Méthodologie de synthèse d'adaptateurs spatio-temporels

Cyrille CHAVET[1], Philippe COUSSY[2], Pascal URARD[1], Eric MARTIN[2]
[1]STMicroelectronics, Crolles, FRANCE. {prénom.nom@st.com}
[2]LESTER, Université de Bretagne Sud, CNRS FRE 2734. {prénom.nom @univ-ubs.fr}

Thème choisi :

Adéquation algorithmes et architectures (5.1)

Problème traité :

Génération automatique d'interface de communication exigeant une flexibilité qui dépasse les capacités des méthodes de conception sur lesquelles s'appuient les outils de synthèse actuels.

Originalité du travail :

Nous proposons (1) un modèle formel basé sur des graphes pour représenter les relations temporelles existant entre les données d'entrée/sortie qui transitent par l'interface à concevoir, (2) une approche de fusion de graphe lorsqu'il existe plusieurs modes d'interfaçage et (3) un algorithme de parcours de graphe pour réaliser l'assignation des données sur des structures de mémorisation de type pile (FIFO), file (LIFO) et registre. L'approche proposée permet une exploration architecturale rapide et performante de l'espace de conception.

Résultats nouveaux :

A partir d'une unique description de la règle d'entrelacement ou d'interfaçage (réalisée à l'aide du langage C), une variété d'architectures spécifiques (décrites en VHDL RTL synthétisable) est obtenue. Nous sommes également capables de générer des architectures (re)configurables pouvant gérer plusieurs modes de fonctionnement.

INTRODUCTION

Les applications du traitement du signal (TDSI) sont maintenant largement utilisées dans des domaines variés allant de l'automobile aux communications sans fils, en passant par les applications multimédias et les télécommunications. La complexité croissante des algorithmes implémentés, et l'augmentation continue des volumes de données et des débits applicatifs, requièrent souvent la conception de circuits intégrés dédiés (ASIC). Typiquement l'architecture d'un composant complexe du TDSI utilise : des éléments de calculs de plus en plus complexes, des mémoires et des modules de brassage de données (entrelaceur/désentrelaceur pour les TurboCodes, blocs de redondance spatio-temporelle dans les systèmes OFDM[1]/MIMO, …) et privilégie des connexions point à point pour la communication inter éléments de calcul.

Pour la conception de la partie calculatoire du système, les concepteurs peuvent réutiliser des blocs IP préconçus, et/ou utiliser des outils de synthèse de haut niveau afin d'optimiser le processus de traduction d'un algorithme en un circuit intégré. Malheureusement, l'hypothèse selon laquelle les IP peuvent être utilisés sans modifications pour la conception de différents systèmes s'applique bien aux composants de type processeur (DSP, …) mais n'est pas réaliste pour des coprocesseurs de type TDSI (FFT, …). La synchronisation et la communication de données entre composants réutilisés requièrent souvent la conception d'une interface supplémentaire car ces composants n'ont pas été pensés et conçus pour communiquer et échanger des données entre eux. Ainsi, chaque combinaison de composants réutilisés implique la conception d'une interface différente, ce qui devient rapidement une tâche lourde dans le flot de conception d'un système (adaptation des contraintes temporelles, des séquences de transferts, de la synchronisation …).

Pour des composants de type entrelaceurs (brassage de données), typiquement utilisés dans les applications à base de TurboCodes, la rapidité d'évolution des standards, et la difficulté d'une conception manuelle, en font une problématique majeure pour la conception de circuits. De fait, leur conception peut amener à des solutions coûteuses en latence, en mémoires et en consommation. Une difficulté majeure vient notamment du fait que ces composants peuvent avoir différents modes de fonctionnement, et passer de l'un à l'autre en cours d'exécution. Les concepteurs doivent alors non seulement optimiser les architectures de chacun de ces modes, mais il leur faut également fusionner ces différentes configurations en un seul et même circuit, sans pour autant engendrer un surcoût trop important (en se contentant d'accoler les différents chemins de données par exemple), ce qui suppose de trouver l'architecture la plus proche de chacun des modes de fonctionnement.

Pour les deux classes de problème présentées (intégration d'IP et conception d'entrelaceurs), on s'aperçoit que les difficultés se situent au niveau de l'architecture mémoire qui devra assurer le réordonnancement et la temporisation (contraintes spatio-temporelles) des données, tout en minimisant les coûts en termes de surface et de consommation.

Aujourd'hui, le coût de ces systèmes en termes d'élément mémorisant est très élevé; les concepteurs cherchent donc à minimiser la taille de ces buffers afin de réduire la consommation et la surface total du circuit, tout en cherchant à en optimiser les performances.. Malheureusement, les outils de synthèse de haut niveau disponibles ne prennent pas en compte ces aspects de façon efficace, et les architectures générées se caractérisent par un surcoût mémoire important. Dans cette problématique globale, nous nous intéressons à l'optimisation des interfaces de communication entre composants De notre point de vue, une approche globale pour la conception d'adaptateur spatio-temporel doit fournir (1) une modélisation des contraintes d'E/S, (2) une analyse de celles-ci et (3) des techniques de synthèse de haut niveau pour générer l'architecture RTL du composant. C'est ce que nous proposons de présenter dans cet article. Celui-ci est organisé comme suit : nous présentons d'abord une formulation générale du problème. Puis, nous développons l'approche et la modélisation formelle que nous proposons. Enfin, nous présentons les premiers résultats que nous avons obtenus pour la conception d'un entrelaceur (brassage de données).

I. ETAT DE L'ART

Parmis les travaux concernant la synthèse d'interfaces, on peut citer ceux de [7] qui, en se basant sur des modèles (*template*) de communication, présentent une architecture générique pour la synthèse d'interfaces (approche *plateform-based*). Dans [3], les auteurs utilisent une architecture basée sur des multiplexeurs/démultiplexeurs et des FIFO. Dans ces deux approches, les auteurs supposent que les séquences de données produites sont identiques aux séquences de données consommées (pas d'adaptation sur les ordres de production/consommation des données). De plus, les FIFO sont dimensionnées par simulation (approche *set and simulate*).

Dans [8] l'approche vise à déterminer, à la compilation, si une FIFO est suffisante pour toute paire producteur/consommateur d'un réseau de processus de Khan. Quand la séquence de données produite est différente de celle des données consommées, il faut alors ajouter une couche de mémorisation et de contrôle spécifique, comme proposé

[1]OFDM : technique de modulation se basant sur le multiplexage fréquentiel de signaux.

dans [10]. Cette couche supplémentaire inclue une CAM (Content Addressable Memory) dans laquelle les données sont accédées par l'intermédiaire d'une table de hachage. Si cette implémentation permet de gérer séquences d'échanges de données non déterministes (ordre) entre les IP, elle ne permet pas de minimiser l'adaptateur puisque le recouvrement entre les entrées et les sorties n'est pas permis. Dans [4] une méthode formelle est proposée pour la conception d'interfaces matérielles. Un modèle d'interface générique, ciblé par l'outil de synthèse, est proposé. Les contraintes d'Entrées/Sorties (E/S) de bas niveau peuvent inclure des spécifications temporelles strictes ou des ordres de transfert. La synthèse d'interface est réalisée par une procédure d'allocation de composant mémoire (FIFO, LIFO, Registre) Toutefois, le dimensionnement de ces éléments n'est pas abordé.

Dans [9] les auteurs proposent une méthodologie de réutilisation d'IP pour lesquelles les circuits sont décrits en trois niveaux. Les transferts de données et les optimisations de stockage sont réalisés par réorganisation d'indices et imbrication de boucles. Malheureusement, les auteurs ne présentent pas la méthodologie utilisée pour produire le composant RTL[2] à partir de la spécification algorithmique.

Enfin, dans [6], les auteurs proposent un ensemble de techniques dédiées à la conception d'applications de type DSP. La synthèse de haut niveau de l'unité de calcul est réalisée sous contrainte d'E/S et d'architecture. L'approche produit un chemin de donnée optimisé mais requière toujours la conception séparée d'une unité de communication.

Dans le domaine des entrelaceurs, le problème du ré-ordonnancement des données est consubstantiel à ce type d'algorithme. Une architecture itérative d'entrelacement typique est composée d'un ensemble d'éléments de calcul connectés à un ensemble de bancs mémoire (RAM) à travers un réseau d'interconnexion. Le principe de l'entrelacement consiste alors à écrire les données produites par les éléments de calculs dans différents bancs mémoires, en fonction de la règle d'entrelacement. A la conception, une contrainte supplémentaire vient s'ajouter : le concepteur doit limiter au maximum les conflits d'accès aux bancs mémoires, [14]. Il existe deux grandes approches pour la conception de ces entrelaceurs : résolution des conflits à l'exécution, ou résolution des conflits lors de la conception. Une approche de résolution des conflits à la volée à été proposée dans [13]. Le principe est d'ajouter de la mémoire dans le réseau d'interconnexion, en entrée des blocs mémoires, pour temporiser les données conflictuelles. Le problème est que l'on augmente le coût en termes d'élément mémorisant, et l'on augmente la latence globale du traitement.

Les approches de résolution des conflits à la conception se classent elles aussi en deux sous-familles. Dans la première, les auteurs définissent leur propre règle d'entrelacement, pour laquelle ils sont certains de ne pas avoir à gérer de conflit d'accès mémoire. C'est par exemple la solution proposée par [11]. Le réseau d'interconnexion est simplifié grâce à l'architecture présentée, toutefois lorsque le concepteur définit sa propre règle d'entrelacement, cela signifie que celle-ci ne se conforme pas à un standard, ce qui n'est pas acceptable dans certaines applications. Une autre approche consiste à déterminer un mapping des données en mémoires pour lequel on garantit un fonctionnement sans conflit. C'est le cas de l'approche proposée dans [12]. Mais cela se fait au prix d'un réseau d'interconnexions plus complexe. De plus, l'algorithme utilisé exploite une méthode de «recuit simulé», on ne peut donc déterminer à priori quand cet algorithme va converger.

II. FORMULATION DU PROBLEME

Considérons un exemple simple : une architecture incluant deux composants qui échangent un ensemble de données $S = \{a, b, c, d, e\}$. S est produite par le bloc #1 et est consommée par le bloc #2 à travers une connexion point à point.

[2] RTL : se dit d'une modélisation synthétisable (Register Transfert Level)

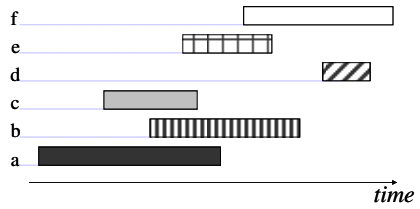

Figure 1: Durée de vie des données dans l'interface

Comme on peut le voir sur la Figure 1, l'ordre de production des données est $S_p = (a,c,b,e,f,d)$, c'est à dire $t^p_a<t^p_c<t^p_b<t^p_e<t^p_f<t^p_d$, alors que l'ordre de consommation de ces données est différent, $S_c = (c,a,e,b,d,f)$ c'est à dire $t^c_c<t^c_a<t^c_e<t^c_b<t^c_d<t^c_f$. Cette différence entre les deux séquences d'E/S peut (1) provenir de l'intégration de deux IP n'ayant pas étés conçus dés le départ pour être assemblés ensemble ou (2) être volontairement recherché dans l'algorithme, par exemple pour introduire de la redondance cyclique (pas dans cet exemple). Il est alors nécessaire d'introduire un adaptateur spatio-temporel entre les blocs #1 et #2. Ce composant supplémentaire peut être conçut en utilisant une mémoire ou une *mer* de registres (6 dans l'exemple de la Figure 1). Toutefois, ces deux premières solutions d'implémentation peuvent être améliorées en termes de surface et de performances en concevant un composant optimisé incluant des éléments mémorisant à sémantique forte (FIFO, LIFO, Registre).

II. PROPOSITION

Nous proposons une méthodologie de conception permettant de générer automatiquement un adaptateur appelé Space-Time AdapteR (STAR), [1]. Notre flot de conception prend en entrée des diagrammes temporels (fichier de contraintes) ou une description en langage C de la règle de brassage des données (par exemple une règle d'entrelacement) et des contraintes utilisateur (débit, latence…). Ce flot formalise ensuite ces contraintes de communication sous la forme d'un Graphe de Compatibilité des Ressources (GCR), dont les propriétés permettent une exploration efficace de l'espace des solutions architecturales afin de générer un composant STAR.

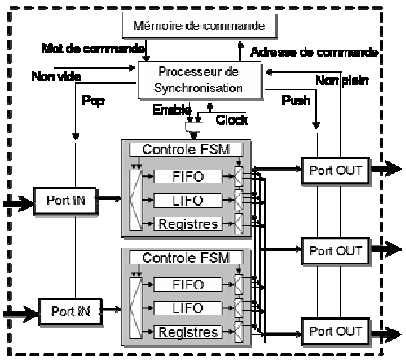

Figure 2: Architecture STAR type

L'architecture STAR (cf. Figure 2) se compose d'un chemin de données (utilisant des éléments mémorisant à sémantique forte : FIFO, LIFO et/ou des registres) et de machines d'état finis permettant de contrôler le système. L'adaptation spatiale (une donnée en peut être transmise de n'importe quel port d'entrée vers un ou plusieurs ports de sortie) est effectuée par un réseau d'interconnexion adapté. L'adaptation temporelle est réalisée par les éléments de mémorisation. Le composant STAR exploite une interface GALS (Globally Asynchronous Locally Synchronous) / LIS (Latency Insensitive System) décrite dans [5].

Le flot de conception proposé peut générer des architectures STAR pouvant intégrer plusieurs modes de fonctionnement (par exemple, plusieurs longueurs de trames pour un entrelaceur, ou bien plusieurs configurations dans une architecture multi-configurations).

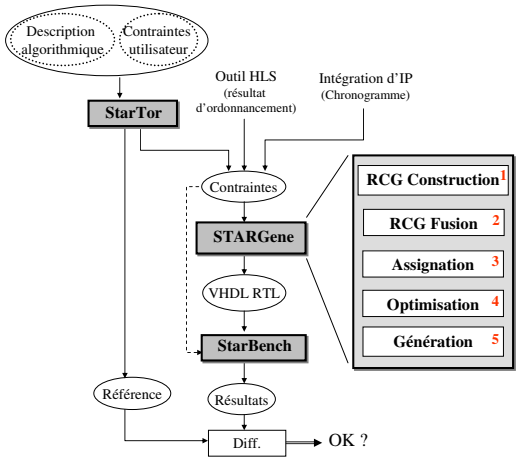

Figure 3: Flot de conception STAR et les outils associés

Le flot de conception (cf. figure 3) est basé sur trois étapes:
- La première, ***StarTor***, prend en entrée la description en langage C de l'algorithme d'entrelacement, et des contraintes de l'utilisateur (latence, débit, interface de communication, parallélisme d'entrée-sortie…). Elle en extrait l'ordre des données d'entrée-sortie en produisant d'une trace à partir de la description fonctionnelle. Ensuite, l'outil génère le fichier de contraintes de communication qui sera utilisé par l'outil *STARGene*.
- La seconde, ***STARGene***, basée sur un flot à cinq étapes, génère l'architecture STAR : (1) construction des graphes de compatibilité des ressources, à partir du fichier de contraintes, correspondant à chacun des modes de fonctionnement du design, (2) fusion des graphes de compatibilités pour chaque mode en un seul graphe multi-modes, (3) assignation des structures de mémorisation (FIFO, LIFO ou Registre) sur le GCR (4) optimisation de l'architecture et (5) génération du VHDL niveau transfert de registre (RTL) intégrant les différents modes de communication.
- La troisième, ***StarBench***, génère un test-bench basé sur les contraintes de communication et permettant de valider les architectures générées en comparant les résultats de simulation avec le comportement théoriquement attendu.

III. FLOT DE CONCEPTION (*STARGENE*)

1 - Construction du graphe de compatibilité des ressources : Les nœuds du RCG représentent les données qui doivent transiter au sein de l'adaptateur; les arcs, étiquetés et orientés, représentent les types de compatibilité entre ces données. L'étiquette est le type d'élément de stockage, T= {Registre, FIFO, LIFO}.
Afin d'assigner des étiquettes aux arcs, nous analysons les relations entre les durées de vie des données. Nous avons défini à cette fin un ensemble de règles basées sur les propriétés temporelles de chaque type d'éléments mémorisant :

- Soient τ_{min} et τ_{max} les dates d'écriture et de dernière lecture de la donnée dans l'interface,
- Soit τ_{fisrt} le premier accès en lecture à une donnée,
- Soit τ_{Ri} le $i^{ième}$ accès en lecture à une donnée,
- Soient *a* et *b* deux données,

Règle 1: *Compatibilité Registre*
Si $(\tau_{min_b} \geq \tau_{max_a})$ *alors on créera un arc étiqueté "Registre".*
Ici, les intervalles de durée de vie des données sont dits "non recouvrants". En d'autres termes, ces deux données peuvent être stockées dans le même mot mémoire.
Règle 2*: ***Compatibilité FIFO
Si $[(\tau_{min_b} > \tau_{min_a})$ *et* $(\tau_{fisrt\ b} > \tau_{max_a})$ *et* $(\tau_{min_b} < \tau_{max_a})]$ *alors on créera un arc étiqueté "FIFO".*
Dans ce cas, les intervalles de durée de vie des données sont dits "partiellement recouvrants". Cela signifie que ces deux données peuvent être mémorisées dans une même FIFO, en augmentant la taille au besoin (+1 case mémoire). Nota: la dernière relation $(\tau_{min_b} < \tau_{max_a})$ permet une distinction formelle avec une compatibilité Registre.
Règle 3*: ***Compatibilité LIFO
Si $[(\tau_{min_b} > \tau_{min_a})$ *et* $(\tau_{fisrt\ a} > \tau_{max_b})]$ *ou* $[(\tau_{Ri_a} < \tau_{min_b} < \tau_{max_b} < \tau_{Ri+1_a})]$ *alors on créera un arc étiqueté "LIFO".*
Dans ce cas, les intervalles de durée de vie des données sont dits "recouvrants-incluants".

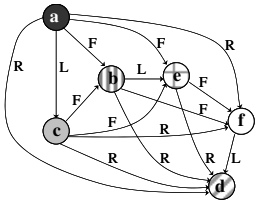

Figure 4: RCG généré pour l'exemple de la figure 1

La figure 4 présente un exemple de graphe de compatibilité des ressources. Celui si a été généré à partir des contraintes de communication présenté dans la figure 1.

2 – Fusion des graphes : Dans le cas d'architectures devant gérer plusieurs modes de fonctionnement au sein d'un même adaptateur, l'approche que nous avons retenue consiste à fusionner, à moindre coût architectural, les différents graphes au sein d'un GCR unifié. Nous avons pour cela définis un algorithme de fusion qui s'inspire de l'algorithme de Kernighan/Lin [17].

3 – Assignation des éléments mémorisant :
- *Identification des ressources* : L'identification des éléments de mémorisation est facilitée par les propriétés intrinsèques de nos arcs étiquetés et par l'utilisation de la notion de chemin dans un graphe. En effet, parcourir un chemin d'un type donné (FIFO ou LIFO) dans notre RCG est équivalent, par construction, à la recherche d'une clique de compatibilité. Ainsi, notre modèle nous permet de nous passer d'un algorithme de recherche de clique maximum (NP-complet).
- *Dimensionnement des ressources* : La taille d'une structure FIFO ou d'une LIFO n'est pas égale au nombre de sommets sur le chemin correspondant. De fait, pour dimensionner correctement ces éléments, il faut identifier le recouvrement maximal de données dans ce même chemin. Notre modèle de représentation, basé sur un graphe GCR, facilite cette exploration :

Soit ***P*** le plus grand chemin de compatibilité FIFO,
Soit ***i*** un noeud du graphe appartenant à *P*,
Soit S_i le nombre d'arc étiqueté FIFO entrant dans le nœud *i* et provenant d'un autre nœud du chemin *P*,
Alors,

Profondeur = 1 + max ({S_i | pour tout noeud *i* de *P*}).

En effet, le problème revient à identifier le nombre maximum de données recouvrantes (en fonction des contraintes E/S) dans le chemin *P*.

- *Assignation des ressources* : Nous proposons un algorithme glouton, basé sur un ensemble de paramètres réglés par l'utilisateur (taille minimum pour les FIFO/LIFO, complexité du multiplexage, taux de remplissage…), pour assigner autant de structures FIFO ou LIFO possibles sur le GCR. Nous utilisons alors un algorithme en deux étapes : (1) identification d'un élément mémorisant, (2) fusion des nœuds mis en jeu au sein d'un nœud hiérarchique. Naturellement, le choix des noeuds à fusionner influence considérablement l'architecture résultante ; de même, les choix de l'utilisateur concernant le réglage des différentes métriques sont de première importance.

4 - Optimisation de l'architecture : Enfin, l'étape d'optimisation vise à maximiser l'utilisation des éléments mémorisant et par là même, réduire le nombre de structures à contrôler. Cette optimisation peut être réalisée à l'aide d'une heuristique dédiée, basée sur un ensemble de métriques (Taux d'utilisation moyen des structures, complexité du

# min.	Usage factor	# structures					# memory points	Throughput
		Mux	FIFO	LIFO	Register	Total	Used	
4	90%	48	45	5	22	174	514	272,2
7	90%	34	34	1	90	159	525	272,2
15	90%	12	12	0	381	415	581	272,2
30	90%	0	0	0	600	600	600	272,2
7	50%	28	27	2	73	130	533	272,2
7	70%	28	28	2	70	129	532	272,2
7	100%	38	38	1	84	162	522	272,2

Table 1: Exploration architecturale pour le mode 600 données

réseau d'interconnexion…), ou bien à l'aide d'un algorithme de type Left-Edge [15]. En fonction de l'algorithme d'optimisation choisi, l'architecture résultante sera optimisée en termes de composants de multiplexage ou de registres.

IV. RESULTATS

Dans cette expérience, le flot STAR a été utilisé pour générer une architecture de type entrelaceur Ultra-Wide Band [16]. Il s'agit là d'un cas d'étude industriel dans le cadre d'une collaboration avec la société STMicroelectronics. Nous avons dans un premier temps cherché à explorer l'ensemble de solution architecturale. Puis, dans une seconde étape nous avons comparé les résultats obtenus avec une architecture d'entrelacement générée à l'aide d'un outil de synthèse de haut niveau du commerce. L'application cible doit être capable de gérer trois modes d'entrelacement en fonction de la longueur de la trame traitée (300, 600 ou 1200 données).

La table 1 regroupe un ensemble d'architectures obtenues pour un parallelisme d'entrée/sortie fixé (6 données en entrée et 5 en sortie). Ces expériences ont étés menées en faisant varier deux paramètres : le nombre minimum de données pour autoriser l'assignation d'un élément FIFO ou LIFO sur le graphe ; et le taux d'utilisation minimum (appelé *usage factor*) pour des FIFO/LIFO. En jouant avec ces deux facteurs, notre approche est capable de générer des architectures complètement différentes, optimisant plus ou moins, soit le nombre d'élément mémorisant, soit le nombre de structures à piloter. Nous aurions également put combiner les effets de ces paramètres en les faisant varier en même temps, ou bien faire varier d'autres paramètres. En d'autres termes, l'exploration architecturale se fait par l'intermédiaire de l'exploration des paramètres.

Mode	Ref		FL 7 (Tx 95%)		FL 15 (Tx 90%)		no FL		Throughput
	Saved	Ctrl	Saved	Ctrl	Saved	Ctrl	Saved	Ctrl	
300	0	300	56	77	60	240	60	240	434,8
600	0	600	83	101	130	470	130	470	438
1200	0	1200	96	117	120	609	168	1032	412,4

Table 2: Comparaison d'architectures STAR avec une architecture de référence

Dans la table 2, nous comparons des architectures d'entrelacement pour les trois modes, en termes de points mémoires économisés et de nombre de structures à piloter. Nous avons pour cela comparé l'architecture de référence générée avec un outil de synthèse de haut niveau du commerce avec trois architectures STAR : deux architectures STAR classiques, en faisant varier deux métriques, et une mer de registre, générée et optimisée avec notre approche. Nous pouvons constater que nous sommes capables de réduire le nombre de points mémoires utilisés et diminuer la latence, par rapport aux approches classiques basées sur des bancs mémoires. De plus, lorsque nous utilisons notre approche, le nombre de structures à piloter est inférieur à celui de l'architecture de référence, obtenue à l'aide d'un outil de synthèse de haut niveau du commerce. Actuellement, la surface totale de notre architecture d'entrelacement est environ 14% plus petite que l'architecture de référence de STMicrolectronics.

CONCLUSION

Nous avons présenté une architecture (STAR) permettant de réaliser l'adaptation de la communication entre deux éléments de calcul. Nous avons également présenté le flot de conception associé. L'approche proposée repose sur un modèle de graphe original (RCG) permettant d'explorer efficacement l'espace des solutions architecturales. Ce flot peut être appliqué pour générer des composants permettant le réordonnancement de données et pour divers champs d'application : intégration d'IP, entrelaceur ou chemin de données reconfigurables (non présenté dans ce papier).

Les résultats montrent la pertinence des solutions retenues puisque les architectures générées se révèlent jusqu'à 14% plus petites que celles générées à l'aide d'outils de synthèse du commerce. Les travaux en cours visent à finaliser l'intégration des architectures pipelines dans le flot STAR. Nous avons montré que l'exploration de l'espace des solutions architecturales se faisait par l'intermédiaire de l'exploration des paramètres. Ce point peut se révéler une difficulté pour l'utilisateur, c'est pourquoi nous envisageons d'explorer la faisabilité d'un outil d'exploration des métriques basé sur un ILP (Integer Linear Programming). Enfin, nous souhaitons également étendre les expériences réalisées et comparer les architectures obtenues avec d'autres outils de synthèse de haut niveau du commerce.

REFERENCES

[1] C. Chavet, P. Coussy, P. Urard et E. Martin, "A Design Methodology for Space-Time Adapter", *GLSVLSI*, 2007.

[2] L. Chiou, S. Bhunia et K. Roy, "Synthesis of Application-Specific Highly Efficient Multi-mode Cores for Embedded Systems", *ACM Transactions on Embedded Computing Systems*, Février 2005.

[3] F. Abbes, E. Casseau, M. Abid, P. Coussy, J-B. Legof, "IP integration methodology for SoC design", ICM04 International Conference on Microelectronics, Tunis, 2004.

[4] A. Baganne, J-L. Philippe, E. Martin, "A Formal Technique for Hardware Interface design", IEEE Trans. On Circuits And Systems, Vol.45, N°5, 1998.

[5] P. Bomel, E. Martin, E. Boutillon, "Synchronization Processor Synthesis for Latency Insensitive Systems", DATE'05, Munich, 2005.

[6] P. Coussy, E. Casseau, P. Bomel, A. Baganne, E. Martin, "A Formal Method for Hardware IP Design and Integration under I/O and Timing Constraints", To appear in ACM Transaction on Embedded Computing Systems, 2005.

[7] D. Hommais, F. Pétrot, I. Augé , "A Practical Toolbox for System Level Communication Synthesis", 9th IEEE International Symposium on Hardware/Software Co-design, CODES, 2001.

[8] A. Turjan, B. Kienhuis and E. Deprettere, "A compile time based approach for solving out-of-order communication in Kahn Process Networks", 13th IEEE International Conference on Aplication-specific Systems, Architectures and Processors (ASAP), 2002.

[9] F. Vermeulen, F. Catthoor, D. Verkest, H. De Man, "Formalized Three-Layer System-Level Model and Reuse Methodology for Embedded Data-Dominated Applications", IEEE Transaction on VLSI Systems, Vol.8, N° 2, 2000.

[10] C. Zissulescu-Ianculescu, A. Turjan, B. Kienhuis and E. Deprettere, "Solving Out of Order communica-tion using CAM memory: an implementation", ProRisc02, Veldhoven, 2002.

[11] D. Gnaedig, E. Boutillon, M. Jezequel, V. C. Gaudet, et P. G. Gulak, "On multiple slice turbo codes," in *Proc. 3rd Int. Symp. Turbo Codes, Related Topics*, Brest, 2003, pp. 343–346.

[12]A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architectures", *IEEE Trans. Inf. Theory*, vol. 50, no.9, pp.2002-2009, Sep. 2004.

[13]M. J. Thul, F. Gilbert, et N.Wehn, "Optimized concurrent interleaving architecture for high-throughput turbo-decoding," in *Proc. 9th Int. Conf. Electron., Circuits, Syst.*, vol. 3, pp. 1099–1102, 2002.

[14]A.Giulietti, L. Van Der Perre et M. Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements", *Electronics Leters, vol. 38, no. 5, pp.232–234*, Feb. 2002.

[15] A. Hashimoto et J. Stevens, "Wire routing by optimizing channel assignment within large apertures," in *Proceedings of Design Automation Workshop*, pp. 155-169, 1971.

[16]IEEE 802.15.3a, *WPAN High Rate Alternative*

[17] B. Kernighan et S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Systems Technical Journal, 49(2),* pp. 291-308, 1970.