RENPAR'16 / CFSE'4 / SympAAA'2005 / Journées Composants
Le Croisic, France, 5 au 8 avril 2005

Modélisation niveau système de SoC reconfigurables

Imène Benkermi[†], Amine Benkhelifa[‡], Daniel Chillet[†], Sébastien Pillement[†], Jean-Christophe Prevotet[‡], François Verdier[‡]

[†]ENSSAT – Université de Rennes I,
IRISA,
6, rue de Kerampont - 22300 Lannion - France
prenom.nom@enssat.fr

[‡]ETIS – UMR 8051 – Université de Cergy-Pontoise,
ENSEA,
6, Avenue du Ponceau - 95014 Cergy-Pontoise - France
prenom.nom@ensea.fr

Résumé
L'objectif de cet article est de proposer un modèle de formalisation de l'ensemble des objets matériels et logiciels constituant une plate-forme reconfigurable. Le modèle proposé permet de spécifier les interactions entre ces objets lors du développement d'une application sur ce type de plate-forme. Nous décrivons dans cet article notre modèle en définissant l'ensemble des objets logiciels et matériels qui composent une architecture reconfigurable. Nous nous intéressons ensuite à décrire les services du système d'exploitation, partie intégrante de notre modèle, dédié à la gestion d'une telle plate-forme.

Mots-clés : SoC, reconfigurable, modèle d'architecture, modèle de tâches.

---

1. Introduction

L'augmentation de la complexité des algorithmes de traitement du signal, de l'image et du contrôle, notamment dans les applications embarquées, nécessite de grandes puissances de calcul pour satisfaire de nouvelles contraintes. C'est le cas par exemple des contraintes liées au contexte temps réel. Ceci conduit les concepteurs à s'orienter de plus en plus vers des architectures matérielles composées de différentes unités de calcul opérant en parallèle, présentant des capacités de calcul différentes et permettant de grandes optimisations. C'est le cas des **systèmes sur puce** ou SoC (*System on Chip*) développés actuellement dans le domaine des télécommunications de troisième et quatrième génération. Les composants présents dans ces systèmes peuvent être des unités de calcul programmables, des unités de calcul spécialisées ou reconfigurables, ou encore des chemins de données spécialisés.
En particulier, la présence d'unités reconfigurables (i.e. FPGA, DART [3]), ou ARD (Accélérateur Reconfigurable Dynamiquement), permet d'adapter l'architecture à différents traitements. Un accélérateur reconfigurable au sein de ce type de plate-forme permet d'apporter un degré important de flexibilité à l'architecture. Le concept de *Platform-based Design* [7] peut ici être pleinement exploité pour assurer la pérennité d'une architecture vis-à-vis de modifications de ses fonctionnalités ou de ses conditions de fonctionnement (changements de standards, de missions, etc).
La flexibilité de cette architecture est (ou sera, à court terme) assurée, en fonction de la technologie reconfigurable employée, par la possibilité d'allouer dynamiquement différents opérateurs de traitement dédiés au sein même de l'accélérateur reconfigurable.
La gestion globale d'une telle architecture fortement hétérogène nécessite de plus en plus la présence d'un exécutif temps réel (RTOS) adapté capable, au minimum, de centraliser la gestion de la mémoire, d'ordonnancer sous différentes contraintes les tâches à exécuter, d'assurer le partage des moyens de communication et d'offrir au mieux aux concepteurs un modèle de déploiement d'applications qui soit indépendant de la technologie et de la structure matérielle.

La thématique RTOS pour les SoC fait l'objet de nombreuses recherches. Pour le cas particulier de la gestion d'architectures reconfigurables, l'identification des services d'un système d'exploitation temps réel a déjà été traitée en 1999 [4]. Depuis, des RTOS pour le reconfigurable ont été implémentés suivant deux concepts principaux. Le premier est l'utilisation d'un RTOS standard (RTAI [16], RTLinux, VxWorks, ...) auquel on ajoute un certain nombre de services pour la gestion du SoC [12, 11]. Le deuxième est l'utilisation d'un RTOS spécifique, c'est à dire un ensemble de fonctions logicielles (et matérielles) nécessaire à la gestion du SoC [22, 19, 20].
Le travail, présenté dans cet article, est issu des réflexions menées au sein de l'équipe projet CNRS POMARD [5]. Il ambitionne une modélisation système complète d'une plate-forme RSoC (*Reconfigurable System on Chip*), incluant l'ensemble des éléments logiciels et matériels ainsi que leurs interactions.
Nous souhaitons donc, au travers de cet article, identifier en premier lieu les différents composants matériels et logiciels intervenant dans les architectures RSoC. En particulier, nous tentons de définir le rôle et l'organisation précise de l'intergiciel d'une telle plate-forme. L'objectif est de se doter de critères formels pour le développement d'un OS spécifiquement adapté à ce type d'architecture.
Enfin, ce travail doit pouvoir fournir une base de réflexion commune aux concepteurs du matériel et du logiciel s'intéressant aux RSoC, afin de mieux appréhender les concepts utilisés et d'éviter de recréer des concepts déjà existants (e.g. services du système d'exploitation).
Notre soucis de rendre notre modèle de RSoC le plus générique possible (supportant la plus grande variété de plates-formes), nous a conduit à adopter le formalisme de représentation de classes UML.
La première partie de cet article présente le contexte du déploiement d'applications sur un RSoC. La deuxième partie présente le modèle général proposé et définit les différents niveaux de représentation des éléments des systèmes considérés. La troisième partie aborde la définition des services d'un OS adapté à la gestion d'un RSoC. Enfin, la conclusion dresse un bilan et trace les perspectives de ces travaux.

2. Déploiement d'applications sur RSoC

Nous présentons dans cette partie la philosophie globale du déploiement d'une application sur une architecture RSoC. Nous détaillons en premier lieu les éléments matériels considérés au sein de la plate-forme (§2.1) puis le découpage matériel/logiciel d'une application (§2.2) et enfin la façon dont nous envisageons son portage sur le RSoC (§2.3).

2.1. Eléments de description architecturale

Dans l'objectif d'une modélisation complète du RSoC et du processus de déploiement d'application sur celle-ci, il est d'abord nécessaire de définir l'ensemble des éléments matériels sur lesquels une application devra s'exécuter.

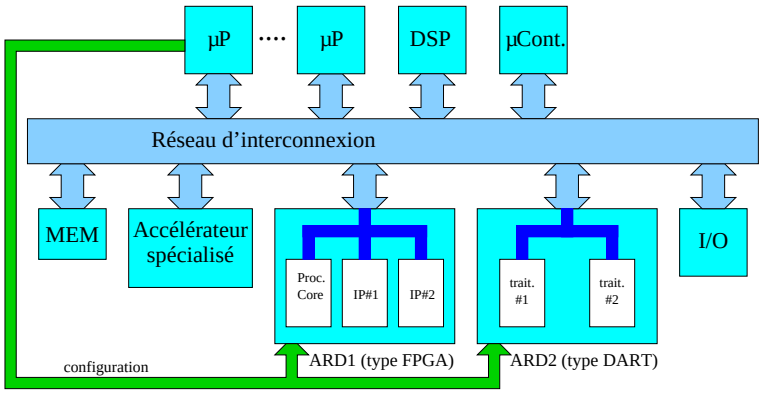

FIG. 1 – Exemple de plate-forme SoC reconfigurable fortement hétérogène.

L'architecture matérielle que nous considérons, peut être schématisée par la figure 1. Elle est fortement hétérogène : de nombreux opérateurs de traitement co-existent au sein de la même puce. Ces opérateurs peuvent posséder des modèles d'exécution différents (séquentiel, parallèle, pipeline, flot de données, micro-programmé...).
Ainsi, nous nous plaçons dans un contexte où l'architecture de la plate-forme RSoC peut comporter :
- Un ou plusieurs processeurs de traitement généraliste. La majeure partie du système d'exploitation est centralisée et donc exécutée sur l'un de ces processeurs ;
- Un ou plusieurs processeurs de traitement dédiés (RISC, DSP, micro-contrôleurs) ;
- Des accélérateurs spécialisés ou IP de traitement dédiés (filtres, convolueurs, décodeurs de VITERBI...) ;
- Des resources de mémorisation et des interfaces d'entrées / sorties ;
- Un ou plusieurs accélérateurs reconfigurable dynamiquement, que nous détaillons plus loin ;
- Un ensemble de ressources de communication (bus, multi-bus, réseau...), avec éventuellement un réseau dédié au transport des informations de configuration vers les ARD, permettant le parallélisme entre le transfert des données et celui des configurations.

La différence majeure entre l'architecture d'un RSoC et celle d'un système sur puce classique réside dans la présence, au sein même du RSoC, d'une zone reconfigurable. L'évolution conjointe de la technologie reconfigurable et du concept de plate-forme évolutive rendent disponibles, sous forme d'IP synthétisables, de tels accélérateurs. Les structures de ces IP reconfigurables peuvent être très variables et dépendent fortement des applications envisagées :
- Zones reconfigurables à grain fin de type FPGA pour les applications légères ou très orientées niveau bit (architecture FPSLIC [1] par exemple) ;
- Zones à grain élevé pour les applications multimédia (SoC XPP [17] avec des traitements arithmétiques principalement) ;
- Zones mixtes associant chemin de données configurable et zone FPGA (architecture DART par exemple [3]).

Le point commun entre toutes ces architectures que nous considérons est la possibilité d'allouer dynamiquement, sur l'ARD, un ou plusieurs traitements spécifiques. Le caractère dynamique indique ici que la nécessité d'effectuer ou non ces traitements peut ne pas être connue au moment de la compilation (dépendance sur les données ou sur les ressources disponibles par exemple). D'autre part, ces traitements ont une durée de vie limitée et très largement supérieure à la durée de la configuration du traitement lui-même sur l'ARD (cf. les travaux sur les mesures de la *rémanence* de différentes architectures reconfigurables dans [2]).

2.2. Partitionnement des applications

La complexité des applications considérées (multimédia, télécommunication, réseau...) implique, lors du développement, un découpage des traitements en tâches. Ces *tâches applicatives* représentent un traitement à réaliser sans connaissance *a priori* du support d'exécution. Compte tenu de l'aspect hétérogène de l'architecture cible, le modèle général que nous avons adopté s'appuie sur l'implémentation de chacune de ces tâches en une ou plusieurs tâches logicielles et/ou matérielles (voir figure 2). Ces tâches devront alors être exécutées sur les différents composants de l'architecture.

Tâche logicielle : On appelle tâche logicielle, une tâche dont l'exécution est prise en charge par un processeur du RSoC. Le terme processeur englobe les processeurs physiquement disponibles dans le RSoC mais aussi les cœurs de processeur préalablement configurés sur un ARD.

Tâche matérielle : On appelle tâche matérielle, un traitement câblé, supporté directement par une IP physiquement disponible ou préalablement configurée sur un ARD.

L'architecture logicielle s'organise autour du système d'exploitation dont l'un des services concerne l'allocation (ou le placement) des différentes tâches matérielles et logicielles sur les différentes cibles disponibles.

2.3. Portage d'une application sur le RSoC

Partant des descriptions architecturale et logicielle précédentes, nous définissons alors différents niveaux de portage des tâches. Ces niveaux sont représentés figure 3. Dans cet exemple, nous avons considéré un RSoC minimal constitué d'un seul processeur généraliste associé à un ARD.

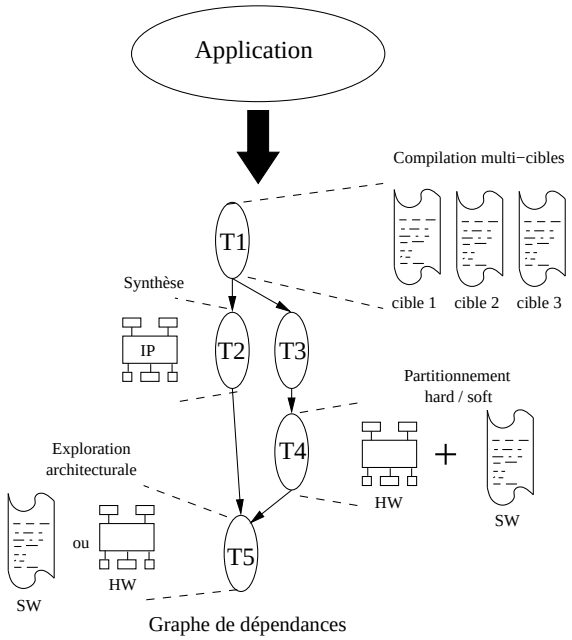

FIG. 2 – Les tâches applicatives (Ti) peuvent posséder, chacune, différentes implémentations logicielles et/ou matérielles possibles

– Le niveau 0 (physique) : correspond à l'architecture physique incluant la zone reconfigurable et le processeur généraliste. Ce niveau comporte aussi un canal de communication (non représenté) et une hiérarchie mémoire. C'est le niveau le plus bas de l'architecture qui ne contient qu'un ensemble de ressources (zones de mémorisation, bande passante sur un bus, ressources reconfigurables, temps du processeur) susceptibles d'être allouées aux traitements que nécessite l'application.
– Le niveau 1 : correspond à l'ensemble des tâches matérielles nécessitant des ressources du niveau 0. En particulier, l'ensemble des objets matériels à configurer sur l'ARD (IP dédiées, cœurs de processeurs, filtres...) sont des tâches de niveau 1. Configurer une tâche matérielle au sein de l'ARD nécessite une étape obligatoire de transport des informations de configuration (notée H sur la figure 3). Il est à noter qu'il est possible d'avoir différentes réalisations matérielles pour une même tâche de niveau 1, ceci permet d'avoir différentes formes et/ou performances pour un même traitement.
– Le niveau 2 : ce niveau regroupe l'ensemble des tâches logicielles de l'application à déployer sur le processeur généraliste ou sur les cœurs de processeurs configurés dans l'ARD. L'assignation d'une tâche logicielle à l'un de ces processeurs correspond à un changement de contexte (noté S sur la figure 3). Dans le cas où la cible est un cœur de processeur sur ARD, une étape de configuration matérielle supplémentaire pourra être nécessaire (la configuration du cœur de processeur lui-même). De plus, à l'instar des tâches matérielles, une tâche logicielle peut avoir différentes descriptions (compilation multi-cibles) permettant d'assigner cette dernière à différents types de processeurs.

A partir de ces niveaux de portage nous avons défini un modèle général permettant de clarifier les interactions entre tous les éléments du système.

3. Le modèle général

Une modélisation graphique basée sur UML a été choisie afin d'avoir une représentation claire des différents éléments du modèle général. Le modèle inclue les tâches applicatives, tous les composants matériels et logiciels du système ainsi que les interactions entre eux. Plus particulièrement, nous nous appuyons sur une représentation à l'aide d'un diagramme de classes UML, tel que le montre la figure 4. Nous avons choisi de décomposer la modélisation en trois parties correspondant chacune à une couche de description différente (*application*, *architecture* et *intergiciel*).

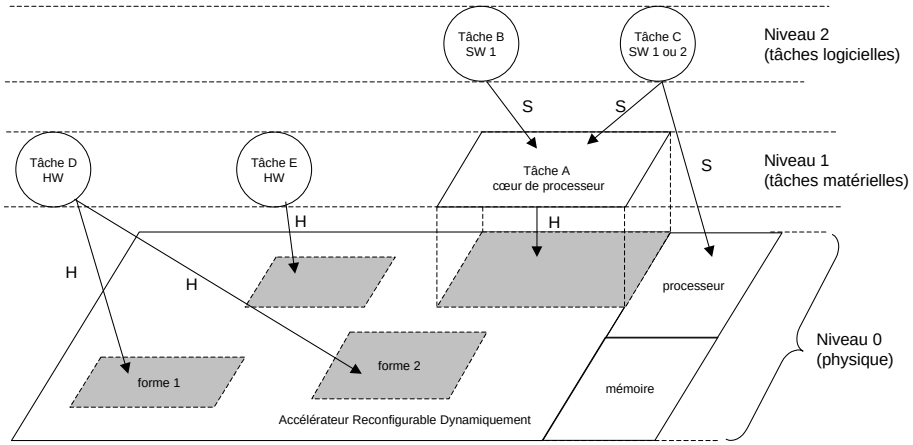

FIG. 3 – Niveaux de portage des tâches sur le RSoC

Chacune de ces couches correspond aussi à des compétences différentes et donc à des équipes de concepteurs disjointes. Ceci permet de spécifier une application hors de toutes considérations matérielles. De ce fait, notre modélisation exprime l'indépendance de la spécification de l'application vis à vis de l'architecture.

3.1. La couche application

La couche *application* est basée essentiellement sur la description de tâches applicatives (*ApplicativeTask*). C'est la couche utilisée par le concepteur "logiciel" pour représenter son application sans se préoccuper de l'architecture matérielle. Ainsi, à ce niveau de description, n'existent que des informations liées aux traitements à réaliser.

Cette représentation suppose que le concepteur a décomposé son application en tâches et qu'il est en mesure de décrire chacune d'elles ainsi que leurs dépendances (cf. figure 2). Plusieurs travaux se sont intéressés à la structuration d'une application en tâches [6, 9], nous ne nous intéressons pas à cet aspect et notre propos suppose que ce travail est déjà réalisé.

La tâche applicative est spécifiée via des paramètres classiques du domaine du temps réel, tels que son identifiant, sa priorité, sa période, son échéance, sa date de validité (ou *ReadyTime*), son type qui spécifie si la tâche est stricte ou relative et un dernier paramètre indiquant son caractère préemptible ou non. Les tâches sporadiques peuvent être transformées en tâches périodiques avec une période égale à l'intervalle minimum entre deux arrivées.

Par ailleurs, si la tâche possède une priorité relative, l'ordonnanceur peut décider de ne pas déclencher ou d'interrompre son exécution si cela cause le dépassement d'échéance d'une tâche stricte. L'interruption d'une tâche sera évidemment conditionnée non seulement par son paramètre d'interruption mais aussi par le paramètre de préemption du support physique final.

Enfin, les dépendances de contrôle et de données entre tâches doivent être spécifiées dans ce niveau. En particulier, la modélisation réalisée sur la production/consommation de données en vue de leur partage avec d'autres tâches ou d'une utilisation interne à la tâche (données temporaires de calcul), fait apparaître la notion d'accès mémoire (*DataAccess*). Les accès mémoire assurent le transfert d'une ou d'un ensemble de données (*Data*). L'objet *Data* permet, de plus, de spécifier dans ce niveau la taille de l'espace mémoire total à allouer à la tâche.

Enfin, dans ce niveau, le concepteur peut être en mesure de spécifier la ou les qualités de services (*QoS*) associées à chaque tâche. Le QoS peut être défini selon plusieurs paramètres (respect du temps réel, qualité du service rendu…) et doit être adapté, par le concepteur logiciel, à chaque application. Le respect des contraintes de qualités de services ne pourra se faire qu'à la condition que les éléments des couches inférieures du modèle et, en particulier, les éléments matériels, supportent certains mécanismes d'exécution. La spécification des contraintes de QoS sort du cadre de cet article.

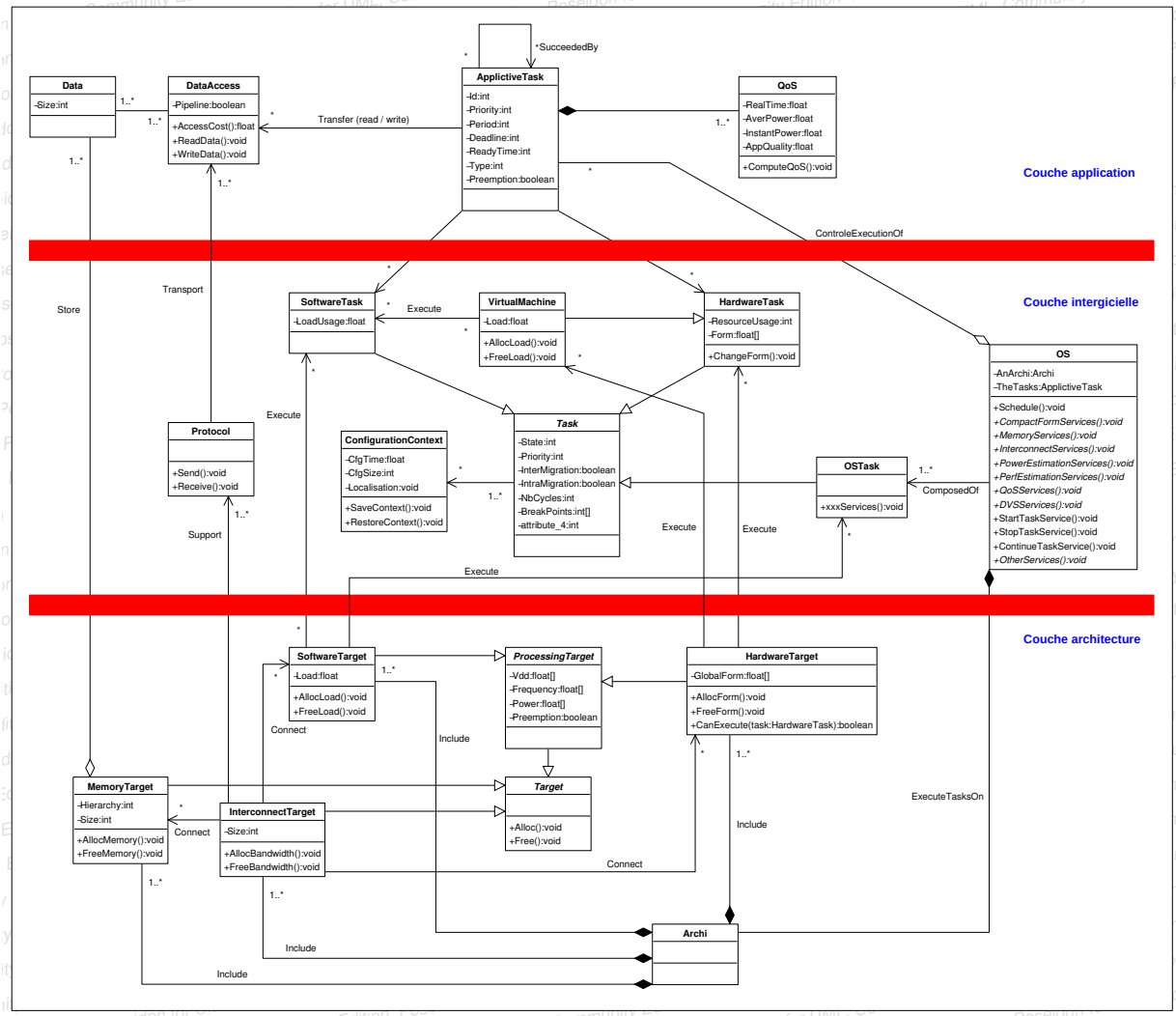

FIG. 4 – Diagramme de classes du modèle système général

3.2. La couche architecture

La couche *architecture* représente les entités matérielles du RSoC présentes physiquement dans la plate-forme (niveau 0). Elle englobe les cibles de traitement (*ProcessingTarget*) qui devront supporter l'exécution des tâches (§3.2.1), les cibles de stockage (*MemoryTarget*) des données manipulées (§3.2.2) et enfin les cibles d'interconnexion (*InterconnectTarget*) permettant l'échange de ces données (§3.2.3).

3.2.1. Les cibles de traitement

Les cibles de traitement regroupent les supports d'exécution. Cette classe (*ProcessingTarget*) est déclinée en deux sous-ensembles représentant respectivement les cibles de type processeur (*SoftwareTarget*) et les cibles matérielles (*HardwareTarget*). Ces deux sous-ensembles peuvent être caractérisés par différentes tensions d'alimentation et fréquences ainsi que différentes puissances (statiques ou dynamiques) consommées. La représentation de ces paramètres permet d'envisager la prise en compte des techniques tels que la variation dynamique de la tension (*Dynamic Voltage Scaling*) [8].

1. Les supports d'exécution logicielle
   L'architecture du RSoC contiendra au minimum un processeur qualifié de central. Les autres processeurs (qualifiés de secondaire) n'auront pas de rôle particulier dans la gestion de la plate-forme.
   (a) Le processeur central

C'est la machine d'exécution principale du système d'exploitation de gestion du RSoC. Il est physiquement relié à tous les éléments de la plate-forme. Ce processeur supporte aussi l'exécution de tâches logicielles.

(b) Les processeurs secondaires

Cette catégorie regroupe tous les autres supports d'exécution logicielle. On trouvera dans ce sous ensemble des processeurs généralistes ou spécialisés de type DSP, micro-contrôleurs... Ils supportent des tâches logicielles. Ils peuvent toutefois nécessiter un micro-noyau capable de gérer plusieurs tâches.

2. Les supports d'exécution matérielle

Ils sont divisés en deux grandes catégories, les modules spécifiques et les accélérateurs reconfigurables dynamiquement.

(a) Les accélérateurs spécialisés

Ce sont des blocs matériels câblés dans l'architecture (figure 1), ils sont optimisés pour une fonctionnalité donnée. Ce sont des modules, non reconfigurables, pouvant accéder aux canaux de communication.

(b) Les accélérateurs reconfigurable dynamiquement

Un ARD est un ensemble de resources reconfigurables dynamiquement dont la structure et le nombre peuvent être très variables d'une architecture à l'autre.

La granularité des ARD correspond à la finesse des opérateurs que l'on peut y configurer. A titre d'exemple, une zone reconfigurable au niveau du bit est associée à un grain fin tandis qu'une zone dans laquelle l'opérateur de base est une ALU est considérée comme étant à gros grain.

Le modèle proposé permet de considérer toutes les formes de granularités et propose une vue générique des zones reconfigurables. Chaque ARD permet d'implanter un certain nombre de tâches matérielles.

3.2.2. Les ressources mémoire

Le modèle dispose au niveau architectural d'une organisation mémoire générique. Les notions de hiérarchie mémoire, d'optimisation de ces hiérarchies, de caches... constituent un enjeu essentiel dans les SoC et notamment dans les RSoC [14]. Des études sont actuellement en cours pour approfondir ces aspects et devraient conduire à une amélioration de la partie correspondante dans le modèle proposé.
Dans ce niveau, la taille, l'organisation et la hiérarchie mémoire (nombre de caches utilisés) doivent être spécifiées.

3.2.3. Les ressources de communications

Dans la couche architecture, les ressources de communications permettent de définir le support physique des transactions entre les différents éléments de la plate-forme.
Le premier type d'interconnexions concerne les communications entre les processeurs, les accélérateurs spécialisés et les accélérateurs matériels. Ce type d'interconnexion est statique et il est optimisé au moment de la conception.
Les interconnexions entre modules matériels à l'intérieur d'une même zone reconfigurable constituent un véritable enjeu et un des points les plus sensibles du modèle. En effet, celles-ci doivent évoluer au rythme des différentes configurations que ce soit au niveau structurel ou fonctionnel. Les médias de communications doivent pouvoir être flexibles et s'adapter à l'architecture. Différentes études ont conduit à l'élaboration de réseaux sur puces dans le cadre du reconfigurable [10, 13] permettant de garantir une très grande flexibilité tout en maintenant une certaine qualité de services.
Certaines interfaces de ces interconnexions sont, du reste, standardisées et leurs règles d'échanges font l'objet de nombreux protocoles (cf. VCI[21], OCP [15]).

3.3. La couche Intergicielle (ou *Middleware*)

La couche *intergicielle* contient toutes les entités ordonnançables par le système d'exploitation, c'est à dire les tâches et leurs caractéristiques liées au matériel (§3.3.1). Ce niveau décrit en particulier les différentes

tâches de configuration ainsi les protocoles de communication (§3.3.2) pouvant être utilisés. Enfin, c'est à ce niveau que vont être spécifiées les interactions entre une application et une implémentation sur une plate-forme reconfigurable (§3.3.3) et le système d'exploitation qui gère toutes ces entités.

3.3.1. Les tâches et leurs supports d'exécution

La couche intergicielle s'articule autour d'un élément central qui est la tâche au sens du système d'exploitation (*Task*). Une tâche, à ce niveau, peut ensuite se décliner de trois façons différentes : (*SoftwareTask*, *HardwareTask*) et un troisième type spécifique pour définir les tâches liées au système d'exploitation (*OSTask*). Du point de vue de l'application, cette représentation permet effectivement de considérer le fait qu'une tâche applicative peut avoir une ou plusieurs implémentations possible suivant la cible d'exécution logicielle ou matérielle considérée (cf. figure 2).
Les tâches logicielles (*SoftwareTask*) s'exécutent sur les cibles de type processeur du RSoC (*SoftwareTarget*). Ce sont des tâches classiques ordonnancées et placées par l'OS.
Les tâches matérielles (*HardwareTask*) s'exécutent sur les ARD (*HardwareTarget*) ou font appels aux éventuels modules spécifiques présents dans la plate-forme. Ces tâches sont principalement des implémentations efficaces de tâches applicatives. Elles nécessitent une certaine quantité de ressources (*ResourceUsage*) et consomment une certaine quantité de la surface de l'ARD (*Form*) lorsque la tâche est affectée à ce dernier.

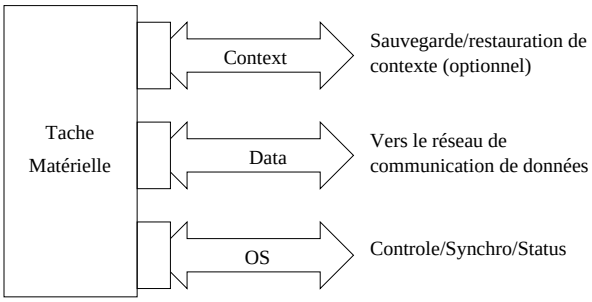

FIG. 5 – La tâche matérielle dans l'ARD et ses interfaces

Une tâche matérielle, comme le montre la figure 5, est capable de communiquer avec les autres ressources du RSoC via trois interfaces. Une première interface dite de contexte qui permet la sauvegarde et la restauration de l'état de la tâche matérielle (le cas échéant). Une seconde interface, de données permet l'interconnexion de la tâche matérielle sur le médium de communication, c'est sur cette interface que circule les données d'entrées/sorties de la tâche. Finalement, l'interface de contrôle permet de communiquer avec l'OS chargé de la gestion des tâches et de l'ordonnancement.
Certaines tâches matérielles particulières peuvent être des supports d'exécution de tâches logicielles à l'instar des processeurs classiques. C'est le cas de cœurs de processeurs "mous" configurés dans l'ARD (Altera NIOS, Xilinx μBlaze) ou de chemins de données spécialisés exécutant des instructions. Le modèle général proposé les regroupe sous le terme de machines virtuelles (*VirtualMachine*).
Comme le montre la figure 6, une machine virtuelle dispose d'une interface de programmation au travers de laquelle le système d'exploitation peut "charger" une tâche, d'une interface de communication avec les autres ressources du RSoC et d'une interface de contrôle vers le système d'exploitation. Elle contient également au minimum un micro-noyau capable de gérer la machine, de lancer l'exécution des tâches qui lui sont attribuées et d'en indiquer l'état (i.e. `EN COURS`, `EN ATTENTE`, `PRÊTE`, `NON OPÉRATIONNELLE`). Ce micro-noyau peut être réduit à une simple machine d'états pour les machines virtuelles les plus simples ou être un noyau de système d'exploitation temps réel local pour les machines de type processeur. Dans le second cas, ce noyau local peut être chargé de l'ordonnancement et de la gestion des ressources locales de la machine virtuelle.
Enfin, chaque tâche, qu'elle soit logicielle ou matérielle, nécessite une phase de configuration (*ConfigurationContext*) pour s'exécuter sur une cible. Lorsque la tâche considérée s'exécute sur un processeur, ce

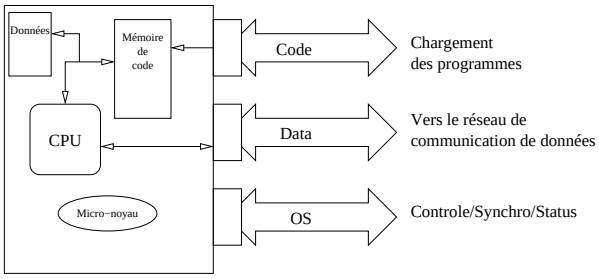

FIG. 6 – La machine virtuelle et ses interfaces dans le modèle du SoC.

contexte est similaire à celui d'une tâche logicielle classique (valeur du pointeur de programme, pointeur de pile...). Dans le cas d'une tâche matérielle, le contexte peut être plus complexe et contenir l'ensemble de la séquence de configuration d'une cible matérielle (par exemple un *bitstream* pour une zone de type FPGA).

3.3.2. Protocoles de communication

Par ailleurs, la couche *intergicielle* prend en charge la liaison entre les transferts de données et la cible matérielle supportant effectivement ces transferts. La notion de protocole (*Protocol*) apparaît donc à ce niveau. Les données sont ainsi transférées par les tâches au travers des accès mémoires. Ces accès mémoires sont gérés par le protocole qui est lui-même supporté par la cible d'interconnexion. La modélisation qui est faite au niveau applicatif permet une abstraction des échanges (communications) inter-tâches [18].

3.3.3. l'OS

Enfin, c'est dans cette couche qu'est situé le système d'exploitation (*OS* pour *Operating System*). Nous avons modélisé l'OS avec un ensemble de services. Chaque service pouvant alors être décliné en une tâche système particulière (*OSTask*). Un certain nombre des services de l'OS relatifs à la gestion de la plateforme ont été identifiés. Les services spécifiques au caractère reconfigurable de l'architecture sont décrits dans la section suivante.

4. Spécificité d'un OS pour un RSoC

Le rôle de l'OS consiste à organiser l'ensemble des traitements sur la plate-forme RSoC. Il s'agit donc d'un rôle de chef d'orchestre qui consiste à définir l'ordre d'exécution des tâches, mais aussi à préparer le bon déroulement de cet ordre d'exécution.
La présence d'ARD dans le système modifie assez profondément plusieurs services de l'OS. Les caractéristiques particulières de la zone reconfigurable nous amènent donc, assez logiquement, à revisiter l'ensemble des services d'un OS pour identifier les adaptations à apporter de ce point de vue. Dans les paragraphes suivants, nous donnons quelques éléments de réflexions pour les services principaux de l'OS pour un RSoC.

4.1. Services des tâches

4.1.1. Sélection

Toutes les tâches, matérielles ou logicielles, sont gérées par l'OS, celui-ci pouvant être amené à gérer plusieurs instances possibles pour une même tâche applicative. En effet, notre modèle de description de l'application repose sur la possibilité de spécifier l'exécution de certaines tâches applicatives au travers de tâches logicielle et/ou matérielle. Le choix d'une tâche logicielle ou matérielle, pour l'exécution d'un traitement, est réalisée dynamiquement par l'OS en fonction de l'état du RSoC.
Par exemple, si le processeur central du RSoC peut supporter la charge d'une nouvelle tâche logicielle tout en respectant l'ensemble des contraintes, alors la tâche pourra être exécutée par celui-ci. Par contre, si le processeur ne peut pas accepter une nouvelle tâche sans provoquer la violation des contraintes d'une ou plusieurs tâches, alors il faut envisager le placement (éventuellement le déplacement, c.a.d la

migration) d'une ou plusieurs tâches vers un ou plusieurs ARD.
Une autre approche est envisageable pour le respect des contraintes temporelles. Elle consiste, lorsque c'est nécessaire et si des resources sont disponibles, à dupliquer des machines virtuelles sur les ARD. Cette approche permet d'envisager ainsi l'équilibrage de charge entre les cibles logicielles et matérielles.

4.1.2. Migration

Nous nous plaçons volontairement dans le cas où l'un des rôles de l'OS serait celui d'assurer la répartition optimale des charges de calcul. Pour remplir cet objectif, les tâches ainsi que leurs supports d'exécution doivent pouvoir supporter une éventuelle préemption par l'OS (caractéristique précisée dans la classe *ProcessingTarget*).
Par ailleurs, une tâche peut être autorisée à migrer d'une cible, matérielle ou logicielle, vers une autre cible équivalente (*IntraMigration*) ou d'un type différent (*InterMigration*). Cela suppose l'existence de deux instances différentes (l'une logicielle, l'autre matérielle) d'une même tâche applicative. Du point de vue du système d'exploitation, la migration s'effectue après sauvegarde de la configuration de la tâche qui va correspondre au point de reprise (*BreakPoint*), antérieur à la date d'interruption, le plus proche parmi les points de reprise spécifiés au sein de la même tâche. A chaque bloc compris entre deux points de reprise peut correspondre une représentation différente suivant la cible d'exécution.
Il est à noter que le problème principal dans l'*InterMigration*, reste l'identification de l'équivalence d'état entre la représentation logicielle et la représentation matérielle de la tâche à faire migrer ; les informations sur l'état doivent être transmises pour pouvoir relancer la tâche à partir de son point d'arrêt [11].

4.1.3. Service de placement/configuration

Lorsque le choix de la cible d'exécution a amené l'OS à placer une tâche sur un ARD, alors un service spécifique peut être requis. Ce service est particulier car il apparaît ici la notion de phase de configuration optionnelle. En effet, si une tâche logicielle doit être affectée à une machine virtuelle, la phase de configuration de cette même machine virtuelle peut être optionnelle, si elle est déjà présente dans l'ARD.

4.2. Service de gestion du QoS

La qualité de service des traitements est un point délicat dans tout système ayant à gérer des tâches irrégulières, c.a.d non périodiques, sporadiques...Ce point est d'autant plus délicat qu'il y a de fortes probabilités que les tâches matérielles prises en charge par un ARD ne puissent pas facilement être préemptées. En effet, lors de l'exécution de tâches sur un processeur et de l'arrivée d'une nouvelle tâche pour ce processeur, l'OS peut modifier dynamiquement le QoS des tâches afin de continuer à assurer l'exécution de l'application, mais dans un mode dégradé. En revanche, dans le cas des tâche matérielles, l'OS pourrait être incapable de modifier le QoS et cela pourrait conduire à un dysfonctionnement de l'application.

4.3. Service de gestion des ressources

Comme pour un processeur classique, l'OS d'un RSoC doit avoir une connaissance très précise de l'occupation des ressources à chaque instant. Toutefois, la notion de ressources dans le modèle de RSoC est une notion dynamique. En effet, la quantité de ressources disponibles dans les ARD évolue au fil des reconfigurations effectuées par l'OS. De plus, l'assignation dynamique d'une tâche sur une zone géographique d'un ARD rend la notion de ressources non seulement dynamique d'un point de vue temporel mais aussi d'un point de vue géographique.
La gestion des ressources pour cette partie du RSoC ne peut donc se satisfaire de renseignements aussi simples que "ressource occupée", "ressource libre". Des informations supplémentaires concernant la quantité, la forme, la position des ressources "consommées" par la tâche sont absolument nécessaires à la bonne gestion de la plate-forme.
La gestion de la mémoire dans un RSoC est aussi un réel enjeu si l'on souhaite conserver les performances et la flexibilité qu'apportent l'utilisation d'ARD.

4.4. Service de compactage / ramasse miettes

Au cours de l'exécution des tâches matérielles dans un ARD, il y a de fortes chances que ce dernier se morcelle (de manière similaire à une mémoire dans laquelle une succession d'allocations et de désalloca-

tions sont effectuées). Ce morcellement, s'il n'est pas maîtrisé, peut conduire à l'impossibilité de trouver une zone dans l'ARD de taille suffisante pour y allouer une nouvelle tâche à exécuter.
Un service de "ramasse miettes" doit alors être envisagé. Le rôle d'un tel service consiste à récupérer la forme rectangulaire la plus grande possible dans l'ARD. Ce service s'appuie sur une fonction de compactage des tâches matérielles. Le déclenchement de ce service doit reposer sur une mesure "de fractionnement" et une évaluation du temps d'exécution du ramasse miettes.
Notons que la fonction de compactage, dont l'un des rôles consiste à déplacer les tâches matérielles au sein de l'ARD, peut aussi prendre en charge la définition d'une nouvelle forme lors du déplacement d'une tâche. C'est la possibilité de modifier la forme des tâches matérielles afin d'optimiser l'utilisation des ressources [19]. On dira alors que la tâche matérielle est malléable ou géométriquement souple.

4.5. Service de communication

Ce service doit pouvoir offrir toutes les possibilités de communications classiques. Cependant, se pose le cas du médium de communication à l'intérieur d'un ARD. En effet les propriétés de migration des tâches au sein d'une zone reconfigurable et l'évocation d'un service de compactage nécessitent de fait la mise en place d'un canal ou réseau de communication flexible.

5. Conclusion

Cet article constitue une première synthèse des réflexions et travaux qui ont été menés au sein du thème OS de l'EPML POMARD. L'objectif initial de ce projet a été de définir avec précision les avantages apportés par un OS embarqué sur une plate-forme SoC reconfigurable ainsi que les contraintes imposées par celui-ci à la fois sur le déploiement d'une application et sur la conception de la plate-forme elle-même. Les premiers travaux menés dans ce sens ont permis d'établir les liens et les interactions entre l'OS (et ses services) et les différents éléments reconfigurables, ou non, de la plate-forme.
Ces travaux nous ont conduit à définir le modèle système général qui est proposé dans cet article. Bien qu'incomplet, il constitue à notre connaissance la première démarche de formalisation de l'ensemble d'une plate-forme reconfigurable menée par une communauté d'architectes de systèmes et doit pouvoir fournir une base solide de perspectives futures.
En particulier, la définition des services précis que doit offrir un OS temps réel embarqué sur un RSoC n'est qu'embryonnaire. Les aspects de qualité de service, volontairement pris en compte dès le début, doivent pouvoir se concrétiser au travers de l'équilibrage des charges de calcul, la migration (et *l'inter*-migration) des tâches, le placement 2D des tâches matérielles au sein des ARD et les interconnexions *flexibles*, autant de services qui n'ont été que mentionnés dans ce document.

Bibliographie

1. ATMEL. – FPSLIC (AVR with FPGA). – `http://www.atmel.com/products/FPSLIC/`.
2. Benoit (Pascal), Sassatelli (Gilles), Torres (Lionel), Demigny (Didier), Robert (Michel) et Cambon (Gaston). – Metrics for reconfigurable architectures characterization : Remanence and scalability. *In : SAMOS'03 : Systems, Architecture, Modeling and Simulation*. – Samos, Grèce, july 2003.
3. David (R.), Chillet (D.), Pillement (S.) et Sentieys (O.). – Dart : A dynamically reconfigurable architecture dealing with next generation telecommunications constraints. *9th IEEE Reconfigurable Architecture Workshop RAW*, April 2002.
4. Diessel (O.) et Wigley (G.). – *Opportunities for Operating Systems Research in Reconfigurable Computing*. – Rapport technique n ACRC-99-018, Advance Computing Research Center, School of Computer and Information Science, Univ. of South Australia, August 1999.
5. EPML POMARD (Thème OS). – Problématique d'intégration OS dans les plateformes à reconfiguration dynamique. – Mai 2004. Wokshop du RTP SOC, disponible sur `http://www.lirmm.fr/rtp_soc/`.
6. Gomaa (H.). – *Software Design Methods for Concurrent and Real-Time Systems*. – Addison Wesley, 1993.
7. Keutzer (Kurt), Malik (Sharad), Newton (A. Richard), Rabaey (Jan M.) et Sangiovanni-Vincentelli (A.). – System-Level Design : Orthogonalization of Concerns and Platform-Based Design. *IEEE Trans. on Computer-Aided Design of Intergrated Circuits and Systems*, vol. 19, n12, December 2000, pp. 1523–1543.

8. Kim (W.), Kim (J.) et Min (S. L.). – A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. *in Proceedings of the Design Automation and Test in Europe DATE, Paris, France*, 4-8 March 2002.
9. Kopetz (H.) et Obermaisser (H.). – Temporal composability. *Computing and Control Engineering Journal*, August 2002, pp. 156–162.
10. Marescaux (T.), Mignolet (J-Y.), Bartic (A.), Moffat (W.), Verkest (D.), Vernalde (S.) et Lauwereins (R.). – Networks on Chip as Hardware Components of an OS for Reconfigurable Systems. *In : Field Programmable Logic and Application : 13th International Conference, FPL*, pp. 595–605.
11. Nollet (V.), Coene (P.), Verkest (D.), Vernalde (S.) et Lauwereins (R.). – Designing an Operating System for a Heterogeneous Reconfigurable SoC. *In : Reconfigurable Architectures Workshop*. – Nice, France, April 2003.
12. Nollet (V.), Mignolet (J-Y.), Bartic (T.A.), Verkest (D.), Vernalde (S.) et Lauwereins (R.). – Hierarchical Run-Time Reconfiguration Managed by an Operating System for Reconfigurable Systems. *In : International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 81–87. – Las Vegas, USA, June 2003.
13. Nollet (Vicent), Marescaux (Théodore) et Verkest (Diederik). – Operating-System Controlled Network on Chip. *In : Design Automation Conference, DAC*, pp. 256–259.
14. Ouaiss (I.). – *Hierarchical Memory Synthesis in Reconfigurable Computers*. – Mémoire de doctorat, University of Cincinnati / OhioLINK, 2002.
15. Partnership (Open Core Protocol International). – Ocp-ip. – `http://www.ocpip.org/`.
16. RTAI. – `http://www.rtai.org/`.
17. Schueler (H.). – Smart media processing with XPP, white paper. – `http://www.pactcorp.com/xneu/px_SMeXPP.html`, April 2003.
18. Segard (A.) et Verdier (F.). – SOC and RTOS : Managing IPs and tasks communications. *In : Field-Programmable Logic and its Applications : 14th International conference (FPL)*, pp. 710–718. – Antwerp, Belgium, September 2004.
19. Steiger (C.), Walder (H.) et Platzner (M.). – Operating Systems for Reconfigurable Embedded Platforms : Online Scheduling of Real-time Tasks. *IEEE Transaction on Computers*, vol. 53, n11, November 2004, pp. 1392–1407.
20. Ullmann (M.), Hübne (M.), Grimm (B.) et Becker (J.). – On-Demand FPGA Run-Time System for Dynamical Reconfiguration with Adaptive Priorities. *In : Field Programmable Logic and Application : 14th International Conference, FPL*. pp. 454–463. – Springer-Verlag Heidelberg.
21. VSIA. – `http://www.vsia.org/`.
22. Walder (H.) et Platzner (M.). – Reconfigurable Hardware Operating Systems : From Design Concepts to Realizations. *In : Proceedings of the 3rd International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA)*. pp. 284–287. – CSREA Press.