboratoire **TIMA**
dans le cadre de **l'Ecole Doctorale** *« Électronique, Électrotechnique, Automatique, Télécommunications, Signal »(EEATS)*

présentée et soutenue publiquement

par

Maxime FIANDINO

le 02 Mai 2007

Titre :

Exploration d'architectures basée sur la génération automatique de plates-formes matérielles et le portage rapide du logiciel

Directeur de thèse :
Ahmed Amine Jerraya

JURY :

M. Frédéric PÉTROT , Président
M. François TERRIER , Rapporteur
M. Ahmed Amine JERRAYA , Directeur de thèse
M. Alain CLOUARD , Co-encadrant
M. Frédéric DESPREZ , Examinateur
M. Tanguy RISSET , Examinateur

Table des matières

Table des figures

Glossaire

BCA (Bus Cycle Accurate) : Modélisation où les interconnexions ont une précision au niveau cycles d'horloges.

BFM (Bus Fonctionnal Model) : Niveau de modélisation aux cycles d'horloges mais avec des composants non tous synthétisables (ISS cycle accurate).

DMA (Direct Memory Access) : Accès direct à une mémoire sans passer par son processeur ce qui permet d'accélérer les transferts et de libérer le processeur.

FIFO (First In First Out) : Zone de stockage où les données sont lues dans l'ordre des écritures, soit une file.

HMPSoC (Heterogeneous MPSoC) : SoC contetant de multiples processeurs différents.

ISS (Instruction Set Simulator) : Programme binaire ou librairie qui simule un processeur en analysant un exécutable qui est destiné au type de processeur simulé (crosscompilé).

ITC (InTerrupt Controler) : Périphérique local d'un processeur chargé de la gestion des interruptions.

MMU (Memory Management Unit) : Unité de gestion de la mémoire. Une MMU se charge de deux tâches. D'une part, la protection de la mémoire, un programme tentant d'accéder à une zone de mémoire à laquelle il n'a pas droit provoquera une exception. D'autre part, la translation des blocs mémoire qui permet de déplacer à volonté la mémoire

physique pour créer un espace de mémoire logique disposé différemment.

MPSoC (Multi-processor System-On-Chip) : un SoC contenant plusieurs processeurs de tous types.

PV (Programmer's view) : Modèle d'une architecture avec une précision correspondant à la vue d'un logiciel embarqué.

PVT (PV timed) : Modèle PV avec une gestion du temps.

RTL (Register Transfert Level) : Précision d'un modèle au niveau des portes logiques.

SoC (System-On-Chip) : Système intégré sur un unique circuit silicium ou monopuce.

SPIRIT (Structure for Packaging Integrating and Re-using Ip within Tool flows) : Un ensemble de normes définies par le consortium spirit. Elles ont pour but d'assurer la compatibilitée des descriptions des IPs, à partir du fournisseur jusqu'aux outils de conception.

TLM (Transaction Level Modelling) : Niveau de modélisation où les communications sont abstraites sous la forme de transactions atomiques. La définition étant large, de nombreux sous niveaux existent.

XML (eXtensible Mark-up Language) : Un langage définissant des balises extensibles qui permettent une structuration des données

Remerciements

Je tiens à exprimer toute ma reconnaissance à M. JERRAYA et CLOUARD pour leur encadrement, leurs nombreux conseils et leur soutient constant tout au long de ma thèse. M. PÉTROT m'a fait l'honneur d'accepter d'être président de mon jury de thèse. Je lui exprime ma profonde gratitude. Je remercie M. DÉRUTIN et TERRIER d'avoir accepté d'être rapporteurs de ma thèse, ainsi que pour leurs jugements sur mon manuscrit, tant sur le fond que sur la forme. Je remercie M. RISSET et DESPREZ pour avoir accepté de faire partie de mon jury de thèse.

Je remercie tous les chercheurs, enseignants et membres du personnel du groupe SLS du laboratoire TIMA.

Je remercie les membres de l'équipe SPG (EX: Sysar) de STMicroelectronics, Frank GHENASSIA pour m'avoir accepté dans son équipe, Laurent Maillet-Contoz pour toutes les discussions sur mes travaux, Jean-Philippe Strassen pour les discutions techniques, et enfin Laurent, Antoine, Herve, Olivier, Julien, Michel, Andreï, Marc, Eric, Nicolas, Vincent, Anthony et Stephane pour me supporter au travail. Il reste mes collègues thésards, Mathieu, Claude, Jérome et Sébastien.

Je remercie mes collègues du groupe SLS du TIMA pour m'avoir accepté parmi eux lors des réunions de groupes.

Pour finir je remercie mes parents pour leur soutient tout au long de mon début d'existence et pour leur soutient logistique à la soutenance, ma fille pour m'avoir laissé dormir de temps en temps, et ma femme pour trop de chose pour pouvoir les citer.

Introduction

L'évolution actuelle des composants électroniques incluant des SoCs montre de nouvelles contraintes. Ces composants doivent traiter des algorithmes plus complexes et plus nombreux, tel que communications, jeux, multimédias. Mais ils doivent aussi offrir plus de flexibilité, supporter de multiples algorithmes dans de nombreuses configurations, voire s'adapter aux futures évolutions des algorithmes. Cette flexibilité et la puissance de calcul nécessaire en constante augmentation amènent l'industrie à se tourner vers des SoCs contenant de multiples processeurs hétérogènes, appelés des HMPSoCs (Heterogeneous Multi Processor System-On-Chi :Système sur puce multi-processeurs hétérogène). Malgré ces améliorations continues, le temps de mise sur le marché reste le facteur clef de la réussite d'un circuit. Il est nécessaire d'adapter le flot de conception à cette nouvelle génération de puces.

Une architecture HMPSoC est un ensemble de composants et d'interconnexions, processeurs, caches, mémoires, réseaux, périphériques matériels. Elle nécessite des programmes binaires pour chacun de ses processeurs. Ces programmes doivent être adaptés aux processeurs et à l'architecture. Le concepteur d'architecture dispose d'un temps limité pour la réalisation d'un système matériel. Comme les logiciels doivent être adaptés suivant l'architecture matérielle, cette étape ne peut être réalisée qu'après la finalisation de l'architecture. Cette finalisation implique d'avoir évalué les performances de cette architecture.

Trois méthodes d'évaluation d'architectures sont possibles pour un futur HMPSoC:

- Par une description en langage de haut niveau.Cette méthode est très rapide mais il n'y a ni notion d'architecture ni de coûts.
- Par calcul avec une évaluation analytique, c'est à dire des équations mathématiques, mais dans ce cas, la formalisation est difficile car il faut formaliser des éléments dynamiques, tel que des processeurs et des caches. Les évolutions de ces équations sont difficiles. La précision est faible au niveau de la plateforme entière, du fait des difficultés de formalisation des interactions entre éléments dynamiques lors des communications, .
- Par simulation de l'architecture matérielle et logicielle ce qui permet une meilleure précision et la possibilité de modifications, mais, dans ce cas, le temps de construction est important. Pour la simulation, la méthode actuelle pour accélérer le développement du modèle exécutable consiste à le concevoir au niveau de modélisation dit "transactionnel" (TLM). La conception, le développement et la simulation sont plus rapides qu'au niveau RTL. Malgré ce gain, ces temps doivent encore être réduits pour s'adapter au nombre de composants importants que contiendront les circuits de type HMPSoC.

En contexte industriel, la simulation est ainsi l'approche la plus adaptée. Un modèle exécutable doit donc être développé par le concepteur d'architecture pour être simulé. Suivant les résultats de simulation l'architecture sera éventuellement modifiée puis resimulé :il s'agit dans ce cas d'une exploration architecturale.

En pratique, il est impossible de réaliser cette exploration avec une évaluation des performances précise pour des HMPSoCs car la construction d'un modèle exécutable reste longue et coûteuse. Il manque des outils d'automatisation.

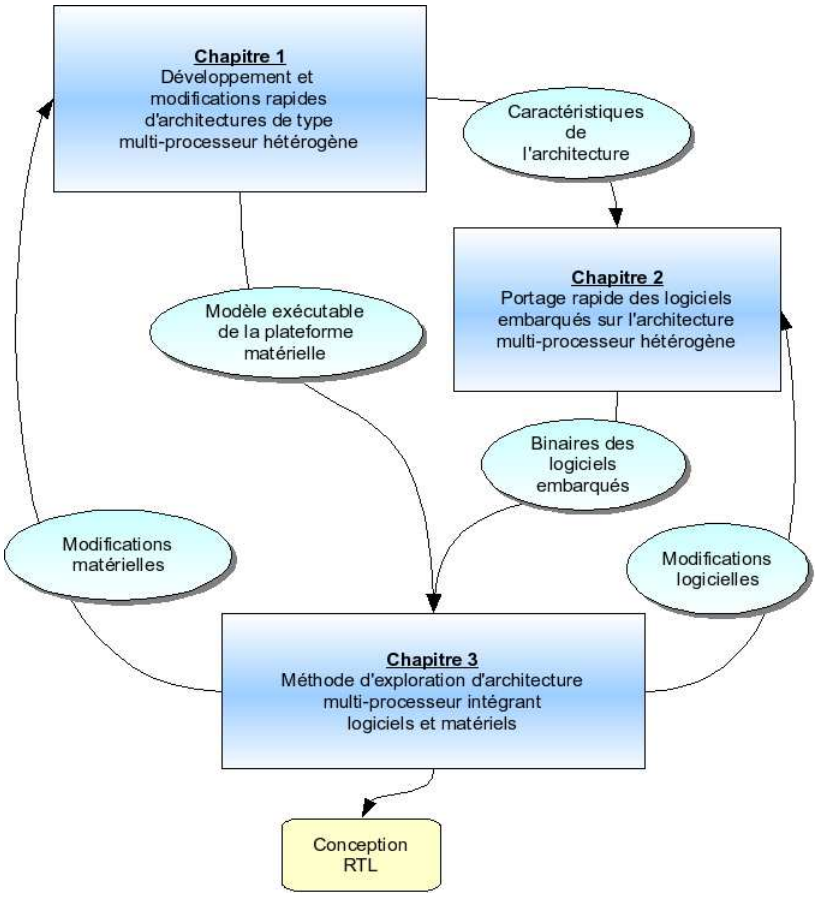

FIG. 1 – Ensemble du flot proposé dans cette thèse découpé par rapport aux chapitres

La solution recherchée dans cette thèse a pour but de fournir des outils automatiques de composition d'architecture HMPSoC logicielle et matérielle qui incluent un grand nombre de processeurs, permettant la simulation au niveau TLM. Ceci permet de garder les avantages de la modélisation TLM tout en diminuant les temps de réalisation et de modification de l'architecture.

Actuellement certains des HMPSoCs modélisés dans l'équipe *"System Platform Group"* (responsable de la modélisation TLM dans STMicroelectronics) contiennent une vingtaine de processeurs. Certains projets envisagent même d'inclure plus de 100 processeurs. L'hypothèse de HMPSoC contenant plus de mille processeurs qui a amené à cette thèse semble donc se profiler, et les résultats de cette thèse permettent au flot de cosimulation d'être prêt à cette évolution.

La proposition consiste en la génération de modèle de plateformes TLM exécutables composées de sous-systèmes de traitement. Ces sous-systèmes

sont générés puis interconnectés entre eux. Ils sont conçus au niveau TLM et contiennent des composants standards. Un générateur de tels sous-systèmes est décrit, section 1.6.1.

Un générateur automatique de modèles TLM exécutables de HMPSoC avec des sous-systèmes de traitement ainsi qu'une description des interconnexions a également été mis en place. Il est décrit dans la section 1.5. Ce générateur a été utilisé sur différentes architectures en vue de démontrer l'approche.

Le générateur été validé par la création rapide de plusieurs plateformes multi-processeurs suivie d'explorations architecturales. Les itérations des explorations d'architectures ont pris de quelques minutes à quelques heures au lieu de quelques jours à quelques semaines.

Les programmes embarqués sur ces architectures peuvent être découpés en deux parties, haut et bas niveau. D'une part, la partie "haut niveau" contient les algorithmes à implémenter, par exemple une fonction de traitement du signal. Elle est souvent développée indépendamment par avance, en tant que programme sur une station de travail. D'autre part, la partie "bas niveau" ou dépendante du matériel qui contient la partie du code spécifique au processeur et à l'architecture. Elle ne peut être conçue que lorsque l'architecture est fixée, ce qui allonge donc la durée de conception totale du HMPSoC final qui comprend le logiciel et le matériel.

De plus dans le cas d'explorations d'architectures, les spécifications matérielles sont modifiées de manière itérative. Le logiciel de bas niveau n'est alors plus adapté. Le travail d'adaptation est à effectuer rapidement.

Ces opérations d'adaptation peuvent nécessiter la réécriture d'une partie du code du logiciel de bas niveau. Il faut trouver les modifications de l'architecture puis en déduire les impacts sur les logiciels. Ce travail est long, fastidieux et donc générateur d'erreurs. Son coût en temps est tel que l'on ne peut se permettre de le réaliser plusieurs fois lors de l'exploration d'architecture.

Pour l'écriture de la partie bas niveau d'un logiciel parallélisé existant sur une architecture matérielle cible définie, il est possible de procéder soit manuellement, soit via des outils qui permettent la génération d'une partie de ce code bas niveau. Cette partie de code peut être le système d'exploitation, la gestion des tâches seule et/ou celle des communications. Dans tous les cas, la description de l'architecture doit être saisie dans un formalisme compréhensible par l'outil. Cette description doit être modifiée lors de chaque exploration d'architecture.

Pour résoudre ce problème il faut automatiser l'extraction des caractéristiques de l'architecture nécessaire aux logiciels embarqués. Un mécanisme

pour extraire automatiquement cette description d'un modèle exécutable d'architecture matérielle est proposé, section 2.2.2.

Ce mécanisme d'extraction a été développé. Il utilise une primitive particulière pour trois types de composants, maître, communication et esclave. De plus un exemple d'utilisation avec génération de code et gestion de la compilation a été implémenté pour des processeurs de type ARM7TDMI et ARM926EJS.

Une utilisation en a ensuite été faite sur un algorithme d'encodage vidéo. Les mécanismes de gestion des logiciels embarqués ont été générés, avec les scripts de compilation, d'édition des liens et les fichiers assembleur d'initialisations. Des explorations d'architectures ont été effectuées: modification du nombre de processeurs, de la taille des mémoires, et présence ou non de contrôleur d'écran, avec la modification, recompilation et chargement automatiques des logiciels embarqués.

Il faut ensuite valider l'exécution conjointe des logiciels avec le matériel, l'ensemble devant permettre des mesures de performances pour différentes instances de l'architecture.

Deux techniques sont possibles pour réaliser cette exécution. La première technique consiste en l'utilisation de machines d'émulation. Cependant cette solution implique l'utilisation de modèles RTL longs à concevoir et modifier. Le coût financier prohibitif des machines limite la disponibilité. La visibilité sur le HMPSoC est limitée et les puces de grande taille ne peuvent être simulées qu'en partie. Par contre la précision est particulièrement importante. Il est préférable de n'utiliser l'émulation qu'en dernière étape, sur l'architecture figée. La seconde technique consiste en la simulation rapide de modèle TLM via l'utilisation du simulateur SystemC, par exemple via la bibliothèque *TLM OSCI*. Diverses techniques sont utilisées pour accélérer la simulation de processeurs. Pour la mesure de performance, ce sont des simulateurs de jeux d'instructions ISS (*Instruction Set Simulator*) qui sont utilisés. Pour la vitesse et la vérification fonctionnelle ce sont des simulations natives. Pour une meilleure complémentarité, il faudrait pouvoir choisir à chaque exécution (voire même pendant l'exécution) entre soit une simulation rapide et peu précise, soit une simulation avec une précision accrue mais au prix d'une simulation plus longue.

Dans cette thèse une méthode est définie pour l'écriture ou le portage de logiciels multi-cibles vers une modélisation et une simulation multi-niveau, à la section 1.3. Celle-ci inclut la gestion des communications et des synchronisations.

Une API de synchronisation et de communication est définie pour le logiciel embarqué et l'adaptateur matériel de *simulation native*, voir section 1.2.2.

Ce mécanisme est incorporé aux outils de génération de la plateforme exécutable et de gestion du logiciel embarqué.

La simulation multi-niveau a été évaluée lors de simulations de modèles logiciel et matériel d'encodage H264. La simulation rapide d'une plateforme avec dix processeurs a permis une vitesse d'encodage avec pré et post visualisation de 1 image de taille 640×480 pixels toutes les 2 secondes. La simulation précise avec ISS réalisée après la validation du logiciel demande environ une heure par image.

La figure 1 décrit le flot proposé dans cette thèse découpé par rapport aux chapitres, ainsi que les intéractions entre eux.

Le premier chapitre concerne un outil pour la création ou modification du modèle exécutable de l'architecture de type multiprocesseurs hétérogènes. Le second chapitre expose un outil pour le portage rapide des logiciels embarqués sur une architecture multiprocesseurs hétérogène. Le troisième chapitre décrit une méthode pour des simulations avec exécution du modèle incluant logiciel et matériel.

Ces trois étapes sont obligatoires pour effectuer l'exploration d'architecture. Elles sont de plus liées entre elles. Le portage du logiciel demande la connaissance des caractéristiques de l'architecture. La validation de l'architecture requiert les binaires des logiciels embarqués portés. La simulation requiert un modèle exécutable de l'architecture ainsi que les binaires des logiciels embarqués qui ont été portés.

Chacune des parties de cette thèse est rapportée respectivement à ces trois étapes pour proposer un flot d'exploration d'architecture HMPSoC complet. Chaque chapitre commence par le diagramme de l'ensemble du flot sur lequel sont précisées les étapes abordées.

Chapitre 1

Développement et modifications rapides d'architectures de type multi-processeurs hétérogène

Ce chapitre propose une méthode de développement rapide de modèle TLM exécutable de plateforme multi-processeurs. Un outil de génération de tels modèles de plateformes matérielles TLM exécutables a été développé. Il prend en entrée des fichiers de composition d'architecture et utilise une bibliothèque de générateurs de sous-systèmes. Ces sous-systèmes sont un assemblage de composants matériels.

La figure 2 représente le flot d'utilisation de l'outil de génération de plateforme matérielle TLM exécutable ainsi que sa position vis-à-vis des parties du flot global de conception de plateforme logicielle et matérielle détaillées dans les autres chapitres.

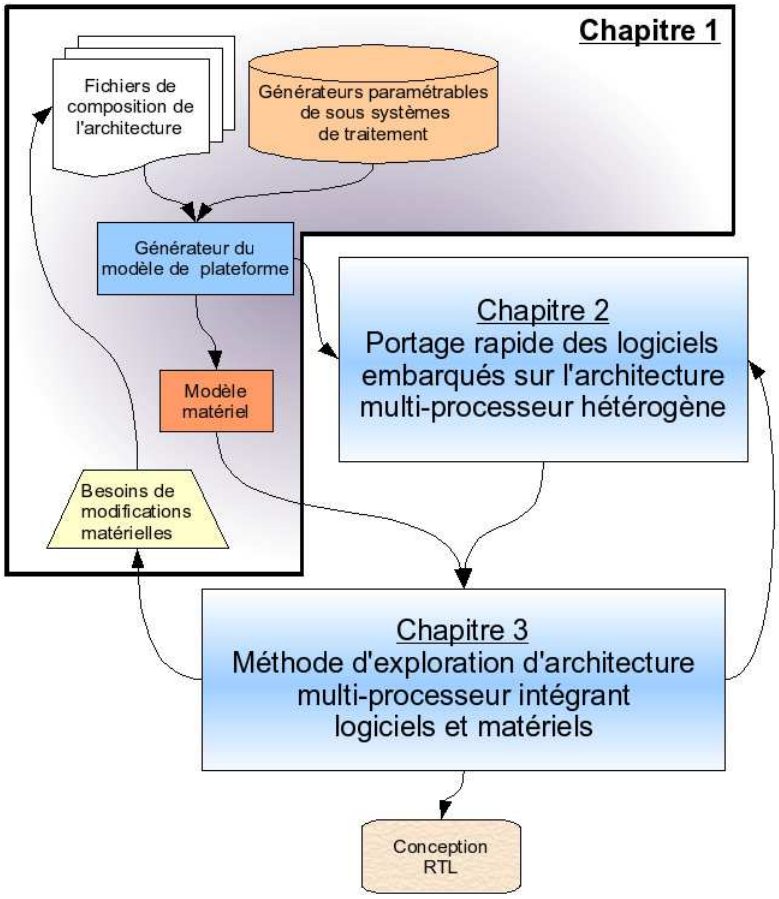

FIG. 2 – Développement du modèle exécutable de l'architecture matérielle

La première section décrit la problématique de la conception de systèmes sur puce multi-processeurs hétérogène au niveau système. Un état de l'art

décrit les différentes solutions pour la simulation des processeurs section 1.2. Une technique est proposée pour réaliser une simulation multi-niveau des logiciels embarqués à partir des même codes sources de haut niveau section 1.3. Ensuite un état de l'art des outils d'évaluation et de modification de processeurs est effectué. Puis, une proposition d'outil pour générer (créer et modifier) un modèle exécutable TLM d'architecture, est détaillé. Enfin un exemple teste le générateur pour un circuit du domaine multimédia.

1.1

La conception de système sur puce multi-processeurs hétérogène avec modélisation au niveau système

Un système multi-processeurs est une architecture contenant plusieurs processeurs et composants matériels interconnectés entre eux. Le système multi-processeurs hétérogène se différencie des systèmes dit homogènes par des irrégularités dans la topologie, par la présence variée (voire hétéroclites) de processeurs ou composants matériels, ou par des interconnexions non symétriques.

L'arrivée de systèmes multi-processeurs hétérogènes contenant un nombre important de processeurs différents entraîne de nouvelles difficultés de conception. Cette conception de systèmes sur puce repose toujours sur un cahier des charges initial. Cependant la complexité accrue de l'architecture ainsi que des logiciels associés demande de nouvelles méthodes pour aider les concepteurs. Un flot de conception avec modélisation au niveau système utilisé actuellement chez STMicroelectronics^{TM}est décrit. Enfin suivant des besoins d'exploration tel que l'ont souhaités divers concepteurs d'architectures, un exemple de flot idéal est proposé à la fin de cette section 1.1 pour la conception de système sur puce multi-processeurs hétérogène avec modélisation au niveau système. Ce flot est comparé aux flots existant et au flot proposé dans cette thèse, dans la section 1.4.

1.1.1 Les motivations pour la conception de puces multi-processeurs hétérogènes

Les systèmes sur puce massivement multi-processeurs hétérogènes contiennent des sous-systèmes de types variés. Ces composants sont : différents processeurs embarqués avec leurs logiciels et données en formats binaires, des circuits électroniques numériques spécialisés et des réseaux d'interconnexion.

La conception de HMPSoC demande donc l'utilisation de nombreux éléments complexes, tel que composants matériels et processeurs, ainsi que des techniques de différents domaines : intégration de systèmes matériels (SoC), logiciel embarqué et programmation parallèle. Cette augmentation importante de la complexité induit de nouveaux défis de conception.

Pour concevoir un flot pertinent, il est nécessaire d'anticiper les architectures de demain, ainsi que les futurs besoins de conception et les méthodes et outils associés [1] [2].

Pour des architectures de type HMPSoC avec un grand nombre de processeurs, les coûts d'un point de vue temporel et financier, de conception de l'architecture et de réalisation du masque, sont tous deux prohibitifs. La puce produite doit donc être rentabilisée par une production de masse. Elle doit cibler un marché important voire plusieurs marchés. La polyvalence est une caractéristique que doivent avoir ces composants futurs. Cette polyvalence peut être envisagée avec des processeurs exécutant des programmes facilement modifiables ou échangeables, ou avec des circuits électroniques reprogrammables. Mais les zones reprogrammables étant très chères en surface et en énergie, le choix va donc plutôt vers l'utilisation massive de processeurs. Néanmoins de nombreuses recherches sont en cours [3] [4] [5]. L'utilisation de zones reprogrammables est à priori compatible avec l'approche proposée, elle n'est cependant pas étudiée dans ces travaux et n'a pas été testée.

Cet aspect réutilisation comprend aussi la vente de composants n'utilisant qu'une partie des ressources. En effet, il peut être plus rentable de vendre un composant existant mais bridé et avec une application plus simple que de faire un composant spécifique supplémentaire, même plus petit (silicium moins cher à produire), mais qui entraînerait des coûts de conception et des délais supplémentaires. Cette utilisation est aussi valable dans le cas d'une erreur de fabrication (impureté) dans un sous-système mais qui n'impacte pas les autres. Ce composant peut être utilisé pour une application plus simple qui ne nécessite pas ce sous-système (ou plusieurs sous-systèmes). Il y a donc une réduction des pertes en fabrication qui entraîne une augmentation de productivité. L'utilisation d'une partie seulement des sous-systèmes peut aussi permettre une baisse de la consommation. La tendance va donc vers

des SoCs modulaires composés de multiples sous-systèmes.

1.1.2 Les systèmes multi-processeurs hétérogènes sur puce : difficultés de conception

Un HMPSoC est un SoC particulièrement complexe avec un grand nombre de composants hétérogènes et d'interconnexions. La présence des processeurs engendre des problématiques nouvelles.

Il faut calibrer le nombre de processeurs nécessaire, puis pour chaque processeur se pose un ensemble de choix. Le type du processeur doit être déterminé : générique, DSP, VLIW, multimédia, vectoriel ou multi-fils d'exécution. Les réseaux d'interconnexions et leurs topologies doivent être fixés. De même, le fonctionnement des interruptions, des changements de contextes peuvent être adaptés à l'application, par exemple le choix des priorités ou des primitives d'interruption. Les fréquences de fonctionnement doivent être calculées. La taille en nombre de portes logiques de chaque composant est aussi un facteur déterminant.

Sur cet ensemble de composants, des ajouts sont possibles pour améliorer l'architecture matérielle : des caches de données et/ou d'instructions permettent suivant leur taille de diminuer le nombre d'accès à la mémoire; des coprocesseurs qui peuvent amener un grand nombre de fonctionnalités ou d'optimisations matérielles au processeur; des composants pour améliorer ou paralléliser les communications avec l'exécution des programmes tels que des DMA, boites à messages ou mémoires partagées. Enfin des instructions spécialisées permettent d'accélérer l'exécution de certains programmes.

Finalement l'architecte choisit les interconnexions en prenant en compte les communications [6] liées aux processeurs, en particulier les effets des caches et les chargements d'instructions. Celles-ci sont multiples et entraînent la nécessité d'une bande passante globale importante et difficile à estimer. Un réseau [7] [8] de communication peut alors faire partie du système : c'est un réseau sur puce *"Network On Chip"*. Ces réseaux peuvent gérer des protocoles différents, zone d'adressage, grand ou petit boutisme (de l'anglais : *endianness, big or little endian*).

De plus des modifications locales de l'architecture matérielle peuvent avoir des impacts sur les communications et donc potentiellement sur tout le reste de la plateforme, par exemple en diminuant la bande passante sur le réseau global pour d'autres sous-systèmes.

1.1.3 Le cahier des charges initial d'une architecture logicielle et matérielle pour le concepteur

Le cahier des charges initial d'une architecture combinant logiciel et matériel pour le concepteur contient l'intégralité de ce que le système doit réaliser : l'ensemble des fonctionnalités, performances et normes à respecter. Ce document contient donc l'ensemble des spécifications sous forme de textes et de schémas.

Le cahier des charges conditionne la conception de l'architecture par assemblage ou création de composant, souvent autour d'une plateforme préexistante. Les contraintes de performances, de taille du composant et de réutilisabilité pour le logiciel impliquent l'utilisation de techniques de conception conjointe de matériels et de logiciels (de l'anglais : *codesign*). Les parties logicielles devront donc être explicitées.

Le cahier des charges va cadrer :

- la création ou réutilisation des composants et des réseaux de communication;
- les interconnexions entre les composants et réseaux;
- le développement des logiciels de test, des composants et réseaux;
- la conception et programmation des logiciels embarqués;
- la validation des tests au niveau des fonctionnalités et performances.

Il est souhaitable, afin de faciliter la mise au point, de valider séparément les performances et fonctions. Les performances dépendent des caractéristiques physiques, telles que le temps, la taille, la consommation d'énergie. La fonctionnalité découle du respect d'une norme, d'un algorithme.

1.1.3.1 Les performances requises par les clients pour le système sur puce

Les performances requises par les clients pour le système sur puce peuvent [9] se classer en trois catégories : surface, temps et énergie.

Le coût de surface du SoC est fonction de la technologie de conception du circuit, et notamment des caractéristiques de la librairie de portes utilisée. Ce coût limite les composants présents et leur taille, notamment pour les mémoires et les caches qui doivent être incorporés [10].

Les temps d'exécution des différentes fonctionnalités disponibles ainsi que les débits de données en entrée et en sortie qui sont particulièrement importants. La consommation énergétique globale du système en activité et au

repos est particulièrement importante pour les composants mobiles, de plus elle prend de l'importance au vu de l'augmentation des coûts de l'énergie.

Il est aujourd'hui prouvé [11] [12] que la résolution de ces contraintes, telles que temps d'exécution, taille mémoire/surface et énergie, se joue à la fois dans la conception de l'architecture et dans celle du logiciel. Les programmes peuvent être optimisés pour le temps d'exécution, la gestion d'un débit, la minimisation des coûts mémoires ou de l'énergie consommée.

1.1.3.2 Les fonctionnalités attendues par les clients pour le système sur puce

Les fonctionnalités attendues par les clients pour le système sur puce sont exprimées par un ensemble de spécifications habituellement textuelles. Ces fonctionnalités peuvent être des normes ou des spécifications techniques. Celles-ci peuvent s'appliquer à des algorithmes, traitements de données ou contrôles, ou à des protocoles de communication.

Le respect scrupuleux de ces consignes est gage de compatibilité avec d'autres systèmes.

Des jeux de tests peuvent aussi être fournis pour à la fois valider l'implémentation des spécifications et éclaircir des points particuliers.

Ces spécifications constituent le point d'entrée pour le travail de l'architecte.

1.1.4 Modélisation de systèmes sur puce multi-processeurs hétérogènes au niveau système

La modélisation de systèmes sur puce multi-processeurs hétérogènes au niveau système lors de la conception est utilisée dans deux buts. Dans un premier temps pour la conception matérielle, afin d'accélérer la conception et la simulation. Puis pour la réalisation et le test en avance de phase du logiciel.

La conception au niveau système est particulièrement utile pour l'exploration architecturale. Elle permet une conception et une simulation plus rapides (fournissant des estimations et mesures) et il est donc possible de réaliser plus d'itérations qu'au niveau des descriptions RTL [13]. L'utilisation de multiples processeurs dans un système induit, par leur présence, de nombreux éléments complexes : les processeurs eux-mêmes car interprétant des logiciels mais aussi les coprocesseurs, les périphériques de calcul ou de

communication et les réseaux pour les communications inter-processeur. Ces architectures matérielles multiprocesseur prennent d'autant plus de temps à simuler. Le gain de la modélisation et de la simulation au niveau système devient encore plus important.

La modélisation au niveau système peut se découper en sous-niveaux. Les niveaux définis à STMicroelectronics^{TM}[14] pour la simulation sont définis comme suit :

	Description	**Utilisation**
Fonctionnel	Un ensemble de tâches communiquent via des services.	Validation fonctionnelle.
PV	Des blocs matériels communiquent à travers un réseau abstrait. Les signaux des interfaces de ces composants sont *"bit true"* pour les adresses et les données.	Validation fonctionnelle avec le logiciel final.
PVT	PV avec une modélisation du temps aux interfaces.	Estimation des débits, latences et interruptions de la plateforme intégrée.
BCA	PVT avec précision au niveau cycle dans le réseau de communication.	Mesures des contraintes dans le réseau.
RTL	Entièrement au niveau cycle, synthétisable.	Vérification des fonctionnalités avant synthèse.

Fig. 1.1.1 – Différents niveaux de modélisation

Pour la conception au niveau système, les niveaux de modélisation utilisés sont PV, PVT et BCA. Ces trois niveaux sont considérés comme étant TLM.

Un autre niveau non défini à STMicroelectronics mais parfois utilisé, est le **BFM** qui correspond à une précision de l'ensemble au niveau cycle mais avec un processeur non instanciable (ISS). Il sert pour la validations de l'instanciation (Composants RTL, connexions).

La comparaison des temps de simulation entre un modèle RTL et un modèle TLM PV n'est pas simple. En effet, l'utilisation de différents simulateurs de processeurs, des différents modèles de simulation de mémoires en RTL, des prorata calcul versus communications des applications embarquées,

font varier grandement ces durées. Ces valeurs sont donc valables, soit pour des ordres de grandeur soit pour un cas d'utilisation spécifique.

Des mesures pour différentes plateformes en passant d'une simulation VHDL à une simulation TLM (avec des processeurs en simulations natives) ont donné :

- architecture avec un processeur et au moins un composant matériel complexe : accélération de 720 fois;
- architecture complexe avec six processeurs : accélération de l'ordre de 10000 fois.

Le nombre plus élevé de processeurs a donc un impact positif sur l'efficacité d'une simulation TLM. Ceci doit être pondéré par, d'une part une perte importante dans le cas d'utilisation d'ISS, et de l'autre l'utilisation d'émulateurs qui permettent des exécutions au niveau portes logiques beaucoup plus rapides.

Pour les logiciels embarqués, le modèle exécutable au niveau système permet un développement plus en avance, soit une réduction de temps de mise sur le marché. Mais le modèle permet aussi une bien meilleure observabilité de l'exécution. L'utilisateur peut visualiser toutes les transactions, et insérer ses tests simplement dans les composants matériels.

Les données exploitables pour l'estimation de performances dépendent de la précision des modèles. Elles peuvent aller de la simple validation des logiciels embarqués à des mesures de performances, de débits, de latences, de temps d'exécution et de consommations énergétiques.

La modélisation d'architecture contenant quelques processeurs (jusqu'à environ 10) est une extension des modèles mono-processeurs. Par contre pour un plus grand nombre de processeurs, les outils de modélisation et de mesure de performances ne sont plus adaptés.

1.1.5 Le flot de conception avec modélisation au niveau système proposé par STMicroelectronics

Le flot de conception de STMicroelectronics^{TM}est utilisé actuellement en division pour les composants complexes. Il n'est pas spécialisé pour les architectures avec un très grand nombre de processeurs. Il commence par la conception de l'architecture. Cette architecture est déduite du cahier des charges grâce à l'expérience des architectes et la reprise de parties connues d'architectures existantes (évolution d'une plateforme entière et réutilisation d'ensembles de composants matériels).

Quand la description de l'architecture matérielle est figée, les développements simultanés des nouveaux composants matériels puis de la variante des plateformes TLM et RTL commencent. En même temps les premiers travaux sur l'implémentation logicielle des algorithmes sont entrepris.

Dès que la plateforme exécutable TLM est achevée, les parties logicielles de bas niveau peuvent être programmées et les algorithmes complexes portés. Toutes les parties logicielles sont testées, validées puis optimisées sur cette plateforme TLM.

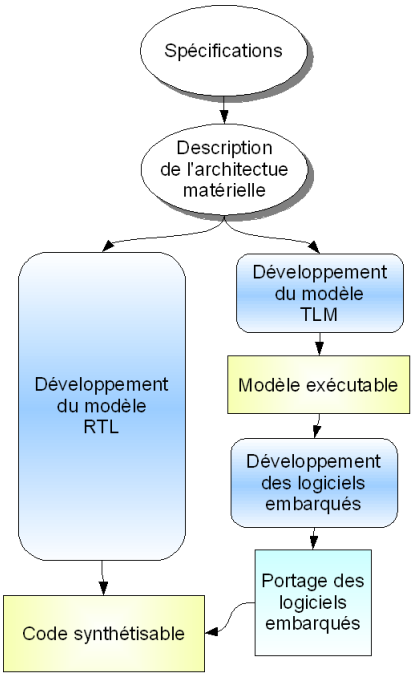

FIG. 1.1.2 – Le flot de conception avec modélisation au niveau système proposé par STMicroelectronics

Les composants RTL sont progressivement vérifiés en simulation grâce aux données de référence fournit par la plateforme TLM.

Quand le RTL est enfin prêt, il passe en validation sur les plateformes d'émulation avec les programmes préalablement testés. La plateforme TLM continue de servir comme référence et pour la reproduction et la différenciation entre erreurs fonctionnelles et celles plus spécifiques des étapes de synthèse.

Lorsque le silicium est disponible, la plateforme TLM sert pour la maintenance logicielle afin de reproduire des scénarios problématiques avec une plus grande visibilité, recherche d'erreurs simultané sur le logiciel et le matériel avec visualisation de tous les états internes.

1.1.6 Les explorations architecturales souhaitées par les concepteurs d'architectures pour l'étude de systèmes sur puce multiprocesseurs hétérogène

L'exploration architecturale est une part intégrante du flot de conception. La performance est difficile à prédire [15], les architectures devenant trop complexes pour être évaluées manuellement. De plus les méthodes d'évaluation analytique de performances se heurtent aux aspects dynamiques du système. Leurs impacts sont particulièrement difficiles à modéliser, notamment en ce qui concerne les antémémoires d'instructions (en anglais *"instruction caches"*).

Pour le modèle d'architecture générée, la flexibilité se situe à deux niveaux. D'abord à celui qui est interne aux sous-systèmes. Puis au niveau macro-architecture qui inclut l'ensemble des sous-systèmes présents et leurs interconnexions.

L'observation des SoCs en cours de développement montre une réutilisation importante de composants matériels. Aujourd'hui celle-ci est encore plus large et utilise même d'anciens systèmes complets, car ces architectures sont déjà testées et des logiciels embarqués existent pour gérer leurs fonctionnalités. Ces programmes incluent des pilotes de périphériques qui sont eux aussi réutilisables en tout ou partie pour ce système. De plus les performances sont connues ainsi que les paramètres physiques de consommation et de surface du circuit. Le dernier point et non le moindre : les programmeurs connaissent déjà ce circuit et sa prise en main est immédiate sans surcoût de formation.

Cette réutilisation demande cependant d'adapter ce système matériel aux évolutions des spécifications. La possibilité d'effectuer ces modifications constitue les flexibilités nécessaires aux flots de création de la plateforme matérielle. Elles comprennent les modifications de l'architecture interne. Celles-ci sont les ajouts, suppressions ou modifications des coprocesseurs, tailles des mémoires et des caches, interfaces vers l'extérieur, des composants de communications, DMA, boites à messages et mémoires partagées.

La polyvalence de plus en plus requise pour les nouveaux systèmes sur puce implique de pouvoir exécuter différents algorithmes ou plusieurs versions d'un algorithme, avec un pipeline plus important, des paramètres différents ou une nouvelle spécification.

L'impact de la parallélisation est donc variable, le nombre de sous-systèmes requis peut varier de manière importante. Les outils de création du modèle matériel doivent donc fournir des possibilités d'extension matérielle.

Le développeur de la plateforme doit disposer de ces possibilités d'exploration. Il doit fournir une description de l'architecture. Un modèle exécutable sera généré à partir de cette description et d'une bibliothèque de générateurs paramétrables. Il est alors possible de simuler et de faire un rebouclage rapide de la modification de l'architecture jusqu'à une autre simulation.

1.1.7 Un flot idéal pour la conception de système sur puce multi-processeurs hétérogène

Le flot idéal pour la conception de système sur puce multi-processeurs hétérogène, figure 1.1.3, permet à l'architecte une modélisation rapide de la plateforme matérielle et logicielle. Ce modèle doit pouvoir être simulé avec des mesures de performances. Suivant les résultats obtenus, l'architecture sera validée ou des modifications seront nécessaires.

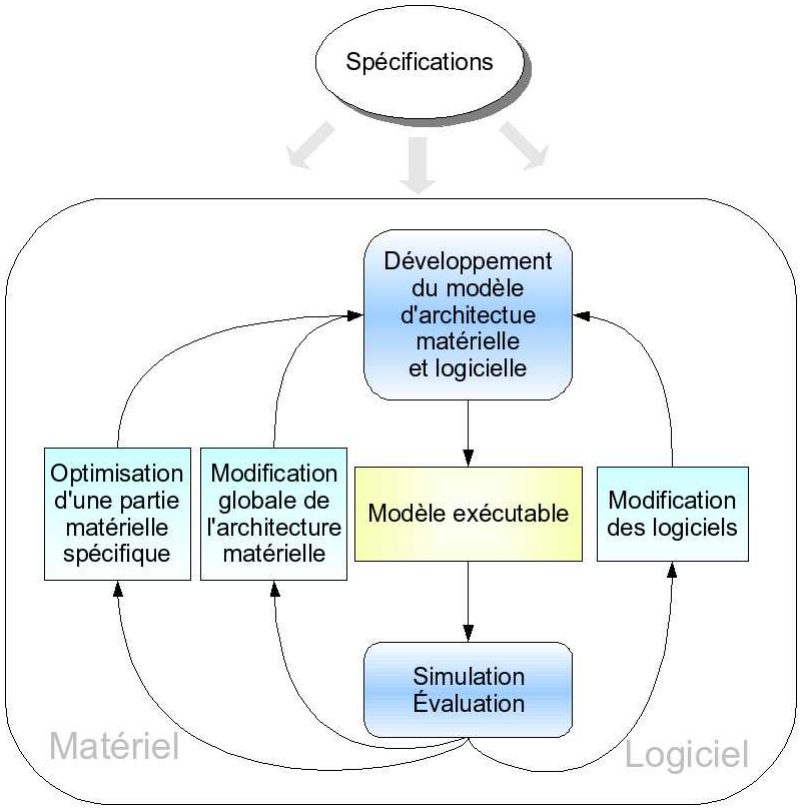

FIG. 1.1.3 – Un flot idéal pour la conception de HMPSOC

Les modifications peuvent concerner des changements sur l'ensemble de la plateforme : changement de la topologie, rajout ou retrait de processeurs. Sinon il peut s'agir de modifications locales : configuration d'un processeur, changement de taille de mémoire. Enfin il peut s'agir de modification du logiciel, tel que optimisations, placement [16][17] ou rajout de couches logicielles. Dans tous les cas l'architecte doit pouvoir réaliser son nouveau modèle rapidement afin de pouvoir relancer une simulation.

Dans la prochaine section il sera traité des outils existants pour évaluer ou modifier un modèle d'architecture.

1.2

État de l'art des techniques de simulation des logiciels dans les modèles exécutables de plateformes matérielles

Des outils spécifiques existent pour simuler les processeurs. Ces outils se confrontent aux deux problèmes opposés, vitesse et précision, de la simulation. La plupart favorisent une seule de ces approches, quelques autres, tel *Vast*$^{\text{TM}}$[18], visent à résoudre les deux contraintes.

1.2.1 Utilisation de simulateurs fonctionnant à partir du jeu d'instruction

Les simulateurs fonctionnant à partir du jeu d'instruction *"ISS : Instruction Set Simulator"* sont des exécutables. Ils décodent le flux binaire des instructions reçues par le processeur. Ils sont souvent prévus pour fonctionner seuls et peuvent charger seul le programme et gérer des mémoires internes. Dans notre cas, tous les accès externes aux processeurs deviennent des transactions dans la simulation. Ces accès comprennent tous les accès mémoire, les accès aux périphériques proches s'ils sont simulés comme composants indépendants, tels que coprocesseurs, caches et gestionnaires d'interruptions.

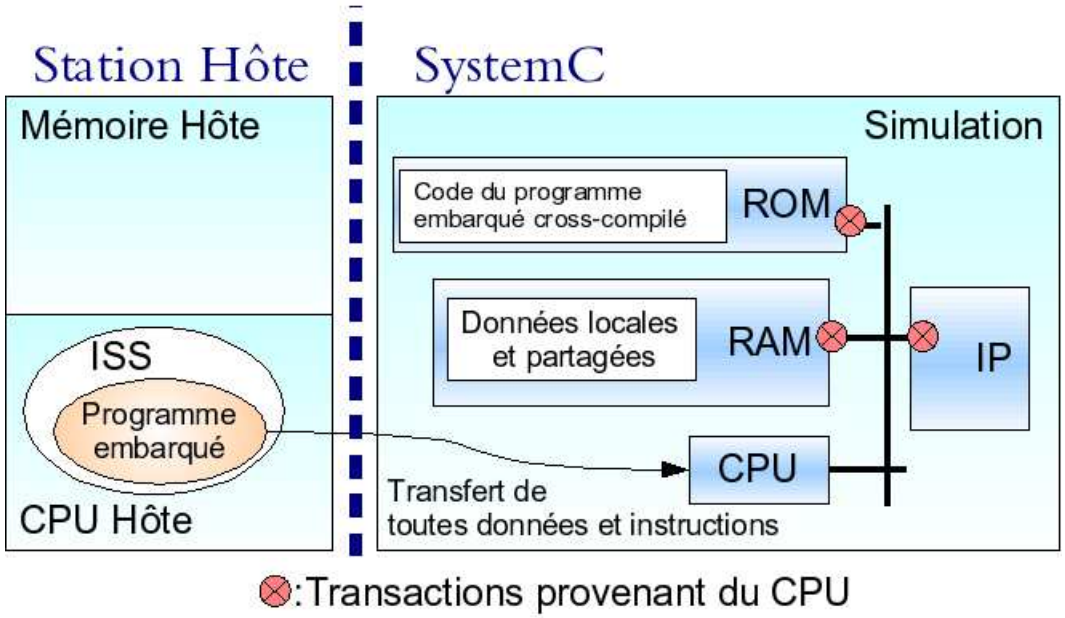

FIG. 1.2.1 – Cosimulation de processeur par ISS et SystemC

La cosimulation demande de co-exécuter le logiciel sur l'ISS en parallèle avec la simulation systemC du reste de la plateforme matérielle.

Dans la figure 1.2.1, un modèle de processeur *CPU* dans la simulation SystemC encapsule le simulateur de processeur *ISS*. Toutes les demandes d'accès mémoire, pour les données et les instructions sont d'abord des appels de fonctions de l'ISS. Ces appels de fonctions sont transformées en transactions dans la modèle TLM. Il est alors possible d'évaluer le trafic sur le modèle TLM du réseau.

1.2.1.1 Simulateur de jeu d'instruction précis au cycle d'horloge

Le premier et principal avantage de ces simulateurs d'instructions est leur précision au cycle d'horloge, *"ISS cycle accurate"*. Ils implémentent tout le comportement interne du processeur cycle après cycle, calculs, registres et états des différents pipelines [19].

En revanche, le revers de cette précision est un temps de simulation particulièrement long. Cependant ces composants sont dans notre cas interconnectés par une simulation *TLM PV*. Ce type de simulation ne permet pas une précision au niveau du cycle. La précision de l'ensemble de la simulation ne peut donc pas être au cycle près. L'utilisation de tels ISS entraîne donc

une perte de temps conséquente sans apporter une grande précision.

L'autre avantage de ce type d'ISS est l'assurance que le comportement du processeur réel est simulé au mieux. Il est important avant d'utiliser des simulateurs moins précis de vérifier que des simplifications ne sont pas faites pour l'utilisateur, par exemple des initialisations de périphériques internes par défaut. En effet le logiciel ne serait alors pas portable sur une instance en silicium du processeur, ou synthétisé, ou synthétisable.

1.2.1.2 Simulateur de jeu d'instruction précis au niveau de l'instruction

Les ISS précis au niveau de l'instruction se basent aussi sur le jeu d'instructions du processeur qui sont lues dans une mémoire, décodées puis exécutées, *"ISS Instruction accurate"*. Ce processeur ne simule pas tous les registres internes du processeur. Il est donc moins précis que le précédent mais permet des simulations plus rapide.

Cet ISS est plus utile pour les simulations TLM et moins vers les simulations RTL. En effet dans une simulation TLM avec gestion du temps, les synchronisations se font lors des transactions entre objets. Pour les processeurs, ces transactions sont liées aux instructions décodées. La précision est donc suffisante.

Ce type d'ISS connaît une croissance importante qui est similaire à celle des simulations au niveau transactionnel. De nombreux fournisseurs d'outils en proposent [20] [21].

1.2.1.3 Simulateur de jeu d'instruction compilé statiquement

Le code source est compilé pour former un simulateur spécifique à ce code d'entré, soit sous forme d'autres sources, soit sous forme d'exécutables [22], *"Compiled ISS"*. Cette sortie simule le logiciel embarqué, elle est préformatée pour faciliter la simulation. L'effet de chaque instruction sur les variables internes du simulateur est déjà écrit. Il n'est plus nécessaire de décoder l'instruction. La compilation est donc plus longue et la simulation plus rapide.

Avec cette méthode il est aisé d'incorporer des modules lors de la création du simulateur. Ces modules permettent des mesures de performances totalement paramétrables sans influer sur les résultats de la mesure.

Par contre ces simulateurs ne fonctionnent pas s'ils doivent gérer du code auto-modifiant, ce qui est le cas pour un chargement d'un système d'exploitation ou l'utilisation d'un logiciel de recherche d'erreurs. En effet, seul le

code connu peut être précompilé avant la simulation.

Un tel ISS est souvent utilisé par les équipes qui développent des compilateurs. Il leur permet de valider aisément les compilations par visualisation du code en langage C équivalant.

1.2.1.4 Simulation rapide avec traduction dynamique

La technique de simulation rapide avec traduction dynamique est une agrégation des deux précédentes, ISS précis à l'instruction et ISS compilé, *"ISS with dynamic binary translation"*. Le simulateur de processeur découpe le binaire d'instructions en zones. Quand une instruction de la zone doit être exécutée, le simulateur traduit l'ensemble des instructions de la zone en algorithmes qui simulent cette partie. Une référence entre la zone et l'algorithme est mémorisée. A chaque fois qu'une instruction de cette zone est appelée le simulateur vérifie qu'elle n'a pas été modifiée. Dans ce cas l'algorithme correspondant est appelé, sinon la zone est redécodée.

De nombreux simulateurs de processeurs [23] [24] [25] utilisent cette technique. En tant que simulation précise à l'instruction, elle entraîne des pertes importantes de précision lors de cosimulation avec des modèles décrit au niveau RTL.

1.2.2 Simulation native des processeurs

La simulation native permet de simuler le comportement d'un processeur. Elle utilise à la fois les codes sources des programmes embarqués qu'il doit exécuter, et certaines spécifications fonctionnelles du processeur, tels que les ports, les interruptions et leur gestion, contenues dans un composant spécifique.

Elle permet une simulation très rapide. Les inconvénients sont l'absence de précision des mesures de performance et l'obligation d'avoir l'ensemble des codes sources en langage de haut niveau (au moins C). Des recherches ont lieu pour donner de la précision aux simulations natives [26][27].

1.2.2.1 Concept de simulation native

Le but d'un grand nombre de simulations est de tester des composants. Si ces composants sont configurés par des logiciels embarqués, ceux-ci doivent être simulés. Une grande partie de la simulation calcule l'exécution du modèle du processeur alors que seules ses communications vers l'extérieur sont importantes. Dans ce but le code embarqué est compilé pour le processeur

de l'ordinateur simulant (en anglais : *HOST*), les opérations d'écritures et de lectures vers l'extérieur sont redirigées vers la simulation. Il n'y a pas de décodage d'instruction ni d'ISS à exécuter. La simulation est plus rapide.

Avec un style de codage défini, l'usage de fonctions pour les accès extérieurs ainsi que la gestion par le simulateur natif des fonctionnalités assembleur utilisées (autorisation et interdiction des interruptions), il est possible d'utiliser le même code source pour une simulation native et une simulation avec un ISS.

La simulation native peut aussi permettre, de tester fonctionnellement des algorithmes ou une première validation des codes sources.

1.2.2.2 Simulation native avec redirection totale des entrées/sorties

La simulation native avec redirection totale des entrées/sorties demande la compilation des logiciels embarqués pour le processeur simulant. Cependant l'édition des liens utilise les spécifications mémoires de l'architecture simulée. Toutes les adresses mémoires utilisées dans les programmes sont celles de l'architecture.

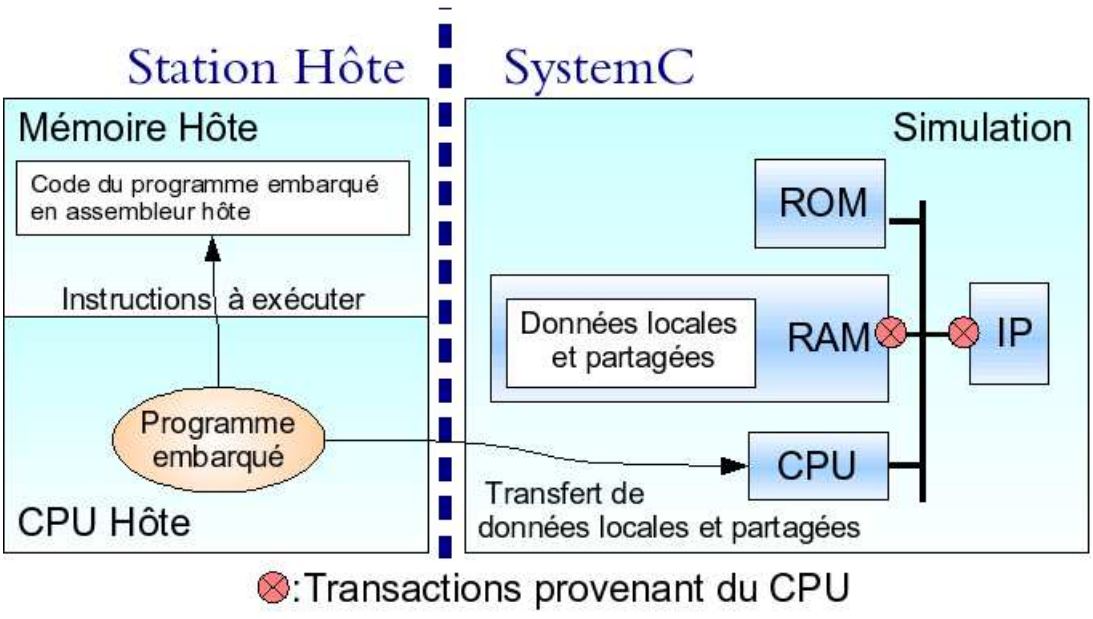

Fig. 1.2.2 – Simulation native avec redirection totale

Lors de la simulation, figure 1.2.2, le code du logiciel embarqué est exécuté

sur le processeur hôte. Le code du programme est donc contenu dans la mémoire de la machine hôte en temps que code exécutable. Dès qu'une lecture ou une écriture mémoire (de données) survient l'accès est transformé en une transaction sur le modèle exécutable d'architecture simulée. Cette méthode permet l'usage sans modifications de programmes qui n'incluent pas de code assembleur.

La cosimulation entre des simulations natives de processeurs de ce type et des simulations sous forme d'ISS est automatique. Cette simulation permet l'évaluation des transits de données sur les réseaux[28] ainsi que la vérification des besoins en tailles de mémoires. Attention, seuls les transferts de données sont visibles, les transferts d'instructions ne le sont pas.

1.2.2.3 Simulation native avec redirection des communications matérielles ou logicielles

Avec la technique de simulation native avec redirection des communications matérielles ou logicielles seules les données nécessaires aux synchronisations et aux communications transitent par la simulation. La mise en œuvre est très simple et rapide. Elle consiste en l'utilisation de fonctions de redirection pour la lecture et l'écriture. Cependant, elle demande une réflexion sur les structures de données partagées, les différentes communications et les points de synchronisation.

Le nombre de transactions plus faible rend la simulation plus rapide et la recherche d'erreurs plus simple.

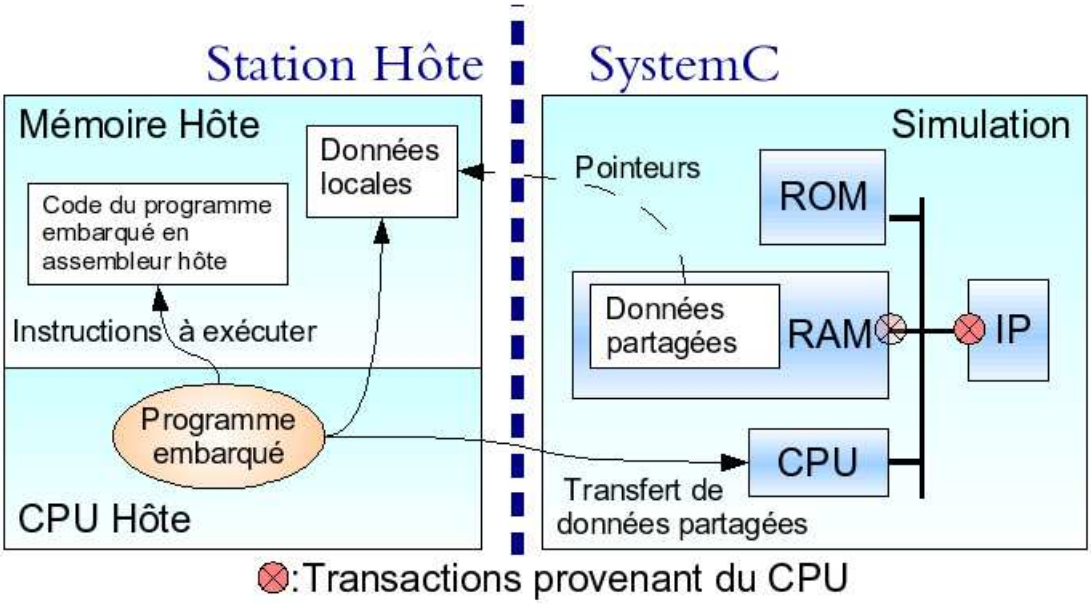

FIG. 1.2.3 – Simulation native avec redirection des communications uniquement

Lors de la simulation, figure 1.2.3, le code du logiciel embarqué est exécuté sur le processeur hôte. Le code du programme est donc contenu dans la mémoire de la machine hôte en temps que exécutable. Dès qu'une lecture ou une écriture mémoire de données survient, celle-ci est effectuée dans la mémoire du processeur hôte. Ce transfert ne transite donc pas par la simulation et ne modifie aucune mémoire ou composant simulé. Les transferts de données partagées ou les accès aux composants matériels doivent être explicités via des fonctions de lecture et d'écritures. Ces fonctions sont transformés par le modèle du processeur en transactions TLM.

Un point particulier spécifique à la communication entre deux processeurs est l'allocation dynamique de mémoire et le partage des données. Si un programme embarqué utilise une primitive d'allocation standard, par exemple de la *libc*, l'espace mémoire sera allouée dans la mémoire de la machine hôte comme une donnée locale. Cependant l'adresse mémoire correspondant à cette zone peut être écrite dans une mémoire TLM simulée, puis lue par un autre programme embarqué sur un autre processeur simulé. Ce programme aura alors accès aux données en utilisant directement le pointeur.

1.2.3 Comparaison des différentes techniques de simulation des processeurs et logiciels embarqués

Les différentes techniques de simulations des processeurs et des logiciels embarqués, figure 1.2.4, s'opposent donc sur trois domaines : vitesse de simulation, précision et fonctionnalités. La fonctionnalité principale requise est le support du code automodifiant (un code capable de se modifier au cours de son exécution). Il est nécessaire pour les outils de recherche d'erreurs, le lancement de système d'exploitation et le chargement de programmes.

Simulation	Vitesse[1]	Précision/Transferts	Code automodifiant
ISS cycles	x1	cycles d'horloge / données&instructions	oui
ISS instructions	x10	instructions / données&instructions	oui
ISS Compilés	x200	instructions / données&instructions	non
Traduction dynamique	x100	instructions / données&instructions	oui
Simulation native redirection totale	x1000	Pas de temps / données	non
Simulation native avec communications	x3000	Pas de mesures	non

FIG. 1.2.4 – Récapitulatif des différentes techniques de simulation de processeurs

La vitesse de la simulation d'un programme sur une architecture ne dépend pas uniquement de la technique utilisée pour le simuler. La modélisation de l'ensemble de l'architecture, le comportement du logiciel, la quantité de communication par rapport aux calculs, et les interruptions la font varier de manière importante. Il est donc plus juste de comparer les ordres de grandeurs entre les différentes simulations. D'autres classements ont été effectués[29].

[1]N'ayant pas pu mesurer avec un programme identique sur toutes les techniques de simulation, j'ai défini les facteurs de vitesses par expérience

1.2.4 Le flot de conception avec exploration d'architecture et simulation multi-niveau

L'exploration d'architectures consiste tout d'abord a créer une nouvelle architecture ainsi que ces logiciels. A partir de cette plateforme logicielle/matérielle sera faite une simulation avec des mesures de performances. L'évaluation des résultats et des tenants de cette simulation permet ensuite de modifier ou de concevoir une nouvelle architecture.

Cette exploration demande donc une exécution conjointe réussie du logiciel et du matériel. Pour arriver à un tel résultat, une étape de recherche d'erreurs est obligatoire. Cette recherche d'erreurs demande une simulation très rapide mais qui n'a pas besoin de mesures de performances.

Quand les erreurs sont corrigées, la simulation avec mesures de performances est lancée. Le cycle peut continuer.

1.3

Simulations multi-niveaux des logiciels et leur mise en place

La simulation multi-niveaux des logiciels implique l'utilisation de simulation de processeurs de différents niveaux d'abstraction avec des simulateurs de différent types. Chacun de ces simulateurs de processeurs défini une interface et des règles de codages différentes. Des adaptations sur les logiciels peuvent être nécessaires pour communiquer avec toutes ces interfaces.

Dans un cas idéal, avec toutes les sources disponibles en langage C ou C++, pas de code assembleur et deux niveaux d'abstraction, simulation native avec redirections totale et un ISS, aucune adaptation n'est à effectuer.

Dans le cas général, les instructions assembleurs doivent être séparées et simulées par le modèle. Les communications ainsi que les points de synchronisation doivent être spécifiés.

1.3.1 Buts de la simulation multi-niveau ou compromis vitesse/précision

La simulation multi-niveaux a trois buts Le premier est de rendre les simulations des logiciels extrêmement rapide pour la recherche d'erreurs sans notion de temps. Le second est la mesure de performances. Le troisième est de n'utiliser qu'un seul code source avec le moins de modifications possibles entre les deux simulations.

L'idée est donc d'allier à la fois la vitesse pour la recherche d'erreurs et la précision dans la même plateforme logicielle et matérielle exécutable mais dans deux simulations successives.

1.3.2 Implémentation nécessaire à cette simulation multiprocesseur multi-niveaux

La simulation multiprocesseur multi-niveaux nécessite des possibilités particulières.

Pour la simulation multi-niveaux, il faut des interfaces vers les primitives de bas niveau, les communications et les synchronisations. Ces interfaces contiennent le code qui change selon le niveau de simulation. Pour la partie bas niveau il faut donc définir une règle de codage avec l'utilisation de fonctions d'accès. Souvent une bibliothèque de fonctions d'accès est fournie avec la chaîne d'outils de compilation. Il suffit dans ce cas de la redéfinir pour la simulation native.

Les principales adaptations sont sur les logiciels. Le modèle doit simplement permettre d'instancier au choix des simulations de processeurs différentes soit rapides soit précises, respectivement natives ou ISS.

Le dernier point consiste à mettre en œuvre un mécanisme de double compilations, croisé et natif.

1.3.2.1 ISS et simulateur natif multi-instanciable

L'utilisation de modèle logiciel matériel exécutable de HMPSoC implique la modélisation simultanée de plusieurs processeurs. Ces processeurs sont modélisés par les ISS ou de la simulation native, ils doivent donc être multi-instanciables.

Pour cela ils doivent être esclaves de la simulation, il faut les concevoir comme une bibliothèque et non comme le programme principal. De plus, pour protéger leurs données, il ne faut pas qu'ils aient de variables globales.

Dans le cas où ces deux points ne sont pas respectés, des techniques logicielles de contournement peuvent être mises en place : communications inter-processus, isolation dans des librairies. Cependant, celles-ci diminuent grandement la vitesse de la simulation et augmentent de manière importante la complexité de la modélisation, ce qui les rend difficilement utilisables.

1.3.2.2 Interfaces d'appel assembleur, de communication et de synchronisation

Les interfaces d'appel assembleur, figure 1.3.1, de communication et de synchronisation contiennent le code source qui est différent entre les simulations natives et les simulations avec ISS. Elles sont à la fois dépendantes du type de processeur, et du simulateur natif. Le simulateur natif étant lui

même dépendant du processeur, il doit implémenter le code de ces interfaces lors des simulations natives. Pour pouvoir effectuer des simulations multi-niveaux, les codes sources des logiciels embarqués doivent impérativement utiliser ces interfaces.

Pour la définition des interfaces assembleurs :

- Une bibliothèque de fonctions pour la programmation du processeur et de ses périphériques proches est disponible. Dans ce cas l'interface doit suivre les spécifications de cette bibliothèque.
- Les logiciels doivent fonctionner sur un système d'exploitation. Ce sont les appels systèmes qui doivent être définis.
- Aucune standardisation existe, il faut définir une interface qui prend en compte les fonctions du processeurs et des périphériques proches.

```
static inline void invalidate_cache(unsigned long int addr);
static inline void enable_IRQ(void);
static inline void disable_IRQ(void);
static inline void enable_FIQ(void);
static inline void disable_FIQ(void);
static inline void sync_interrupt(void);
```

Fig. 1.3.1 – Exemple d'interface de bas niveau

Les primitives de communication, figure 1.3.2, servent pour la redirection des lectures et écritures de communications et synchronisations. Ces accès sont soit entre logiciels, soit entre logiciel et matériel.

Ces primitives de communications et de synchronisations doivent obligatoirement être explicitées lors de simulations natives avec redirection des communications matérielles ou logicielles. Dans le cas de simulations natives avec redirection totale, elles sont facultatives mais permettent une meilleure structuration du code, et des vitesses de simulation plus importantes.

```
static inline void write_mem(unsigned long int address, unsigned
long int data);
static inline unsigned long int read_mem(unsigned long int address);
static inline void write_mem_block(unsigned long int address,
unsigned long int * src, int number);
static inline void read_mem_block(unsigned long int address, unsigned
long int * dest, int number);
static inline void memcopy(unsigned long int addr, unsigned long
int dest, unsigned long int number, unsigned int size);
```

FIG. 1.3.2 – Exemple d'interface de communication

L'interface de synchronisation, figure 1.3.3, sert dans les simulations natives. Elle permet d'expliciter les points de synchronisation.

```
static inline void sync(unsigned int multiplicator);
```

FIG. 1.3.3 – Exemple d'interface de synchronisation

1.3.2.3 Les points de synchronisation dans les logiciels

La notion de point de synchronisation en simulation native est proche du concept de "céder la main" (*Yield*) en programmation parallèle coopérative.

Le but de ces points de synchronisation est d'expliciter que le programme attend un évènement extérieur. Cet évènement peut être logiciel, par exemple attendre qu'un autre logiciel arrive à un point particulier de son exécution repéré par le changement de la valeur d'une variable globale. Il peut aussi être matériel : attendre une interruption ou un test sur un registre.

Donc, certains comportements demandent un point de synchronisation :

- attente d'interruption;
- boucle infinie, attente de réinitialisation;
- attente active (*Polling*).

Le cas de simulation dans un environnement coopératif (SystemC) force à l'utilisation de ces points de synchronisation. Dans un environnement préemptif (pthread), leur utilisation permet de diminuer les attentes inutiles, ce qui améliore les performances, et d'expliciter les points de synchronisation, ce qui aide à la vérification.

1.3.2.4 Mécanisme de double compilation

Le mécanisme de double compilation, figure 1.3.4, permet de compiler deux fois les mêmes sources, pour obtenir deux binaires différents. Un pour la simulation avec ISS et l'autre pour la simulation native, respectivement compilation croisée et compilation vers la machine de simulation.

Les raisons de cette compilation multiple sont doubles :

- disposer des deux binaires pour pouvoir exécuter la simulation, soit pour la recherche d'erreurs, soit pour les mesures de performances;
- valider lors de l'écriture que le code est compatible avec les deux compilations, interprétation du langage, fonctions disponibles.

Les mécanismes de compilation sont séparés mais les codes sources de l'application restent communs.

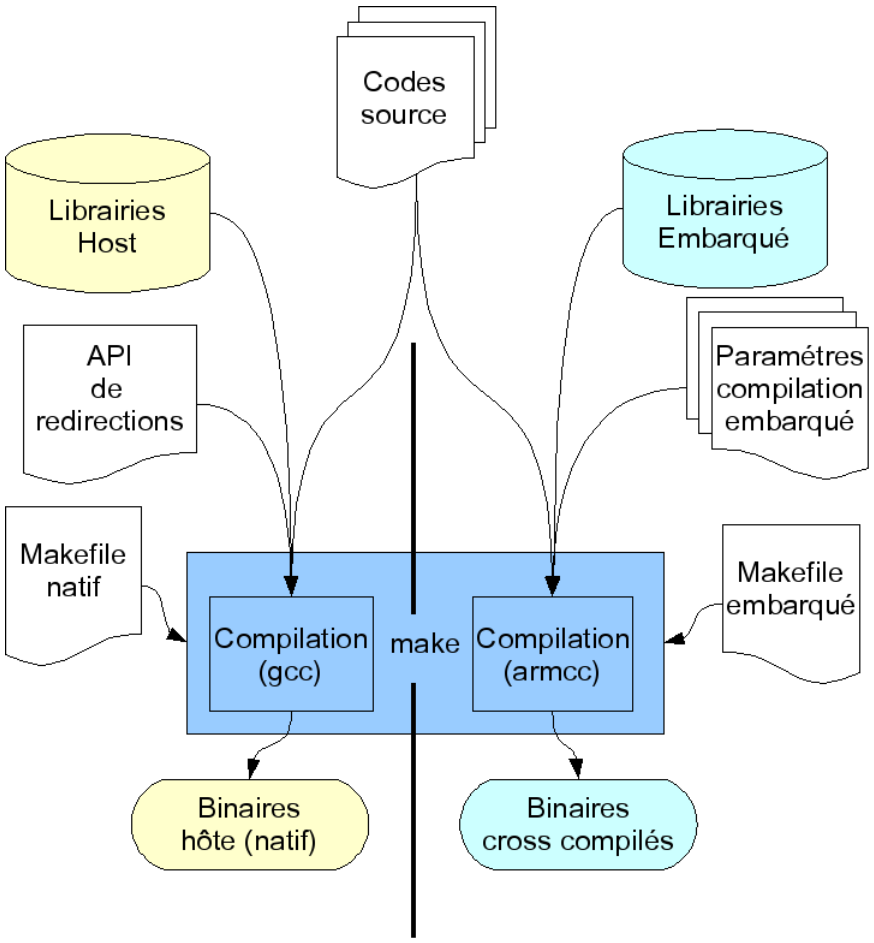

Fig. 1.3.4 – Mécanisme de double compilations natif et ISS

Les spécificités de chaque compilation sont :
- la chaîne d'outils à utiliser, compilateurs, édition des liens;
- les paramètres des différents outils malgré quelques standards de fait (option -g);
- les fichiers de configuration, fichier d'édition des liens pour les ISS, ou de redirection en natif;
- les librairies à utiliser, machine simulant ou chaîne d'outils;
- le résultat de la compilation : une bibliothèque ou un binaire exécutable pour la simulation native, un fichier de données binaires chargeable dans une mémoire pour l'ISS.

Cette vérification à la compilation est importante mais ne permet malheureusement pas de vérifier que les fonctionnalités des deux binaires sont équivalentes. En effet les spécifications des langages de programmation laissent souvent des points dépendants de l'implémentation. Des règles de codage plus strictes sont conseillées pour résoudre ces problèmes. D'autant qu'ils sont particulièrement complexes à résoudre.

1.3.3 Contraintes de modélisation pour une simulation multi-niveaux efficace

Le but premier est de pouvoir exécuter des simulations multi-niveaux efficaces à partir du même code source des logiciels embarqués. Par efficace, s'entend d'effectuer des exécutions avec simulation natives des processeurs très rapides, mais aussi de faire apparaître le plus d'erreurs possible avant de commencer les exécutions de modèle incluant logiciel et matériel avec les mesures de performance.

Les contraintes de modélisation pour une simulation multi-niveaux efficace peuvent se classer en cinq catégories : les points de synchronisation, les accès par blocs, les structures de données partagées, l'initialisation des périphériques matériels et des règles de codage spécifiques.

1.3.3.1 Les points de synchronisation entre tâches logicielles/matérielles

Les points de synchronisation entre tâches servent à rendre la main explicitement à l'environnement de simulation. Celui-ci va évaluer les tâches à exécuter, puis choisir celle qui a le niveau de priorité le plus élevé.

Ce mécanisme induit deux changements de contexte, tâches vers environnement de simulation puis vice versa, et un calcul d'ordonnancement. Il est

donc d'un coût non négligeable en temps de calcul et ne doit être utilisé que si nécessaire, il doit être appelé lors des synchronisation avec le matériel ou d'autres logiciels.

Pour la synchronisation matérielle, on déclare un point de synchronisation par exemple, changement de valeur d'un registre ou lors de l'attente d'interruptions. On peut remarquer que le mécanisme d'un processeur qui lui permet de se mettre en veille jusqu'à la prochaine interruption permet une implémentation très efficace. Ces attentes ont souvent lieu soit dans le système d'exploitation soit dans des primitives logicielles de synchronisation, telles que signaux logiciels, mécanismes de barrières ou d'exclusions mutuelles. Pour les synchronisations logicielles, le mécanisme est celui des attentes actives sur des données partagées. En vue d'améliorer les performances ces attentes sont souvent couplées à des attentes d'interruptions ce qui ramène au cas précédent.

1.3.3.2 Les transactions par bloc de données

L'utilisation de ces transactions par blocs est très importante pour la vitesse de simulation. En effet, il est possible de transférer en une seule fois des données qui auraient demandées de multiples transferts sur le composant réel.

Ces transactions peuvent se classer en deux sous catégories :
- communication de blocs de données entre une tâche logicielle et un composant matériel;
- communication de blocs de données entre deux tâches logicielles.

La communication de données entre un processeur en simulation native et un processeur simulé par ISS est vue comme une communication entre la simulation native et le composant matériel de la mémoire partagée, ou du composant de communication.

Pour la communication entre processeurs en simulation native et du matériel, deux primitives de communications par bloc sont à utiliser, lecture et écriture. L'ensemble des données est transféré en une seule fois par le réseau. Le destinataire gère le bloc comme une suite d'accès. Une attention particulière doit être portée aux écritures de blocs de données qui ont des tailles inférieures aux mots du protocole de communication. Les octets activés doivent être précisés.

La communication entre processeurs en simulation native est encore plus simple. Il suffit d'échanger les données dans la mémoire du processeur simulant. L'adresse vers cette zone mémoire est transmise par une mémoire partagée simulée. Lors du passage en simulation ISS l'adresse transmise doit pointer vers une zone d'une mémoire partagée simulée. Si les deux processeurs

ne voient pas la mémoire à la même adresse un mécanisme supplémentaire de modification doit être ajouté.

L'utilisation de tel transfert par pointeur rend impossible toute analyse du trafic. Cette limitation est cependant peu gênante car l'évaluation du trafic est déjà faussée par la simulation native.

1.3.3.3 Gestion des structures de données destinées aux mémoires partagées

Le problème de la gestion des structures de données destinées aux mémoires partagées généralise celui des communications directes, utilisant seulement une mémoire, entre programmes exécutés sur des simulations natives de processeurs.

Le style de codage le plus efficace consiste à créer une structure de données. Cette structure contient l'ensemble des données à partager entre les deux tâches sous forme de pointeurs. Dans le cas de simulation native avec redirection des communications tous les accès à cette structure doivent être déclarées via des primitives de l'interface de communication.

Ces modifications concernent des grandes parties du code. De plus elles peuvent être complexes dans le cas de structures de données accèdées par pointeurs. La solution consiste à communiquer le pointeur vers cette structure dans la mémoire partagée. La structure elle même et ses données sont dans la mémoire hôte. Les mécanismes de synchronisation des accès à ses données sont gérés via la plateforme. Les modifications du code sont moins importantes et la simulation est plus rapide.

Enfin les primitives d'allocation mémoire, *new/delete* ou *malloc/free* ne concernent généralement que la zone de mémoire de données, souvent locale, déclarée dans le script d'édition des liens. Les primitives d'allocation dans les mémoires partagées doivent être écrites pour les simulations natives avec redirection totale ou lors de l'utilisation d'ISS.

1.3.3.4 Problème de l'initialisation des périphériques et des mémoires

Le problème de l'initialisation des périphériques et des mémoires est général aux plateformes multi-processeurs. Il concerne aussi les simulations natives de ces plateformes. Un mécanisme de synchronisation explicite et des règles concernant les initialisations permettent d'éviter ces erreurs dont l'origine est complexe à déterminer.

Premièrement une tâche doit être choisie pour réaliser l'initialisation de

chaque composant qui le nécessite. Ceci est particulièrement important pour les périphériques partagés, notamment les mémoires. Si le temps de lancement n'est pas un facteur limitant, une seule tâche peut initialiser tous les composants, ce qui limite encore les problèmes d'interactions lors de l'initialisation.

De plus, la fin des initialisations est un point de synchronisation entre les tâches. Un mécanisme d'attente de style *barrière* est conseillé.

1.3.3.5 Règles supplémentaires de codage pour limiter les erreurs de changement de niveau

Des erreurs peuvent apparaître entre la simulation native et la simulation avec ISS. Certaines dépendent directement du niveau d'abstraction et ne peuvent pas être résolues. D'autres dépendent des processeurs et des compilateurs, elles demandent de figer certaines règles. De manière plus générale le langage de programmation est sensé protéger le programmeur de ces variations. Or, pour des raisons d'efficacité des choix ne sont pas spécifiés.

Quelques exemples :

- le caractère signé ou non de certain type, en langage c le *char*, résolu en spécifiant *signed char* ou *unsigned char*;
- l'alignement des adresses, souvent configurable dans le compilateur;
- la taille des pointeurs et des types, en langage c le *int* sur 2 ou 4 octets;
- l'ordre d'évaluation des arguments des appels de fonction;
- l'ordre de rangement des variables dans les structures de données (notamment les *bit field* en langage C);
- l'impact du boutisme du processeur sur les données accédées selon plusieurs types;
- le nombre d'octet utilisé pour stocker un type, *int* sur deux, quatre ou huit octets.

Toutes les erreurs engendrées par ces choix, non spécifiées peuvent être résolues par des règles de codages strictes et des options de compilation adéquates.

1.4

Modifications locales aux processeurs - Utilisation de processeurs configurables et personnalisables : État de l'art

Les modifications locales aux processeurs concernent leurs caractéristiques internes ainsi que celles de leurs périphériques proches, tels que coprocesseurs, mémoires, contrôleurs d'interruption, composants matériels spécifiques.

Deux approches permettent de spécialiser un processeur [30].

- L'utilisation de processeurs paramétrables, certaines caractéristiques sont variables et peuvent être spécifiées par l'utilisateur. Ces paramètres concernent : la taille des bus, le nombre de registres, les coprocesseurs, les caches, des instructions optionnelles. Les outils associés à ce processeur sont configurables à partir de ces paramètres.
- Partir d'un langage de description de processeurs, l'ensemble du processeur est décrit grâce à cette sémantique. Ensuite une chaîne d'outils analyse la description pour créer le compilateur, le modèle RTL synthétisable, les différents simulateurs d'instructions qui y sont associés.

1.4.1 Processeurs paramétrables

L'utilisation de processeurs paramétrables est l'approche la plus simple, mais aussi plus limitant puisque la liste des paramètres est finie et figée. Un exemple de paramètres est décrit dans la figure 1.4.1. Il existe déjà plusieurs processeurs de ce type. Pour une liste non exhaustive, NIOS d'ALTERA [31], ARC600 et ARC700 [32], Xtensa [33], Jazz DSP [34].

		ARC 600	**ARC 700**	**Xtensa**	**Jazz DSP**
Bus de données		16/32	32	32	16/32
Registre		35 à 63	35 à 63	4 à 32	25 à 58
Cache de données		1,2,4 de 0/32ko	8/64ko	1,2,4 de 0/32ko	
Caches d'instructions		1,2,4 de 0/32ko	8/64ko	1,2,4 de 0/32ko	
Étages du pipeline		5	7	5	2
Type de branchement		statique	dynamique		
Nombre d'interruptions		16 à 32		32	
Instructions	type	RISC	RISC	RISC	VLIW/DSP
	nombre		86		76
	taille	32	16/32	16/24	16/32
Gestion d'énergie		Sleep mode	Sleep mode		
Instructions personnalisées		128 en 32bits 128 en 16bits	128 en 32bits 128 en 16bits	ISA	ISA
Taille en portes Configuration basse		27000	100000	25000	50000
Horloge	0.13	290MHz	400MHz	350MHz	
	0.18	200MHz	266MHz	200MHz	
Énergie	0.13@1V	$0.04mW/MHz$	$0.15mW/MHz$	$0.1mW/MHz$	$0.1mW/MHz$
	0.18@1.8V	$0.13mW/MHz$	$0.5mW/MHz$	$0.4mW/MHz$	$0.6mW/MHz$

FIG. 1.4.1 – Caractéristiques modifiables (voir intervalle) pour différents processeurs paramétrables

1.4.2 Langages de description de processeurs

L'utilisation de langages de description de processeurs est une approche beaucoup plus flexible, car tous les aspects du processeur sont créés à la demande. Le compilateur [35], les ISS et le modèle RTL synthétisable [36] sont générés automatiquement. Par contre certaines extensions, par exemple les instructions complexes, peuvent ne pas être gérées par le compilateur. La charge de travail est cependant beaucoup plus importante que pour l'approche avec des processeurs paramétrables. Il faut décrire tout le processeur avec un langage spécifique , dit *Architecture Description Language* [37], comme LISA [38], ISDL [39], EXPRESSION [40], MIMOLA [41] ou nML [42]. Une description est présente dans la figure 1.4.2. Enfin il faut valider ce code et rechercher les erreurs qu'il contient.

```
STORAGE
   {
   REGISTERS  unsigned short  REG([0..7])6;
   MEMORY_RW  signed int long RWMEM([0..15]);
   }

OPERATION CONCAT
   {
   BEHAVIOR
   USES (IN REG[];
   OUT RWMEM[];)
      {
      /* C-code */
      RWMEM[address] = REG[reg1]<<8 & REG[reg2]>>8;
      }
   }
```

FIG. 1.4.2 – Partie d'un source LISA

Si la description de nombreux processeurs est disponible sous forme de modèles existants (sources *LISA*). Alors, les coûts des développements sont moindres, et la spécialisation puis le lancement de la chaîne d'outils sont facilités.

1.4.3 Position du flot proposé vis à vis des processeurs configurables

Les processeurs configurables permettent l'optimisation d'une partie spécifique de l'architecture. De plus ils peuvent aider à la gestion des logiciels embarqués si des outils sont fournis pour adapter la chaîne de compilation aux modifications du processeur. Cette solution permet donc de répondre efficacement à une partie de la problématique, soit l'optimisation d'une partie spécifique dans le cas d'un processeur. Il faut étendre cette solution dans le cas d'autres composants ou de modifications globales.

Dans le cadre de cette étude seul des processeurs peu configurables (bien qu'avec différentes tailles de bus et possibilités de coprocesseurs) ont été utilisé. Cependant leur utilisation est orthogonale à l'approche. Des processeurs paramétrables pourraient donc être intégrés dans des sous-systèmes.

1.5

Description du générateur de plateformes exécutables conçu dans cette thèse

Le générateur de plateforme proposé permet la création rapide de modèle exécutable TLM d'architecture HMPSoC. Il lit une description de l'architecture d'ensemble et appelle ensuite des générateurs spécialisés de sous-systèmes. Enfin, il interconnecte les sous-systèmes générés. L'architecture est donc une interconnexion de sous-systèmes.

En effet, pour intégrer un HMPSoC avec un grand nombre de processeurs, des caches, des mémoires embarquées, des composants matériels spécifiques, il faut utiliser un circuit de grande taille. Or, sur ces circuits il ne sera possible d'utiliser une horloge synchrone que sur des zones réduites. L'utilisation d'un réseau sur puce [43] [44] permet aussi de pallier à ce problème en gérant des communications entre des composants ayant des domaines d'horloges différents. Le réseau est alors interconnecté avec un ensemble de nœuds de traitement.

Ces nœuds de traitement sont un agrégat de composants matériels, tels que des mémoires, des processeurs, des interconnexions locales. Ils peuvent donc être des composants simples ou des systèmes multiprocesseurs complets. Les nœuds de traitement sont connectés entre eux via une interface spécifique, soit directement soit à travers des sous-systèmes de communication contenant des réseaux globaux, voir figure 1.5.2.

Différentes technologies peuvent être employées, telles que connexions par bus, crossbar ou point à point. Dans le cas d'un grand nombre de nœuds, une connexion par bus offre une bande passante trop faible tandis qu'un crossbar utilise un trop grand nombre de transistors sur le composant. Les

technologies de type point à point (non totalement connecté, grille, arbre) ou de bus hiérarchiques semblent donc s'imposer.

De ces constatations a résulté le choix de modéliser le HMPSoC sous la forme de sous-systèmes.

1.5.1 Architecture du flot de génération d'une plateforme exécutable modélisée au niveau transactionnel

Le générateur de plateforme TLM exécutable permet la création puis la modification rapide du modèle TLM exécutable. Le générateur prend en entrée des fichiers de composition de l'architecture. Il s'appuie ensuite sur un ensemble de générateurs paramétrables de sous-systèmes.

Le flot de génération d'une plateforme, est visible figure 1.5.1. En entrée il prend des fichiers de composition de l'architecture qui sont, d'une part un fichier de déclaration des sous-systèmes présents et de leurs interconnexions; d'autre part, un ensemble de fichiers (un par sous-système) contenant les paramètres destinés au générateur de ce sous-système traitement (que ce soit traitement de calcul ou d'interconnexions). Ces fichiers sont écrits par l'utilisateur de l'outil. L'autre entrée de l'outil est un ensemble de générateurs de sous-systèmes.

L'outil interprète le fichier de composition de l'architecture. Il appelle le générateur de chaque sous-système présent avec ses paramètres de génération. Enfin il interconnecte les différents sous-systèmes entre eux. Le résultat est un modèle TLM exécutable de la plateforme matérielle.

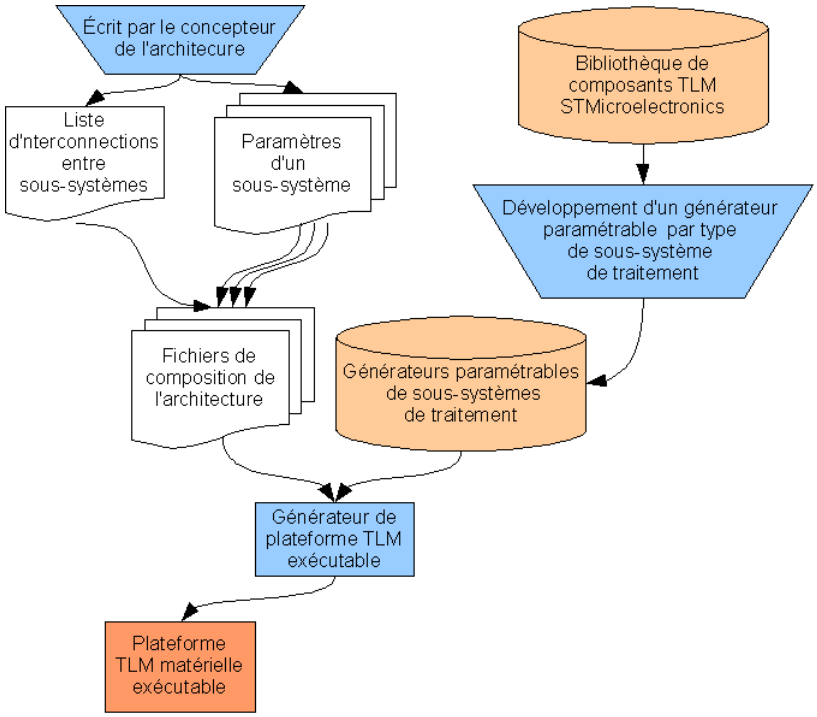

FIG. 1.5.1 – Architecture du flot de génération d'une plateforme TLM exécutable

Les générateurs de sous-systèmes sont écris manuellement. Ils ont été conçus dans le cadre de cette thèse. Ils se servent, lors de la génération, de composants TLM provenant d'une bibliothèque existante à STMicroelectronics^{TM}.

1.5.2 Modèle de l'architecture générée

Le modèle d'architecture généré est un ensemble de sous-systèmes adaptables. Chacun des sous-systèmes exporte une interface vers l'extérieur. Cette interface comprend un nombre variable de ports (avec ou sans gestion d'adresse) ainsi que des fonctions logicielles. Ces sous-systèmes sont interconnectés via leurs ports.

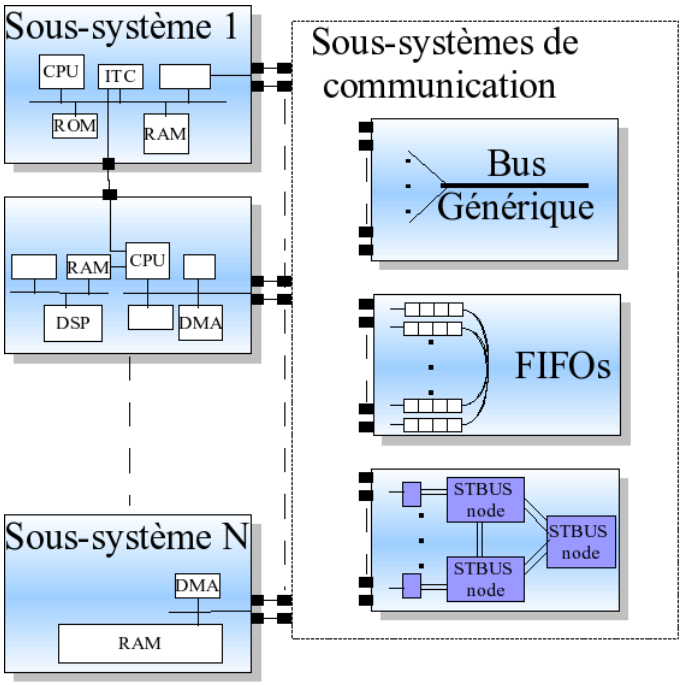

FIG. 1.5.2 – Exemple d'architecture générée, assemblage de sous-systèmes (de calcul et de communication)

Lors de la création d'une telle architecture il faut définir préalablement le découpage en différents sous-systèmes. Le travail est simplifié lors de l'utilisation d'anciens sous-systèmes existants.

1.5.3 Modèle générique des sous-systèmes de traitement

Le sous-système de traitement est un ensemble de composants matériels. Il est encapsulé dans un composant TLM paramétrable.

Les sous-systèmes peuvent être classés en trois familles :

Sous-systèmes de calcul : ils contiennent des processeurs avec leurs composants associés, tels que mémoires, gestionnaires d'interruptions ainsi que le réseau local.

Sous-systèmes matériel : ils comprennent des composants matériels (c'est-à-dire sans logiciels) tels qu'une mémoire ou des circuits spécifiques.

Sous-systèmes de communication : ils incluent un réseau de communication, tels que bus, réseau sur puce et/ou composants matériels de communication, mémoires partagées, DMA.

Chaque sous-système de traitement présente des ports (interface pour la simulation) et une interface logicielle vers l'extérieur (utile aux outils du flot). Pour l'étude, les ports sont soit des ports basés sur la librairie *TLM_TAC* réalisée avec *SystemC* [45] [46] et contenant un port initiateur et un port receveur, soit des signaux *SystemC* [47]. Les paramètres des générateurs comprennent des paramètres des composants internes au sous-système, par exemple la taille d'une mémoire, et ceux pour l'algorithme de génération, tels que le nombre de processeurs pour un nœud multi-processeurs symétrique *"Symmetrical MultiProcessing"* ou la présence d'un composant optionnel. Ces paramètres sont fourni en entrée dans le fichier de paramètres du sous-système qui est parcouru et interprété par le générateur associé au sous-système.

Les sous-systèmes de communication sont vus comme des composants à part entière. Ceci permet de s'abstraire de leur réalisation, autant au niveau de la topologie que celui du niveau d'abstraction (TLM, BCA).

1.5.3.1 Le fichier de composition de l'architecture pour le générateur de plateforme TLM exécutable

Un fichier de composition de l'architecture pour le générateur de plateforme TLM exécutable contient les données concernant les sous-systèmes de traitement à générer, les liens vers leurs fichiers de paramètres, leur nom ainsi que les interconnexions, voir figure 1.5.3. Dans l'exemple, les sous-systèmes a générer sont *node_STxP70_simple*, *node_easy_arm* et *node_generic_memory*. Un nom est spécifié pour chacun, respectivement *node1*, *node2* et *node3* ainsi que son fichier de paramètres pour la génération *./fichierparametre*. Suivent les interconnexions entre les sous-systèmes de traitement. Par exemple pour la première connexion (voir *bind*) le port 1 de *node1* est connecté au port 2 de *node3*. Les interconnexions doivent être point à point. L'architecture résultant de cette description est visible sur la figure 1.5.4.

```
<?xml version="1.0"?>
<hmpsoc:arch
xmlns:hmpsoc="http://SchemaMacroArchitectureDescription"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://MacroArchitectureDescription
/schema.xsd">

<hmpsoc:nodes>
<hmpsoc:node hmpsoc:name="node1"
hmpsoc:type="node_STxP70_simple">./fichierparametre1
</hmpsoc:node>
<hmpsoc:node hmpsoc:name="node2"
hmpsoc:type="node_easy_arm">./fichierparametre2
</hmpsoc:node>
<hmpsoc:node hmpsoc:name="node3"
hmpsoc:type="node_generic_memory">./fichierparametre3
</hmpsoc:node>
</hmpsoc:nodes>

<hmpsoc:binds>
<hmpsoc:bind hmpsoc:init="node1" hmpsoc:ip="1"
hmpsoc:target="node3" hmpsoc:tp="2"></hmpsoc:bind>
<hmpsoc:bind hmpsoc:init="node2" hmpsoc:ip="1"
hmpsoc:target="node1" hmpsoc:tp="2"></hmpsoc:bind>
<hmpsoc:bind hmpsoc:init="node2" hmpsoc:ip="2"
hmpsoc:target="node3" hmpsoc:tp="1"></hmpsoc:bind>
</hmpsoc:binds>
</hmpsoc:arch>
```

FIG. 1.5.3 – Fichier de composition de l'architecture - Sous-systèmes et interconnexions

La syntaxe du fichier de composition d'architecture est sous un format *XML*; un schéma de description lui est associé (il est disponible annexe A). Le fichier de paramètres est volontairement séparé et son format dépend seulement du générateur de sous-systèmes.

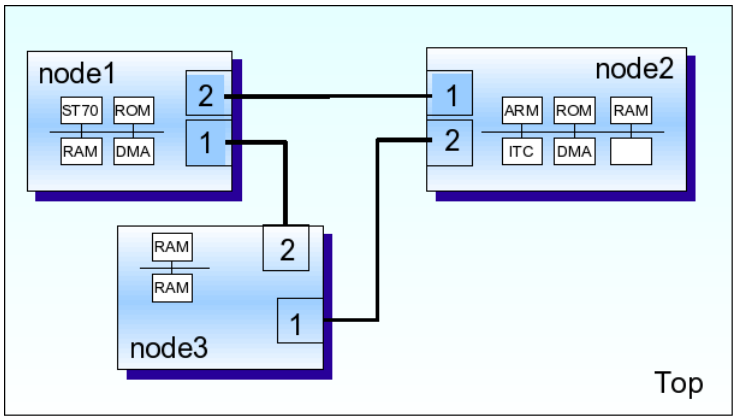

FIG. 1.5.4 – Architecture résultant du fichier de composition de l'architecture de la figure 1.5.3

Cette architecture matérielle, figure 1.5.4, est construite avec le fichier de composition d'architecture de la figure 1.5.3. L'architecture contient trois sous-systèmes interconnectés via des liaisons point à point.

Le générateur de plateforme accède aux références des ports via l'interface logicielle (pour les outils, voir 1.5.3) des sous-systèmes. Il utilise ces références pour connecter les sous-systèmes.

1.5.3.2 Organisation logicielle du générateur

Le générateur de plateforme (ou encore de modèle matériel de l'architecture complète), dont la description est visible figure 1.5.5, est composé de générateurs de sous-systèmes de traitement. Chacun de ces générateurs peut construire des sous-systèmes spécialisés selon des paramètres de flexibilité.

Le générateur de plateforme prend en entrée le fichier de composition de l'architecture. Ce fichier définit les sous-systèmes à créer, référence un fichier de paramètres pour chaque sous-système, décrit les interconnexions entre les sous-systèmes. Le générateur de plateforme appelle chaque générateur de sous-systèmes en lui donnant un fichier de paramètres. Le générateur de sous-systèmes interprète le fichier de paramètres, vérifie leur validité et construit un sous-système. Enfin le générateur de sous-système renvoie le sous-système généré au générateur de plateforme. Une fois tous les générateurs de sous-systèmes appelés, le générateur de plateforme interconnecte les sous-systèmes puis il retourne la plateforme générée.

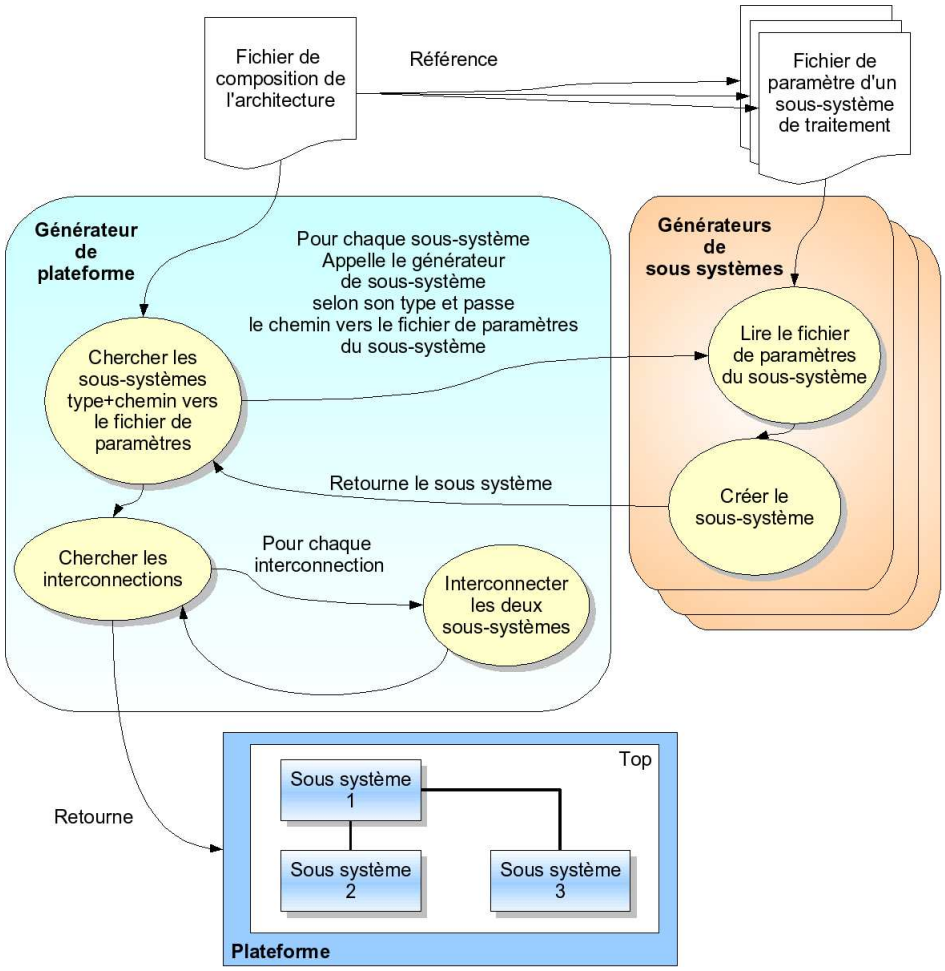

FIG. 1.5.5 – Description du fonctionnement du générateur de plateforme

Le but de cette répartition est de séparer la génération de la plateforme, de l'analyse des paramètres des sous-systèmes. Cette modularité rend plus aisée le changement de format pour les fichiers d'entrées ou de sorties. La structure du programme suit le modèle objet *"Design Pattern Factory"*[48] qui permet justement d'avoir une gestion uniforme de sous-systèmes qui sont pourtant spécialisés.

1.5.3.3 Interface logicielle à respecter par les sous-systèmes de traitement, utilisée par les outils

L'ingénieur de développement de sous-systèmes doit implémenter trois interfaces. Le générateur de plateforme a besoin d'appeler ces interfaces dans les sous-systèmes, dans deux circonstances : d'une part lors de l'assemblage

de la plateforme matérielle (connexion des sous-systèmes), d'autre part lors de l'extraction des caractéristiques de l'architecture (voir chapitre 2.2) en vue de l'adaptation des logiciels embarqués.

Leurs prototypes sont définis dans une classe C++ d'interface. Ces interfaces servent pour la connexion entre sous-systèmes, et pour l'extraction (pour l'adaptation du logiciel embarqué) des caractéristiques de l'architecture. Elles sont :

- Une interface pour l'accès aux ports via une référence.
  *sc_port * get_port(port_id id)*
- Une interface pour connecter un port à un autre.
  *void bind(port_id id, sc_port *ptr)*
- Une interface pour demander une extraction des caractéristiques sur un intervalle d'adresses à travers un port esclave. L'utilisation de ces caractéristiques est expliquée dans le chapitre 2.
  *caracteristics_t * get_caracteristics(port_id source, interval_type interval)*

1.5.3.4 Le fonctionnement du générateur de plateformes

Le générateur de modèles exécutables de plateformes fonctionne en trois étapes : la validation des paramètres, l'instanciation des composants et enfin leur connexion.

La validation des paramètres consiste à valider leurs formats, le respect de bornes mais aussi de leurs interactions, par exemple détecter l'absence d'un composant requis par un autre. Une réalisation exhaustive de cette étape est nécessaire pour une bonne qualité des sous-systèmes générés.

L'instanciation des composants est simplement l'appel des constructeurs C++ associés. Les paramètres d'appel sont soit calculés par le générateur du sous-système, soit proviennent directement du fichier de paramètres du sous-système.

Les connexions internes concernent les bus et réseaux locaux, les connexions directes entre composants de type adresse et données, et les signaux simples. Des bouchons sont utilisés dans le cas des signaux ou ports sans provenance ou sans destination.

Via les interfaces logicielles des sous-systèmes de traitement, il est possible au sous-système de :

- renvoyer une copie de l'intervalle d'adresse (*Address Map*) interne visible par l'interface;
- fournir les caractéristiques de l'architecture pour le logiciel embarqué;

– retourner son nom de sous-système, son type et son identifiant unique[1];
– permettre au générateur (pour la connexion) d'appeler les fonctions de connexion et en préalable d'obtenir les références vers les interfaces matérielles du sous-système (ports).

Le générateur crée puis connecte les modèles de sous-systèmes sans générer de sources à compiler; cette étape d'élaboration une plateforme immédiatement simulable.

1.5.4 Flexibilité apportée par la génération du modèle de l'architecture matérielle : plateforme et sous-systèmes

La génération apporte une grande flexibilité pour modifier la plateforme. Ces modifications peuvent être classées en deux catégories : les modifications au niveau de la plateforme et les modifications au niveau des sous-systèmes de traitement.

1.5.4.1 Flexibilité de composition au niveau plateforme

La flexibilité de composition de l'architecture (au niveau plateforme) a pour but de simplifier les modifications sur la composition globale de l'architecture. L'ajout, la suppression ou la duplication d'un sous-système de traitement est simple : modifications correspondantes de quelques lignes dans le fichier de composition de l'architecture. La modification d'un sous-système de traitement se fait aussi par le biais de ce fichier. La transformation en modèle exécutable est ensuite automatique par le générateur de plateforme.

Il faut noter que la flexibilité est encadrée par deux contraintes :

D'une part, les interfaces entre les sous-systèmes doivent être définies au préalable de manière cohérente. Il serait possible de définir dynamiquement un ensemble d'interfaces mais le réseau devient plus complexe car il doit les gérer. Pour l'étude, l'interface définie est celle de ports spécifiques. Ces ports spécifiques contiennent deux ports TLM de la bibliothèque STMicroelectronics™ *"TAC"* : un port est maître, l'autre esclave.

D'autre part, les logiciels embarqués doivent respecter les contraintes des sous-systèmes qui les exécuteront (notamment pour les composants adressables). Chaque sous-système doit fournir ses caractéristiques de l'architecture pour le logiciel embarqué. La partie spécifique de génération de code pour

[1]Spécifiés initialement par son générateur de sous-système, qui partage avec les autres générateurs de sous-systèmes la classe de base "générateur" : Design Pattern Factory

un processeur donné *(ARM926, ST200)* peut être commune à un ensemble de générateurs de sous-systèmes de traitement incluant des processeurs de même type. Cette seconde contrainte est levée à la section 2.3.

1.5.4.2 Flexibilité interne aux sous-systèmes de traitement

Les flexibilités internes aux sous-systèmes de traitement doivent être simples à mettre en œuvre. Leur utilisation doit rester plus facile que la réécriture du sous-système.

La flexibilité interne aux sous-systèmes de traitement correspond aux modifications de la composition du sous-système via ses paramètres.

Cette flexibilité est réalisée par l'utilisation conjointe des paramètres et du générateur du sous-système pour spécifier ces modifications. Elle permet aussi une plus importante réutilisation du code, et donc augmente sa fiabilité. La flexibilité concerne :

- la présence de composants optionnels, tels que des composants matériels, des coprocesseurs;
- la modification des paramètres des sous-composants, tels que taille des mémoires, de caches, de FIFOs;
- des paramètres pour l'algorithme de génération, tels que la scalabilité (sous-système SMP), ou encore les connexions internes. Leurs impacts peuvent être importants.

L'algorithme de génération doit gérer les caractéristiques associées aux différents paramètres. La présence de composants ou les paramètres comme la taille des adresses influent sur la carte des adresses et les interruptions à gérer.

1.5.5 Les paramètres du générateur de sous systèmes

Les paramètres du générateur de plateformes sont dans le fichier de composition d'architecture. Ces paramètres sont :

- le nombre de sous-systèmes de traitement et leurs types (identifiant du générateur de sous-systèmes de traitement correspondant);
- les paramètres de génération de chaque sous-système de traitement; Ceux-ci sont transmis au générateur de sous-systèmes de traitement associé. Ils sont dans des fichiers séparés avec une syntaxe spécifique

pour le sous-système visé. Ces fichiers de paramètres sont indiqués dans le fichier de composition de l'architecture pour chaque sous-système.
- la présence de composants réseau et leurs paramètres;
- les interconnexions entre les interfaces des composants, sous-systèmes et réseaux.

Chaque format est spécifié par le générateur associé, plateforme, sous-système de traitement, de calcul ou de communication. Ils sont définis dans des fichiers séparés.

1.5.6 Les possibilités d'évolutions du générateur de plateformes exécutables

De nombreuses améliorations peuvent être ajoutées au générateur de plateformes exécutables. La première envisagée est l'utilisation du standard SPIRIT[49] comme format de description de l'architecture. Ce format pourrait aussi être utilisé en sortie du générateur de plateformes exécutables après la résolution des paramètres de flexibilité. Ceci permettrait l'utilisation de l'architecture figée dans d'autres outils.

Ensuite la flexibilité peut être augmentée par utilisation de composants sous forme de librairies dynamiques. Cette modification apporterait de nombreux avantages :
- composants compilés, à part, un par un;
- ajout/Retrait de composants sans recompilation;
- générateur indépendant des composants;
- composants avec une interface d'accès aux ports unifiée.

On peut remarquer qu'il serait possible d'améliorer le générateur de plateformes exécutables pour supporter des sous-systèmes avec différents types de ports. En effet, la fonction pour accéder à un port reste valide. La seule obligation est que le générateur de plateformes exécutables connaisse les différents ports afin de valider que la connexion est licite.

1.6

Exemples de générations de plateforme matérielle TLM exécutable par assemblage de plateformes matérielles

Cette section traite de la génération de plateforme matérielle TLM exécutable par assemblage de plateformes matérielles. La première partie détaille un sous-système de traitement flexible avec processeur. Ensuite des exemples de sous-systèmes réseaux qui ont été générés sont décrit. Enfin une utilisation de l'outil de génération sur une plateforme matérielle multiprocesseurs hétérogène à mémoires distribuées est effectuée.

Chaque composant sous-système contient l'algorithme de génération de son type de sous-système, de ses sous-composants (dont processeurs) et de ses connexions internes. Le sous-système de traitement généré dépend de l'algorithme et des paramètres d'entrée. Les sous-systèmes de traitement peuvent aussi se différencier par les valeurs des paramètres. Le générateur de sous-systèmes interprète le fichier de paramètres et appelle le constructeur du sous-système.

Ces sous-systèmes peuvent se construire de trois manières :

- sous la forme d'un composant qui contient son algorithme de génération avec une interface dynamique (nombre de ports variable);
- sous la forme d'un composant qui contient son algorithme de génération mais avec une interface figée;
- être écrit manuellement, topologie et interface figées.

1.6.1 Détail d'un générateur de sous-système de traitement flexible avec processeur

Cet exemple s'appuie sur la génération d'un type de sous-système de traitement avec CPU. Le processus de génération peut être appelé plusieurs fois avec des paramètres différents pour créer une plateforme multi-processeurs hétérogène complexe. Les spécifications initiales du sous-système sont celles de la plateforme *easy* de ARM [50]. Certains composants, initialement figés, sont transformés en paramètres. Un composant DMA a aussi été ajouté, en prévision de contraintes de communications supplémentaires du à l'architectures multi-processeurs.

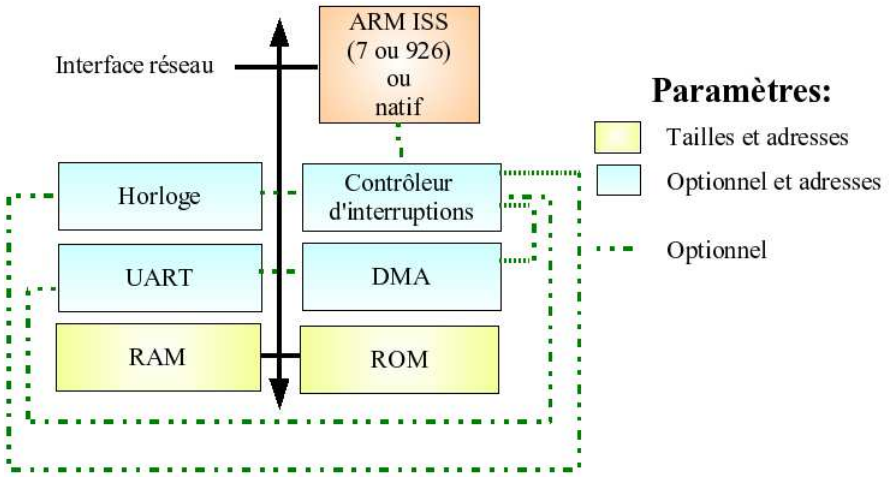

FIG. 1.6.1 – Diagrammes des sous-systèmes générés

Dans cet exemple, les fils d'interruptions entre les composants,tels que horloge, UART, DMA et le contrôleur sont gérés par l'algorithme du générateur de sous-systèmes de traitement. Si le composant est demandé dans le fichier de paramètres, le fil est créé et connecté. En cas d'instanciation illogique, par exemple une horloge sans contrôleur d'interruption, un message d'avertissement provenant du générateur du sous-systèmes est transmis au concepteur.

1.6.1.1 Les composants matériels des sous-systèmes de traitement générés

La plateforme peut contenir six types de composants provenant de la bibliothèque TLM *TAC* de STMicroelectronics^{TM}:

- un contrôleur d'interruptions : *ITC "Interrupt Controller"*;
- une interface série : *UART "Universal Asynchronous Receiver-Transmitter"*;
- une horloge : *TIMER*;
- une mémoire de code : *ROM "Read Only Memory"*;
- des mémoires locales : *RAM "Random Access Memory"*;
- Direct Memory Access : *DMA "Direct Memory Access"*.

Les paramètres des mémoires (tailles) sont modifiables. Les mémoires sont des composants obligatoires. Les quatre autres , l'UART, le DMA, le TIMER et l'ITC, sont optionnels.

Les antémémoires et les coprocesseurs sont, dans ce cas, inclus dans l'ISS. Donc leur gestion et leurs paramètres dépendent du modèle de processeur. Par exemple, les types d'antémémoire différents entre les processeurs ARM926 et ARM946 respectivement *"write-thru"* et *"write-back"*.

1.6.1.2 Les processeurs possibles et leurs paramètres

Le générateur de sous-systèmes de traitement décrit permet le choix entre différents processeurs. Les composants d'encapsulation des simulateurs de processeurs proviennent de la bibliothèque TLM *TAC* de STMicroelectronics^{TM}.

Ces processeurs sont eux-même paramétrés via un fichier spécifique qui doit être écrit manuellement pour chaque configuration de caches et de co-processeurs souhaitée par le concepteur pour un type de processeur.

Cette restriction dans l'automatisation du processus vient du fait que les outils employés (ici l'ISS) ne sont pas initialement prévus pour être utilisés dand un contexte automatique. L'utilisation principale (voire unique) jusqu'à peu était l'ISS seul dans une configuration incluant ses périphériques locaux, dont les mémoires.

L'augmentation de l'utilisation et de la standardisation de TLM (notamment autour du consortium *OSCI*) tend à faire disparaître ce problème. La multi-instanciation d'un ISS est maintenant souvent requise par les clients lors de l'achat de licence.

Les processeurs que ce générateur peut instancier dans le sous-système sont une partie de la gamme ARM^{TM}, ils fournissent chacun une interface vers l'extérieur similaire (ports, interruptions) :

- **ARM7TDMI**
- ARM7TDMI-S
- ARM720T
- ARM9TDMI
- ARM920T
- ARM940T
- ARM9E-S

- **ARM926EJ-S**
- ARM946E-S
- ARM1020E
- ARM1022E

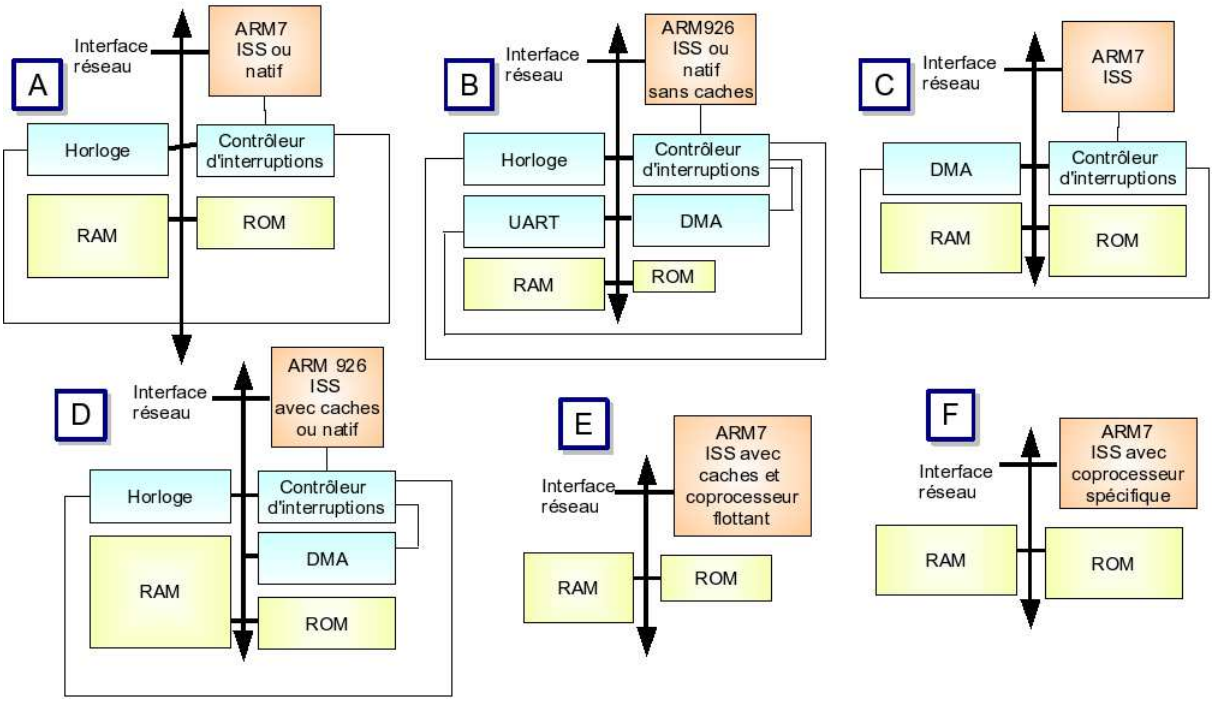

FIG. 1.6.2 – Exemples de génération possible pour un générateur spécifique de sous-système de traitement

Les différents paramètres sur les composants et les processeurs permettent la génération d'un grand nombre de sous-systèmes de traitement différents, quelques exemples sur la figure 1.6.2 ci-dessus.

1.6.1.3 Le fichier de paramètres pour le générateur d'un sous-système de traitement avec processeur

La syntaxe du fichier contenant les sous-systèmes et interconnexions est figée car liée au générateur de plateforme. La syntaxe des paramètres des sous-systèmes est dépendante de chaque générateur de sous-système de traitement. L'exemple suivant a pour but de fournir un modèle type dans le cas de sous-systèmes de traitement avec processeur.

```
<?xml version="1.0"?>
<node version="node_easy_arm">
 <parameters>
  <PROCESSOR name="ARM" type="ARM7TDMI_rev4"
debug="st_armsd" endianess="bi" path="ARM_1/bin/ISS/">
   <LEVEL>ISS</LEVEL>
   <COPROS />
  </PROCESSOR>
  <ITC></ITC>
  <DMA></DMA>
  <ROM>0x2000</ROM>
  <RAM>0x4000</RAM>
  <NET size="0xF7FFFFFF">0x08000000</NET>
 </parameters>
</node>
```

FIG. 1.6.3 – Fichier de paramètres d'un générateur de sous-système de traitement

Dans l'exemple de la figure 1.6.3, les paramètres du générateurs de sous-systèmes de traitement sont : la présence d'un contrôleur d'interruption et d'un DMA, les tailles des mémoires $0x2000$ octets pour la ROM et $0x4000$ octets pour la RAM. La balise NET représente l'espace d'adressage vers l'extérieur du sous-système. Ceci correspond au sous-système C de la figure 1.6.2.

La syntaxe générale proposée est donc la liste des processeurs avec leurs paramètres de configuration. Chaque processeur inclut la liste de ses coprocesseurs et leurs paramètres. Ensuite viens la liste des composants du sous-système avec leurs paramètres. Si un composant est nécessaire à d'autres composants, il peut inclure ceux-ci entre ses balises.

1.6.2 Exemples de générateur de sous-systèmes de traitement de type réseaux

Trois sous-systèmes réseaux ont été utilisés, ils sont de deux types, dont un décrit en deux niveaux d'abstraction. Ils ont tous les trois des interfaces identiques vers les autres sous-systèmes. Ces interfaces sont une série de deux ports TLM. Un de ces ports est un port maître et l'autre est un port esclave. Chacun de ces trois réseaux est abstrait sous la forme d'un composant sous-système de traitement, ceci vise trois possibilités :

La scalabilité : le nombre de connexions du composant réseau peut être un paramètre du sous-système. Le sous-système contient l'algorithme de génération du réseau avec les interconnexions et composants internes.

L'interchangeabilité : les interfaces entre le réseau et les sous-systèmes sont fixées et connues. Il est facile de remplacer un réseau par un autre. Il serait possible d'utiliser plusieurs interfaces avec un mécanisme de négociation entre le réseau et les sous-systèmes.

La réutilisabilité : Un réseau créé de cette manière peut être repris et utilisé directement dans une autre plateforme.

Ces réseaux sont :

- Deux réseaux basés sur les adresses, lectures ou écritures avec adresse et donnée. Il sont décrit à deux niveaux d'abstractions différents;
- un réseau de type crossbar avec des FIFOs, fonctionnant avec des identifiants de sous-systèmes.

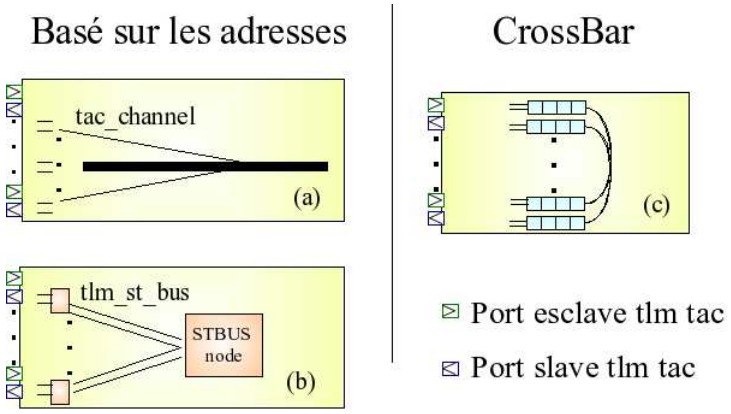

FIG. 1.6.4 – Les réseaux scalables, des composants SystemC

Le premier type de réseau a été implémenté à deux niveaux d'abstraction différents, voir figure 1.1.1. D'abord avec un bus au niveau PV (*tac_channel*) qui est utilisé pour simuler le réseau. Ensuite, dans un niveau d'abstraction plus précis, BCA, le réseau est remplacé par un nœud STBus et ses interconnexions. Le protocole de communication découpe les accès en une requête et une réponse. La taille des données et les mots de commande du STBus sont respectés. Le comportement vis-à-vis des conflits (arbitre des nœuds) est modélisé. Des interfaces standards sont utilisées pour transformer les transactions PV "tac" en transaction BCA "STBus".

Le second type de réseau est décrit seulement à un niveau fonctionnel. Le modèle du composant contient le comportement du réseau sous la forme de code C++. Ce réseau gère trois types de requêtes :

– transmission de données, émission ou réception;
– demande d'informations, message présent, numéro de l'émetteur, taille du message;
– configuration, sous-système destinataire, taille du message.

Cet algorithme gère la transmission des écritures vers la FIFO(logicielle) du sous-système de destination. En cas de requête de lecture les données sont extraites de la FIFO et transmises comme réponse.

1.6.3 Utilisation du générateur de plateforme matérielle TLM exécutable sur une architecture définie par un concepteur

Le générateur décrit section 1.5 a été développé en construisant des plateformes de test. Pour le valider il a ensuite été utilisé pour construire une plateforme expérimentale avec une architecture pour un encodeur H264. Cette architecture est construite suivant les demandes d'un concepteur de HMPSoCs.

1.6.3.1 Architecture avec une mémoire distribuée

L'architecture avec une mémoire distribuée est conçue selon le modèle utilisé par le générateur. Un ensemble de sous-systèmes de traitement interconnectés par un réseau global.

Chaque sous système de traitement contient un processeur, deux mémoires et différents composants matériels. Des mémoires globales sont interconnectées au réseau global pour les communications et les synchronisations inter-tâches.

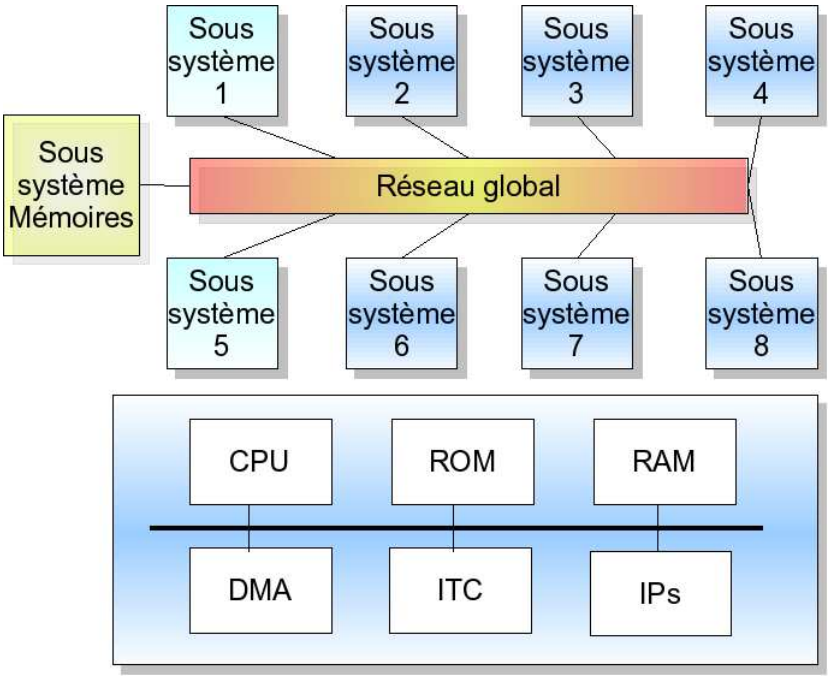

FIG. 1.6.5 – Une configuration de l'architecture H264 créée

Le nombre de sous-systèmes dépend de l'exploration choisie et des contraintes suivant des paramètres dans l'algorithme d'encodage vidéo.

1.6.3.2 Description des étapes de réalisation avec leur niveaux d'abstraction

Il a d'abord fallu développer un nouveau système flexible contenant des mémoires pour implémenter les mémoires globales partagées, ensuite créer les fichiers de paramètres des différents sous-systèmes, enfin écrire le fichier de composition de l'architecture avec les sous-systèmes et les interconnexions.

La première architecture a été conçue avec une simulation native pour les processeurs. Ensuite les paramètres des sous-systèmes de traitement ont été modifiés pour remplacer les simulations natives par des ISS.

Quand ces différents fichiers sont finis, l'outil de génération est exécuté pour créer dynamiquement la plateforme TLM exécutable.

1.6.3.3 Explorations architecturales souhaitées par les concepteurs

Ce flot a été présenté à plusieurs architectes de systèmes sur puce. Les explorations architecturales qu'ils souhaitent peuvent se séparer en trois catégories.

1. L'architecture locale au processeur
2. L'architecture globale du système
3. Le réseau d'interconnexion global

Pour la première catégorie, les modifications touchent le processeur et ses périphériques proches. Il peut s'agir de changement de type ou de version de processeur, de la modification des coprocesseurs ou des instructions spécifiques. Il est aussi courant de vouloir modifier la taille des mémoires, ainsi que les composants matériels présents, ajout, modification ou suppression.

L'exploration de l'architecture globale consiste à la modification du nombre et des types de sous-système présents. L'exploration du nombre de processeurs nécessaire pour l'application est particulièrement demandée. La modification de l'architecture mémoire de l'ensemble du système rentre aussi dans cette catégorie.

Enfin la modification du réseau d'interconnexion global consiste en l'essai de différents réseaux d'interconnexion et avec différentes topologies.

Le but du générateur de plateforme est d'aider à la mise en place de ces explorations, d'une part par la génération automatique, d'autre part avec les règles de codage. Par exemple l'écriture du réseau global sous la forme de sous-systèmes permet de le remplacer rapidement.

1.7

Conclusion relative au générateur et proposition d'améliorations

Depuis le début de ces travaux, plusieurs demandes pour des explorations d'architectures multi-processeurs (entre vingt et trente cœurs hétérogènes) ont été faites par des équipes de conceptions (STMicroelectronics). Des points communs peuvent être extraits.

Le nombre de processeurs sur l'architecture finale ne peut être prévu par calcul. Des expérimentations sont nécessaires pour le déterminer. Dans le modèle, ce nombre doit être un paramètre modifiable facilement (de préférence en paramètre de la simulation).

La vitesse de la simulation est un souci constant. L'impact du grand nombre de processeurs doit donc être réduit en rendant possible des simulations natives.

La programmation de tels systèmes avec un grand nombre de processeurs fait aussi partie des problèmes principaux. Il est nécessaire que les programmeurs puissent expérimenter les différentes solutions. Un modèle de l'architecture, de préférence à haut niveau d'abstraction (simulation rapide) doit être disponible rapidement.

L'ensemble de ces points valide l'approche suivie. La méthode utilisée dans l'expérimentation convient aux besoins. Suivant les demandes des spécificités peuvent être prises en compte : simplification de l'interface des sous-systèmes de traitement par un nombre de port figé, nombre de sous-systèmes de traitement différent fini. Mais l'approche correspond à la demande pour ces études architecturales.

1.7.1 Non séparation entre sous-systèmes de traitement et sous-systèmes de communication

Au début de la conception de l'outil de génération de modèles de plateformes TLM exécutables, l'architecture orientée objet interne du générateur séparait les sous-systèmes de traitement et les sous-systèmes de communication avec deux classes de base différentes. Ce partitionnement devait permettre à chacun de définir et implémenter une interface différente.

Finalement après les développements et les différentes expérimentations ces interfaces sont devenues identiques, deux fonctions pour la connexion des interfaces matérielles et une troisième pour l'extraction des caractéristiques architecturales.

Il semble donc que la différenciation entre ces deux types de sous-systèmes soit inutile. Ils peuvent donc être unifiés sous le seul concept de sous-système de traitement avec une classe de base logicielle commune.

1.7.2 Possibilité de flexibilités supplémentaires dans la génération du modèle TLM exécutable de l'architecture matérielle

La génération du modèle TLM exécutable de l'architecture et des sous-systèmes de traitement qui la composent permet au concepteur d'architecture une réalisation et des modifications rapides. Cependant certains points ont montré leurs limites et devraient être améliorés. Ces points sont aux niveaux des sous-systèmes de traitement, des composants TLM et des interfaces de communication.

1.7.2.1 Possibilité de flexibilités supplémentaires au niveau des sous-système de traitements

Une limitation importante de l'outil est l'obligation de modifier le code source et de recompiler pour pouvoir utiliser un nouveau type de sous-système. Le chargement via une bibliothèque dynamique permettrait de résoudre cette contrainte. Les sous-systèmes devraient simplement satisfaire à une fonction d'interface supplémentaire pour leur instanciation. On peut remarquer que cela pourrait aussi être utilisé pour les composants TLM standards. Ceci serait une première technique qui permettrait de les incorporer

directement dans la description de l'architecture sans passer par un sous-système.

L'autre aspect est la création graphique de l'architecture. Une telle solution a été envisagée et semble possible notamment via le standard industriel SPIRIT.

1.7.2.2 Possibilité de flexibilités supplémentaires au niveau des composants TLM

Actuellement l'ajout d'un nouveau type de composant dans un sous-système demande la modification du générateur. Il serait possible de rajouter la possibilité de connecter des sous-systèmes à des composants matériels. Ceci demanderai l'ajout d'un constructeur du composant prenant ses paramètres dans un fichier pour chaque composant TLM. Cependant dans le cas de composants polymorphes, les instances possibles devraient être listées et précompilées comme autant de composants séparés.

Il serait aussi possible de permettre la gestion de multiples interfaces sur un port. Lors de la connexion les deux composants s'accordent pour choisir les meilleures interfaces compatibles.

1.7.2.3 Possibilité de flexibilités supplémentaires au niveau des interfaces de communications des sous-systèmes de traitement

On peut remarquer que l'utilisation des interfaces pourrait aussi être étendue pour les composants TLM standards. Ceci serait une seconde technique qui permettrait de les incorporer directement dans la description de l'architecture sans passer par un sous-système. Il parait utile de pouvoir le faire pour des composants découplés des processeurs, comme des accélérateurs matériels ou des mémoires partagées.

1.7.3 Architectures matérielles générées

Le modèle d'architecture matérielle choisi n'impose pas de limitation sur les architectures possibles. Par contre, la modélisation en elle même est contrainte. Tous les éléments doivent être décrit sous la forme de sous-systèmes. On peut remarquer qu'un sous-système ne peut contenir qu'un unique composant.

Lors de cette modélisation, plusieurs voies sont possibles suivant deux axes :

– la taille des sous-systèmes en nombre de composants;
– le nombre de paramètres pour la génération de différents sous-systèmes.

Dans les deux cas les conséquences sont identiques.

Pour un petit sous-système avec peu ou pas de paramètres de génération, la création du sous-système est simple, peu de risque d'erreurs mais la réutilisabilité et l'exploration d'architectures sont limités.

Pour un grand sous-système avec de nombreux paramètres, le développement est plus long mais le sous-système pourra être utilisé dans de nombreux cas de figures et permettra de meilleures adaptations.

1.7.4 Conclusion sur la génération de plateformes matérielles décrites au niveau transactionnel exécutables

Dans ce chapitre est proposé une méthode et un générateur de plateforme associé pour le développement et la modification rapide d'architectures de type multi-processeurs hétérogènes. Un générateur de plateforme TLM exécutable fonctionnant avec un ensemble de générateurs de sous-systèmes de traitement a été développé. Il a été utilisé avec succès pour générer puis modifier rapidement un modèle TLM d'architecture matérielle multiprocesseurs hétérogène.

Cette architecture matérielle ainsi que les modifications apportées ont pu être testées en portant un logiciel de tests parallèles simples. Certaines modifications de l'architecture ont demandé des adaptations de ce logiciel. Ces adaptations demandent de modifier les couches bas niveau du logiciel, ce qui est long et source d'erreurs. Les avantages de la création et la modification de l'architecture sont donc limités par le temps nécessaire à l'adaptation des logiciels. Il faut donc une technique pour pourvoir porter et adapter rapidement les logiciels embarqués sur l'architecture multi-processeurs hétérogène avant de pouvoir envisager de disposer d'une véritable approche pour la conception de système sur puce avec un grand nombre de processeurs.

Chapitre 2

Portage rapide des logiciels embarqués sur l'architecture multi-processeurs hétérogène

Dans ce chapitre, une méthode pour réaliser un portage rapide des logiciels embarqués sur une architecture multi-processeurs hétérogène est proposée. En premier lieu une technique pour extraire de l'architecture les caractéristiques nécessaires pour les logiciels embarqués, ou *"modèle abstrait de l'architecture pour les logiciels embarqués"*, est décrite section 2.2. Une implémentation de ce mécanisme d'extraction des caractéristiques a été réalisé dans des plateformes TLM matérielles exécutables. Ensuite un outil qui utilise ces caractéristiques nécessaires pour les logiciels embarqués pour générer la couche bas niveau du logiciel ainsi que les fichiers nécessaires à la compilation a été développé, il est décrit section 2.3. Enfin ce flot et les outils associés sont utilisés sur un exemple réel d'encodeur logiciel H264, à partir d'un code source parallèle de haut niveau existant, section 2.4.

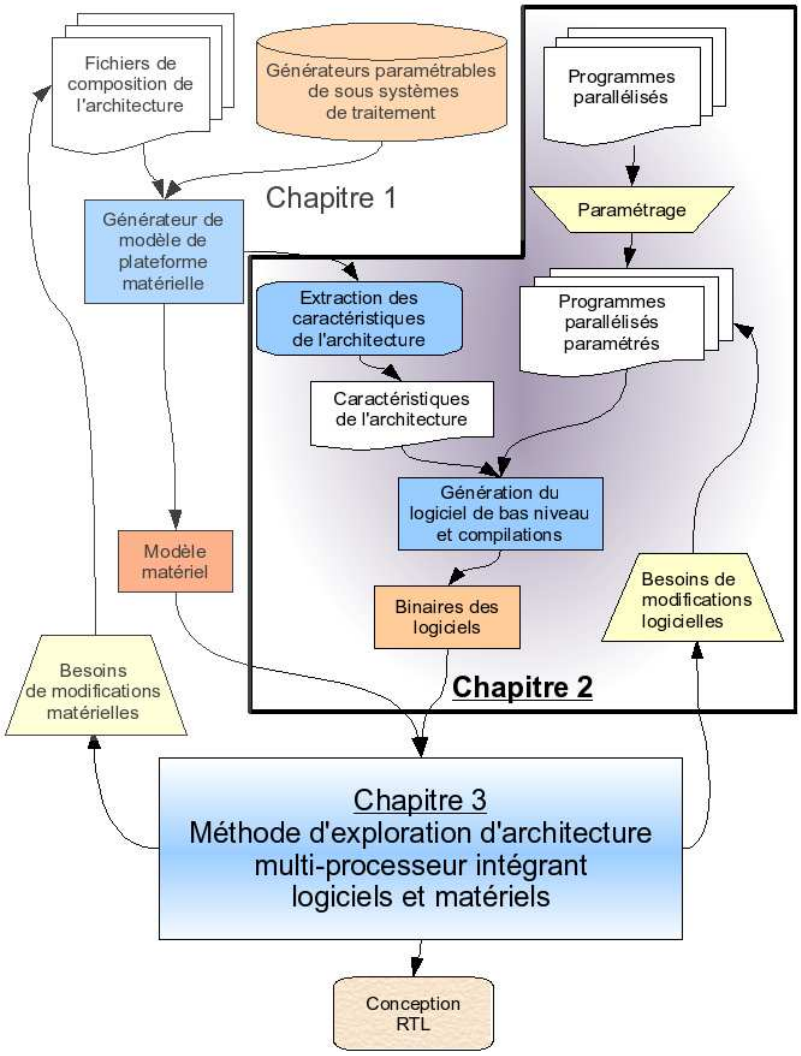

FIG. 3 – Adaptations, paramétrages et compilations des logiciels

Le flot, figure 3, décrit dans ce chapitre prend deux entrées, des programmes parallélisés de haut niveau et la sortie d'un mécanisme d'extraction des caractéristiques d'une architecture. Les programmes parallélisés de haut niveau doivent être paramétrés manuellement. Un outil permet d'extraire automatiquement les caractéristiques d'architecture pour des plateformes développées selon le flot exposé dans le chapitre 1. Ce mécanisme d'extraction produit des fichiers de caractéristiques de l'architecture. Ces fichiers de caractéristiques de l'architecture sont utilisés par un outil d'adaptation du logiciel et de génération du logiciel de bas niveau. Cet outil génère selon l'architecture les parties logicielles de bas niveau, tels qu'un fichier de résolution des

paramètres, des scripts de compilation et d'édition des liens. Ces scripts sont utilisés sur des programmes parallélisés paramétrés pour produire les binaires des logiciels embarqués.

2.1

Les logiciels embarqués lors des modifications de l'architecture

2.1.1 Modèle de l'architecture d'un logiciel embarqué sur une plateforme multiprocesseur hétérogène

L'architecture d'un logiciel embarqué sur une plateforme multiprocesseurs hétérogène peut généralement être décrite comme un assemblage de couches logicielles figure 2.1.1. Il est possible de distinguer deux couches principales. La première est un ensemble de programmes, dite couche de haut niveau, écrit sous la forme d'un langage portable (dans le monde des logiciels embarqués souvent en langage C). La seconde est la partie logicielle dépendante du matériel; elle est souvent décrite avec un mélange de langage C et d'assembleur spécifique au processeur et à l'architecture matérielle.

La couche bas niveau du logiciel peut être séparée en différentes parties, notamment la gestion du matériel, initialisation du processeur, gestion des périphériques et la partie communication, soit les communications entre programmes sur un processeur et entre processeurs.

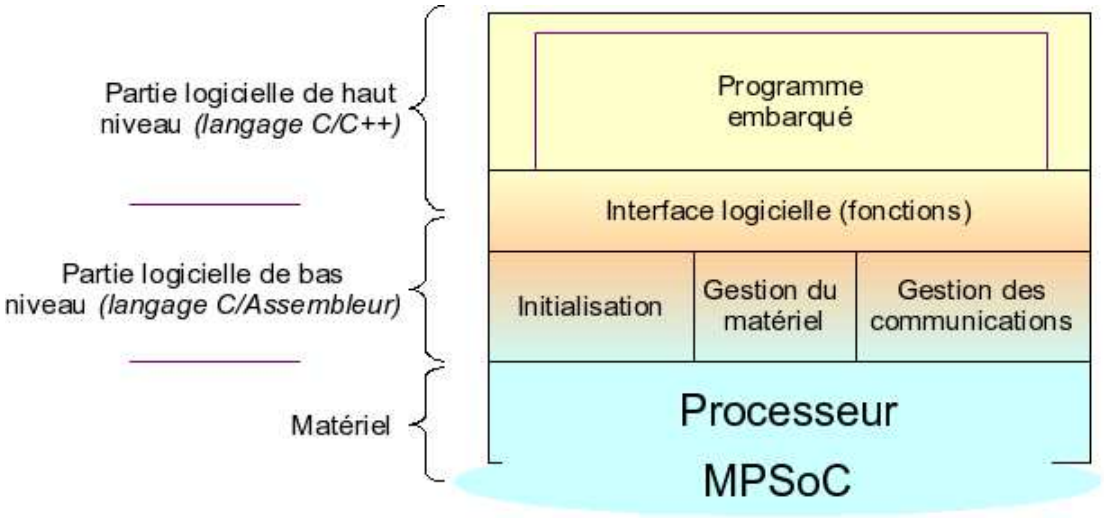

FIG. 2.1.1 – Modèle de l'architecture d'un logiciel embarqué sur une plateforme multiprocesseur hétérogène

Dans ce modèle de l'architecture d'un logiciel embarqué sur une plateforme multiprocesseurs hétérogène, la partie logiciel de haut niveau peut être écrite et validée sur un ordinateur standard. Lors du portage sur la plateforme multiprocesseurs hétérogène, la problématique est donc centrée sur la partie bas niveau du logiciel. Cette partie peut nécessiter des modifications si l'architecture change.

2.1.2 Les difficultés et la pérennité du portage des logiciels embarqués

Le portage demande des modifications du logiciel pour l'adapter à la plateforme matérielle, par exemple le processeur, les composants matériels ou l'architecture mémoire. Ce travail demande l'écriture de codes en langages assembleurs ou l'utilisation de codes déjà portés, comme un système d'exploitation.

Les modifications d'architectur :nombre et adresses de registres, adresses mémoires, de composants ou d'instructions spécifiques disponibles, peuvent influer sur les logiciels[51].

Les adaptations nécessaires à ces changements doivent être déduites des modifications matérielles. Ensuite, il faut trouver les parties de codes impactées parmi l'ensemble du logiciel et réaliser les modifications. De plus, les

erreurs qui sont provoquées par ces modifications matérielles peuvent apparaître dans des zones de codes non concernées. Elles sont donc particulièrement complexes à corriger.

De surcroît, ces adaptations coûteuses devront être reproduites à chaque exploration, il est donc crucial de les faciliter.

2.1.3 Outils existants pour le portage et la génération de codes embarqués existant

De nombreuses recherches visent à faciliter cette adaptation des logiciels embarqués sur l'architecture. Ces aides peuvent cibler différents problèmes de l'adaptation, soit les couches de bas niveau via la génération d'un système d'exploitation, ou alors la génération des interfaces logicielles de communication, ou encore la gestion du placement et ordonnancement d'une application sur un réseau de processeurs.

2.1.3.1 Génération d'un système d'exploitatio :ASOG

L'outil ASOG s'intègre dans le flot de conception ROSES développé par le groupe SLS du laboratoire TIMA [52].

L'outil *ASOG* a pour but la génération d'un système d'exploitation minimal adapté au mieux à une architecture et une application [53]. Ce système est assemblé à partir de briques de base et d'un graphe de dépendance. Chaque besoin d'interface exprimé au sein de l'application va entraîner une chaîne de dépendances de briques qui peuvent aboutir à des dépendances vers des composants matériels. L'outil *ASOG* permet donc la gestion de la couche bas niveau avec un système d'exploitation spécifique.

2.1.3.2 Génération des mécanismes des communications de l'applicatio :Peakware4SoC

Le but de l'outil *Peakware4soc* est de rendre les tâches indépendantes du placement sur l'architecture. Pour cela, il génère le code source des primitives de communication. Il se sert de graphes en entrée. Un graphe flot de données représente le logiciel. Un graphe décrit l'architecture au niveau d'une part des sous-systèmes comprenant processeurs et mémoires, et d'autre part les interconnexions. Et un tableau contient le placement désiré par l'utilisateur des différentes tâches sur les processeurs. Les tâches sont écrites en langage C. La sortie du programme est un binaire exécutable pour chaque processeur.

Le code source de l'algorithme doit être partitionné par l'utilisateur sous la forme de tâches communicantes formant un graphe de type flot de données. Les communications se font à travers un ensemble de primitives définies par l'outil, implémentées au dessus des drivers du système d'exploitation, de pilotes de périphériques ou directement sur le matériel. Pour définir ces primitives, l'outil se base sur le graphe logiciel. Toutes les communications se font par une interface de type passage de messages.

Les processeurs et le réseau sont représentés dans sous la forme d'un graphe matériel. Des annotations spécifiques sont ajoutées pour la gestion de ce réseau. De plus, chaque type de réseau correspond à un module logiciel qui gère l'implémentation des primitives de communication. Un tableau de placement des tâches sur les processeurs, facilement modifiable, sert pour cette implémentation. Le placement est statique.

Deux modules sont préexistants, la gestion d'une mémoire partagée ainsi que celle d'un DMA simple.

Les composants matériels sont décrits dans le graphe matériel avec les processeurs et réseaux. Ils peuvent servir à définir des adresses physiques de base pour le logiciel.

L'outil *Peakware4soc* permet donc la gestion de la couche bas niveau de communication.

2.1.3.3 Placement de tâches logicielles à partir de codes source :Design Trotter

L'outil *Design Trotter* [54] prend en entrée les spécifications de l'application écrites en langage C. Il les traduit sous la forme d'un graphe hiérarchique *"Hierarchical Control Data Flot Graph"*. L'ordonnancement est calculé suivant les caractéristiques de l'application, partie flot de contrôle ou flot de données.

Pour réaliser ces estimations l'outil utilise un fichier de description appelé *"User Abstract Rules"* qui contient les fonctions que peuvent réaliser les composants matériels de l'architecture et la hiérarchie mémoire (locale/globale). Ce fichier de description est écrit par l'utilisateur.

Cet outil complexe permet l'évaluation statique d'architecture logicielle et matérielle décrite sous forme de graphe de haut niveau [55].

2.1.3.4 Placement de tâches logicielles et communications à partir d'un graphe de type flot de donnée :*Syndex*

Syndex est développé dans le cadre du thème de recherche Adéquation Algorithme Architectures, un des buts [56] du logiciel *Syndex* est la génération d'un code optimal pour l'architecture cible.

Le logiciel est écrit sous la forme d'un graphe flot de données. Le test de ce graphe se fait en trois étapes.

- Placement et ordonnancement statique
- Dépendance temporelle
- Génération de l'exécutable

Syndex génère aussi les communications inter-processus.

Ce logiciel particulièrement élaboré permet aussi de la conception d'architecture mais reste limité à des applications décrites sous la forme de flots de données.

2.1.3.5 Conclusion sur les outils pour le portage et la génération de codes embarqués

Le point commun entre ces outils d'aide pour le portage et la génération de codes embarqués est la nécessité de connaître l'architecture matérielle. Des fichiers de description de l'architecture doivent actuellement être écrits par l'utilisateur de l'outil.

Or, l'ensemble de ces données peut être déduit de l'architecture matérielle. La principale contribution de ce chapitre est la mise en place un mécanisme d'extraction des caractéristiques de l'architecture. Ce fichier permettra ensuite d'utiliser ou de créer des outils pour aider à l'adéquation entre les logiciels et l'architecture.

2.2

Extraction des caractéristiques de l'architecture pour le logiciel

Cette section traite du mécanisme d'extraction des caractéristiques de l'architecture. Une possibilité d'implémentation dans le standard de communication *TLM OSCI* est aussi évoquée. Le but de ces caractéristiques est de fournir un résumé, qui contient seulement les informations de l'architecture nécessaires aux logiciels.

Ces informations sont ici utilisées directement, mais des interfaces logicielles avec des modèles de programmation de plus haut niveau pourraient s'en servir pour s'adapter à l'architecture [57].

2.2.1 Besoin de paramétrisation du logiciel dans le flot de compilation lors de son adaptation

Afin de rendre le logiciel portable sur une classe d'architecture nous proposons de paramétrer le logiciel selon des modifications matérielles possibles. Les adresses des registres doivent être fournies par un fichier de configuration. Les lignes de codes sources de programmes dépendantes d'un composant spécifique sont entourées de balises pour soit provoquer un avertissement si le composant n'est pas présent, soit utiliser un algorithme logiciel associé. Le programme embarqué prend aussi comme paramètre le nombre de processeurs présents et le type de processeur cible pour s'adapter à l'architecture.

Les possibilités de configuration sont très nombreuses et dépendantes de l'architecture, des algorithmes et des volontés des concepteurs.

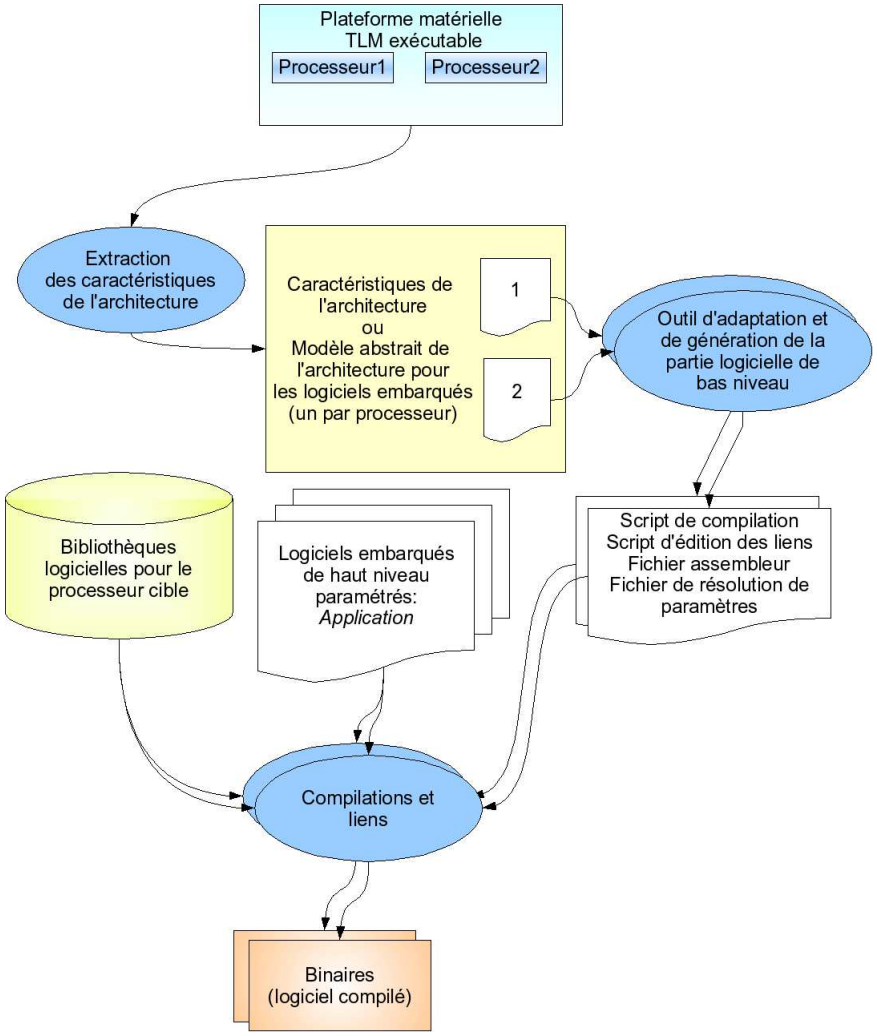

Fig. 2.2.1 – Flot de compilation avec utilisation des caractéristiques de l'architecture

Dans le flot de la figure 2.2.1, la plateforme matérielle TLM exécutable générée par l'outil du chapitre précédent est le point de départ. De cette plateforme est extrait pour chaque processeur un fichier qui contient la vision de l'architecture pour les logiciels sur ce processeur. Ce fichier contient la description de l'ensemble des composants auxquels le processeur peu accéder. C'est un modèle abstrait de l'architecture pour un processeur donné.

Un fichier associé à un processeur est utilisé lors de la compilation des logiciels embarqués sur ce processeur.

2.2.2 Description de l'architecture matérielle pour les logiciels embarqués

Les caractéristiques pertinentes de l'architecture matérielle pour les logiciels embarqués doivent être extraites indépendamment pour chaque processeur, ainsi que pour chaque composant qui peut écrire sur un bus, appelé composant maître. Elles contiennent des données sur ce composant maître puis sur tous les composants adressables, dit composants esclaves. Si des caractéristiques ne sont pas applicables à un composant elles sont laissées vides. Enfin elles contiennent aussi l'ensemble des connexions sans adresse (*"par fils"*) avec les caractéristiques de la connexion.

Ce modèle contient d'abord les caractéristiques du processeur ou du composant qui peut écrire sur un bus associé. Ensuite vient l'ensemble des caractéristiques des composants dans les registres desquels le composant associé peu écrire. Enfin vient la liste des composants connectés sur des ports sans adresse comme des fils d'interruptions.

Pour chaque composant maître instancié dans la plateforme matérielle, un fichier de description de l'architecture matérielle pour les logiciels embarqués est présent. Un exemple est visible figure 2.2.2. Un schéma de description du langage utilisé est en annexe B. Il commence par le nom du sous-système qui contient le composant, avec le numéro d'identification unique de ce sous-système et le nombre de sous-systèmes dans le plateforme matérielle. Puis il contient, *1* structure de caractéristiques pour le processeur ou pour le composant qui peut écrire sur un bus associé, les N structures de caractéristiques pour chaque composant esclave adressable et les M structures de caractéristiques pour chaque connexion sans adresse. N et M représente respectivement, le nombre de composants adressables et le nombre de connexions sans adresse du composant maître associé.

```
<HARDWARE node="ARM_NODE" number="9" all="11">
<!– PROCESSOR DESCRIPTION> <PROCESSOR
name="ARM" type="ARM926EJS_rev0"
endianess="LITTLE"> <LEVEL>ISS</LEVEL>
</PROCESSOR> <!– SLAVES DESCRIPTION> <SLAVE
name="DATA_MEMORY" type="RAM_type"
address_base="0x00300000"
address_size="0x00500000"></SLAVE> <SLAVE
name="CODE_MEMORY" type="ROM_type"
address_base="0x00000000"
address_size="0x00100000"></SLAVE> <SLAVE
name="TIMER" type="APBtimer" address_base="0x00200000"
address_size="0x00001000"
register_map="timer_easy/include/timer_easy_registermap.h">
</SLAVE> ... <SLAVE name="ARM_NODE_0_DMA"
type="DMAeasy" address_base="0x08220000"
address_size="0x00001000"
register_map="dma_easy/include/dma_easy_registermap.h">
</SLAVE> <SLAVE name="ARM_NODE_0_ITC"
type="APBitc" address_base="0x08100000"
address_size="0x00001000"
register_map="itc_easy/include/itc_easy_registermap.h">
</SLAVE> </HARDWARE>
```

FIG. 2.2.2 – Extrait d'un fichier de description d'architecture pour un processeur

Les caractéristiques pour le processeur ou le composant qui peut écrire sur un bus sont:

- le nom du composant dans la plateforme, choisit par le concepteur de l'architecture;
- le numéro de version du composant, *1.0*, *20060731a*;
- le type, soit le nom usuel du composant, *ARM7TDMI*, *DMA_PL080*;
- la présence, le nombre, les types et paramètres de co-composant (ou coprocesseur);
- les paramètres éventuels spécifiques avec leur nom et valeur, boutisme, taille du bus (*défini par l'utilisateur*);
- L'adresse du binaire et des sources.

Les caractéristiques pour chaque composants esclaves:

- le nom du composant dans la plateforme, choisit par le concepteur de l'architecture;

- le numéro de version du composant;
- le type, soit le nom usuel du composant, *uart_16550*;
- les paramètres spécifiques éventuels avec leur nom et valeur, taille de FIFO, version de firmware;
- l'adresse, le nom et le type de chaque registre ou banc de registres;
- des codes sources de drivers (*Code C/C++ paramétré*);
- le réseau à utiliser si la communication n'est pas basée que sur les adresses.

Les caractéristiques pour chaque connexion sans adresse:

- le nom du port sur le processeur;
- le nom dans la plateforme et le type de chaque composant connecté ainsi que son type, et le nom du port sur ce composant.

2.2.3 L'extraction des caractéristique :un mécanisme d'auto description

Les données disponibles de l'architecture sont les codes sources en langage C++ des sous-systèmes de traitement paramétrables, des interfaces TLM et les fichiers contenant les interconnexions entre systèmes et les paramètres de chaque système.

Les fonctions d'extraction des caractéristiques sont incluses dans la plateforme et les composants. Après une étape d'élaboration, la plateforme lance le mécanisme d'extraction.

Cette procédure est donc similaire aux techniques logicielles d'auto description, héritage, paramètres et agrégation. Mais la description est du type composants et adresses.

L'autre solution qui consisterait à analyser et interpréter le code source a été rapidement écartée pour cause de trop grande complexité sans apporter d'avantages supplémentaires.

2.2.4 L'implémentation du mécanisme d'extraction des caractéristiques de l'architecture matérielle

Le mécanisme d'extraction est effectué lors d'une exécution du modèle de la plateforme TLM avec un paramètre particulier. L'ensemble de la plateforme et des composants sont instanciés (étape d'élaboration) mais la simulation n'est pas lancée. A la place, le programme appelle la fonction d'extraction

de chaque sous-système de traitement inclut dans le modèle. Cette fonction va calculer les composants et ports externes joignables pour chaque composant maître. Les paramètres des composants esclaves locaux sont enregistrés dans le fichier de configuration. Pour les ports externes il est fait appel aux autres sous-systèmes.

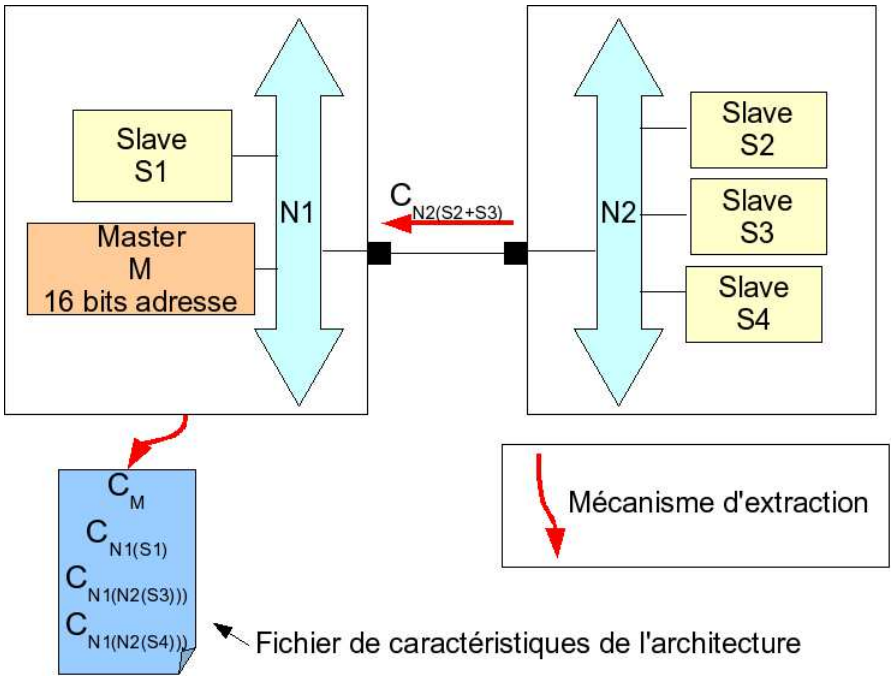

FIG. 2.2.3 – Le mécanisme d'extraction des caractéristiques de l'architecture à travers les sous-systèmes

Dans le schéma 2.2.3, le mécanisme d'extraction est appelé pour le sous-système de gauche. Le sous-système de gauche appelle la primitive d'extraction du sous-système de droite. Celui-ci retourne les caractéristiques de l'architecture associé au bus *N2*. Le sous-système de gauche effectue un tri sur les composants visibles. Enfin le sous-système de gauche écrit dans un fichier, car il contient un unique composant maître, les informations reçues et celles le concernant.

Extraction (*Intervalle*)
Pour chaque composant adressable
  Si le composant **est** un port extérieur
    Ajouter à Liste de caractéristiques **Appliquer TableDeRoutage (** sous-système connecté Extraction(*Intervalle du port*))
  Sinon si le composant est un IP
    Ajouter à Liste de caractéristiques, **Appliquer TableDeRoutage(** Caractéristiques du composant)
Retourner Liste de caractéristiques;

FIG. 2.2.4 – Algorithme du cas général de l'extraction des caractéristiques d'architecture

L'algorithme de cette extraction des caractéristiques de l'architecture, figure 2.2.4, est appelé pour chaque composant ayant un port maître. Le paramètre de l'algorithme est l'intervalle sur lequel le composant peut adresser le bus (addresse de base et taille fixées par le composant maitre au premier appel puis par les différents routeurs suivant les tables de routage). Par exemple, pour un processeur 16bits sur bus d'adressage 32bits, l'intervalle est limité de 0 à 2^{16}. Si le composant connecté est un bus il appelle la même sur chaque composant adressable sur cet intervalle. Il modifie les valeurs retournées selon ses caractéristiques, décalages ou masques et les concatène. Sinon si le composant connecté est une terminaison du réseau, il renvoie ses caractéristiques.

2.3

Utilisation des caractéristiques de l'architecture pour la génération du logiciel de bas niveau

Les caractéristiques de l'architecture vont être utilisées pour générer des couches logiciels de bas niveau ainsi que les fichiers de configuration de l'ensemble de la compilation et de l'édition des liens.

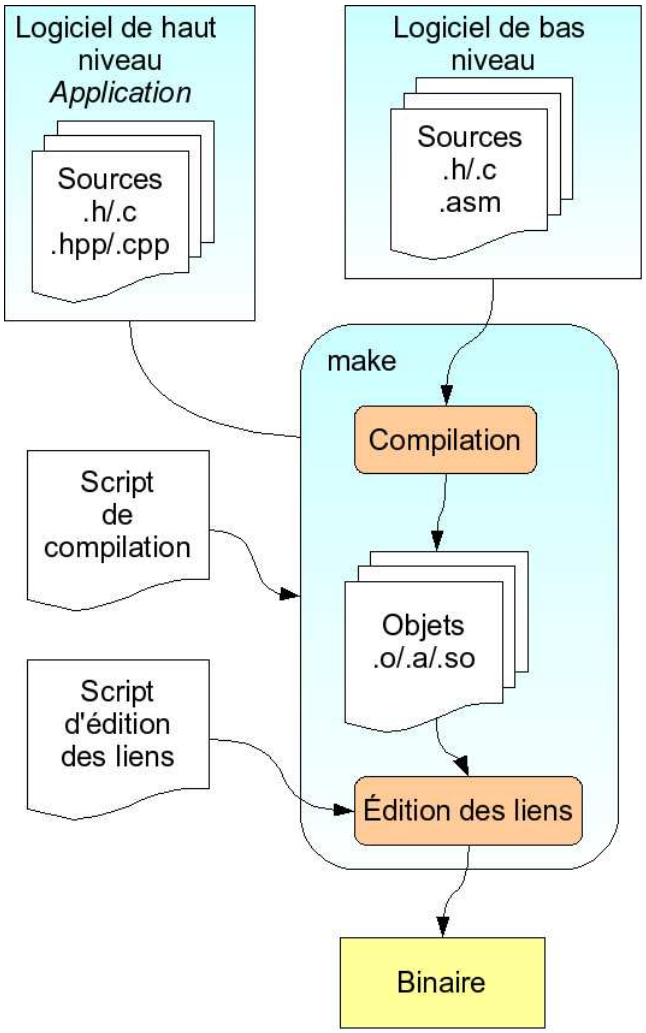

FIG. 2.3.1 – Mécanisme de compilation

Dans ce flot, figure 2.3.1, pour transformer un logiciel de haut niveau en un binaire exécutable, il est nécessaire d'adapter le logiciel à l'architecture matérielle cible. Cette adaptation se fait grâce à la partie logicielle de bas niveau, ainsi que les scripts de compilation et d'éditions des liens. Cet ensemble de fichiers doit être récrit pour chaque processeur de chaque architecture. L'adaptation du logiciel de haut niveau passe par la génération de logiciels de bas niveau ainsi que de scripts pour la compilation et l'édition des liens.

2.3.1 Le logiciel de bas niveau

Certaines contraintes d'écriture et de compilation sont spécifiques aux logiciels embarqués sur un processeur sans système d'exploitation. Ces contraintes concernent l'initialisation du processeur et de ses périphériques proches. L'unité de gestion de la mémoire, le gestionnaire de translation d'adresse, les caches doivent être configurés. Il faut initialiser les composants matériels, et enfin gérer les outils de compilation et d'édition des liens avec la gestion de la carte des adresses, notamment des différentes mémoires et d'éventuels *remaping*.

2.3.1.1 Initialisation d'un processeur et de ses périphériques proches

L'initialisation est dépendante du processeur et de ses périphériques internes et proches. Elle concerne principalement les mécanismes de gestion des exceptions, des niveaux d'utilisation et de la mémoire, mémoire virtuelle, différentes piles et tas. Il est aussi possible de configurer les caches notamment avec la définition de zones sans cache pour les communications.

2.3.1.2 Initialisation des composants matériels

L'initialisation des composants externes est soit faite par le système d'exploitation soit par le début du programme, soit encore laissée à l'application. Des pilotes de périphériques sont souvent fournis avec les composants matériels pour gérer ces initialisations et fournir une interface simple et unifiée.

Ces initialisations consistent le plus souvent en l'écriture de plusieurs registres de configuration. Mais il peut aussi s'agir de mécanismes complexes dépendants de périphériques externe au SoC et d'initialisation de structures en mémoire.

2.3.1.3 La gestion de la compilation et de ses paramètres

Ces paramètres de la compilation peuvent varier selon la configuration du processeur, sa version, ses coprocesseurs, les options choisies. Les coprocesseurs, les instructions voire les caches doivent être déclarés pour que le compilateur les utilise et optimise le code de manière efficace. Ces paramètres sont encore plus flagrants dans le cas de processeurs reconfigurables.

Des paramètres de compilation peuvent aussi être ajoutés pour paramétrer le logiciel, comme des définitions de constantes, adresses, numéro de processeur.

2.3.1.4 L'édition des liens

L'édition des liens peut demander un fichier de configuration spécifique à la chaîne d'outils. Ce fichier contient la carte des mémoires visibles par le processeur avec de nombreux paramètres. Toutes les zones utilisées par le programme doivent être définies, zone en lecture seule, zone de données ou de programme, adresse des piles.

En programmation traditionnelle sur un ordinateur, ce fichier est fournit par le système d'exploitation, cette partie et alors automatique. Dans le cas d'un système d'exploitation embarqué, il est nécessaire de fournir ce fichier pour la compilation du noyau du système d'exploitation.

2.3.1.5 Caractéristiques de l'architecture nécessaire pour la génération

Les informations principales nécessaires à la génération du logiciel de bas niveau sont les adresses des registres des composants et le type du composant ainsi que les caractéristiques du processeur. Ces informations sont écrites dans un fichier de description de l'architecture associé au processeur cible, un exemple figure 2.3.2.

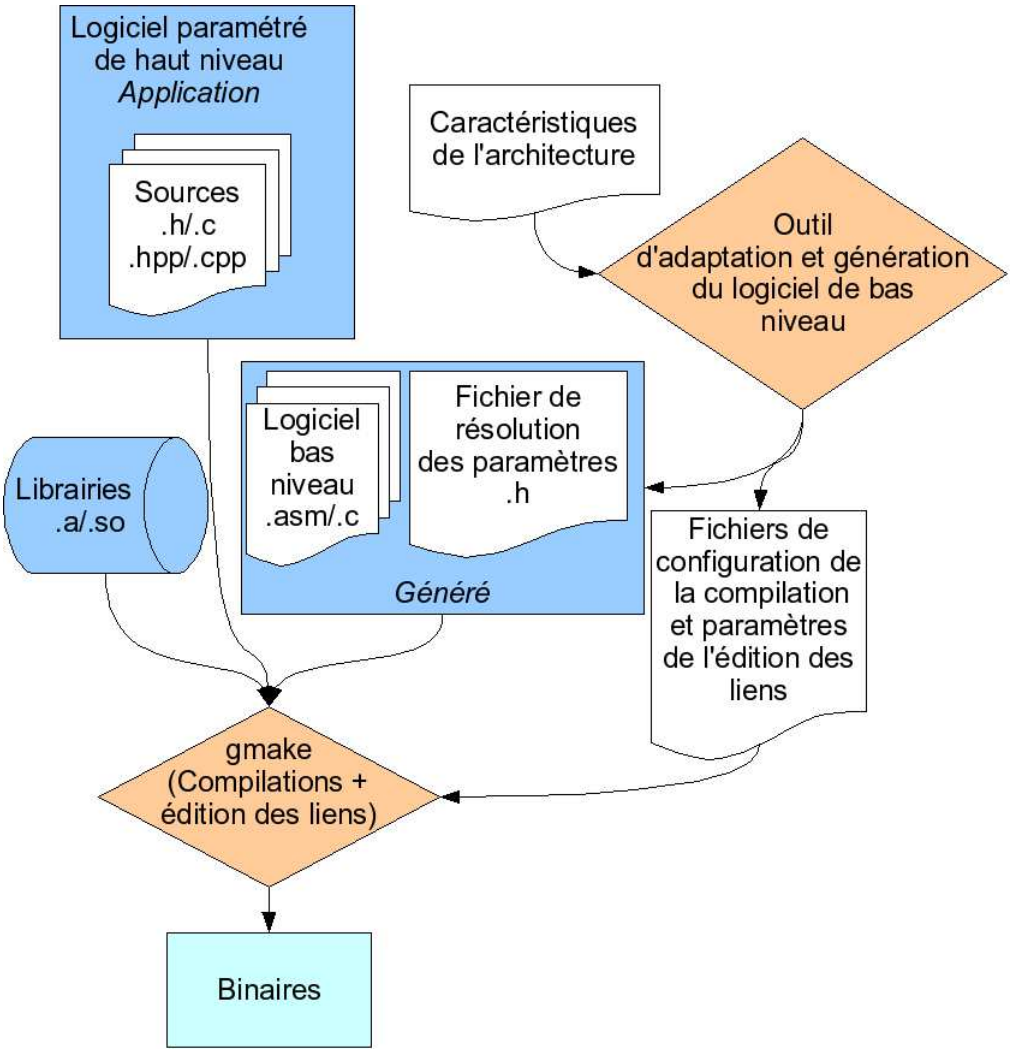

FIG. 2.3.2 – Mécanisme de compilation avec utilisation des fichiers générés

2.3.2 Relations entre l'architecture matérielle et les logiciels de bas niveau

L'architecture matérielle a de nombreux impacts sur le comportement des logiciels embarqués. Il est possible de classer ces impacts sur les logiciels selon les zones de l'architecture concernées, en deux parties, d'une part l'architecture locale, et de l'autre part l'architecture globale comprenant les réseaux d'interconnexions.

2.3.2.1 Relations entre l'architecture locale à un processeur et les logiciels de bas niveau

L'architecture locale peut avoir un impact sur l'ensemble des logiciels de la plateforme. La présence ou l'absence de composant induisent des erreurs dans les algorithmes les utilisant. Les erreurs n'engendrent pas forcément un arrêt immédiat du programme. La modification des adresses a un effet similaire, les accès en écriture n'ont pas d'effet et la lecture retourne soit des valeurs indéterminées soit zéro. Ces types d'erreurs peuvent ne devenir visible que plus tard dans l'algorithme.

Le choix du processeur peut avoir des effets plus drastiques. Il est évident qu'en cas de changement de processeur les binaires ne sont plus compatibles. Mais des changements de version ou des modifications de périphériques proches ont des effets plus complexes qui peuvent aussi entraîner des erreurs, valeurs corrompues ou instructions invalides principalement. Le changement de paramètres du processeur, tel que le boutisme ou la taille des bus, ont aussi un impact sur les communications avec les autres processeurs ou composants matériels ainsi que sur le chargement des données.

2.3.3 Le mécanisme de génération pour le logiciel de bas niveau

Cette section traite du mécanisme d'exemple de génération des fichiers d'initialisation et de gestion des codes sources embarqués. Le mécanisme est d'abord décrit. Ensuite un démonstrateur pour deux processeurs spécifiques est exposé.

2.3.3.1 Description du mécanisme et de l'utilisation des fichiers générés

Le début du mécanisme commence par l'exécution de la plateforme avec les paramètres de génération des caractéristiques de l'architecture. Ces caractéristiques sont triées dans un arbre de répertoire, d'abord par sous-système, puis pour chaque processeur.

Ensuite l'outil de génération analyse tous les fichiers de caractéristiques. Chaque fichier est ensuite transmis au générateur spécialisé selon le type de processeur associé.

2.3.3.2 Un démonstrateur de générateur pour l'adaptation sur *ARM7TDMI* ou *ARM926EJS*

Deux générateurs spécifiques ont été développés, un pour le *ARM7TDMI* et l'autre pour le *ARM926EJS*. Le second induit une complexité plus importante due à l'initialisation de la MMU et à la gestion des caches. Ces générateurs prennent comme seule entrée les fichiers de description de l'architecture. Le flot de génération avec les fichiers d'entrées et de sortie et inclue dans le flot de compilation figure 2.3.2.

Génération de l'assembleur d'initialisation

La génération de l'assembleur d'initialisation, exemple figure 2.3.3, crée quatre fichiers assembleurs, qui servent respectivement à la configuration du tas, des piles, des exceptions et l'initialisation.

Les fichiers d'initialisation du tas et des piles ne dépendent que de l'emplacement et de la taille de la mémoire de données. Une syntaxe fixe vers les fonctions de récupération est utilisée pour le fichier d'exception.

```
     ORR R3,R2,R1,lsl#20
     STR R3,[R0,R1,lsl#2]
     SUBS R1,R1,#1
     BPL TTB_init_loop
     LDR R0, =TransTableBase    ; Start address of translation table (TTB)
     MCR p15, 0, R0, c2, c0, 0  ; CP15 register 2
     LDR R1, =0x1               ;; ROM section modification
     LDR R3, =0x0               ; first entry translation table ( first 1M )
ROM
     LDR R2,[R0,R3,lsl#2]       ; Load value entry
     ORR R2,R2,#2_000000001000  ; 1M section cacheable ( bit 3)
     BIC R2,R2,#2_000110100000  ; Set ROM domain 2 (b8..5)
     STR R2,[R0,R3,lsl#2]       ; Store modified entry
     SUBS R3,R3,#1              ; Next entry (dec R3)
     SUBS R1,R1,#1              ; Next entry (dec R1)
     BPL ROM                    ; Until R1 = 0
     LDR R1, =0x5               ;; RAM section modification
     LDR R3, =0x7               ; Entry Correspond to RAM section
RAM
     LDR R2,[R0,R3,lsl#2]       ; Load value entry
     ORR R2,R2,#2_110000000000  ; Set User R/W  (b11..10)
     ORR R2,R2,#2_000000001100  ; 1M section cacheable(b3)&bufferable(b2)
     BIC R2,R2,#2_000111100000  ; Set RAM domain 0 (b8..5)
     STR R2,[R0,R3,lsl#2]       ; Store modified entry
     SUBS R3,R3,#1              ; Next entry (dec R3)
     SUBS R1,R1,#1              ; Next entry (dec R1)
     BPL RAM                    ; Until R1=0
```

FIG. 2.3.3 – Début de l'initialisation de la TLB et des caches, fichier généré

Pour le fichier d'initialisation, il commence par la mise à zéro des mémoires, puis pour les *ARM9*, le boutisme du processeur est déclaré, enfin suivent les configurations de la MMU et des caches. Le choix a été d'interdire tout accès mémoire, puis d'autoriser une zone pour chaque composant présent. Les caches sont initialisées sur le même principe. Le cache est déclaré en lecture pour la mémoire d'instructions, par contre pour la mémoire de données il est déclaré à la fois en lecture et en écriture. Pour les autres composants le cache est désactivé. Le générateur crée une boucle de configuration assembleur par composant esclave adressable. Ces composants esclaves et leurs adresses sont contenus dans le fichier de caractéristiques de l'architecture matérielle.

Le fichier finit par l'appel au programme principal *main*.

Générations des fichiers de compilation et d'édition des liens

La génération des fichiers de compilation, figure 2.3.4 et d'édition des liens figure 2.3.5 crée trois fichiers. Deux fichiers concernent la compilation et un fichier l'édition des liens.

```
ARMASM = $(ARMHOME)/$(arm_arch)/bin/armasm
ARMCC = $(ARMHOME)/$(arm_arch)/bin/armcc
ARMCPP = $(ARMHOME)/$(arm_arch)/bin/armcpp
ARMLINK = $(ARMHOME)/$(arm_arch)/bin/armlink
FROMELF = $(ARMHOME)/$(arm_arch)/bin/fromelf

Make.dep:
        $(ARMCC) $(DEPEND) $(INCDIR) $(ESWSRCS1) $(DEFINE) >
Make.dep
        $(ARMCPP) $(DEPEND) $(INCDIR) $(ESWSRCS2)
$(ESWSRCS3) \
$(DEFINE) >> Make.dep

$(TARGET).axf: $(ESWOBJS)
        $(ARMLINK) $(ESWOBJS) -info unused -scatter \
$LOCAL_PATH/config/esw.scf -entry 0x0 -o $(TARGET).axf

$(TARGET).bin: $(TARGET).axf
        $(FROMELF) $(TARGET).axf -bin -output $(TARGET).bin

.s.o:
        $(ARMASM) -$(PROC_ENDIAN) $(ASMFLAGS) $< -o $@

.c.o:
        $(ARMCC) -$(PROC_ENDIAN) $(CFLAGS) $(DEFINE) -c $<
```

FIG. 2.3.4 – Partie du fichier de compilation, fichier généré

Les deux fichiers de compilation correspondent à une compilation croisée et une compilation native. L'utilisation de ces deux compilations sera décrite à la section 1.3.2.4. Pour chaque fichier, le compilateur associé est appelé avec ses paramètres spécifiques. Pour la compilation croisée, des paramètres dépendent du boutisme et de la présence de coprocesseurs. Pour chaque compilation, les éléments correspondant sont paramétré :librairie croisée de la chaîne d'outils, librairies systèmes et interface vers les protocoles TLM.

```
ROM_LOAD 0x0
{
    ROM_EXEC 0x0
    {
        vectors.o (Vect, +First)
        * (+RO)
    }
    RAM 0x300000 0x500000
    {
        * (+RW,+ZI)
    }
    HEAP +0 UNINIT
    {
        heap.o (+ZI)
    }
    STACKS 0x800000 UNINIT
    {
        stack.o (+ZI)
    }
}
```

FIG. 2.3.5 – Un fichier d'édition des liens, fichier généré

Le fichier d'édition des liens est généré pour la chaîne d'outils ARM™. Il varie en fonction des tailles et des adresses des mémoires de codes et de données.

Fichier de résolution des paramètres de l'architecture matérielle

Le fichier de résolution des paramètres de l'architecture matérielle, visible figure 2.3.6, sert au programmeur pour adapter les algorithmes à l'architecture cible. Ce fichier est généré automatiquement avec les parties bas niveau du logiciel ainsi que les scripts de compilation et d'édition des liens. Il est spécifique à une architecture matérielle. Il contient des constantes, telles que les adresses de base des composants visibles par le processeur, le type de processeur, le nombre de sous-systèmes, le numéro du sous-système dans lequel est le processeur. Il inclue les fichiers de déclaration des registres des composants visibles.

```
#ifndef _HARDWARE_H_
#define _HARDWARE_H_
//The node
#define PROC_NUM 0
//Other nodes
#define ALL_NODES 11
//Proc caracteristics
#define PROC_NAME ARM
#define PROC_TYPE ARM926EJS_rev0
#define ENDIAN LITTLE
// - - BASE ADDRESS - - - -
#define DATA_MEMORY_BASE 0x00300000
#define CODE_MEMORY_BASE 0x00000000
#define TIMER_BASE 0x00200000
#define DMA_BASE 0x00220000
#define ITC_BASE 0x00100000
#define SCREEN_BASE 0x00240000
...
#include "dma_easy/include/dma_easy_registermap.h" ... #endif
```

FIG. 2.3.6 – Extrait d'un fichier de résolution, fichier généré

Ce fichier de paramètres peut être utilisé pour adapter l'algorithme à l'architecture matérielle pour laquelle le programme est compilé.

```
#ifdef DMA_BASE
if (!tlm_read( DMA_BASE+dma_easy_RUNNING))
        {
        tlm_write(DMA_BASE+dma_easy_ADDRESS_READ,src);
        tlm_write(DMA_BASE+dma_easy_ADDRES_WRITE,dest);
        tlm_write(DMA_BASE+dma_easy_NUMBER_DATA,nb);
        tlm_write(DMA_BASE+dma_easy_RUNNING,1);
        }
#else
else
#endif
        memcpy((unsigned long int *)dest,(unsigned long int *)src,nb<<2);
```

FIG. 2.3.7 – Résolution des paramètres dans le programme embarqué, modification manuelle

Dans l'exemple de source de programme embarqué, figure 2.3.7, l'adaptation concerne l'utilisation d'un DMA. La présence du DMA est d'abord testé via son adresse de base *DMA_BASE*. Si le programme est compilé sur une plateforme matérielle sans DMA, le transfert est effectué par le processeur. Si un DMA est présent, le programme teste si un transfert est en court. Si le DMA est occupé le processeur transfère lui même les données, sinon il configure le DMA pour qu'il effectue le transfert. On peut remarquer que cet exemple concerne un DMA très simple sans transfert multiple ni liste d'attente. De plus, dans cet exemple le transfert était dans le code source un point de synchronisation entre tâches, il ne pouvait donc être mis en attente.

2.3.4 Restrictions de l'outil d'adaptation et de génération du logiciel bas niveau développé

2.3.4.1 Générations dépendantes du processeur

La génération des codes de bas niveau et des fichiers de gestion des compilations et édition des liens est dépendante du processeur et de sa version. Il est donc obligatoire de créer un générateur spécifique pour chaque processeur avec la gestion des différentes versions.

2.3.4.2 Gestion des différents types d'initialisation

Les générateurs réalisés ne peuvent créer que les fichiers pour un type d'initialisation avec des choix prédéfinis. Il serait intéressant de rendre ces choix modifiables avec des paramètres supplémentaires.

Ces paramètres permettraient de rendre le générateur utilisable pour un plus grand nombre de logiciels. Il peut s'agir des choix d'initialisation des MMU ou du choix des différents modes d'utilisation du processeur à configurer.

2.3.4.3 Limitations dues à l'absence de système d'exploitation

La génération actuelle considère une initialisation pour un programme embarqué simple mono tâche. Ceci simplifie le code d'initialisation notamment par rapport aux modes d'utilisation du processeur.

On peut cependant remarquer que dans le cas d'un système d'exploitation, celui-ci gère habituellement lui-même l'initialisation du processeur. Il serait nécessaire de paramétrer certaines parties du code du système d'exploitation pour l'adapter automatiquement à la plateforme matérielle.

L'utilisation de l'approche semble envisageable pour **un système d'exploitation déjà porté sur le processeur cible**. L'adaptation ne concernerait que la plateforme *"board"*.

2.3.4.4 Perspectives sur l'adaptation des logiciels embarqués

Il serait particulièrement intéressant d'aller plus loin dans ce mécanisme avec l'utilisation d'outils externes[58][59] de parallélisation et de placement de tâches et de génération des communications. L'adaptation se ferait sur les sources parallélisés suivant le placement fournit.

2.4

Utilisation de l'adaptation avec le logiciel parallélisé d'encodage H264

Le générateur pour l'adaptation du logiciel a été utilisé sur un programme complexe, un algorithme d'encodage H264 parallèle de haut niveau [60]. Celui-ci a du être adapté sur une plateforme matérielle avec plus de 10 sous-systèmes de traitement hétérogènes, l'ensemble contenant plus de cinquante composants matériels. Durant toutes les expérimentations, il n'a pas été nécessaire d'écrire ou de modifier manuellement ni de code assembleur d'initialisation du processeur ni de scripts de compilation et d'édition des liens.

2.4.1 Description de l'algorithme utilisé

L'algorithme est un compresseur H264 qui prend en entrée un flux vidéo non compressé. Ce flux peut être dans différents formats allant du *qcif* à la haute définition *1920p*. Cet algorithme est décrit de manière succincte puis les mécanismes de synchronisation qu'il utilise sont détaillés.

2.4.1.1 Description de l'algorithme d'encodage H264 parallèle de haut niveau utilisé

L'algorithme d'encodage H264 parallèle de haut niveau a une première parallélisation sous la forme d'un pipeline avec une dizaine d'étages. Les communications entre ces étages sont complexes avec des rebouclages.

Ensuite les étages les plus conséquents sont découpés avec des calculs vectoriels sur des groupes de données ou des parties d'images.

Les synchronisations peuvent être autant inter que intra étages.

L'algorithme en lui même prend de nombreux paramètres, taille de l'image, nombre de *B-frame*, période des *I-frame*, paramètres de quantification ainsi que des paramètres pour les choix d'encodage et l'estimateur de mouvements.

L'ensemble des allocations mémoires est effectué avant l'encodage d'un flux. Le code source parallèle, environ 22000 lignes, est entièrement en langage C. Des fonctions sont aussi fournies en double, soit en C soit en assembleur INTEL *"SSE2"*. Elles ont été utilisées dans leur versions vectorielles (assembleur INTEL *"SSE2"*) pour des expérimentations en simulation native uniquement.

2.4.1.2 Les mécanismes de synchronisation

L'algorithme parallèle de haut niveau initial utilisait deux types de communications avec synchronisation. Des synchronisations par mécanisme de barrières pour gérer le flux des images et des communications par FIFOs pour la modification des donnés partagées.

Ces deux mécanismes utilisent des synchronisations par signaux et verrous (*Mutex*). Un mécanisme de signaux permet à une tâche d'attendre l'émission du signal par une autre tâche. L'émetteur du signal peut choisir de réveiller toutes les tâches en attentes *"Broadcast"* ou une seule. Un mécanisme de verrous permet de sérialiser de l'accès à des données partagées.

int *pthread_cond_init*(**pthread_cond_t** *cond, **pthread_condattr_t** *cond_attr);
int *pthread_cond_signal*(**pthread_cond_t** *cond);
int *pthread_cond_broadcast*(**pthread_cond_t** *cond);
int *pthread_cond_wait*(**pthread_cond_t** *cond, **pthread_mutex_t** *mutex);
int *pthread_mutex_init*(**pthread_mutex_t** *mutex, **const pthread_mutexattr_t** *mutexattr);
int *pthread_mutex_lock*(**pthread_mutex_t** *mutex);
int *pthread_mutex_trylock*(**pthread_mutex_t** *mutex);
int *pthread_mutex_unlock*(**pthread_mutex_t** *mutex);

FIG. 2.4.1 – Les primitives de synchronisation utilisé par le logiciel parallèle d'encodage H264 de haut niveau

Toutes les primitives de synchronisation sont basées sur les primitives de la bibliothèque *pthread*.

2.4.2 L'architecture multi-processeurs hétérogène à mémoire partagée

Il a été nécessaire d'abord de définir et de construire une architecture multiprocesseur, puis celle-ci a été modélisée à partir de sous-systèmes flexibles à travers le générateur de plateformes exécutables, enfin l'algorithme et particulièrement les synchronisations ont du être portés sur l'architecture. Ce travail réalisé en commun par le programmeur qui avait parallélisé l'algorithme et par l'auteur a été accompli en moins d'une semaine.

L'architecture comporte 11 sous-systèmes de traitement:

- 7 sous-systèmes "processeur :ARM7, DMA, ROM, RAM et ITC
- 2 sous-systèmes "processeur :ARM7, DMA, ROM, RAM, TIMER, ITC, contrôleur d'écran
- 1 sous-système de communication
- 1 sous-système mémoire

2.4.2.1 Portage du mécanisme des pthread sur l'architecture multi-processeurs hétérogène à mémoire partagée

Un sous ensemble des mécanismes de synchronisation entre pthread à dû être écrit pour l'architecture.

Ces mécanismes ont concernés:

- les verrous pour rendre unique l'accès à des zones de code sensible;
- les signaux pour l'attente et le réveil lors de demande d'accès à des ressources occupées;
- les barrières pour la synchronisation globale entre les tâches, soit l'arrivée de l'ensemble à un point de code précis.

Le mécanisme des verrous et celui des barrières s'appuient sur celui des signaux.

2.4.2.2 Restriction sur les données partagées et la répartition des tâches

Il a été nécessaire de choisir la répartition des données entre les mémoires locales d'accès rapide et les mémoires globales accessibles à plusieurs tâches. De plus il est nécessaire de rajouter une mise en accord entre les différentes tâches sur l'utilisation des zones de mémoires partagées.

Les structures de données ont été séparées manuellement en fonction de

leurs accesseurs. Ces structures sont ensuite à répartir au mieux sur les différentes mémoires.

2.4.3 Utilisation de la génération de la partie logicielle bas niveau et des scripts de compilation et d'édition des liens avec le logiciel d'encodage H264

La plateforme matérielle TLM exécutable a déjà été créée, et l'extraction des caractéristiques de l'architecture effectuée.

Des dossiers sont créés contenant les fichiers d'initialisation et les scripts pour la compilation. Il faut créer dans cette structure de fichier les liens vers les fichiers sources des programmes embarqués. Enfin il faut exécuter un script de compilation qui exécute le script correspondant à chaque processeur. Les binaires sont créés. Il ne reste plus qu'à lancer la simulation.

2.4.3.1 Les fichiers assembleur d'initialisation

Les fichiers de script et d'initialisation ont été générés pour des processeurs ARM7 et pour des processeurs ARM9 grands et petits boutistes. Le code généré assembleur généré varie entre 400 et 800 lignes de codes sources. Ces lignes sont réparties entre les quatre fichiers.

Ces fichiers ont été validés par compilation et lien avec l'ensemble des programmes exécutés.

2.4.3.2 Les scripts de compilation et d'édition des liens

Les scripts de compilation sont créés pour la compilation native et pour la compilation croisée vers le processeur cible. Les deux scripts ont servi sans modifications manuelles pour compiler le code parallélisé du H264.

Par contre la compilation croisée a nécessité des modifications de syntaxe dans le programme H264. Le compilateur croisé est en effet moins permissif que celui qui avait servi au développement initial.

2.4.3.3 Génération automatique lors de modifications de l'architecture matérielle

Ces fichiers générés ont servi lors de la mise en place des programmes sur l'architecture. Après avoir validé la fonctionnalité des programmes, des

explorations d'architectures ont été faites.

Lors de chaque exploration le mécanisme de génération a été relancé avec succès. De même pour l'exécution des scripts de compilation et du fichier de paramètres de l'édition des liens.

2.4.4 Exemple de modification de l'architecture avec variation du nombre de processeurs

Cette section décrit les étapes d'une exploration effectuée sur l'architecture matérielle. La génération de plateformes exécutables à partir de sous-systèmes de traitement flexibles, l'extraction des caractéristiques de l'architecture et la génération des logiciels de bas niveau sont utilisées.

2.4.4.1 La modification d'architecture choisie

L'architecture initiale comporte 11 sous-systèmes de traitement:
- 7 sous-systèmes processeu :ARM7, DMA, ROM, RAM et ITC
- 2 sous-systèmes processeu :ARM7, DMA, ROM, RAM, TIMER, ITC, contrôleur d'écran
- 1 sous-système de communication
- 1 sous-système mémoire

Le but de cette première exploration a été de tester les performances dans le cas d'un parallélisme plus important. La nouvelle architecture visée se compose de 13 sous-systèmes de traitement:
- 3 sous-systèmes processeu :*ARM9, ROM, RAM et ITC*
- 6 sous-systèmes processeu :*ARM7, ROM, RAM et ITC*
- 2 sous-systèmes processeu :*ARM9, DMA, ROM, RAM, TIMER, ITC, contrôleur d'écran*
- 1 sous-système de communication
- 1 sous-système mémoire

Les mémoires locales ont des tailles inférieures. Des composants DMA sont supprimés avec un changement d'adresse des autre composants dont les mémoires.

Le logiciel est paramétré suivant le nombre de processeurs. Une table des placements des tâches suivant le nombre de processeurs avait été défini. Dans ce cas le passage à 13 sous-systèmes augmente le nombre d'images de références utilisé pour l'encodage.

2.4.4.2 Description de la séquence d'opérations à effectuer pour l'adaptation des logiciels embarqués

La séquence d'opérations se découpe en deux étapes principale :la modification du modèle de plateforme matérielle exécutable, puis la régénération et recompilation des logiciels.

Modification de la plateforme et extraction des nouvelles caractéristiques architecturales

La modification de la plateforme demande l'ajout de 2 sous-systèmes et de leur connexion au sous-système de calcul. Le fichier de paramètres des sous-systèmes de traitement ARM7 sans TIMER doit être copié et modifié pour intégrer un ARM9. Ensuite, tous les fichiers de configuration doivent être modifiés pour diminuer la taille des mémoires. Un dernier fichier de configuration doit être altéré pour changer le type de processeur.

Enfin, il suffit de lancer le générateur de plateforme exécutable.

Modifications logicielles, placements et paramètres

L'exploration consistant à augmenter le nombre de processeurs avait été prévue lors du portage du logiciel. A cet effet, le code est paramétré pour que la compilation configure le programme lancé sur chaque processeur suivant le nombre de processeurs présents dans la plateforme. Sinon, le placement aurait du être modifié manuellement ou nécessiter un outil utilisant les caractéristiques de l'architecture.

Un seul point n'a pas été paramétré par manque de temps. Il est nécessaire de modifier une variable de configuration de l'algorithme pour indiquer le nouveau nombre d'images de référence.

2.4.5 Proposition d'implémentation du mécanisme d'extraction de paramètres de l'architecture dans le protocole transactionnel de STMicroelectronics

Le mécanisme d'extraction de paramètres de l'architecture dans le protocole TLM *"TAC"* de STMicroelectronics^{TM}est une proposition qui n'a pas encore pu être implémentée. Ce mécanisme serait inclut directement dans le

protocole et ne dépendrait pas des sous-systèmes. Il pourrait donc être utilisé sur toutes les plateformes exécutables TLM.

2.4.5.1 Description de l'implémentation du mécanisme d'extraction de paramètres de l'architecture dans le protocole transactionnel de STMicroelectronics

Le mécanisme d'extraction est lancé par la plateforme. Celle-ci appelle la fonction d'extraction de chaque composant maître de la plateforme. Les composants maîtres appellent les fonctions des composants esclaves qui sont connectés sur chaque port. Les composants esclaves renvoient leurs informations, et celles des composants connectés à travers eux, si ils font office de réseau.

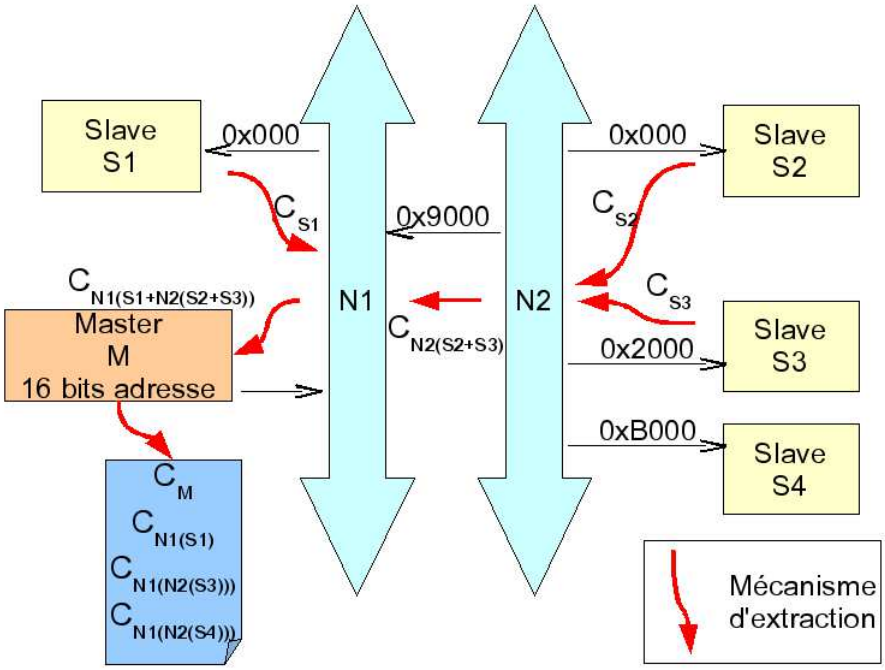

FIG. 2.4.2 – Le mécanisme d'extraction des caractéristiques de l'architecture dans le protocole TLM

Dans le schéma, le composant maître demande au réseau *N1* les composants accessibles sur la plage 0×0 à $0\times FFFF$. Le réseau demande les informations au composant *S1* et au réseau *N2* sur la plage 0×0000 à $0\times6FFF$. Le réseau *N2* demande les informations à *S2* et *S3*, applique ses modifications

puis les renvoie. Le réseau *N1* applique ses modifications aux informations reçues et les renvoie au composant maître. Enfin le composant maître écrit dans un fichier les informations reçues et celles le concernant.

2.4.5.2 Proposition d'implémentation pour le protocole au niveau transactionnel du consortium OSCI

Une implémentation pour le protocole TLM du consortium OSCI de l'extraction des caractéristiques de l'architecture matérielle pour les logiciels embarquées peut être réalisée. Le protocole définit quatre fonctions d'accès au port *read*, *write*, *put* et *get*. Une solution d'implémentation serait l'ajout d'une primitive prenant en paramètre une plage d'adresses et retournant une liste de structures de caractéristiques de composants esclaves.

Dans les composants de communication cette primitive serait implémentée comme une communication vers tous *"broadcast"* qui appellerait la fonction sur tous les composants destinataires. Les composants maîtres ou esclaves retourneraient leurs informations via la même primitive. Sur le retour le composant de communication appliquerait ses modifications, tel que par exemple un décalage des adresses.

L'interface de composants maîtres implémenterait une fonction de récupération des informations, sur lui-même et appellerait celles des composants sur lesquels il est connecté. Et enfin écrirait ces informations dans un fichier.

2.4.5.3 Ajout de caractéristiques de performances

Une autre possibilité d'évolution de ce mécanisme serait l'ajout de caractéristiques de performance, temps d'accès, latence. Cependant ces informations sont souvent dynamiques. Une étude serait donc nécessaire pour évaluer la meilleure façon d'ajouter ces fonctionnalités, soit par calcul avec des algorithmes, soit de manière analytique avec des temps moyen.

2.4.6 Impact de ces travaux sur les standards utilisés

L'extraction et l'utilisation des caractéristiques de l'architecture, section 2.3, dans cette thèse a permis d'énumérer et de valider les types de données nécessaires à une étude d'architecture assistée.

Ce résultat est utilisé dans des propositions quant aux évolutions du format SPIRIT. Le but est de pourvoir extraire directement les caractéristiques de la description standardisée de l'architecture finale, après la résolution des éventuels paramètres de flexibilité.

2.5

Conclusion sur le portage rapide des logiciels embarqués sur une architecture multi-processeurs hétérogène

Le but de ce chapitre est de décrire une méthode pour réaliser un portage rapide des logiciels embarqués sur une architecture multi-processeurs hétérogène et les outils associés. En premier lieu un mécanisme d'extraction des caractéristiques de l'architecture a été proposé, avec une description de ces caractéristiques. Ensuite ces caractéristiques ont été utilisées avec le développement d'un outil démonstrateur pour générer la couche bas niveau du logiciel ainsi que les fichiers nécessaire à la compilation. Enfin cet outil a été utilisé avec succès sur un logiciel de haut niveau d'encodage H264.

L'ensemble des codes sources de l'application ont pu être portés sur l'architecture. La génération du code source de bas niveau a permis de réaliser ce travail sans écrire ni modifier de code assembleur.

Le programme a pu être paramétré, à la compilation ou l'exécution, pour les explorations d'architecture souhaitées. Les explorations ont donc pu se faire sans avoir à réadapter l'application. L'étape de paramétrage de l'application a de plus entraîné une répartition de code plus propre.

Par contre de nombreuses limitations sont apparues lors de l'utilisation du préprocesseur pour paramétrer les codes sources. L'utilisation de scripts insérés dans les codes sources devraient être expérimentée pour permettre plus de flexibilité.

Chapitre 3

Méthode d'exploration d'architecture multi-processeurs intégrant logiciels et matériels

L'exploration d'architecture [61] commence par deux étapes, la construction de l'architecture, et le portage du logiciel.

Ensuite il faut simuler la plateforme matérielle avec l'ensemble des logiciels embarqués. Cette simulation permet des mesures de performances. L'architecture est évaluée à partir de ces mesures. Si les performances ne sont pas suffisantes l'architecture et les logiciels embarqués sont modifiés et un nouveau cycle de l'exploration d'architecture recommence, avec la modification de l'architecture matérielle et logicielle, le portage des logiciels, l'intégration, la simulation et l'évaluation.

Dans ce chapitre est proposé une solution pour une exploration d'architecture multi-processeurs intégrant logiciels et matériels. Une technique de simulation multi-niveau sera validée par une expérience sur une plateforme complète section 3.2 suivit d'une exploration d'architecture 3.3. Cette expérience reprend l'ensemble des flots et outils décrit dans cette thèse.

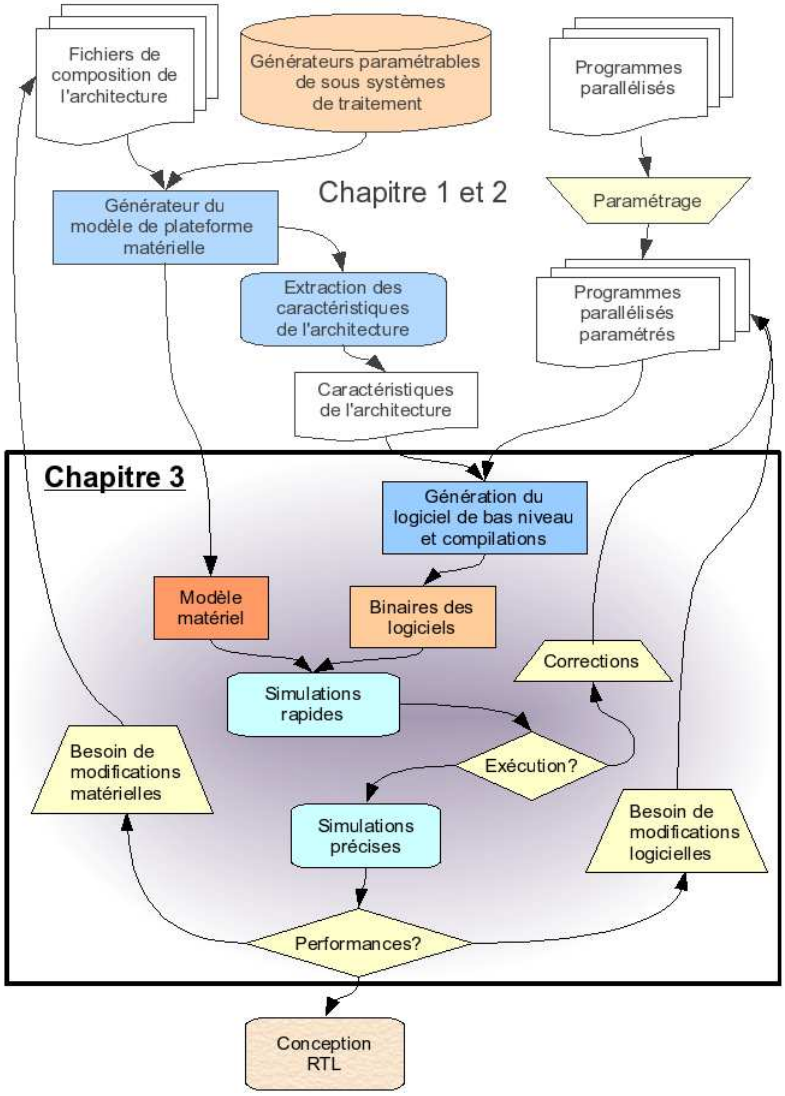

FIG. 4 – Mise en place de la simulation multi-niveau

Le flot d'exploration, figure 4, proposé dans ce chapitre utilise deux entrées, un modèle TLM exécutable d'une plateforme matérielle, des binaires des logiciels à deux niveau d'abstraction, soit pour simulateur de processeur, soit pour simulation native. La plateforme matérielle est obtenue via le flot décrit au chapitre 1, les binaires des logiciels le sont via le flot du chapitre 2. Les modifications de l'outil d'adaptation et de génération de la partie logicielle de bas niveau pour obtenir des binaires à plusieurs niveaux d'abstraction est décrite dans ce chapitre, section 1.3.

3.1

Description de l'ensemble du flot et interactions logiciel matériel

Le flot de conception de HMPSoC proposé dans cette thèse prend quatre entrées. La première entrée est *la bibliothèque de générateurs paramétrables de sous-systèmes de traitement* qui est utilisée par le générateur de plateformes matérielles exécutables. La seconde entrée est *les fichiers de compositions de l'architecture* qui contiennent les sous-systèmes de traitement à générer avec leurs paramètres ainsi que les interconnexions. La troisième entrée est *les programmes parallélisés*, soit l'ensemble des codes sources des applications embarquées. La quatrième entrée est *une série de paramètres aux outils logiciels* qui sont rentrés manuellement pour la génération des binaires des programmes embarqués, par exemple le placement des programmes sur l'architecture.

La première sortie des outils est la plateforme TLM exécutable matérielle, à partir de laquelle il est possible d'extraire les caractéristiques de l'architecture pour les logiciels embarqués. Ces caractéristiques sont utilisées par les outils logiciels, notamment pour la génération du code de bas niveau et l'adaptation. Ces outils permettent aussi la compilation et l'édition des liens. Ils fournissent donc en sortie les binaires des programmes embarqués.

A partir de cette étape, la plateforme logicielle et matérielle exécutable est disponible pour une première tentative de simulation.

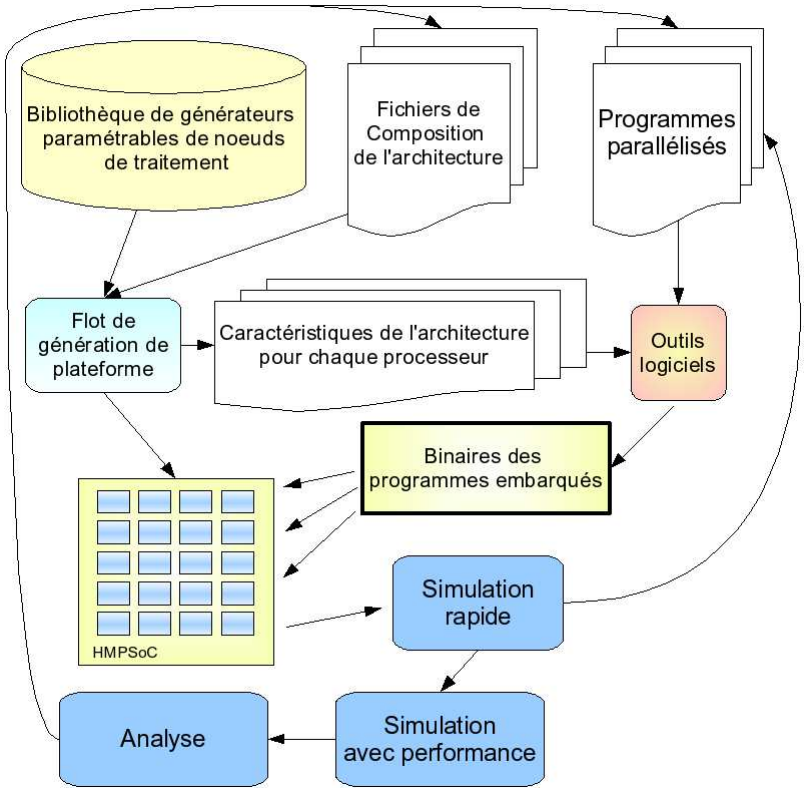

FIG. 3.1.1 – Le flot d'exploration d'architecture logicielle et matérielle proposé

Sur ce diagramme, figure 3.1.1, on peut voir différentes interactions entre les logiciels et l'architecture. D'abord, la partie d'adéquation et de portage autour des "outils logiciels" utilisant les caractéristiques de l'architecture qui définie aussi le placement et les communications inter tâches. Ensuite le chargement des programmes compilés, soit les binaires, dans les mémoires de l'architecture ou le chargement par les simulateurs de processeurs. Enfin la simulation conjointe entre les logiciels et les matériels qui nécessitent l'utilisation de composants spécifiques de simulation de processeurs, ISS ou autres.

3.2

Travaux réalisés sur l'architecture multi-processeurs hétérogène à mémoire partagée et le logiciel H264 pour la simulation

Ces travaux se divisent en deux parties, la conception de la plateforme et le portage des logiciels embarqués. Ensuite les simulations et les explorations d'architectures peuvent commencer.

L'ensemble du flot proposé est utilisé. D'abord l'outil de génération pour crée le modèle exécutable de l'architecture. L'algorithme est porté avec l'utilisation des caractéristiques de l'architecture. Le modèle exécutable de plateforme TLM logiciel matériel est utilisé en simulation.

3.2.1 Utilisation de l'outil de génération pour créer le modèle exécutable de l'architecture

Pour utiliser l'outil de génération pour créer le modèle exécutable de l'architecture il faut d'abord décrire la plateforme. Celle-ci à été faite suivant les demandes d'un architecte, elle est semblable à la plateforme de la section 2.4.2. L'architecture originale demandait plusieurs types de sous-systèmes contenant différents processeurs dans des sous-systèmes de calcul avec DSP ou orientés transfert de données ou encore avec un processeur spécialisé dans les flux. Pour des questions de temps et de disponibilité des simulateurs de processeurs il a été décidé de n'utiliser que le sous-système

processeur flexible existant en les spécialisant suivant les fonctionnalités nécessaires.

La création du modèle exécutable a demandé le développement d'un nouveau sous-système paramétrable. Celui-ci contient une hiérarchie de mémoires.

Enfin les fichiers de description de l'architecture et de paramètres des sous-systèmes flexibles ont été écrits. Le modèle est créé par l'outil de génération à partir des fichiers de description.

L'étape de développement proprement dite, hors spécifications, à pris quatre jours. Ce travail effectué, l'extraction des caractéristiques à pu être lancée pour commencer le portage des algorithmes logiciels.

3.2.2 Le logiciel parallélisé de l'application H264

Le logiciel parallélisé de l'application H264 est un encodeur vidéo. L'algorithme est partagé en 17 taches distinctes. Une tache concerne l'initialisation du matériel. Huit taches forment (deux à deux) un parallélisme sur les donnée :Le flot de données peut être divisé en quatre, chaque partie et utilisé par deux taches puis regroupée. Le reste des taches forme un pipeline mais avec des synchronisations vers d'autres taches.

Les données sont transférées entre les taches via des structures de données de type FIFO incluant des verrous. Les messages de synchronisation utilisent des primitives de type barrières et signaux.

3.2.3 Validation, exécution de la plateforme matérielle et logicielle

Cette validation se sépare en deux étapes, lors des simulations natives puis sur ISS.

Les simulations natives ont permis la correction de multiples erreurs:

- oublis de zones mémoires partagées;
- erreurs dans les mécanismes de synchronisation;
- initialisation des composants matériels.

Puis les simulations avec ISS ont pu commencer. L'utilisation de code de bas niveau généré et juste par construction a permis d'éviter de nombreuses erreurs complexes. Deux erreurs résiduelles ont tout de même du être corrigées:

- La première est due à la compilation. En effet, le compilateur sur la machine "simulant" considère les variables de type *char* comme signées.

Le compilateur croisé les considère comme non signés. Les tests de comparaison d'égalité avec la valeur -1 donnent des résultats différents, ce qui change le comportement du programme. Ceci aurait pu être évité en utilisant des règles de codage plus stricte.

– La seconde s'est révélée être une erreur d'alignement dans la primitive d'allocation dynamique dans les mémoires partagées. Cette fonction avait due être écrite lors du portage multi-niveaux à la section 1.3.3.3. Cette erreur aurait été trouvée avec une simulation native avec redirection totale.

3.2.3.1 Résultats d'exécutions

Les résultats d'exécutions sont de plusieurs ordres, validation fonctionnelle de l'ensemble et visualisation du résultat premières estimations en simulation native puis les mesures de performances avec l'utilisation d'ISS.

La plateforme utilisée permet en natif et avec ISS de visualiser les flux vidéos grâce à des contrôleurs d'écran. Dans la plateforme finale les flux ne sont pas affichés, les périphériques sont donc éliminés lors des mesures de performances. Le logiciel est paramétré vis-à-vis de leurs présences.

FIG. 3.2.1 – Visualisation[1] lors de l'exécution de la plateforme logicielle et matérielle

[1] A gauche vidéo non compressée, à droite vidéo encodée-décodée

La spécification de l'encodage H264 laisse de nombreuses variables de configuration au choix du concepteur et selon ses possibilités de calcul. Il est donc particulièrement important de pouvoir visualiser les pertes amenées par l'encodage. Dans la figure 3.2.1 la qualité de l'encodage rend la différence difficilement visible, mais par exemple des volutes sont visibles sur l'écran central d'arrière plan sur la vidéo de gauche, elles sont lissées et ne sont donc plus visibles sur celle de droite.

3.2.3.2 Possibilités d'évaluation du logiciel embarqué en simulation native et limitations

Les possibilités d'évaluation du logiciel embarqué en simulation native et les limitations sont de deux ordres, la recherche d'erreurs, les premières évaluations de performances.

Au niveau de la recherche d'erreurs, il est plus simple de lister les erreurs qui ne peuvent pas être trouvées:

- Les erreurs dans les codes de bas niveau, ou assembleur. En effet, ils ne sont pas interprétés mais émulé par un code source différent.
- Des erreurs de temporisation, par exemple des procédures d'interruptions trop longues qui ne permettent pas au programme principal de s'exécuter.
- Des erreurs de synchronisation, notamment des accès multiples non protégés à des variables partagées.
- Des erreurs dues à des interruptions hors de points de synchronisation.

Il est aussi possible de réaliser de premières évaluations de performances en simulation native, mais avec des limitations très importantes. Lors de simulations natives avec redirection des communications, voir section 1.2.2.3, il est possible de réaliser un profil d'exécution des codes sources sur les différents processeurs natifs. Cette manipulation permet d'estimer si la répartition entre des processeurs identiques est correcte ou non. Dans l'expérimentation H264 la répartition s'est avérée mauvaise avec des processeurs en sous charge. L'instrumentation est réalisée via le logiciel de recherche d'erreu : *Valgrind* [62], et l'interprétation des résultats via le logiciel de visualisatio :*KCachegrind* [63]. Un exemple de visualisation des résultats de profilage via *KCachegrind* figure 3.2.2.

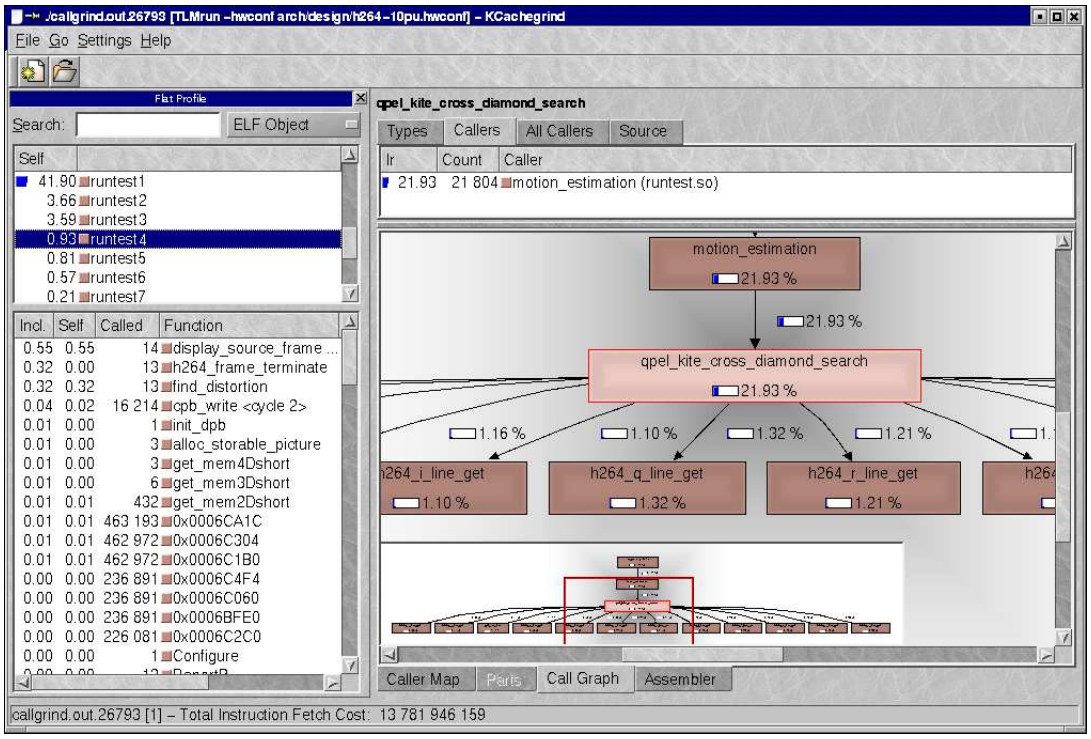

FIG. 3.2.2 – Profilage de code en simulation native

En exécutant une simulation native avec redirection de toutes les communications vers la simulation, des mesures de trafics de données auraient été possibles. Dans notre cas, il est nécessaire d'attendre les simulations avec ISS.

3.2.3.3 Possibilités d'évaluation du logiciel embarqué avec simulateur d'instructions et limitations

Les possibilités de mesures de performances avec des simulations précises de processeurs (pour l'instant des ISS) sont nombreuses. Les deux principales sont la mesure des trafics sur les réseaux et le nombre d'instructions exécutées sur chaque processeur pour effectuer une fonction précise.

Pour la mesure des trafics du réseau, un mécanisme d'enregistrement des transactions est mis en œuvre. Toutes les données concernant une transaction, moment, durée, adresse, données, types, connexion sont enregistrés dans une base de données. Il est ensuite possible de visualiser, figure 3.2.3, les transactions pour chaque connexion de l'architecture.

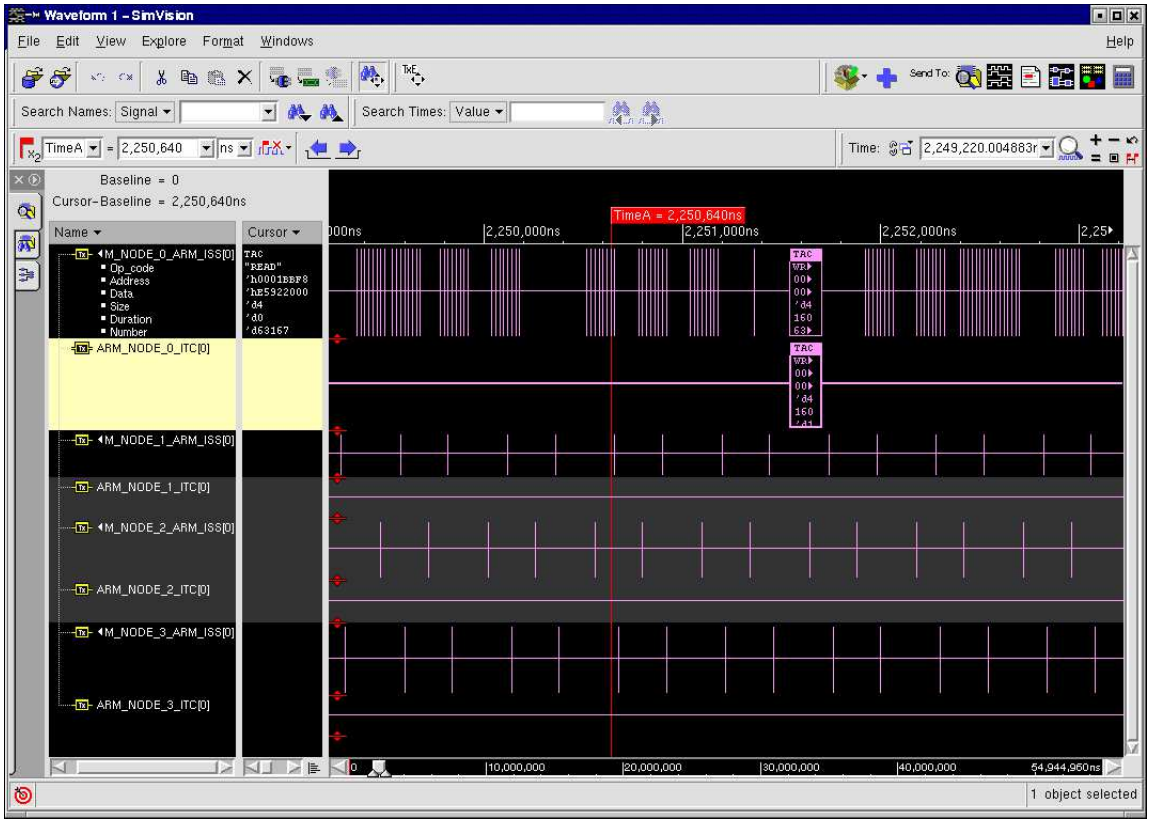

FIG. 3.2.3 – Visualisation[1] des traces lors d'une simulation avec ISS

Des outils supplémentaires permettent aux concepteurs d'architectures de faire des tris, des calculs ou statistiques sur ces transactions.

Le développement de modèle TLM avec temporisations étant en cours, les modèles utilisés pour cette expérience, à part le processeur, n'ont pas de précisions temporelles. Au mieux il est possible de configurer des temps d'accès moyens aux périphériques et aux réseaux.

[1] Visualisation via l'outil *SimVision* de Cadence™[64]

3.3

Exemple d'exploration architecturale avec variation du nombre de processeurs

L'étude de cas vise une exploration d'architecture qui initialement comportait 11 sous-systèmes de traitement dont 9 avec processeurs, c'est celle qui a été présentée dans la section 2.4.4.1, soit:

- 7 sous-systèmes processeu :*ARM7, DMA, ROM, RAM et ITC*
- 2 sous-systèmes processeu :*ARM7, DMA, ROM, RAM, TIMER, ITC, contrôleur d'écran*
- 1 sous-système de communication
- 1 sous-système mémoire

Suite à des mesures de performances et des évaluations cette architecture a du être modifiée. L'ensemble des outils et méthodes proposés sont utilisés.

3.3.1 L'exploration d'architecture matérielle choisie

L'étude des mesures de performances réalisées sur l'architecture de départ a permis plusieurs observations:

- Surcharge de travail sur un processeur ARM7, celui-ci va donc être transformé en ARM9 et des caches lui seront ajoutés.
- Certains sous-systèmes communiquent peu avec la mémoire partagée, les composants DMA seront retirés.
- Des tailles mémoires ont été sur évaluées, elles seront réduites.

La nouvelle architecture choisie consiste en:

- 3 sous-systèmes processeu :*ARM7, DMA, ROM, RAM et ITC*

- 4 sous-systèmes processeu :*ARM7, ROM, RAM et ITC*
- 1 sous-systèmes processeu :*ARM7, DMA, ROM, RAM, TIMER, ITC*
- 1 sous-systèmes processeu :*ARM9, DMA, ROM, RAM, TIMER, ITC*
- 1 sous-système de communication
- 1 sous-système mémoire

3.3.2 Description des modifications matérielles de la plateforme lors d'une exploration d'architecture

Trois modifications de la plateforme matérielle sont effectuées lors de l'exploration. D'abord, deux types de sous-systèmes processeurs, soit 4 sous-systèmes, sont peu modifiés, changement de la taille des mémoires et de la carte des adresses. Ensuite un nouveau sous-système dérivé du précédent est construit sans le composant DMA et avec des modifications des mémoires. Enfin un nouveau sous-système est créé avec la transformation du ARM7 en ARM9 et l'ajout de caches d'instructions et de données.

Donc deux fichiers de paramètres des sous-systèmes doivent être créés. Les deux fichiers existant doivent être modifiés. Le sous-système de communication et le sous-système mémoire restent identiques.

3.3.3 Impacts des modification de l'architecture matérielle sur les logiciels embarqués

Les impacts sur les logiciels embarqués peuvent se classer en deux catégories.

La première concerne les impacts sur le logiciel quelque soit le niveau de simulation. Ces modifications concernent la fonctionnalité des algorithmes.

La seconde inclue les impacts sur les simulations avec ISS, ce sont uniquement les modifications sur le code de bas niveau.

Les modifications fonctionnelles sont la suppression des composants DMA. Les zones de codes concernées avaient été paramétrées selon la présence ou non de ce composant. S'il est absent, le processeur transfère lui-même les données.

Les modifications du code de bas niveau concerne le sous-système dans lequel le processeur est modifié. L'assembleur d'initialisation est différent

et doit de plus initialiser le composant interne de MMU et les caches. Cette partie est entièrement générée. Aucune modification manuelle du logiciel n'est nécessaire pour cette exploration d'architecture.

3.4

Conclusion sur la méthode d'exploration d'architecture incluant logiciel et matériel

Ce chapitre décrit une méthode d'exploration d'architecture multi-processeurs intégrant logiciels et matériels. En premier lieu est décrite une technique pour mettre en place une simulation multi-niveaux des logiciels embarqués. Un exemple d'une telle simulation est développé sur une plateforme multiprocesseur hétérogène embarquant un logiciel d'encodage H264. Finalement une expérience d'exploration architecturale utilisant tous les outils et méthodes décrits dans cette thèse est réalisée en exploitant cet exemple.

Cette expérience montre que l'utilisation du flot proposé permet une itération rapide de différentes étape :modification de l'architecture, adaptation du logiciel, validation du fonctionnement et retour aux mesures de performances rapide.

La première étape de l'expérience a été la création de l'architecture et le portage du logiciel. La création de l'architecture a demandé le développement d'un nouveau générateur de sous-système de type sous-système mémoire. L'adaptation du logiciel a demander la modification de l'interface bas niveau du logiciel parallèle existant ainsi que l'ajout de paramètres. La simulation multi niveau des logiciels a été utilisée pour accélérer la mise en place de cette plateforme. L'exploration d'architecture a été effectuée. Au cours de cette exploration, avec compilations et simulation conjointe du logiciel et du matériel, aucune erreur n'est apparue malgré les nombreuses modifications. Ceci semble principalement du au fait que la majorité des parties modifiées, autant sur le matériel que sur les logiciels, ont été générées automatiquement.

Conclusion

Cette étude propose une méthode de conception pour des HMPSoCs. L'approche proposée dans cette thèse permet notamment des explorations d'architectures. Cette méthode se déroule selon un flot itératif en trois étapes. La première étape concerne le matériel avec la modification et le développement rapide du modèle exécutable de l'architecture. La seconde étape vise le portage rapide des logiciels sur la nouvelle architecture. La troisième étape est l'exploration d'architecture logicielle et matérielle avec simulations multi-niveaux des logiciels embarqués.

Un outil a été développé et est proposé pour la création et les modifications rapides d'architecture HMPSoC. Il utilise des sous-systèmes de traitement paramétrables et permet la composition automatique de modèles d'architecture HMPSoC. Il a été validé lors de créations et de modifications de plateformes matérielles. Ces temps de création et de modification vont de quelques minutes à quelques heures, alors que auparavant cela nécessitait de quelques jours à quelques semaines.

Pour rendre le logiciel adaptable, une méthode a été développée qui inclue les trois techniques suivantes: la modification manuelle du logiciel applicatif pour le paramétrer, l'extraction automatique de l'architecture des valeurs des paramètres, la génération des sources de bas niveau. Cette méthodologie permet le portage rapide, voire automatique, du logiciel sur l'architecture initiale et lors de l'exploration. Elle a été validée sur un logiciel d'encodage

H264 parallélisé. Celui-ci a été porté sur l'architecture, paramétré, et les parties de bas niveau (codes assembleurs) ont été générées.

Enfin une méthode pour réaliser le flot d'exploration d'architectures intégrant logiciel et matériel est exposée. Dans ce flot une méthode d'écriture des programmes permet d'effectuer des simulations multi-niveaux des processeurs. Les simulations de haut niveau servent pour exécuter rapidement les logiciels embarqués. Il est alors aisé d'effectuer les recherches d'erreurs fonctionnelles, dont l'intégration logicielle sur le matériel. Puis les simulations précises en mode bas niveau (ISS) avec mesures de performances sont effectuées. Suivant les évaluations des résultats, l'architecture et les logiciels sont modifiés, en s'appuyant sur le flot, et le cycle peut reprendre. Ce cycle a été mis en œuvre avec succès sur une application d'encodage H26 :Simulations conjointes (matériel et logiciel) sans erreur, cycle de décision des modifications matérielle à la simulation conjointe de l'ordre de la demi journée au lieu des nombreux jours que nécessitaient précédemment la mise au point (notamment les codes de bas niveaux dont assembleur)

Ce flot a également été démontré sur une architecture comprenant 1024 sous-systèmes ayant chacun un processeur, avec leurs logiciels bas niveau générés, une application simple de test et divers composants. Cette expérience a été réalisée avec succès. Bien que n'apportant pas de concept supplémentaire (et donc n'étant pas décrite dans cette thèse), cette expérimentation contribue à montrer que ce flot de conception supporte le passage à l'échelle avec des HMPSoCs massivement parallèle.

Perspectives

Le travail de cette thèse ouvre des perspectives concernant les trois points suivant :modélisation du matériel, adaptation des logiciels et flot de conception global de HMPSoC.

En parallèle de cette thèse, pour l'aspect matériel, un standard de description d'architectures et de composants (SPIRIT) a été créé par l'industrie et est en cours de maturation. Il serait intéressant de modifier les générateurs de sous-systèmes pour en faire des générateurs compatibles SPIRI :la sortie serait la description SPIRIT de l'architecture du sous-système. Par ailleurs au sein de l'équipe des outils de génération de modèle TLM d'architecture matérielle SoC à partir de descriptions SPIRIT sont en cours de développement. Ce mécanisme crée une étape intermédiaire dans le flot de cette thèse et rallonge donc le temps de création. Cependant il permettrait de profiter de l'ensemble des outils compatibles avec le format SPIRIT, et de réutiliser les composants matériels externes avec une description respectant ce format.

De même que pour l'aspect matériel, l'adaptation du logiciel pourrait aussi se faire de manière compatible en s'appuyant sur un nouveau format standard de description de l'architecture, tel SPIRIT. Il serait en particulier possible d'extraire à partir d'une description, respectant ce format standard, les caractéristiques de l'architecture nécessaires pour le logiciel. La partie génération des couches de bas niveau de cette thèse est alors inchangée.

Par ailleurs, il serait intéressant de développer des générateurs de ces couches logicielles bas niveau pour d'autres processeurs afin de certifier que tous les paramètres sont pris en compte. De plus, il serait possible de paramétrer cette génération pour l'adapter aux différents à différent environnement

d'utilisation (types d'initialisation). Ou encore augmenter la capacité du flot à exploiter les paramètres dans le logiciel grâce à l'insertion de scripts pour la transformation du code source avant la compilation.

L'utilisation de cette exploration d'architecture HMPSoC intégrant logiciel et matériel en aval d'outils de plus haut niveau partant des spécifications (par exemple des diagrammes UML[65]) est évalué. Enfin l'utilisation supplémentaire de ce flot, pour estimer des performances matérielles à partir de modèle de composants matériels temporisés, pourrait être étudiée notamment en exploitant la standardisation prochaine de modèle TLM au niveau de modélisation PVT.

Bibliographie

[1] Philippe Magarshack and Pierre G. Paulin. System-on-chip beyond the nanometer wall. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 419–424, New York, NY, USA, 2003. ACM Press.

[2] Wayne Wolf. The future of multiprocessor systems-on-chips. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 681–685, New York, NY, USA, 2004. ACM Press.

[3] Reiner Harenstein. Reconfigurable computing: A new business model and its impact on soc design. In *DSD*, pages 103–111. IEEE Computer Society, 2001.

[4] Lewis Bryan, Bolsens Ivo, Lauwereins Rudy, Wheddon Chris, Gupta Bhusan, and Tanurhan Yankin. Reconfigurable soc - what will it look like? In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, pages 660–662, Washington, DC, USA, 2002. IEEE Computer Society.

[5] Sami Khawam, Tughrul Arslan, and Fred Westall. Synthesizable reconfigurable array targeting distributed arithmetic for system-on-chip applications. In *IPDPS*. IEEE Computer Society, 2004.

[6] Mirko Loghi, Federico Angiolini, Davide Bertozzi, Luca Benini, and Roberto Zafalon. Analyzing on-chip communication in a mpsoc environment. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 20752, Washington, DC, USA, 2004. IEEE Computer Society.

[7] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Design Automation Conference*, pages 684–689, 2001.

[8] Axel Jantsch et Hannu Tenhunen. *Networks on Chip*. Kluwer Academic Publishers, Boston, 2003.

[9] Richard Goering. Multiple processors mean multiple challenges. *EE Times*, September 2005. http://www.eetimes.com/showArticle.jhtml?articleID=170703813.

[10] Ron Wilson. What is memory's role, anyhow? *EE Times*, 22 Mai 2003.

[11] J. Russell and M. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *ICCD '98: Proceedings of the International Conference on Computer Design*, page 328, Washington, DC, USA, 1998. IEEE Computer Society.

[12] Sébastien Pignolo et Eric Martin et Nathalie Julien et Eric Senn et Brigitte Saget. Optimisation de la consommation d'énergie des applications de traitements du signal et de l'image embarquées sur dsp. http://lester.univ-ubs.fr:8080.

[13] Sudeep Pasricha, Nikil Dutt, and Mohamed Ben-Romdhane. Extending the transaction level modeling approach for fast communication architecture exploration. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 113–118, New York, NY, USA, 2004. ACM Press.

[14] Frank Ghenassia, editor. *Transaction Level Modeling With SystemC: TLM Concepts And Applications for Embedded Systems*. Kluwer Academic Publishers, Boston, 2005.

[15] Chris Rowen and Steve Leibson. Flexible architectures for engineering successful socs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 692–697, New York, NY, USA, 2004. ACM Press.

[16] Deplanche Anne-Marie, Elloy Jean-Pierre, and Sorel Yves. Le placement. État de l'art, IRCyN-INRIA, April 1999. AEEE: Architecture Electronique Embarqué.

[17] Andre Françoise and Pazat Jean-Louis. Le placement de taches sur des architectures parallèles. Publication Interne 338, IRISA, Campus Universitaire De Beaulieu, Avenue du général Leclerc, 35042 RENNES, FRANCE, January 1987.

[18] www.vastsystems.com. Internet.

[19] Realview instruction set simulator. www.arm.com/products/DevTools/RealViewISS.html.

[20] Improvsys. Jazz processor standard tools. www.improvsys.com/registered/general/tools/Jazz_Standard_Tools.pdf.

[21] Coware. Lisatek. www.coware.com/products/lisatek_description.php.

[22] Erik R. Altman, Rene Miranda, Jaime Moreno, and C. Brian Hall. An integrated approach to architectural simulation, timing and memory hierarchy evaluation. Technical report, IBM Research, 1996. www.research.ibm.com/vliw/Pdf/sim.paid96.pdf.

[23] David Sharp. A dynamic recompiling arm emulator. Rapport de master, University of Warwick, 2001.

[24] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *FREENIX Track*, page 41 à 46. USENIX 2005 Annual Technical Conference, 2005.

[25] Gxemul an instruction-level machine emulator under bsd licence. http://gavare.se/gxemul/.

[26] Sungjoo Yoo, Iuliana Bacivarov, Aimen Bouchhima, Yanick Paviot, and Ahmed A. Jerraya. Building fast and accurate sw simulation models based on hardware abstraction layer and simulation environment abstraction layer. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, page 10550, Washington, DC, USA, 2003. IEEE Computer Society.

[27] Aimen Bouchhima. *Modélisation du logiciel embarqué à différents niveaux d'abstraction en vue de la validation et la synthèse des systèmes monopuces*. PhD thesis, INPG, 2006.

[28] Luca Benini and Giovanni De Micheli. Networks on chips: A new soc paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[29] Moo-Kyoung Chung and Chong-Min Kyung. Improvement of compiled instruction set simulator by increasing flexibility and reducing compile time. In *IEEE International Workshop on Rapid System Prototyping*, pages 38–44. IEEE Computer Society, 2004.

[30] Prabhat Mishra and Nikil Dutt. *The EDA Handbook*, volume 3 of *Industrial Information Technology*, chapter Processor Modelling and Design Tools. CRC Press, g. martin, l. lavagno, and l. scheffer edition, 2005.

[31] Nios 2 processor reference handbook. http://www.altera.com.

[32] http://www.arccores.com. ARC cores.

[33] Tensilica. http://www.tensilica.com.

[34] improv systems inc, jazz dsp. http://www.improvsys.com/Architecture/JazzDSPCore.cfm.

[35] Hohenauer, M. and Scharwaechter, H. and Karuri, K. and Wahlen, O. and Kogel, T. and Leupers, R. and Ascheid, G. and Meyr, H. and Braun, G. Compiler-in-loop Architecture Exploration for Efficient Application Specific Embedded Processor Design. In *Design & Elektronik*, Munich, Germany, February 2004. WEKA Verlag.

[36] Oliver Schliebusch, A. Chattopadhyay, R. Leupers, G. Ascheid, H. Meyr, Mario Steinert, Gunnar Braun, and Achim Nohl. Rtl processor synthesis for architecture exploration and implementation. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 30156, Washington, DC, USA, 2004. IEEE Computer Society.

[37] Andreas Hoffmann, Oliver Schliebusch, Achim Nohl, Gunner Braun, Oliver Wahlen, and Heinrich Meyr. A methodology for the design of application specific instruction set processors (ASIP) using the machine description language LISA. In *ICCAD01*, pages 625–630. IEEE, 2001.

[38] Andreas Wieferink, Tim Kogel, Rainer Leupers, Gerd Ascheid, Heinrich Meyr, Gunnar Braun, and Achim Nohl. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 21256, Washington, DC, USA, 2004. IEEE Computer Society.

[39] George Hadjiyiannis and Srinivas Devadas. Techniques for accurate performance evaluation in architecture exploration. *IEEE Trans. Very Large Scale Integr. Syst.*, 11(4):601–615, 2003.

[40] Prabhat Mishra, Peter Grun, Nikil Dutt, and Alex Nicolau. Processor-memory co-exploration driven by a memory-aware architecture description language. In *VLSID '01: Proceedings of the The 14th International Conference on VLSI Design (VLSID '01)*, page 70, Washington, DC, USA, 2001. IEEE Computer Society.

[41] Rainer Laupers. Hdl-based modeling of embedded processor behavior for retargetable compilation. In *ISSS '98: Proceedings of the 11th international symposium on System synthesis*, pages 51–54, Washington, DC, USA, 1998. IEEE Computer Society.

[42] A. Fauth, J. Van Praet, and M. Freericks. Describing instruction set processors using nml. In *EDTC '95: Proceedings of the 1995 European conference on Design and Test*, page 503, Washington, DC, USA, 1995. IEEE Computer Society.

[43] Claire Tristram. It's time for clockless chips. *Technology Review Magazine (based in MIT)*, 104(8):36–41, October 2001.

[44] John Bainbridge and Stephen B. Furber. Chain: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, 2002.

[45] Luc Charest et El Mostapha Aboulhamid Nadir Boudina. Évaluation de la performance de systemc dans la modélisation de systèmes multiprocesseurs et de processeurs réseaux. Université de Montréal.

[46] Wolfgang Müller, W. Rosenstiel, and J. Ruf, editors. *SystemC - Methodologies and Applications.* Kluwer Academic Publishers, Dordrecht, June 2003.

[47] www.systemc.org. Internet.

[48] A. Beugnard. *Les design patterns.* http://perso-info.enst-bretagne.fr/ beugnard/cours/DP.pdf, 1999.

[49] http://www.spiritconsortium.com/. Internet.

[50] www.arm.com. Internet.

[51] Journal Robert Day. The challenges of an embedded software engineer. *Embedded Technology Journal*, October 2005. Director of marketing, Accelerated Technology, A division of Mentor Graphics.

[52] L. Guthier, S. Yoo, and A. Jerraya. Automatic generation and targeting of application specific operating systems and embedded systems software. In *DATE '01: Proceedings of the conference on Design, automation and test in Europe*, pages 679–685, Piscataway, NJ, USA, 2001. IEEE Press. http://tima.imag.fr/sls/documents/scopes2001.pdf.

[53] Lovic Gauthier. *Génération de système d'exploitation pour le ciblage de logiciel multitâche sur des architectures multiprocesseurs hétérogènes dans le cadre des systèmes embarqués spécifiques.* PhD thesis, INPG, 2001.

[54] Yannick Le Moullec et jean Philippe Diguet et Dominique Heller et Jean-Luc Philippe. *Estimation du parallélisme au niveau système pour l'exploration de l'espace de conception de systèmes enfouis*, volume 3. RSTI TSI, 2003.

[55] Yannick Le Moullec. *Aide à la conception de systèmes sur puce hétérogènes par l'exploration paramétrable des solutions au niveau système.* PhD thesis, Université de Bretagne Sud, 2003.

[56] M. Raulet, M. Babel, J.-F. Nezan, O. Déforges, and Y. Sorel. Automatic coarse-grain partitioning and automatic code generation for heterogeneous architectures. In *Proceedings of IEEE Workshop on Signal Processing Systems, SiPS'03*, Seoul, Korea, August 2003.

[57] JoAnn M. Paul. Programmers' views of socs. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 156–181, New York, NY, USA, 2003. ACM Press.

[58] Roch Jean-Louis, Gautier Thierry, and Revire Rémi. Athapascan : an api for asynchronous parallel programming user's guide. Technical report, INRIA - Rhone-Alpes , Equipe : APACHE, 2003.

[59] Coudarcher R. et Serot J. et Derutin J. P. Implementation of a skeleton-based parallel programming environment supporting arbitrary nesting. pages 71–85, Chicago, USA, April 2001. 6th international workshop on high level programming models and supportive environments.

[60] Thomas Kunlin. *Algorithmes et architectures pour l'estimation de mouvement dans le standard de codage vidéo H.264*. PhD thesis, IEF, 2007. A paraitre.

[61] Amer Baghdadi. *Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques*. PhD thesis, INPG, 2002.

[62] Nicholas Nethercote and Julian Seward. Valgrind: A program supervision framework. *Theoretical Computer Science*, 89(2), 2003.

[63] Kcachegrind - profiling visualization. http://kcachegrind.sourceforge.net/cgi-bin/show.cgi/KcacheGrindDocs.

[64] www.cadence.com. Internet.

[65] François Terrier and Sébastien Gérard. Platform modeling in uml + uml profile example. Lille, France, May 4, 2004.

Chapitre 4

Annexe

A

Schéma de définition du fichier de description d'architecture

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:hmpsoc="http://SchemaMacroArchitectureDescription"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace="http://SchemaMacroArchitectureDescription"
elementFormDefault="qualified" attributeFormDefault="qualified">
    <xsd:element name="arch">
      <xsd:complexType>
        <xsd:attribute name="version" type="xsd:string"/>
      <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="1" name="nodes">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" minOccurs="0" name="node">
                <xsd:annotation><xsd:documentation>
For this element, "node":
-name is the name you want to give at this sub system in the platform
-type is which subsystem you want
The string is mandatory and is a valid file path to this subsystem parameter file.
                </xsd:documentation></xsd:annotation>
                <xsd:complexType>
                  <xsd:simpleContent>
                    <xsd:extension base="xsd:string">
                      <xsd:attribute name="name" type="xsd:string"/>
                      <xsd:attribute name="type" type="xsd:string"/>
                    </xsd:extension>
                  </xsd:simpleContent>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element maxOccurs="1" minOccurs="0" name="binds">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element maxOccurs="unbounded" minOccurs="0" name="bind">
                <xsd:annotation><xsd:documentation>
For this element, "bind":
-init and target are the name of a subsystem
- ip and tp are the numbers of the initiator and target ports
              </xsd:documentation></xsd:annotation>                <xsd:complexType>
                <xsd:attribute name="init" type="xsd:string"/>
                <xsd:attribute name="ip" type="xsd:unsignedLong"/>
                <xsd:attribute name="target" type="xsd:string"/>
                <xsd:attribute name="tp" type="xsd:unsignedLong"/>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:key name="NodeInstanceNameKey">
    <xsd:selector xpath="hmpsoc:nodes/hmpsoc:node"/>
    <xsd:field xpath="@hmpsoc:name"/>
  </xsd:key>
  <xsd:keyref name="NodeInstanceNameKeyRef1" refer="hmpsoc:NodeInstanceNameKey">
    <xsd:selector xpath=".//hmpsoc:binds/hmpsoc:bind"/>
    <xsd:field xpath="@hmpsoc:init"/>
  </xsd:keyref>
  <xsd:keyref name="NodeInstanceNameKeyRef2" refer="hmpsoc:NodeInstanceNameKey">
    <xsd:selector xpath=".//hmpsoc:binds/hmpsoc:bind"/>
    <xsd:field xpath="@hmpsoc:target"/>
  </xsd:keyref>
</xsd:element>
</xsd:schema>
```

B

Schéma de définition de la description de l'architecture pour un processeur

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="HARDWARE">
    <complexType>
      <sequence>
        <element maxOccurs="1" minOccurs="1" name="PROCESSOR">
          <complexType>
            <sequence>
              <element maxOccurs="unbounded" minOccurs="0" name="COPRO">
                <complexType>
                  <attribute name="name" type="string"/>
                  <anyAttribute/>
                </complexType>
              </element>
              <element name="LEVEL">
                <simpleType>
                  <restriction base="string">
                    <enumeration value="ISS"/>
                    <enumeration value="NATIF"/>
                  </restriction>
                </simpleType>
              </element>
            </sequence>
            <attribute name="name" type="string" use="required"/>
            <attribute name="type" type="string" use="required"/>
            <attribute name="debug" type="string" use="required"/>
            <attribute name="path" type="anyURI"/>
            <attribute name="endianess" type="string" use="required"/>
          <anyAttribute/>
          </complexType>
        </element>
        <element maxOccurs="unbounded" minOccurs="0" name="SLAVE">
          <complexType>
            <attribute name="name" type="string" use="required"/>
            <attribute name="type" type="string" use="required"/>
             <attribute name="address_base" type="unsignedLong" use="required"/>
            <attribute name="address_size" type="unsignedLong" use="required"/>
            <attribute name="register_map" type="anyURI"/>
            <anyAttribute/>
          </complexType>
          <unique name="nameConstraint">
            <selector xpath="."/>
            <field xpath="name"/>
          </unique>
        </element>
      </sequence>
      <attribute name="node" type="string" use="required"/>
      <attribute name="number" type="unsignedLong" use="required"/>
      <attribute name="all" type="unsignedLong" use="required"/>
    </complexType>
  </element>
</schema>
```

Résumé

L'approche proposée se déroule selon un flot itératif en trois étapes. L'une concerne la modification et le développement rapide du modèle exécutable de l'architecture. Une autre vise le portage rapide des logiciels. La troisième est l'exploration dŠarchitecture logicielle et matérielle. Un outil a été développé pour créer et modifier rapidement un HMP-SoC à partir de sous-systèmes de traitement paramétrables. Une méthode permet d'adapter le logiciel sur une architecture, elle inclut: paramétrer manuellement le logiciel applicatif, l'extraction automatique des caractéristiques de lŠarchitecture, la génération des sources de bas niveau. Enfin une méthode permet dŠeffectuer des simulations multi-niveaux des processeurs. Les simulations de haut niveau servent pour exécuter rapidement les logiciels embarqués, les simulations précises en mode bas niveau (ISS) pour mesurer les performances. Suivant les résultats, l'architecture et les logiciels sont modifiés et le cycle peut reprendre.

Abstract

The proposed approach is an iterative flow in three steps. The first one is the fast development and modification of the architecture executable model. The second one is the adaptation of the embedded software. The third one is the hardware and software architecture exploration. A tool has been developed in order to create and modify quickly a hardware architecture model. It uses flexible sub-systems. One method in order to adapt the embedded software is exposed, it includes: to manually add some parameterization in the software, an automatic extraction of the architecture characteristics, the generation of the low level code sources. To finish a method allow to simulate processors at different level of simulation with their embedded software, high level for fast simulation, low level for performance measurements. Following results, hardware and software are modified and the flow can restart. This flow was tested on a real application, a parallelized H264 encoder.

ISBN : 978-2-84813-101-6