

Méthodologie de conception automatique pour multiprocesseur sur puce hétérogène

Xinyu Li

▶ **To cite this version:**

Xinyu Li. Méthodologie de conception automatique pour multiprocesseur sur puce hétérogène. domain_other. Université Paris Sud - Paris XI, 2009. Français. NNT : . pastel-00005864

HAL Id: pastel-00005864
https://pastel.hal.science/pastel-00005864

Submitted on 18 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° D'ORDRE 9609





THÈSE DE DOCTORAT

SPECIALITE : PHYSIQUE

Ecole Doctorale « Sciences et Technologies de l'Information des Télécommunications et des Systèmes »

Présentée par : Mr. Xinyu LI

Sujet :

Méthodologie de Conception Automatique pour Multiprocesseur sur puce Hétérogène

Soutenue le ……5 Nov. 2009……..devant les membres du jury :

M. COLLETTE Thierry, Examinateur, Directeur de recherches, LIST CEA SACLAY, France

M. ERNST Rolf, Rapporteur, Professeur, Université de Braunschweig, Allemagne

M. GAUDIOT Jean-Luc, Examinateur, Professeur, Université de California, USA

M. HAMMAMI Omar, Co-directeur, Enseignant-chercheur, ENSTA, France

M. JERRAYA. Ahmed, Rapporteur, Directeur de recherches, CEA-LETI, MINATEC, France

M. MERIGOT Alain, Directeur, Professeur, Université Paris-Sud, Orsay, France

ACKNOWLEDGMENT

I would first like to extend my deepest gratitude and appreciation to my advisor Prof. Omar Hammami at ENSTA ParisTech, for the confidence he has shown me and all his valuable time to lead this thesis. I thank him also for all the technical discussions, for the valuable advices he gave me and for everything he taught me during this thesis and in the preparation of this manuscript. His guidance and instruction has played an invaluable part in both my graduate studies and PhD Work.

I would also like to thank Prof. Alain Merigot, my advisor at IEF for all the support, discussion and suggestion.

I am really honored to have Dr. A.A.Jerraya, CEA LETI and Prof. R.Ernst, IEEE Fellow, as my Phd thesis reviewers. I am very grateful to them for their comments which greatly helped me improve my manuscript.

I am really honored and grateful to have Prof. Gaudiot, IEEE Fellow and AAAS Fellow, University of California, Irvine, for accepting being member of this Phd committee and for presiding this Phd committee.
I am very grateful Dr. T.Collette CEA Saclay and Dr. Liu Dong (Intel Research Labs., China) for being members of my PhD committee and I am thankful to them for providing me valuable feedback on my research work.

It has been a pleasure to work with my colleagues at ENSTA ParisTech. I would specially like to thank Zhoukun Wang, Guangye Tian and Mazen Khaddour for sharing their experiences in all ups and downs during these three years.

I would like to thank EVE and Arteris companies for their excellent cooperation.

Almost last, but certainly not least, I would like to thank my entire family for the tremendous love and support they have shown me during my entire life. I am thankful to them for

teaching me the value of hard work and dedication. And finally I want to thank my wife Dong Xin for her encourage and support during my PhD work.

ABSTRACT

ITRS Semiconductor roadmap projects that hundreds of processors will be needed for future generation multiprocessor system on chip (MPSOC) designs. Current research topics contain modelling of multiprocessors and adequate levels of abstraction (TLM, RTL), performance evaluation and design space exploration, verification and test trough simulation or emulation. Design productivity is one of the most important challenges, which is a relatively new and open research issue. We propose to improve design productivity by raising IP reuse level to small scale multiprocessor (SSM) IP and by combining fast extension techniques for system level design automation in the framework of multi-FPGA emulator.

In the thesis, different state-of-art NoC and MPSoC design methodologies are analyzed and compared to better understand the design approaches and to overcome their shortcomings. Then a fully automatic multi-objective design workflow is proposed for network on chip (NoC) at TLM (Transaction Level Modeling) level. The timing and area criteria extracted from RTL level are explored but not limited using the TLM NoC models of NoCexplorer, tool from Arteris. A linear programming methodology is provided as a solution for the organization and dimensioning of eFPGA reconfigurable area to maximize the efficiency of network on chip mapping.

The main contribution is the automatic design flow for large scale MPSoC design based on the reuse of SSM IP. Based on it, an automatic design flow is proposed for data parallel and pipelined signal processing applications on multiprocessor with NoC, using cryptographic application TDES (Triple Data Encryption Standard) as an example. High level synthesis tool is used to generate hardware accelerators, which are added to explore the tradeoff in area-performance while still privileging multiprocessor basis for the implementation. OCP-IP NoC benchmarks are executed on the generated 48-core and 672-core multi-processor for performance evaluation.

All the work done in this thesis is the basis of “MPSOC explorer”, an ongoing industrial project for large scale MPSoC design exploration supported by European Union and French government.

Résumé

La feuille de route d'ITRS Semi-conducteur prévoit que des centaines de processeurs seront nécessaires pour les futures générations du multiprocesseur (MPSoC). La modélisation des multiprocesseurs, le niveau adéquat d'abstraction (TLM, RTL), l'évaluation de la performance et l'exploration d'espace de conception, la vérification et la simulation ou l'émulation sont les sujets actuels de recherche. L'efficacité de conception qui est l'un des défis les plus importants, est un problème de recherche relativement nouveau et ouvert. Nous proposons d'améliorer l'efficacité de conception en augmentant la taille d'IP SSM, et en combinant les techniques d'extension rapide au niveau du système avec multi-FPGA émulateur.

Dans la thèse, avoir analysé et comparé les différentes méthodes pour la conception de NoC et de MPSoC, nous proposons une procédure automatique et multi-objective pour NoC au niveau TLM (Transaction Level Modeling). Les critères du timing et de surface du niveau RTL sont explorés mais non limités avec des TLM modèles du NoC dans NoCexplorer. Une méthodologie de la programmation linéaire est fournie comme solution au problème de l'organisation et du dimensionnement de eFPGA reconfigurable pour maximiser l'efficacité du NoC.

Notre contribution principale est la procédure automatique pour la conception de MPSoC à grande taille basée sur la réutilisation de SSM IP. Basée sur ce principe, une procédure de conception automatique pour des données parallèles et des traitements en pipeline est proposée pour l'application au traitement du signal sur le multiprocesseur avec NoC, utilisant l'application cryptographique au TDES (Triple Data Encryption Standard) comme un exemple. La synthèse de haut niveau est ajoutée à cette procédure pour la génération de hardware accélérateur, qui permet d'étudier le compromis entre la performance et la surface. OCP-IP NoC benchmarks sont exécutés sur notre multiprocesseur de 48 coeurs et de 672 coeurs pour l'évaluation de performance.

Tous les travaux réalisés dans cette thèse rendent possible MPSOC explorer, un projet industriel pour l'exploration de MPSoC à grand taille, soutenu par l'Union Européenne et le gouvernement français.

1. Introduction

La feuille de route d'ITRS Semi-conducteur prévoit que des centaines de processeurs seront nécessaires pour les futures générations du multiprocesseur (MPSoC). La densité croissante du dispositif permet exponentiellement plus de cœurs sur une seule puce. Les fabricants de processeur ont décalé vers la production des processeurs multi-coeurs pour respecter les contraintes de la puissance et du rafraîchissement tout en maintenant les avancées de performance d'exécution attendues avec chaque nouvelle génération de processeur. Le processeur à 8 coeurs de l'Intel est attendu d'ici 2009 et la performance de Tflops avec 80 coeurs en 45 nm technologie a déjà été démontrée. Des processeurs graphiques ont déjà des centaines de cœurs, tels que le récent GeForce 295 du NVIDIA avec 480 coeurs en 55 nm technologie. D'ailleurs, ITRS prévoit que la même tendance continuera également, et qu'une amélioration jusqu'à dix fois meilleure sur la productivité de conception sera nécessaire au cours des dix prochaines années, jusqu'en 2019 afin de maintenir l'effort de conception constante.

1.1 Motivation

En conséquence, il y a deux grands défis dans la conception de la nouvelle génération de MPSOC :

1. Comment améliorer la productivité de la conception afin de réduire le temps de mise en marché (TTM) de système électronique qui est de plus en plus complexe?
2. Comment s'assurer que le projet de conception actuel est adaptable à la technologie de semi-conducteurs qui évolue rapidement?

La productivité de la conception du système sur puce est le défi majeur en matière de technologie de conception. L'écart de productivité de la conception représente le fait que la loi de Moore génère un certain nombre de transistors disponibles, qui croît plus vite que la capacité à les utiliser d'une manière significative. La complexité de silicium et la complexité des systèmes sont à l'origine de cette croissance exponentielle de l'écart de productivité de conception. La complexité de silicium est le résultat des propriétés physiques de la technologie des semi-conducteurs et de la agrandissement de l'interconnexion globale. Les défis associés à la complexité des systèmes sont la réutilisation, la vérification et le test, ainsi

que l'optimisation de la conception axée sur la rentabilité, la conception de logiciels embarqués, les plateformes d'implémentation fiable, et la gestion des processus de conception.

1.2 Les objectifs de recherche

Des nouvelles méthodologies de conception efficaces sont nécessaires pour surmonter les complexités des systèmes et du silicium afin de remplir les écarts de la productivité de conception croissante. Pour atteindre cet objectif, on propose trois stratégies :

1. la combinaison de divers niveaux de conception du système.
2. la réutilisation des IPs et des composants.
3. l'utilisation des nouvelles technologies.

Dans cette thèse nous présentons une nouvelle méthodologie qui implémente ces 3 stratégies pour résoudre les défis de conception.

Cette nouvelle méthodologie combine ensemble différents niveaux de conception pour prendre les avantages de chaque niveau et surmonter le défaut séparé de chaque niveau. De cette façon, nous pouvons tirer profit des modèles au niveau d'abstraction élevé de la conception de haut niveau du système pour passer au niveau RTL pour alimenter l'information électronique du système autant que possible. Cette méthode garantit aussi que la conception du système est toujours adaptative à la nouvelle technologie de semi-conducteurs. De plus, les développeurs avec des simulateurs de système font face à "un mur de simulation" à cause du temps de simulation des systèmes avec des centaines ou plus de cœurs. L'émulation du CMP de grande taille sur la plate-forme multi-FPGA est une des solutions proposées pour accélérer l'exploration de l'espace de conception des systèmes.

Cette nouvelle méthodologie réutilise les blocs et composants d'IP existants pour accélérer le processus de conception et faciliter la vérification du système. Jusqu'à présent, les IPs réutilisables sont toujours trop élémentaires pour construire rapidement des multiprocesseurs de grande taille. Il est alors nécessaire d'augmenter la taille et la complexité d'IPS que nous appelons le multiprocesseur à petite échelle (small scale multiprocessor SSM) IPs. La conception de multiprocesseurs de grande taille basées sur des SSM IP permet la duplication

et la construction rapide en un temps raisonnable utilisant l'émulation sur multi-FPGA pour une validation et une évaluation rapide de performance.

De nouvelles technologies pour la conception de MPSOC sont appliquées dans cette nouvelle méthodologie :
- le Réseau sur la Puce (NoC), une architecture de communication est utilisée pour régler le problème d'évolutivité du système à grande taille.
- Synthèse de Haut Niveau (HLS), outils de génération de VHDL coprocesseur accélèrent la conception et la réutilisation de la IP.
- La technologie de FPGA reconfigurable embarquée permet la mise en œuvre de NoC reconfigurable sur ASIC.
- la plate-forme multi-FPGA rend l'émulation de système de grande taille réalisable.
La mise en oeuvre complète de ces stratégies et technologies dans notre nouvelle méthodologie constitue une bonne solution pour améliorer la productivité de conception du système et sa fabrication.

2. Définition et état de l'art

Le MPSoC (Multiprocessor System on Chip) est différent du multi-cœur distribué ou le multiprocesseur, parce que tous les éléments de traitement sont intégrés sur une puce. La différence majeure vient de l'architecture de communication. Le multiprocesseur est connecté par un réseau d'interconnexion externe avec une grande bande passante et une latence haute ; tandis que la communication sur puce de MPSoC doit être rapide et la gestion de réseau doit être simple et efficace. Les couches d'intégration à très grande échelle (VLSI) fournissent de nombreux fils pour transférer des signaux de contrôle et des données. La proximité locale des éléments de traitement et des mémoires accélère le transport. Mais le compromis doit toujours être fait entre la surface, la performance et la consommation d'énergie.
Le Réseau sur puce (NoC) est une nouvelle méthode pour les communications au sein de grands systèmes VLSI mis en œuvre sur une seule puce de silicium. Comme la complexité de systèmes intégrés continue à grandir, le NoC fournit la performance améliorée et l'évolutivité en comparaison avec des solutions simples de communication sur puce tels que le point-à-

point et le bus partagé. Avec l'avènement du multiprocesseur de grande taille, le NoC est un choix naturel pour la conception d'architecture. Il peut offrir une séparation entre le calcul et la communication, soutenir la modularité et la réutilisation de la IP via des interfaces standards, gérer des problèmes de synchronisation, servir de plateforme pour le test du système et partant, accroître la productivité de conception.

Le NoC est un réseau de communication qui est utilisé sur une puce. Il est construit de liens multiples de interconnectés par des routeurs. Les données peuvent être transférées de la source à la destination sur plusieurs liens, en faisant la décision de routage sur les routeurs. Un haut niveau de parallélisme est obtenu, parce que tous les liens dans le NoC peuvent fonctionner simultanément sur les différents transferts de données.

Un NoC élémentaire se compose de routeurs, de liens et d'interfaces de réseau. Les routeurs dirigent des données sur plusieurs liens selon le politique de routage. Les connexions logiques des liens sont mentionnées comme la topologie de réseau. L'interface de réseau (l'adaptateur) doit découpler le calcul (les ressources) de la communication (le réseau). Chaque cœur IP est connecté au NoC par une interface de réseau.

Les mesures les plus importantes des NoCs sont la bande passante, la surface de silicium, la consommation d'énergie, et la latence. Tous ceux-ci doivent être réduits au minimum. Une solution de Pareto est prévue pour l'exploration à grande échelle.

Différentes méthodes d'évaluation peuvent être utilisées pour mesurer la performance du système: l'analyse basée sur modèle, la simulation et l'exécution sur des puces réelles. L'analyse mathématique est rapide, mais peu précise. Elle peut être utilisée à la première étape pour la vérification rapide et pour enlever les options inutiles et diminuer l'espace d'exploration. La simulation SystemC est largement utilisée dans la recherche. Lorsque le système devient de plus en plus complexe, la vitesse de simulation n'est plus suffisante. L'émulation sur plate-forme FPGA est proposée comme solution pour l'exploration à grande échelle. Nous soutenons que le temps d'exécution doit être mesuré en temps réel plutôt que le nombre de cycle. La fréquence du système doit être mesurée pour des résultats de simulation.

La communication entre processeurs est importante pour la conception de MPSoC. Le bus est une architecture d'interconnexion traditionnelle pour la conception de système sur puce (SoC). L'AMBA bus d'ARM et le CoreConnect bus d'IBM sont les choix connus pour la conception de processeur commercial. Le bus d'OCP (Open Core Protocol) est proposé

comme un moyen efficace pour simplifier l'interconnexion par la standardisation de protocole d'interface. Comme le nombre de processeur dans des MPSoCs grandit exponentiellement, le réseau sur puce (NoC) est proposé comme la seule solution pour la bande passante de communication requises, l'évolutivité de conception et l'énergie limitée.
Des multiprocesseurs homogènes et hétérogènes sont les deux branches importantes et distinctes de la conception de MPSoC. Les communications inter processeur sont très importantes pour la conception. Jusqu'à présent, il n'y a pas beaucoup de conceptions de MPSoC qui sont basées sur l'architecture de NoC. Une étude montre qu'il n'y a aucune procédure de conception pour les MPSoCs plus grands de 32 cœurs interconnectées par la technologie de NoC.

3. Exploration de conception multi-objectif de NoC au niveau TLM

A la première étape de la thèse, on propose une exploration de l'espace pour la conception multi-objectif de NoC au niveau TLM. L'exploration automatique est nécessaire afin de garantir l'évaluation de toutes les solutions possibles. Bien que certains travaux ont été réalisés dans ce domaine, l'explorations de l'espace de conception proposées sont basées sur différents niveaux d'abstraction. Les modèles de SystemC TLM cachent beaucoup de détails d'implémentation de bas niveau. Il permet une rapide simulation des systèmes complexes au prix de moins de précision. Comment gérer l'exploration de l'espace de conception avec ce manque de précision est l'objectif de ce travail. L'exploration de l'espace de la conception multi-objectif devient un défi parce que la surface et timing du système doivent être extraites des niveaux plus bas de l'abstraction. La représentation au niveau TLM de NoC exige la perspicacité profonde et l'expérience d'implémentation pour interpréter correctement la sémantique correspondante à ce niveau d'abstraction.
La version 2.0 de SystemC TLM a été publiée récemment et elle permet la modélisation du système sur puce au plus haut niveau d'abstraction. Comme le SystemC TLM sont basées sur les transactions, la communication entre les IPs doit été effectuée au niveau des transactions. L'analyse d'architecture peut utiliser les trois styles de codage disponibles qui sont unlimited, loosely-timed et approximately-timed.

3.1 Les modèles de surface

Nous avons utilisé dans cette étude des outils de conception industrielle : NoC Solution de l'Arteris. Il contient deux outils de CAO: NoCexplorer et NoCcompiler, qui se concentrent sur les différents niveaux de simulation. NoCexplorer est un outil de génération et simulation système utilisant le langage SystemC TLM. Les modèles cycle basé de l'Arteris accélère la vitesse de simulation. NoCcompiler peut être utilisé pour générer des codes en VHDL ou SystemC RTL pour NoC.

À ce niveau TLM, la topologie de NoC est représentée par des 'liens' (links). Un lien est caractérisé par son horloge et sa largeur, et il est éventuellement associé à une capacité de mémoire tampon FIFO. Les liens portent les paquets de requête et réponse entre chaque paire de source et destination. N'importe quelle type de topologie, régulière ou irrégulière, peut facilement être décrite utilisant des 'liens'. Pour chaque paire de source et de destination, nous décrivons la séquence des liens comme son routage, dans lequel passe la communication. Pour définir l'architecture du NoC, dans le script de NoCexplorer, on décrit le routage sur un réseau de liens au lieu de routeurs. Le point où se mêle de liaisons représente un routeur au niveau RTL, qui n'est pas un module au niveau TLM. Le NoC et les modèles d'esclave et de maître sont décrits dans un fichier de script comme l'entrée de NoCexplorer.

Après une simulation rapide au niveau SystemC TLM, NoCexplorer donne un rapport de la performance du système. Ensuite, nous pouvons transférer la topologie de NoC à NoCcompiler, pour la simulation et l'implémentation au niveau RTL. Les performances, incluant la latence et la bande passante sont prises comme les fonctions d'objectifs dans l'exploration. La méthodologie de conception décrite a été appliquée à une étude de cas de taille significative.

On construit les modèles de surface au niveau TLM pour les liens et routeurs utilisant le rapport des surfaces au niveau RTL estimée par NoCcompiler. La surface est calculée dans l'unité de porte NAND2. Les modèles de NoC au niveau TLM cachent beaucoup de détails d'implémentation comparés aux modèles correspondants de niveau RTL et ce pour obtenir une simulation rapide. Nous pouvons construire des modèles de surface de niveau TLM pour l'estimation. Le composant de RTL est mis aux mêmes configurations que le modèle TLM correspondant, si ces options sont présentés au niveau TLM. Sinon, les options de composant de RTL restent par défaut. Dans cette étude, la profondeur des liens et le nombre d'ES de

routeur sont modifiées en fonction des différentes configurations de l'exploration. En conséquence, nous changeons la capacité du tampon FIFO du composant et le nombre d'ES pour trouver les relations entre la consommation de surface et ces variables.

Selon les données de l'estimation des surfaces de NoCcompiler, la proportion entre la capacité de FIFO et la consommation de surface est linéaire, qui peut être présenté comme:

$$gates = \begin{cases} 0, depth = 0 \\ 372 * depth - 30, depth >= 1 \end{cases}$$

où le 'gates' représente la surface de lien et le 'depth' représente la capacité de FIFO de ce lien.

La relation entre la surface et le nombre d'ES du routeur est plus complexe. Grâce à une analyse numérique des données, la relation entre la surface du routeur et son nombre d'ES est:

$$Gates = 72*X*Y+273*Y+39*X+18$$

où le 'gates' représente la surface ; le 'X' représente le nombre d'entrée de ce routeur et le 'Y' représente le nombre de sortie de ce routeur.

3.2 Multi-objective NOC TLM DSE

La procédure de conception, NOCDEX2 est décrite ci-dessous.

NOCDEX2

générer des populations de configuration du NoC aléatoirement par la modification du scénario

while (les critères de terminaison non atteint)

for (toutes les configurations du NoC)

simuler au niveau TLM et enregistrer les performances

estimer la surface

classer toute les configurations

générer une nouvelle génération de NoC

analyser le front de Pareto final

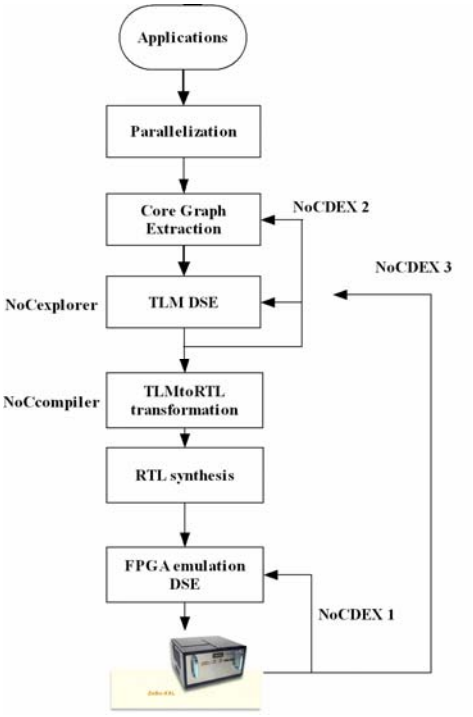


Figure 1 la procédure de conception NoCDEX 3

Combinant NoCDEX 2, notre procédure d'exploration au niveau TLM et NoCDEX, notre procédure d'émulation basée sur la plateforme FPGA, nous proposons une procédure complète de la conception du niveau TLM au RTL, NoCDEX3 qui est montrée dans la figure 1.

Après la parallélisation, le core graphe est extrait d'applications réelles. D'abord NoCDEX 2 est utilisé pour trouver des solutions de Pareto, qui sont utilisées pour l'émulation précise de NoCDEX 1. Si les résultats ne répondent pas à l'objectif de la conception, les résultats sont retournés à la simulation de haut niveau pour la nouvelle parallélisation d'application.

4. NoC reconfigurable sur eFPGA

Le multiprocesseur système sur puce (MPSoC) devrait être utilisé pour des applications multiples qui pourraient présenter des modèles de communication distincts. Comme le nombre d'IP augmente exponentiellement, le problème le plus important de la communication sur puce est de garantir la qualité du service. Le réseau sur puce (NoC) offre une solution éprouvée pour la communication des systèmes sur puce (SoC) complexes. Plusieurs conceptions ont été réalisées. Toutefois, peu d'études ont été faites sur la conception de NoC

pour des demandes d'applications multiples. La conception d'un NoC efficace commun pour ces demandes d'applications multiples pourrait être impossible en raison des exigences divergentes.

Le NoC reconfigurable est une solution potentielle pour ce problème, parce que le réseau est reconfiguré avant l'exécution d'applications afin de répondre aux besoins spécifiques des applications multiples. L'implémentation de cette reconfiguration pourrait être faite en utilisant le circuit d'eFPGA (embedded FPGA). Nous proposons une méthodologie pour spécifier la dimension de secteur d'eFPGA reconfigurable pour NoC. Les résultats d'expérience montrent l'efficacité de notre approche.

L'avantage majeur d'eFPGA est sa capacité de faire des changements après la fabrication du circuit SoC. Sa reconfigurabilité faite sur l'eFPGA est approprié pour des composants sur puce, comme des accélérateurs hardware pour les processeurs afin d'accélérer les applications embarquées, des unités de cryptage des données dans les appareils sans fil qui ont besoin d'être changées de temps en temps, des interfaces d'entrée-sortie pour la transmission de données, les routeurs de NoC qui doivent changer de configuration et routage pour s'adapter aux trafics dynamiques. Les avantages de cette approche permettent d'approvisionner des clients différents avec une seule puce programmable qui peut accommoder des changements des standards ou des spécifications.

4.1 La définition du problème de NoC reconfigurable

Nous supposons comme données du problème que la bibliothèque de fréquence et de surface pour chaque configuration du routeur dans le NoC est disponible. L'objectif est de trouver un NoC hétérogène de haute performance sous contraintes de surface totale d'eFPGAs.

Entrée: (1) un NoC avec N routeurs, (2) la contrainte de surface totale d'un ou plusieurs eFPGAs disponible.

Sortie : le choix de configuration pour chaque routeur dans le NoC et la position de chaque router sur les eFPGA, si plusieurs eFPGAs sont utilisés.

Contraintes: la surface totale des routeurs sur chaque eFPGA ne peut pas dépasser la surface maximum de cet eFPGA où ces routeurs sont placés.

4.2 La solution de programmation linéaire et l'algorithme

Nous allons placer un NoC avec N routeurs sur K eFPGAs. Sur chaque eFPGA, il y a LUT_k luts et RAM_k rams (k = 1,...,K). Les routeurs peuvent être placés sur n'importe quel eFPGA. Nous devons trouver la configuration et la position de chaque routeur pour maximiser les fréquences du NoC.

Nous introduisons une variable binaire $x_{i,j,k}$ pour représenter le choix de la configuration *j* et la placement *k* du routeur *i* :

$X_{i,j,k} \in \{0,1\}$ pour i = 1,…,N ; j = 1,…,M et k = 1,…,K.

$x_{i,j,k}$ = 1, si routeur i est fixé à sa configuration j et il est mis sur le eFPGA k. Sinon, $x_{i,j,k} = 0$

Mettez $lut_{i,j,k}$ et $ram_{i,j,k}$ comme le nombre de luts et rams de switch *i,* si le routeur est fixé à sa configuration j et il est mis sur le eFPGA k. Et $f_{i,j,k}$ représente la fréquence du routeur i. La formulation ILP de ce problème est la suivante:

Max: $$\sum_i \sum_j f_{i,j,k} \cdot x_{i,j,k} \qquad (1)$$

s.t. $$\sum_j \sum_k x_{i,j,k} = 1, \forall i \qquad (2)$$

$$\sum_i \sum_j lut_{i,j,k} \cdot x_{i,j,k} \le LUT_k, \forall k \qquad (3)$$

$$\sum_i \sum_j ram_{i,j,k} \cdot x_{i,j,k} \le RAM_k, \forall k \qquad (4)$$

L'objectif est de maximiser la fréquence de tous les routeurs dans (1). La contrainte (2) permet de s'assurer que chaque routeur est fixé à une seule de ses configurations et est placé à un seul eFPGA. Dans la contrainte (3), nous nous assurons que les luts totaux de tous les routeurs sur eFPGA k ne dépasseront pas le maximum LUT_k. Et dans la contrainte (4), nous nous assurons que les rams totaux de tous les routeurs sur eFPGA k ne dépasseront pas le maximum RAM_k.

4.3 Algorithme pour NoC reconfigurable sur eFPGAs minimum

Dans le cas de eFPGAs nombreux, il n'est pas nécessaire d'utiliser tous les eFPGA pour le placement du NoC. Donc on cherche à minimiser le nombre d'eFPGAs utilisés. Nous utilisons l'algorithme basé sur la formulation ILP pour résoudre ce problème.

L'idée majeure est de tester la faisabilité du placement des routeurs sur des eFPGAs, d'un eFPGA juste qu'à K-1 eFPGAs. Si la valeur de la fonction objectif de la solution n (moins de

K) eFPGAs est égale à la Fmax maximum de la solution K eFPGAs, et les autres valeurs des fonction objectif des solution m (moins de n) eFPGAs sont inférieures à Fmax, puis on trouve une solution minimale pour le problème du NoC sur eFPGAs nombreux.
Dans cet algorithme 1, la liste de combinaison des eFPGAs K est construite pour le problème eFPGAs i. Un exemple combinaison de 3 eFPGAs (1,2,3) est:
(1); (2); (3); (1,2); (1,3); (2,3); (1,2,3)

Algorithm 1 NoC on Minimal eFPGAs Algorithm

1: calculate ILP value F_{max} of K eFPGAs problem
2: build list of all the combination of K eFPGAs
3: for i =1 to K do
4: calculate ILP value Fi of i eFPGAs problem
5: if $F_i >= F_{max}$ then
6: break
7: end if

5. Multiprocesseur de petite échelle (SSM IP)

La future génération de systèmes multiprocesseurs sur puce (MPSoC) sera basée sur des centaines de processeurs connectés par un réseau sur puce (NoC). Un des défis est d'augmenter la productivité de la conception. Nous proposons un multiprocesseur de petite échelle basé sur NoC (SSM IP) comme un élément constitutif des multiprocesseurs à grande taille. Nous décrivons l'architecture d'un tel SSM IP ainsi que les résultats de prototypage sur une seule puce FPGA. Les applications de traitement d'image sont utilisées comme évaluation préliminaire de logiciels parallèles.

5.1 Architecture

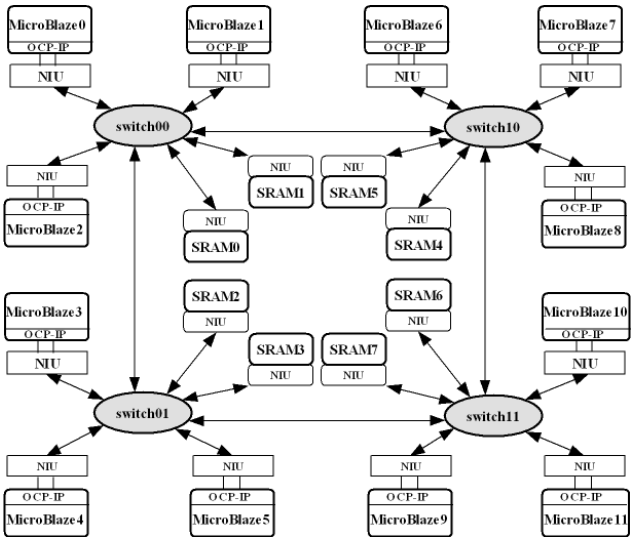


Figure 2 l'architecture du multiprocesseur de petite échelle (SSM IP)

La topologie de notre SSM IP est une maille de groupes (cluster). Le SSM IP est composé de 12 processeurs connectés par une maille 2x2 de routeurs, avec 3 processeurs de MicroBlaze et 2 SRAMs par routeur. La topologie de réseau est maille à cause de ses propriétés d'évolutivité et l'extensibilité. La régularité de mise en page et les retards d'interconnexion sont primordiaux. La topologie maillée fournit des liens courts et est plus facile à placer et router. La performance du dispositif est mieux gérée grâce à des liens courts. Plusieurs projets récents de système sur puce ont utilisé la topologie de maille.

Toutefois, notre architecture d'une maille de clusters est meilleure qu'une maille complète, parce qu'on peut mieux profiter de la localité de données dans un cluster pendant le traitement de l'image pour une application multimédia. Les images peuvent être regroupées également entre les mémoires partagées de chaque cluster, afin que les processeurs appartenant à un cluster puissent traiter la partie d'image associée à ce cluster.

Notre IP SSM est une soft IP. Il est composé des soft IP, qui sont décrites dans le tableau suivant.

Table 1 IPs de SSM multiprocesseur.

Composant d'IP	Description	Source	Version	Nombre
Processeur	Soft IP	Xilinx MicroBlaze	6.00 b	12
Mémoire	Soft IP	Xilinx Coregen 96KB	2.4.	8
Routeur du NoC	Soft IP	VHDL Arteris Danube library	1.10	4

Il y a beaucoup de configurations disponibles pour notre SSM IP. Les 4 exemples se trouvent dans le tableau, qui modifient les configurations de processeur et routeur. Bien que cette exploration de l'espace n'est pas grande, elle illustre les variations de SSM IP. L'application est mis en œuvre sur ces 4 architectures afin de comparer leurs performances.

Table 2 Les 4 versions d'architecture.

	NoC	**MicroBlaze**
Arch. V1	Fwdpipe	Multiplier
Arch. V2	Fwdpipe+Bwdpipe+Pipe	Multiplier
Arch. V3	Fwdpipe	Multiplier+FPU
Arch. V4	Fwdpipe+Bwdpipe+Pipe	Multiplier+FPU

5.2 Implementation and Results of NL-means filter

L'algorithme de NL-means est programmé en langage C pour chaque processeur dans notre SSM IP. L'image grise de 64x48 est divisée en 12 blocs avec la même taille de dimension de 16x16. Une ligne de 3 blocs est mappée vers un SRAM d'un cluster. Chaque processeur lit un bloc d'image de local SRAM. Après le filtrage de NL-means, les résultats sont envoyés dans un autre SRAM local de ce cluster.

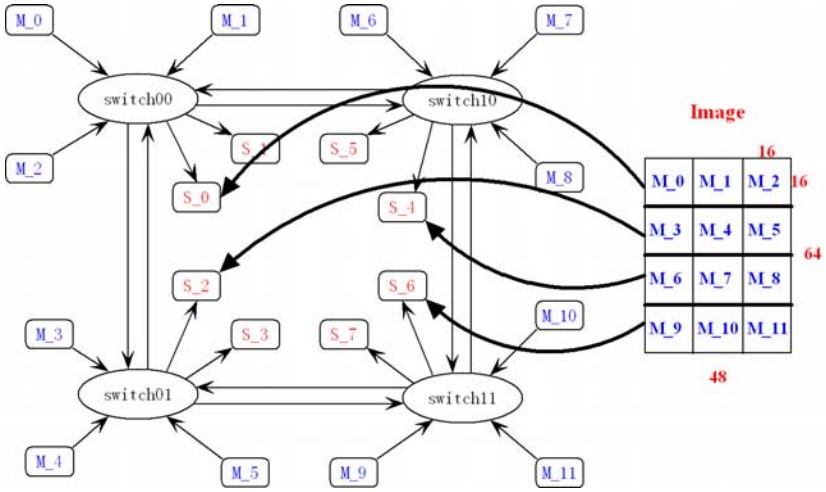


Figure 3 Le mapping d'image aux SRAMs pour l'application de NLMeans filtrage

La Figure 4 montre les performances de l'application NL-means sur les 4 architectures différentes. Afin de mieux analyser la corrélation entre le temps d'exécution et la consommation de hardware, toutes les informations sont fournies dans la même figure. L'axe

Y droit fournit l'information de surface (slice), tandis que l'axe Y gauche fournit l'information de temps d'exécution.

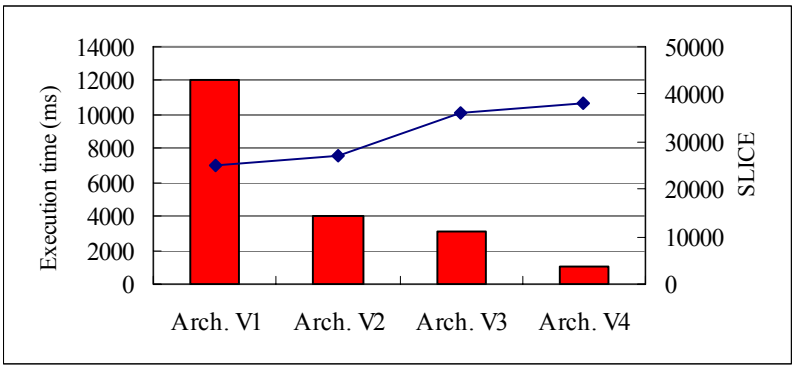


Figure 4 résultats d'implémentation de NLMeans filtrage

L'augmentation de l'utilisation de surface diminue le temps d'exécution dans tous les cas. Toutefois, le gain n'est pas linéaire. Par exemple, entre Arch. V4 et V3 ou entre Arch. V1 et V2, la variation de la surface est moins de 5%, mais le temps d'exécution est réduit de plus de 50%. La fréquence du système est augmentée utilisant plus de pipelines dans le NoC, qui prend peu de slices dans les architecture V2 et V4. Cela souligne l'importance de notre soft SSM, dont les configurations peuvent être changées selon les besoins de l'application. L'unité de traitement flottant (FPU) peut grandement améliorer les performances de calcul du processeur MicroBlaze. La performance du système peut être améliorée par un ordre de grandeur entre les architectures V4 et V1.

6. Multiprocesseur de grande taille (LSM)

La productivité de conception est un des défis les plus importants de la future génération de multiprocesseur sur puce (MPSoC). Nous proposons d'augmenter la productivité de conception en réutilisant notre SSM IP combinée avec les techniques de l'extension rapide. Une implémentation d'un multiprocesseur de 48 coeurs sur une plateforme de 4 grand FPGA a validé notre approche.

6.1 Extension du SSM IP au LSM

Afin d'atteindre une productivité de conception rapide de MPSoC de grande taille, nous avons besoin: (1) de réutiliser notre multiprocesseur à petite échelle (SSM IP) et d'ajuster

automatiquement les configurations du NoC (2) d'intégrer les outils EDA industrielles dans la procédure de conception. En raison de sa grande taille et temps de simulation prohibitifs au niveau RTL, nous avons besoin d'émulation en place de simulation des performances de multiprocesseur de grande taille.

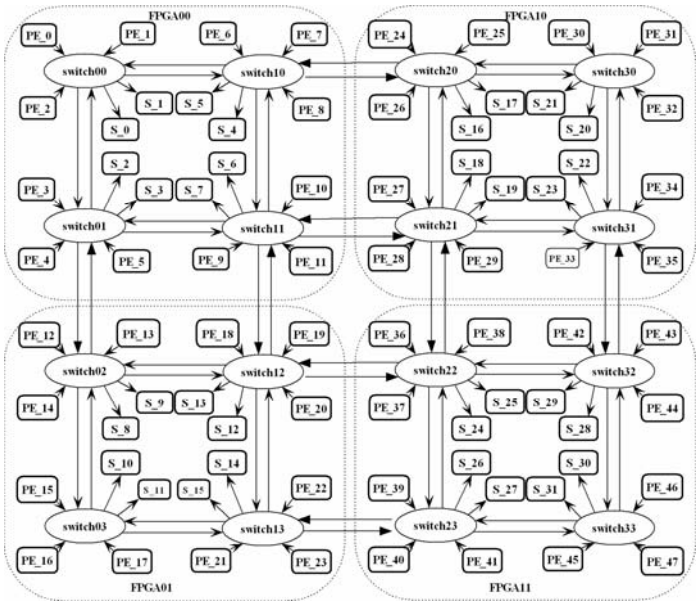


Figure 5 l'architecture du multiprocesseur de 48 coeurs

Un multiprocesseur de 48 coeurs est implémenté sur la plate-forme ZEBU-UF 4. Notre SSM IP est réutilisée pour accélérer la conception de multiprocesseur. Tous les fichiers de configuration de SSM IP sont réutilisées pour l'extension, qui peuvent largement réduire le temps de synthèse de ce multiprocesseur. En double sur 4 FPGA, tous les composants SSM ne seront pas changés, sauf le NoC adapté pour la nouvelle 4x4 topologie maillée.

6.2 Intégration des outils CAO et la procédure de conception

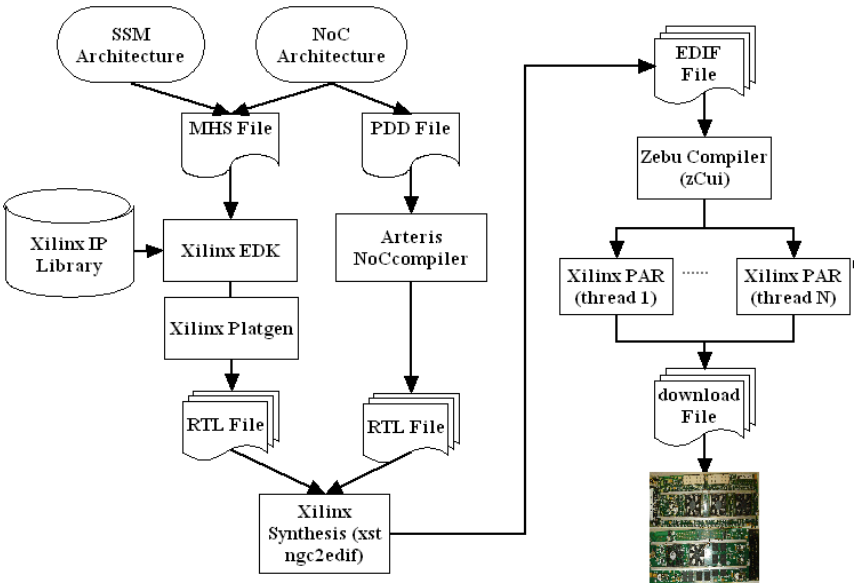


Figure 6 La procédure de conception pour MPSoC sur la plateforme multi-FPGA

Les outils CAO de trois sociétés commerciales sont mis en semble pour générer notre MPSoC de 48 coeurs. L'outil EDK de Xilinx est utilisé pour générer nos SSM multiprocesseurs. Une fois que les fichiers de RTL de SSM sont produits, ils sont réutilisés pour la synthèse de multiprocesseur de grande taille, ce qui peut largement réduire le temps de conception du système. Différents fichiers de NoC sont synthétisés pour chaque SSM sur les différentes puces FPGA de la plateforme en changeant le tableau de routage de chaque routeur selon le politique de routage. Ces fichiers RTL des NoCs sont générés par NoCcompiler, outil d'Arteris. Le compilateur ZEBU d'EVE prend les fichiers EDIF convertis par des outils de synthèse de Xilinx pour l'implémentation sur FPGA. Enfin l'outil de placement et routage est utilisé pour générer les fichiers de téléchargement au FPGA. Cette phase peut être parallélisée afin de réduire le temps de conception. Les résultats de surface et performance sont obtenus par l'émulation sur la plateforme multi-FPGA.

6.3 Modèles de parallélisme et l'implémentation

Une application typique de traitement du signal peut être divisée en plusieurs blocs de fonction et parallélisée par le mapping des blocs sur des différents éléments de traitement. Ces éléments de traitement peuvent travailler en pipeline pour la parallélisation de tâche.

6.3.1 Modèle de Fork-Join

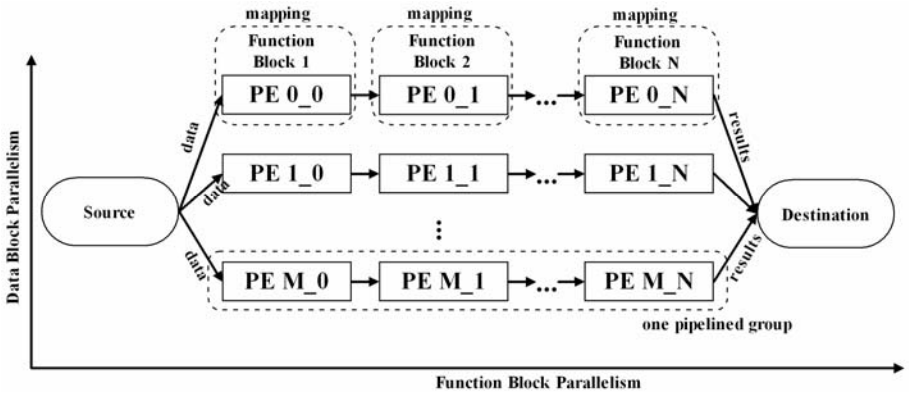


Figure 7 Modèle de Fork-Join avec les parallélismes de donnée et tâche

Deux types des parallélismes, le parallélisme de donnée et le parallélisme de tâche, sont combinés ensemble pour atteindre une meilleure performance. Les deux parallélismes sont réunis pour travailler en modèle de Fork-Join, montré dans la Figure 7.12. Dans le parallélisme de donnée, nous distribuons différents blocs de données aux groupes de traitement différents. Ces groupes travaillent sur les données reçues en parallèle. Dans le parallélisme de tâche, toutes les fonctions d'application sont divisées en blocs et placées sur des PEs (Processor Element) séquentiellement. Chaque PE obtient des données d'entrée, calcule les blocs de fonction associés et envoie les résultats au processeur PE suivants et enfin à la mémoire de destination. Les PEs mappés avec des blocs de fonction travaillent ensemble comme un groupe en pipeline.

6.3.2 L'implémentation de parallélisme sur MPSoC basé sur NoC

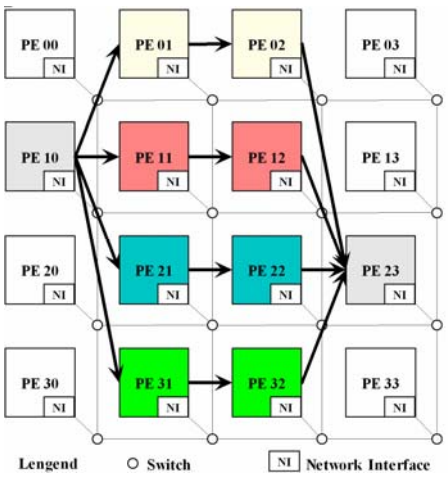


Figure 8 Implémentation de modèle de Fork-Join sur MPSoC avec NoC

Le MPSoC basé sur NoC est la tendance du future multiprocesseur à grande taille en raison de la flexibilité du NoC. Pour paralléliser l'application, plusieurs PEs dans le MPSoC sont divisés en groupes distincts pour la parallélisation de donnée. Afin de minimiser les temps de latence de la communication, les PE dans le même groupe doivent être aussi proches que possible. L'architecture simplifiée d'un MPSoC et une implémentation de notre modèle de Fork-Join sont illustrés dans la figure. Dans cet exemple, PE10 travaille comme source et envoie des blocs de données distincts à 4 groupes différents de PE en des couleurs différentes dans la figure. L'application est divisée en 2 blocs de fonction et ils sont placés sur les deux PEs dans chaque groupe. Enfin chaque groupe envoie les résultats vers PE23 qui sert comme destination. Il y a au total 4 * 2 = 8 PEs utilisés dans cet exemple d'application.

6.4 L'implémentation des Parallélismes sur MPSoC

Le code C séquentiel de cryptage TDES est composé d'une permutation Forward (FO), 48 appels à une macro F et enfin une permutation inverse (IP). Pour utiliser pleinement notre multiprocesseur, le parallélisme de données et le parallélisme de tâche sont combinés pour obtenir une meilleure performance. Un exemple est donné dans la Figure 7.19:

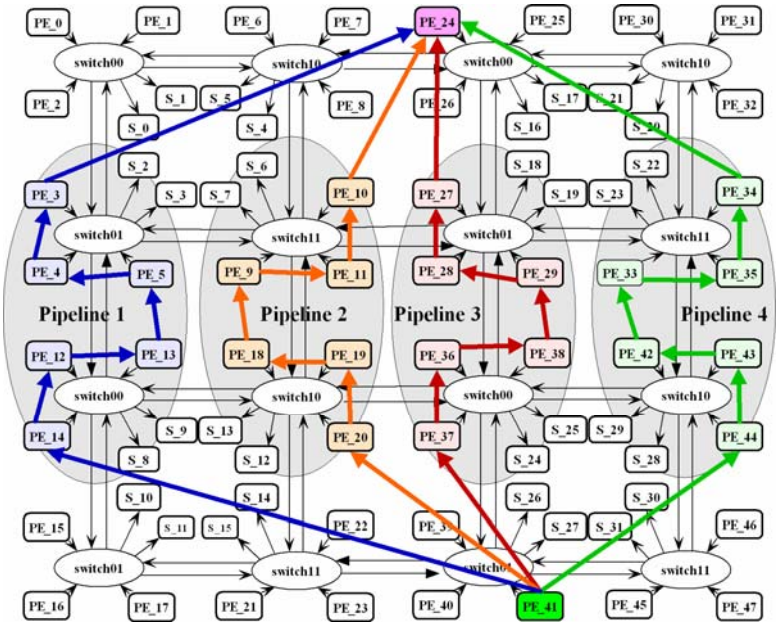


Figure 9 Un exemple de mapping de modèle Fork-Join sur 48-PE multiprocesseur

1. 24 PEs de MicroBlaze sont choisis pour l'implémentation;
2. Toutes les données sont divisées en 4 blocs et les 24 PEs sont divisés en 4 groupes;
3. Pour crypter chaque bloc de données, chaque groupe dispose de 24 / 4 = 6 PEs;
4. Dans chaque groupe en pipeline, chaque PE MicroBlaze calculera 48 / 6 = 8 appels de macro F.

L'exploration aide à trouver un bon compromis entre le Parallélisme des tâches et le parallélisme de données. Dans cet exemple, au maximum 24 PEs sont utilisés, la combinaison du parallélisme de données et le parallélisme des tâches est répertorié dans la Tableau 3:

Table 3 combinaison des parallélismes de donnée et tâche

Nombre de group en pipeline	Nombre de PE dans un groupe	Nombre de micro associé a un PE
24	1	48
12	2	24
8	3	16
6	4	12
4	6	8
3	8	6
2	12	4
1	24	2

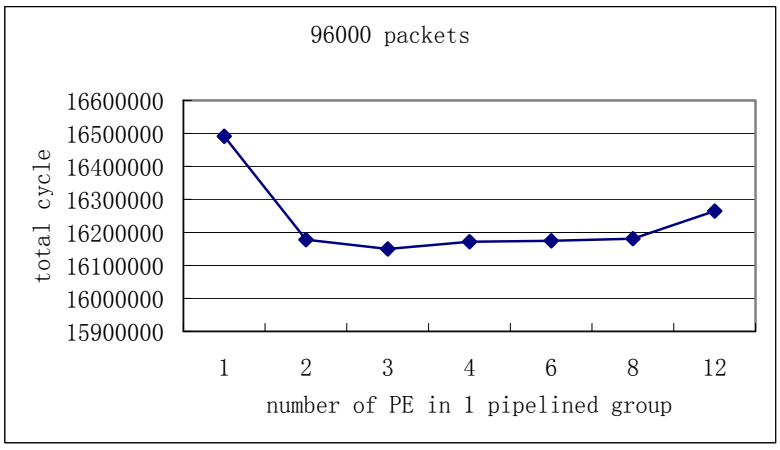


Figure 10 temps d'exécution de 96000 paquets

Dans la Figure 10, il est clair que le temps d'exécution est le plus court dans la combinaison de 8 groupes de PE en pipeline, chacun avec 3 processeurs MicroBlaze dans le groupe. C'est donc le meilleur compromis entre le parallélisme de données et le parallélisme des tâches. Et ce résultat montre l'impact de l'architecture du système sur les performances des applications parallèles.

7. Conclusion

Dans cette thèse nous avons présenté une nouvelle méthodologie de conception de MPSoC à grande taille pour résoudre le défi de la complexité de conception et augmenter la productivité. Pour atteindre cet objectif, on propose et utilise trois stratégies:

1. la combinaison de divers niveaux de conception du système : du niveau TLM à l'émulation de FPGA
2. la réutilisation des IPs et des composants : extension de SSM IP à 48-coeur et 672-coeur MPSoC
3. l'utilisation des nouvelles technologies : Arteris NoC, M2000 eFPGA, EVE plateforme multi-FPGA

Une étude des travaux récents sur la conception de NoC et de MPSOC montre qu'il n'y a aucune procédure de conception mûre pour la future génération de MPSoC à grande taille. L'architecture d'intercommunication de MPSOC affecte énormément la performance du système. Trois méthodes d'intercommunication différentes ont été utilisées : bus, crossbar et réseau sur puce (NoC). Et le NoC est proposé comme la seule solution d'intercommunication pour le futur MPSoC à grande taille. Les outils d'Arteris pour la conception de NoC sont utilisés comme le soutien industriel pour notre procédure de conception.

La contrainte de temps réel doit être prise en compte pendant la conception de MPSOC. Des multiprocesseurs homogènes et hétérogènes sont les deux branches importantes et distinctes de la conception de MPSOC. L'analyse et la comparaison des méthodologies différentes pour la conception de MPSOC aident à mieux les comprendre et à surmonter leurs défauts. Le mapping de "core graphe" aux topologies de NoC est connu comme NP-Hard. Des algorithmes heuristiques sont les seules solutions pour obtenir le résultat proche à l'optimal selon des fonctions objectif différentes. Différents algorithmes d'approximation sont proposés pour réduire le temps d'exécution de la programmation linéaire en nombres entiers.
L'exploration de l'espace de conception du réseau sur puce peut être faite à plusieurs niveaux d'abstraction, de la transaction juste qu'à l'émulation. D'abord on propose un procédure de

conception entièrement automatique pour le réseau sur puce au niveau de TLM. La combinaison de cette procédure avec notre travail d'émulation suivant permettra d'obtenir des solutions satisfaisantes.

Le réseau sur puce reconfigurable exige le support de hardware reconfigurable efficace dans l'environnement d'ASIC. L'apparition d'eFPGA IPs permet l'intégration de secteur reconfigurable dans des puces d'ASIC. L'organisation et le dimensionnement de cette zone sont des questions importantes à traiter pour maximiser l'efficacité du réseau sur puce reconfigurable. Notre méthodologie de programmation linéaire y apporte une solution.

La prochaine génération de MPSoC sera basée sur des centaines de processeurs. la conception de MPSoC est très complexe. Pour accomplir le travail en un temps raisonnable, nous proposons un SSM IP comme un composant élémentaire pour la conception de multiprocesseur à grande taille. Ce SSM IP est basé sur un NoC de topologie Cluster-Mesh. Il a été entièrement évalué sur une grande plateforme FPGA. La conception de multiprocesseurs à grande taille utilisant notre SSM IP est très rapide. L'effort de conception principal est la connexion entre IPs et l'adaptation d'adressage de NOC.

Nous avons validé notre approche sur un multiprocesseur de 48 coeurs en étendant automatiquement notre SSM IP de 12 coeurs et nous avons étendu finalement à un MPSoC de 672 cœurs sur la plate-forme multi-FPGA Zebu-XXL d'EVE. Différents outils de conception industrielle ont été mis ensemble dans notre procédure de conception automatique.

Une procédure de conception automatique pour des données parallèles et des traitements en pipeline est proposée pour l'application au traitement du signal sur le multiprocesseur avec NoC, utilisant l'application cryptographique au TDES (Triple Data Encryption Standard) comme un exemple. Notre procédure explore par l'exécution sur la plate-forme d'émulation multi-FPGA pour la mise en oeuvre de logiciel parallèle avec l'exploration de placement de tâche et l'analyse de granularité de tâche. Un moniteur hardware conduit le processus de placement de tâche pour réduire les congestions de la communication. Dans la deuxième phase, des accélérateurs hardware générés par la synthèse de haut niveau sont ajoutés à notre

procédure pour explorer les compromis entre la performance et la surface en privilégiant toujours la base de multiprocesseur.

L'association de l'OCP-IP a proposé une suite de micro-benchmark pour réseau sur puce. Nous avons évalué ces benchmarks par l'exécution réelle sur nos multiprocesseurs de grande taille avec NoC.

7.1 Travaux Futurs

La conception de MPSoC à grande taille est un secteur de recherche nouveau et ouvert. Dans la thèse, on a proposé des approches qui peuvent être étendues dans plusieurs directions de recherche différentes.

La consommation d'énergie est très importante pour le système électronique, non seulement pour la raison commerciale, mais aussi pour la protection de l'environnement. La bibliothèque de modèle de consommation d'énergie peut être intégrée dans notre procédure de conception pour explorer les compromis entre la performance, la surface et la consommation d'énergie. La combinaison de cette procédure avec nos travaux précédents au niveau RTL permettra une solution complète.
L'exploration de MPSoC à grande taille est très complexe et prend beaucoup de temps. Profitant de la combinaison de différents niveaux, de TLM à RTL, l'espace d'exploration est diminuée à haut niveau pour accélérer l'exploration. La théorie de réseau de neurone et la technologie PR de FPGA peuvent aussi aider à atteindre ce but.
Les travaux futurs consistent à ajouter la gestion de reconfiguration dans la zone d'eFPGA reconfigurable pour des accélérateurs hardware ainsi que la parallélisation automatique de logiciel.

7.2 Contribution

Notre contribution principale est la procédure automatique pour la conception de MPSoC à grande taille basée sur la réutilisation de SSM IP. Basée sur ce principe, une procédure de conception automatique pour des données parallèles et des traitements en pipeline est proposée pour l'application au traitement du signal sur le multiprocesseur avec NoC, utilisant

l'application cryptographique au TDES (Triple Data Encryption Standard) comme un exemple. La synthèse de haut niveau est ajoutée à cette procédure pour la génération de hardware accélérateur, qui permet d'étudier le compromis entre la performance et la surface. OCP-IP NoC benchmarks sont exécutés sur notre multiprocesseur de 48 coeurs et de 672 coeurs pour l'évaluation de performance.

Nous proposons une procédure automatique et multi-objectif pour NoC au niveau TLM (Transaction Level Modeling). Les critères du timing et de surface du niveau RTL sont explorés mais non limités avec des TLM modèles du NoC dans NoCexplorer. Une méthodologie de programmation linéaire est fournie comme solution au problème de l'organisation et du dimensionnement d'eFPGA reconfigurable pour maximiser l'efficacité du NoC.

Pour la première fois dans le monde, nous avons évalué par l'exécution réelle les micro-benchmarks d'OCP-IP sur nos multiprocesseurs à grande taille avec le réseau sur puce utilisant la plate-forme d'émulation multi-FPGA.

Tous les travaux réalisés dans cette thèse rendent possible MPSOCexplorer, un projet industriel pour l'exploration de MPSoC à grand taille, soutenu par l'Union Européenne et le gouvernement français.

CONTENTS

Méthodologie de Conception Automatique pour Multiprocesseur sur puce Hétérogène ... i
ACKNOWLEDGMENT ... iii
ABSTRACT ... v
Résumé ... vii
CONTENTS ... xxxi
LIST OF TABLES ... xxxvi
LIST OF FIGURES ... xxxviii
1. Introduction ... 1
1.1 Motivation ... 1
1.2 Research Objectives ... 2
1.3 Thesis organization ... 4
2. Multiprocessor and Network on Chip State of the Art ... 7
2.1 Multiprocessor on chip ... 7
2.1.1 Definitions, Architecture and Performance Evaluation ... 7
2.1.2 Academic and Commercial MPSOCs ... 8
2.2 Network on chip ... 9
2.2.1 Definitions, Architecture and Performance Evaluation ... 9
2.2.2 OCP-IP NoC micro-Benchmarks ... 14
2.2.3 Academic and Commercial NOCs ... 15
2.2.4 Case study: Arteris Technology ... 17
2.3 Conclusion ... 24
3. MPSOC Design Methodologies ... 27
3.1 MPSOC Design and Synthesis ... 27
3.1.1 Benchmarks ... 28
3.1.2 Design methods of MPSoC ... 30
3.2 NOC Design and Synthesis ... 40

3.2.1 Graph theory related material and GLPK 42
3.2.2 Workflow for regular topology 43
3.2.3 Workflow for application specific topology 46
3.3 Conclusion 48
4. Multi-objective TLM Level NOC Design Space Exploration 53
4.1 NOC SystemC TLM Modelling and Traffic Generators 53
4.1.1 SystemC TLM 54
4.1.2 TLM NoC modeling with commercial tools 55
4.2 NoC Multi-objective Optimization: NSGA-II 61
4.2.1 Multi-objective modeling formulation 61
4.2.2 Multi-objective Evolutionary Algorithm 61
4.3 NOC Multi-objective Optimization under Constraints 63
4.3.1 Design Flow 64
4.3.2 Multi-objective NOC TLM DSE 65
4.3.3 Chromosome: 66
4.4 Performance Evaluation and Comparison 68
4.4.1 Best effort exploration 68
4.4.2 Experiment with different constraints 70
4.4.3 Results of 16 processors with NoC 71
4.4.4 Architecture of exploration results 76
4.5 Conclusion 78
5. Reconfigurable NoC on eFPGA 81
5.1 eFPGA and Reconfigurable NoC State of Art 82
5.1.1 eFPGA technology 82
5.1.2 Reconfigurable NoC implementation with FPGA 84
5.2 eFPGA: M2000 Case Study 85
5.2.1 M2000 eFPGA technology 85

5.2.2 Switch Area and frequency characterization 87
5.2.3 Reconfigurable NoC problem definition 88
5.3 NoC Optimization on eFPGA with Place and Route Constraints 88
5.3.1 Reconfigurable NoC on one eFPGA 89
5.3.2 Reconfigurable NoC on numerous eFPGAs 89
5.3.3 Reconfigurable NoC on minimal eFPGAs 90
5.4 Performance Evaluation and Results 91
5.4.1 Experimental setup 91
5.4.2 Results of NoC on one eFPGA 92
5.4.3 Results of NoC on numerous eFPGAs 92
5.5 Conclusion 93
6. SSM IP: Small Scale Multiprocessor IP 95
6.1 Cluster MPSOC and NoC design 95
6.2 SSM IP FPGA Design and Implementation 97
6.2.1 Small Scale Multiprocessor Soft IP 97
6.2.2 Architecture 98
6.2.3 Design Automation Flow 100
6.2.4 Processor Element and NoC communication service 102
6.2.5 Implementation 103
6.3 Design Space Exploration of SSM IP 104
6.3.1 SSM Soft IP potential variations 104
6.3.2 Synthesis, Place and Route Time and Target frequency 105
6.4 Application Performance Evaluation and Comparison 107
6.4.1 NL-means image filter 107
6.4.2 Implementation and Results of NL-means filter 108
6.4.3 Implementation and Results of other basic applications 109
6.4.4 Results Analysis 111

6.5 Conclusion 111
7. Large Scale Multiprocessor (LSM) DSE 113
7.1 Flow Methodology from SSM to LSM 113
7.2 Extension of SSM IP to LSM 114
7.3 Automatic EDA Support 116
7.3.1 SSM IP Reuse and Automatic Composition 116
7.3.2 Eve Zebu-UF Platform 117
7.3.3 Eve Zebu Design Flow 119
7.3.4 EDA Tools Integration and workflow 119
7.4 Performance Evaluation and Comparison on synchronization 120
7.5 Automatic Exploration of Pipelined Data Parallel Applications 123
7.5.1 Related work on data parallelization 124
7.5.2 Application parallel implementation model and target 126
7.5.3 Exploration flow 128
7.5.4 TDES algorithm 131
7.5.5 Parallel Implementation of algorithm 133
7.5.6 NoC Monitoring and Traffic monitoring results example 138
7.5.7 Conclusion 139
7.6 Heterogeneous design flow with HLS 140
7.6.1 Heterogeneous design flow with HLS 140
7.6.2 High Level Synthesis: C-Based 142
7.6.3 EDA Tools Combination 143
7.6.4 Data parallelism with coprocessor 145
7.7 OCP-IP Micro-benchmarks on LSM processors 147
7.7.1 OCP-IP Micro-benchmarks 147
7.7.2 Performance Evaluation 149
7.8 Conclusion 153

8. Very Large Scale Multiprocessor Design Automation 159
8.1 State of the art 160
8.1.1 BEE multi-FPGA platform 161
8.1.2 RAMP Blue 162
8.2 General VLSM framework 166
8.3 BB-672 VLSM 168
8.3.1 Zebu XXL multi-FPGA platform 168
8.3.2 OCP-IP benchmarks 170
8.3.3 Performance evaluation 171
8.4 Comparison between RAMP Blue and our VLSM 176
8.5 VLSM and Benchmarking: where are the benchmarks ? 179
8.6 Conclusion : EDA vs Computer Architecture 180
9. Conclusion and Perspective 183
9.1 Summary of the Thesis 183
9.2 Future Research 185
9.3 Contribution 185

LIST OF TABLES

Table 1 IPs de SSM multiprocesseur. xix
Table 2 Les 4 versions d'architecture. xx
Table 3 combinaison des parallélismes de donnée et tâche xxvi
Table 2.1 Multicore Implementation 8
Table 2.2 comparison of regular and custom topologies 10
Table 2.3 characteristic of NoC design tools 15
Table 2.4 Arteris Danube Library main components 21
Table 2.5 Arteris Danube Library Switch component options 23
Table 3.1 characteristic of current MPSoC design workflow 30
Table 3.2 characteristic of network analysis program 42
Table 3.3 characteristic of regular noc workflow 44
Table 3.4 comparison of custom NoC workflow 46
Table 4.1 NOCDEX2 exploration with different constraints 70
Table 5.1 Switch Options 87
Table 5.2 results of NoC on one eFPGA 92
Table 5.3 area constraint of 4 eFPGAs 92
Table 5.4 results of NoC on 4 eFPGAs 92
Table 5.5 results of NoC on minimal of 4 eFPGAs 93
Table 6.1 IPs of SSM multi-processor. 99
Table 6.2 Implementation results 103
Table 6.3 Switch options. 105
Table 6.4 the 4 versions of architecture. 105
Table 6.5 Area of 4 version architectures 106
Table 6.6 difference between NoC based system and bus-based system. 109
Table 7.1 Generic Rout-Table of FPGA00 116
Table 7.2 EVE Zebu-uf4 platform details 117
Table 7.3 EVE Zebu-uf4 operating mode and performance 118
Table 7.4 Different Versions of SSM IP Architecture 122
Table 7.5 Synchronization Time of MicroBlaze36 using different methods 123
Table 7.6 combination of data and task parallelism (24 cores case) 136

Table 7.7 HLS based TDES IP vs optimized IPs 146
Table 7.8 Temporal Distribution 148
Table 7.9 Selected Micro-benchmark 148
Table 8.1 RAMP systems summary 160
Table 8.2 NAS Parallel Benchmarks performance results on RAMP Blue system 165
Table 8.3 OCP-IP specific spatial distribution 170
Table 8.4 comparison between RAMP Blue and BB-672 176

LIST OF FIGURES

Figure 1.1 SOC-PE Design complexity trends 1
Figure 1.2 ITRS 2007 Design productivity gap 2
Figure 2.1 (a) distributed memory, (b) shared memory and (c) mixed memory architecture MPSoC 7
Figure 2.2 Basic NoC realization 10
Figure 2.3 regular mesh topology (a) vs. irregular mesh topology (b) 11
Figure 2.4 Cycles Based Results vs. Time Based Results 14
Figure 2.5 Measurement configuration 15
Figure 2.6 Example SOC Design 18
Figure 2.7 NoCexplorer workflow 18
Figure 2.8 NoCexplorer workflow 19
Figure 2.9 NoCcompiler Flow 20
Figure 2.10 packet transport portion 21
Figure 2.11 Arteris switch interface 22
Figure 2.12 Switch internal architecture 22
Figure 2.13 functional view of switch 23
Figure 3.1 Design refinement from application codes to low-level implementation 32
Figure 3.2 HOPSE design flow 33
Figure 3.3 MAMPS design workflow for multiple use-case 34
Figure 3.4 Daedalus design workflow with ESPAM and Sesame 36
Figure 3.5 System-CoDesigner design workflow with behavioral SystemC model 38
Figure 3.6 SoCDAL design workflow overall 39
Figure 3.7 CoMPSoC design workflow overall 40
Figure 4.1 TLM, Use Cases and mapping 54
Figure 4.2 TLM Communication 55
Figure 4.3 Aretris NoC design tools. 56
Figure 4.4 Example of NoC modeling with NoCexplorer and NoCcompiler 56
Figure 4.5 FIFO area estimation of the TLM link model with different depth. 57
Figure 4.6 Switch area estimation with different number of IO. 58
Figure 4.7 two-cut string crossover operation 62

Figure 4.8 Classification of individuals in fronts with constraints threshold. 63
Figure 4.9 Base Architecture 64
Figure 4.10 Target Board 64
Figure 4.11 TLM Network-on-chip design flow 65
Figure 4.12 NoCDEX 3 design exploration flow 66
Figure 4.13 Examples of different individuals in GA exploration 67
Figure 4.14 Chromosome representation of DES for NSGAII. 67
Figure 4.15 Results of best effort exploration 69
Figure 4.16 Best Effort Exploration Results. 71
Figure 4.17 Depth of Master links in Best Effort Exploration. 72
Figure 4.18 Busy Status of Master Links in Best Effort Exploration. 72
Figure 4.19 Latency Constraint Exploration Results. 73
Figure 4.20 Depth of Master Links in Latency Constraint Exploration. 74
Figure 4.21 Busy Status of Master Links in Latency Constraint Exploration. 74
Figure 4.22 Latency and Area Constraint Exploration Results 75
Figure 4.23 Depth of Master Links in Latency and Area Constraint Exploration. 75
Figure 4.24 Busy status of master links in latency and area constraint exploration. 76
Figure 4.25 TLM level NoC architecture of no constraint exploration's conf1 solution. 77
Figure 4.26 TLM level NoC architecture of latency constraint exploration's conf2 solution. 77
Figure 4.27 TLM level NoC architecture of latency and area constraint exploration's conf3 solution. 78
Figure 5.1 ASIC with eFPGA (a) centralized (2) scattered 81
Figure 5.2 Placement of NoC onto eFPGAs 82
Figure 5.3 Abstract view of Menta eFPGA architecture 83
Figure 5.4 Abound Multi-function cell and layout of Raptor FPGA 86
Figure 5.5 Area and Frequency of Switches 87
Figure 5.6 Design flow of NoC on eFPGA 88
Figure 5.7 NoC testbench architecture 91
Figure 6.1 Small Scale Multiprocessor IP Interfaces 97
Figure 6.2 Small Scale Multiprocessor IP Composition (a) NOC based general case (b) NOC Mesh organized 98

Figure 6.3 Small Scale Multiprocessor IP architecture (a) Full mesh (b) Cluster based 98
Figure 6.4 Switch Area Variations as Function of Number of Inputs-Outputs 100
Figure 6.5 SSM Design Automation Work Flow 101
Figure 6.6 (a)Processor Tile (b) MicroBlaze core block diagram 102
Figure 6.7 Multiprocessor Xilinx FX140 floorplan (un-optimized) 104
Figure 6.8 Alpha-Data Board ADM-XRC (a) board architecture (b) board 104
Figure 6.9 DSE Execution Time 106
Figure 6.10 Image mapping to SRAMs for NLMeans filter application 108
Figure 6.11 Execution results of NLMeans filter 108
Figure 6.12 Execution time of dot product 110
Figure 6.13 Execution time of matrix multiplication 110
Figure 6.14 Execution time of filter conservative 110
Figure 7.1 MPSOC Design Space Exploration 113
Figure 7.2 routing example of 2x2 multi-SSM 114
Figure 7.3 Multi-FPGA SRAM Addresses 115
Figure 7.4 Tx output of FPGA00 115
Figure 7.5 Small Scale Multiprocessor IP Reuse and Automatic Composition 117
Figure 7.6 Eve Zebu-UF4 Platform. 118
Figure 7.7 ZeBu Compilation Flow Overview. 119
Figure 7.8 Workflow of Multi-FPGA MPSoC 120
Figure 7.9 Execution cycle of synchronization using different pilot and SRAM. 121
Figure 7.10 Execution time of synchronization with different architecture. 122
Figure 7.11 Block-by-block synchronization 123
Figure 7.12 Fork-Join Model of data and function block parallelism 126
Figure 7.13 MPSoC with NoC and Fork-Join Model Implementation Example 127
Figure 7.14 Data parallel and pipeline exploration flow 129
Figure 7.15 Feistel function F (SBoxes) 131
Figure 7.16 TDES encryption and Decryption schemes (Feistel Network) 132
Figure 7.17 ECB operation mode for the TDES block cipher 132
Figure 7.18 CBC operation mode for the TDES block cipher 133
Figure 7.19 Fork-Join Model mapped to 48-PE multiprocessor example 134

Figure 7.20 Results of one pipelined group with different size of data 135
Figure 7.21 Single pipelined group vs. multiple pipelined groups 137
Figure 7.22 Tradeoff between task and data parallelism (24 core limited case)..... 137
Figure 7.23 Performance monitoring concept..... 138
Figure 7.24 performance monitoring network on each FPGA..... 138
Figure 7.25 NOC monitoring results example 139
Figure 7.26 Automatic heterogeneous Design Flow with HLS 141
Figure 7.27 (1) Block Diagram of Accelerator Connection Forms (2) C-based HW Accelerated System Design Workflow 143
Figure 7.28 Workflow of Multi-FPGA MPSoC with HLS..... 144
Figure 7.29 5 Stage pipeline TDES 145
Figure 7.30 Parallel Software vs Coprocessors on a 48 PE Multiprocessor..... 146
Figure 7.31 Measurement configuration 147
Figure 7.32 (a) Hot spot traffic b = 0.3 = 0.5 (b) locality b = 0.3 149
Figure 7.33 Hot spot benchmark packets latency with target S0: (1) b = 0.3 ρ = 0.5 (2) b = 0.3 ρ = 0.7 (3) b = 0.5 ρ = 0.5 (4) b = 0.5 ρ = 0.7 150
Figure 7.34 Hot spot benchmark packets latency with target S24: (1) b = 0.3 ρ = 0.5 (2) b = 0.3 ρ = 0.7 (3) b = 0.5 ρ = 0.5 (4) b = 0.5 ρ = 0.7 150
Figure 7.35 Hot spot benchmark data transferred at MB0 b = 0.3 ρ = 0.5 150
Figure 7.36 Hot spot benchmark – MB0 (1) S0 b = 0.3 ρ = 0.5 (2) S24 b = 0.3 ρ = 0.7..... 151
Figure 7.37 Locality benchmark packets latency : (1) b = 0.3 (2) b = 0.5..... 152
Figure 7.38 Locality benchmark data transferred at MB0 b = 0.3 152
Figure 7.39 Locality benchmark packets latency MB0 : (1) b = 0.3 (2) b = 0.5 153
Figure 8.1 general architecture of BEE2 module..... 161
Figure 8.2 Architecture of MicroBlaze processor for RAMP Bleu system interconnection . 163
Figure 8.3 Intra and inter board communication and RAMP Blue 3D mesh architecture..... 164
Figure 8.4 1008 cores RAMP Blue system on 21 BEE2 boards with management server and monitor [5] 165
Figure 8.5 General methodology for VLSM multiprocessor design..... 166
Figure 8.6 target Automatic workflow for VLSM multiprocessor 167
Figure 8.7 Cluster-mesh architecture of 672-core VLSM 168

Figure 8.8 Eve Zebu XXL multi-FPGA platform 169
Figure 8.9 Hot spot benchmark packets latency S0: b = 0.3 ρ = 0.5 (672 core) 171
Figure 8.10 Uniform benchmark packet latency with b = {0.5, 0.4, 0.3, 0.2} 172
Figure 8.11 N complement benchmark with b = {0.5, 0.4, 0.3, 0.2} 172
Figure 8.12 Bit rotation benchmarks with b = {0.5, 0.4, 0.3, 0.2} 173
Figure 8.13 Hot spot benchmark packets latency with b = {0.5, 0.3}, ρ = {0.5, 0.7}, and S0 as hot spot 174
Figure 8.14 Hot spot benchmark packets latency with b = {0.5, 0.3}, ρ = {0.5, 0.7}, and S102 as hot spot 174
Figure 8.15 Locality benchmark packets latency with b = {0.5, 0.4, 0.3, 0.2} 175

1. Introduction

1.1 Motivation

ITRS Semiconductor roadmap projects that hundreds of processors will be needed for future generation multiprocessor system on chip (MPSOC) designs. Increasing device density enables exponentially more cores on a single die. Processor manufacturers have shifted towards producing multicore processors to remain within the power and cooling constraints of modern chips while maintaining the expected performance advances with each new processor generation. Intel's 8 core processors are expected within 2009 and Tflops with 80 cores in 45 nm technology has already been demonstrated. Graphic processors have already hundreds of cores, as NVidia's recent GeForce 295 with 480 stream processors in 55 nm process. Moreover, ITRS also predicts the same trend to continue assuming that 10x design productivity improvements will be required for the newly designed portion in the next ten years till 2016 in order to keep the design effort constant.

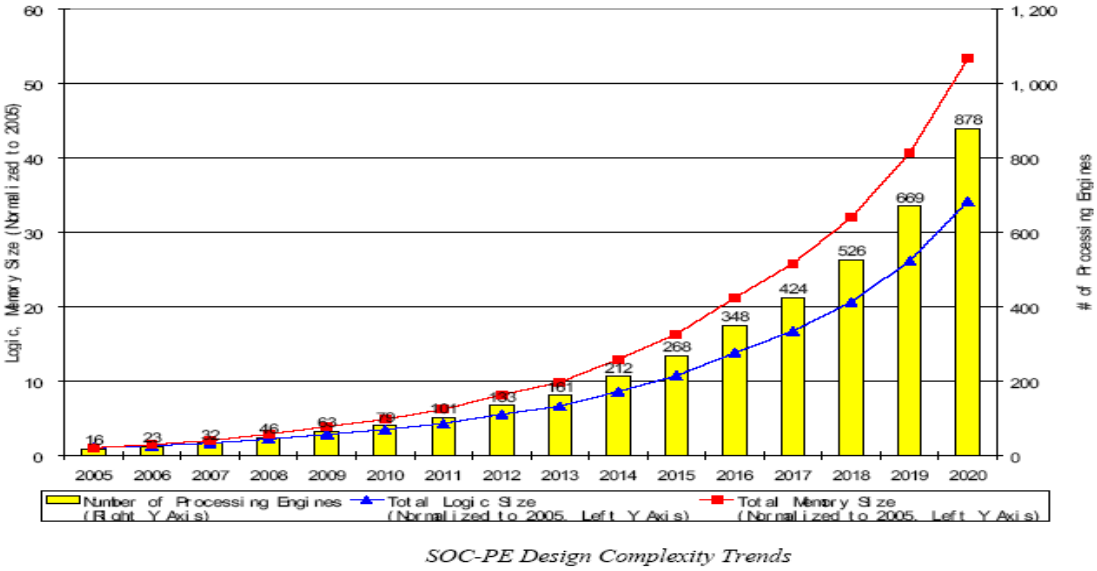


Figure 1.1 SOC-PE Design complexity trends

As a result, there are two great challenges in designing new generation of MPSoC:

- How to improve design productivity in order to shrink the time to market (TTM) of electronic system which is getting more and more complex?
- How to make sure the current design project is adaptive to the quickly evaluating semiconductor process technology?

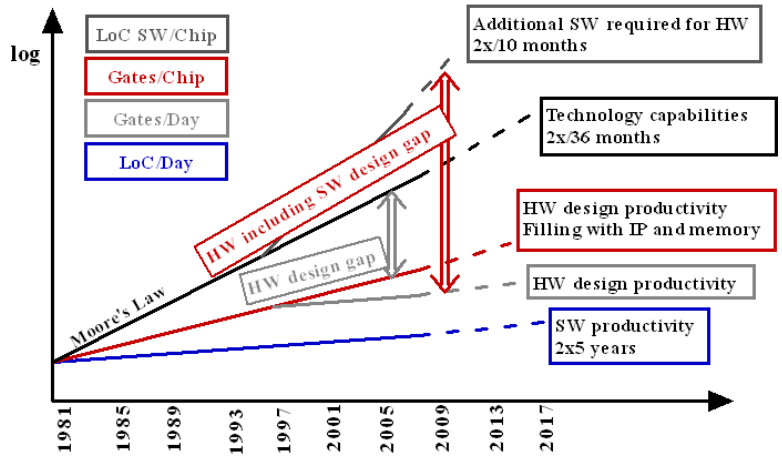


Figure 1.2 ITRS 2007 Design productivity gap

Design productivity of system on chip is the major challenge in design technology. The design productivity gap represents the fact that Moore's law regular progress generates a number of available transistors which grows faster than the ability to use them in a meaningful way. Silicon complexity and system complexity combining together cause this exponentially increasing design productivity gap. Silicon complexity is the result of physical properties of semiconductor process technology(like non ideal scaling of device parasitic and supply threshold/voltages, manufacturing and process variability, complexity of manufacturing handoff, decreased reliability) and scaling of global interconnect. System complexity associated challenges are reuse, verification and test, cost-driven design optimization, embedded software design, reliable implementation platforms, and design process management.

1.2 Research Objectives

New efficient design methodologies are needed to overcome these silicon and system

complexity and fill the increasing design productivity gaps. To achieve this object, 3 strategies are proposed:

1. Combination of different system design levels
2. Reuse of IPs and components
3. Utilization of new technologies

In this thesis we present a new methodology which implements these 3 strategies to resolve the design challenges.

This new methodology combines different design levels together to takes each level's advantages and get over each level's separate shortcoming. More automation is demanded from high system design level, SystemC TLM level, until RTL design level and FPGA emulation or ASIC verification. In this way, we can take advantage of the high abstraction models from high level system design and pass through the RTL level to supply system electronic information as much as possible. This also makes sure that system design is always adaptive to the coming new semiconductor process technology. What is more, developers with functional full-system simulators are facing a simulation wall because of its limited throughput when simulating systems with hundreds or more cores. Large scale CMP emulation with multi-FPGA platform is one solution proposed to accelerate system exploration of design space.

This new methodology reuses the existing IP blocks and components to speed up the design process and make easy the system verification. Until now, the basic reusable IPs are still too elementary to quickly build large scale multiprocessors. It is then necessary to raise the size and complexity of IPs to which we call as small scale multiprocessor (SSM) IPs. Designing large scale multiprocessors based on small scale multiprocessors allows quickly duplicating building elements and building a large scale multiprocessor in reasonable design time with multi-FPGA emulation for fast validation and performance evaluation.

New technologies for MPSoC design are applied into this new methodology:

- Network on Chip (NoC) communication architecture are used to settle the large scale system scalability problem;
- High Level Synthesis (HLS) VHDL generation tools accelerate coprocessor conception and IP reuse;
- Reconfigurable embedded FPGA technology allows the implementation of reconfigurable NoC;
- Multi-FPGA platform make large scale system emulation realizable.

Full implementation of these strategies and technologies in our new methodology makes it a very good solution to improve system design productivity and manufacturability.

1.3 Thesis organization

Chapter 2 of this thesis presents some general background of NoC and MPSoC. Definitions, architecture and performance evaluation are given, then academic and commercial NoCs and MPSoCs are analyzed and compared. Finally Arteris technology is presented as study case.

Chapter 3 compares state-of-art NoC and MPSoC design methodologies. This chapter includes a detailed description of some design workflow. These different approaches are compared and analyzed to overcome their shortcomings. Benchmarks and linear programming tools are also introduced in this chapter.

Chapter 4 presents a fully automatic multi-objective design space exploration of NoC at TLM level. The timing and area criteria extracted from RTL level are explored but not limited using the TLM NoC models. We propose an automatic approach to this problem with three instances: (1) best-effort optimization (2) latency constrained and (3) area and latency constrained.

Chapter 5 provides a linear programming method as a solution for the organization and dimensioning of eFPGA reconfigurable area to maximize the efficiency of network on chip mapping.

Chapter 6 describes the details of SSM IP, basic building block for large scale multiprocessor. Architecture as well as prototyping results are given on a single FPGA chip. Image processing applications are used as preliminary parallel software evaluation and demonstrate the potential of design space exploration at small scale multiprocessor.

Chapter 7 presents the workflow for large scale multiprocessor design based on SSM IP. An automatic design flow is proposed for data parallel and pipelined signal processing applications on multiprocessor with NoC, using cryptographic application TDES (Triple Data Encryption Standard) as an example. High level synthesis tool is used to generate hardware accelerators, which are added to explore the tradeoff in area-performance while still privileging multiprocessor basis for the implementation. OCP-IP NoC benchmarks are executed on the generated 48-core and 672-core multi-processor for performance evaluation.

Chapter 8 summarizes the work presented, suggests topics for future work and lists contributions.

2. Multiprocessor and Network on Chip State of the Art

2.1 Multiprocessor on chip

The trend of implementing multicore in a single chip is increasingly dominating the landscape of new electronics products. Multiprocessors on chip (MPSoC) are strongly emerging and several products or ongoing R&D projects are tackling the issues related to MPSoC [30-37].

2.1.1 Definitions, Architecture and Performance Evaluation

MPSoC is different to distributed multicore or multiprocessor as all the processing elements are integrated onto one chip. The major different comes from the communication architecture: the distributed multiprocessor is connected by Ethernet like network with large bandwidth but high latency, while MPSoC on-chip communication must be fast and networking must be simple and effective. Layers on very large scale integration (VLSI) circuit supply wide of wires to transfer data and control signals. Local proximity of processing elements and memories accelerates the transport. But still trade-off must be made among bandwidth, latency and energy consumption.

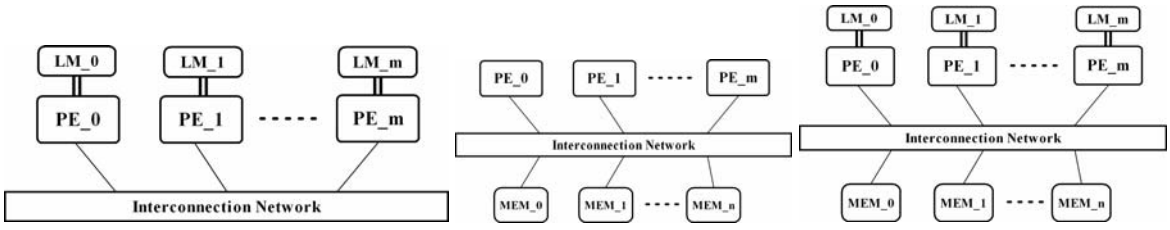


Figure 2.1 (a) distributed memory, (b) shared memory and (c) mixed memory architecture MPSoC

Flynn's taxonomy of computer architecture defines 4 classifications: SISD (Signal Instruction Single Data), SIMD (Single Instruction Multiple Data), MISD (Multiple Instruction Single Data) and MIMD (Multiple Instruction Multiple Data). MPSoC is most appropriate to MIMD architecture. And MIMD can be further divided into 2 categories:

SPMD (Single Program Multiple Data) and MPMD (Multiple Program Multiple Data) according to the program number. Typically, MPMD system works as Fork-Join programming model. With the position of memories, MPSoC can be divided into shared memory architecture and distributed memory architecture.

Ideally, the system with *n* processor show n times faster performance than a signal processor. But in reality, its speed-up ranges from a lower-bound $\log_2 n$ to an upper bound $n/\ln n$ due to conflicts over memory access, IO limitations and inter-processor communication [29]. The interconnection architecture of MPSoC greatly impacts the system performance. Three different interconnection methods have been used: bus, crossbar and network on chip (NoC). And NoC is proposed as the only solution for future large scale MPSoC design.

2.1.2 Academic and Commercial MPSOCs

Table 2.1 provides A few examples of commercial multicore implementations have been proposed. They can be globally divided in 2 categories: (1) general purpose (2) application specific. In the first category we can place the ARM ARM11MPcore [30], the MIPS MIPS32 1004 Core [31] and the Renesas/Hitachi SH-X3 [32]. In the second category we can place Texas Instruments TMS320C6474/ TMS320VC5441DSP [33-35], Freescale QorIQ P4080 [36] and the Toshiba Venezia multicore [37].

Table 2.1 Multicore Implementation

MPSOC	Part	Com	PE nbr
ARM	ARM11	Shared Bus	4
Texas Instruments	TMS320C6474	Switch Central Resource	3
Texas Instruments	TMS320VC5441	Shared Bus/HPI	4
Freescale	QorIQ™ P4080	Corenet Coherency fabric	8
MIPS	1004K™ Core	Coherence Manager	4
Toshiba	Venezia	Bus	8 VeneziaEX

The ARM11 MPcore is a classical shared memory 4 processors based multiprocessor based on a shared bus architecture with a snoopy cache coherency protocol (MESI). The MIPS32 1004 is a 1 to 4 multi-threaded "base" cores (up to 8 hardware threads) with Coherence Management (CM) unit - the system "glue" for managing coherent operation between cores and I/O, I/O Coherence Unit (IOCU) - hardware block for offloading I/O

coherence from software implementation on CPUs. Several multicore architectures are proposed by Texas Instruments. The Texas Instruments TMS320C6474 is a 3 DSP based multicore architecture with switch central resource (SRC) as the interconnection between the 3 DSP and the memories. The 6474 device contains 2 switch fabrics through which masters and slaves communicate: (1) data switch (2) configuration switch. The data switch fabric is a high-throughput intreconnect mainly used to move data across the system and connects masters to slaves via 128-bits data buses (SCR B) and 64-bit data buses (SCR A). The configuration switch is used to access peripheral registers. The Texas Instruments TMS320VC5441 is a 4 core multicore with shared bus between 2 cores and HPI for external accesses. The Freescale QorIQ™ P4080 is an 8 core multicore architecture with a Corenet coherency fabric. Each core is a high-performance Power Architecture e500mc cores, each with a 32-KByte Instruction and Data L1 Cache and a private 128-KByte L2 Cache. The CoreNet fabric is Freescale's next generation front-side interconnect standard for multicore products. However, these devices have a limited number of processors and are not customizable.

2.2 Network on chip

Network-on-Chip (NoC) is an emerging paradigm for communications within large VLSI systems implemented on a single silicon chip. As the complexity of integrated systems keeps growing, NoC provides enhanced performance and scalability in comparison with simple on chip communication solutions such as dedicated point-to-point signal wires and shared buses. From a system design viewpoint, with the advent of multi-core processor systems, a network is a natural architectural choice. NoC can provide separation between computation and communication; support modularity and IP reuse via standard interfaces; handle synchronization issues; serve as a platform for system test; and hence, increase design productivity.

2.2.1 Definitions, Architecture and Performance Evaluation

NoC is a communication network that is used on chip. It is constructed from multiple point-to-point data links interconnected by switches (a.k.a. routers), such that data can be transferred from source to destination over several links, by making routing decisions at the

switches. A high level of parallelism is achieved, because all links in the NoC can operate simultaneously on different data transfer.

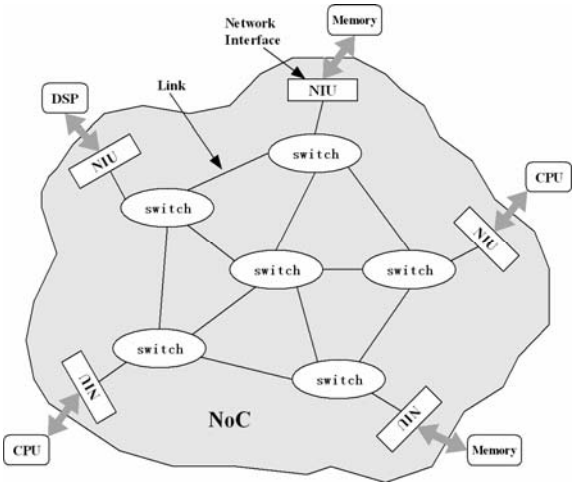


Figure 2.2 Basic NoC realization

A basic NoC consists of switches, links, and network interfaces. Switches direct data through several links according to switching policy. The logical connections of links are referred as network topology which is also restricted by physical layout floor-planning. The function of a network interface (adapter) is to decouple computation (the resources) from communication (the network). Each IP core is connected to the NoC through a network interface.

A. Topology

NoC architectures can be designed with both regular and irregular topologies.

Table 2.2 comparison of regular and custom topologies

Regular topologies	**Custom topologies**
Mesh, torus …	Irregular topologies
General purpose SoC	Application specific SoC
Homogeneous routers	Heterogeneous routers
Reuse of topologies	Design reuse of routers
Lower design time	Longer design time
Lower performance	Higher performance
Higher power	Lower power

The primary advantage of a regular NoC architecture is topology reuse and reduced design

time. They are suitable for general purpose architectures, such as the RAW processor that include homogeneous cores. Regular topologies assume that every core has equal communication bandwidth with every other core which does not hold in custom SoCs. Application-specific SoC architectures consist of heterogeneous cores and memory elements which have vastly different sizes. Consequently, even if the system-level topology is regular, it does not remain regular after the final floor-planning stage. The alternative option of regular layout results in a large amount of area overhead. The custom NoC architecture is superior to regular architecture in terms of power and area consumption under identical performance requirements. [12, 17, 18] In custom topologies, the switch architecture itself is regular and can be easily parameterized for reuse.

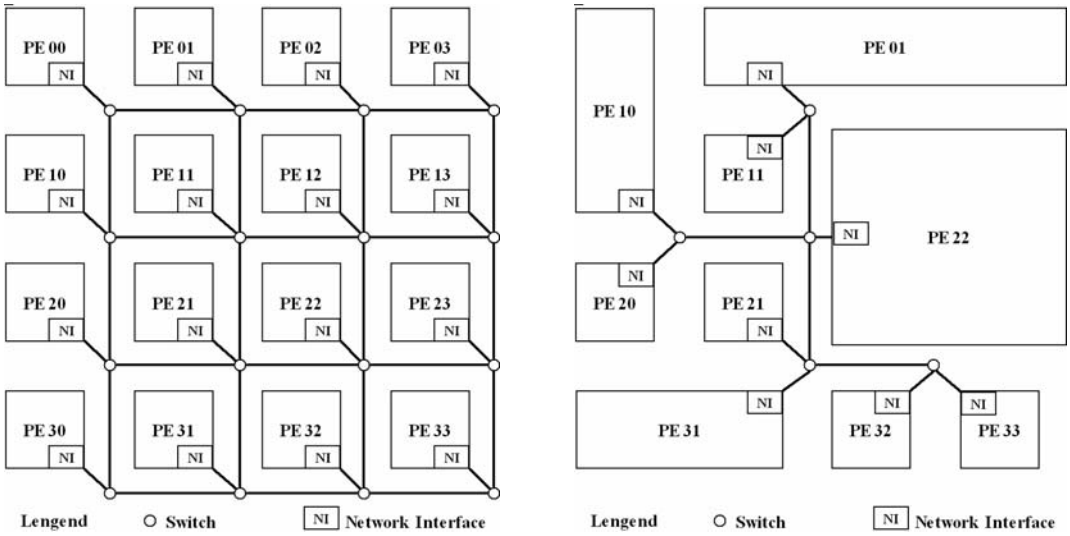


Figure 2.3 regular mesh topology (a) vs. irregular mesh topology (b)

Some important topologies are listed here and the area cost and performance of each one are discussed.

Crossbar: NoC are fully connected when all switches are connected to all others in a matrix manner. The single crossbar does not scale up to large number of network.

Mesh: 2-D mesh is the most common topology. As the switches can be placed regularly in layout and all links have the same length, commercial multicore processor use mesh topology for the manufacture facility. The number of network interface (NI) per switch can be one as usual or more as we propos cluster based mesh. Regular mesh with one NI per switch has relatively large average distance between NIs and affects negative power dissipation.

While cluster based mesh overcomes these disadvantages. Traffic load distribution on mesh topology is an open issue to avoid traffic accumulation in the center of mesh as a hot spot.

Torus: torus adds wrap-around links to the mesh topology. The area of torus is roughly the same as mesh, but the power dissipation and performance are better because the average distance is less than in mesh.

Irregular: Application-specific networks can obtain superior performance while minimizing both area and energy. Foe example, an irregular mesh topology has sufficient performance comparing to regular mesh but at lower cost, shown as in Figure 2.3 .

B. Switching policy

There are two basic modes of switching: circuit switching and packet switching. In Circuit switching, a physical path of links and switches is reserved from source to destination to transfer messages. Packet switching sends message in packet basic. A message make their way independently from source to destination, no link is reserved for the message.

There are three packet switching choices: store and forward (SAF), cut-through (CT) and wormhole (WH). SAF method stocks the whole packet in switch input buffer before sending it to the next switch. The latency per switch is equal or more than the size of packets. CT method reduces the per switch latency by forwarding the packet as soon as the head information is available for switch arbiter. The payload of packet is sent after without delay is there is space available in the next switch. Other whiles, the whole packet is buffered. Both SAF and CT methods require critical buffer capacity, while WH method reduces the buffer requirement to one flit at smallest by forwarding each flit to the next switch as soon as there is buffer space for that flit. The whole packet is cut into flits and spreads over the NoC coined as wormhole switching.

Circuit switching is better to guarantee bandwidth for relatively static communication, which is frequency sent and long enough to amortize the high setup latency due to the initial set-up phase before data transmission. Packet switching has no dedicated reserved circuits and therefore supports more concurrent traffic. However, the buffering and concurrency introduces unpredictable latency. As packet switching does not need set-up and torn down phases, it is more suitable for small and dynamic traffics.

C. Network Interface

Network interface (NI) is glue logic to adapt communication cores to the network. Standardized interface is required for integrating intellectual property (IP) cores of diverse communication bus-specific interfaces: such as AMBA (Advanced Microcontroller Bus Architecture), AHB (Advanced High-performance Bus) and OCP (Open Core Protocol) bus. These NIs support standard IP interface protocol make easy the reuse of IP core into NoC based system on chip and greatly speedup the productivity of system design.

Network interface adapt the IP core communication protocol to the proper communication protocol of NoC by packetization service. It encapsulates and decapsulates data between bus and NoC interfaces by specifying the transaction service. Attentions are paid to guarantee the bandwidth and minimize latency caused by NIs. A software library of instructions should be supplied to support the access of memory. And cash coherence, which is easy to achieve on a bus by snooping, needs new protocols on NoC to realize multiprocessor on chip at low cost.

D. Performance Evaluation

The most important metrics for NoCs are application execution time, silicon area, power consumption, and latency. All these are to be minimized and Pareto solution of large scale exploration is expected.

Different evaluation methods can be used to measure system performance: model based analysis [40,41], simulation and real chip execution. Mathematical model analysis is fast but not precise, which can be used at the first step for fast verification to minimize exploration space. SystemC based simulation, is widely used in academic research but as the system getting more and more complex, the simulation speed is not enough. FPGA platform based emulation is proposed as the solution for large scale exploration.

We argue that execution time should be measured on real time rather than cycle numbers [42]. A comparison example is given in Figure 2.4. These results are obtained by emulation on a multi-FPGA platform. Results are recorded both on cycle and time with the same configurations.

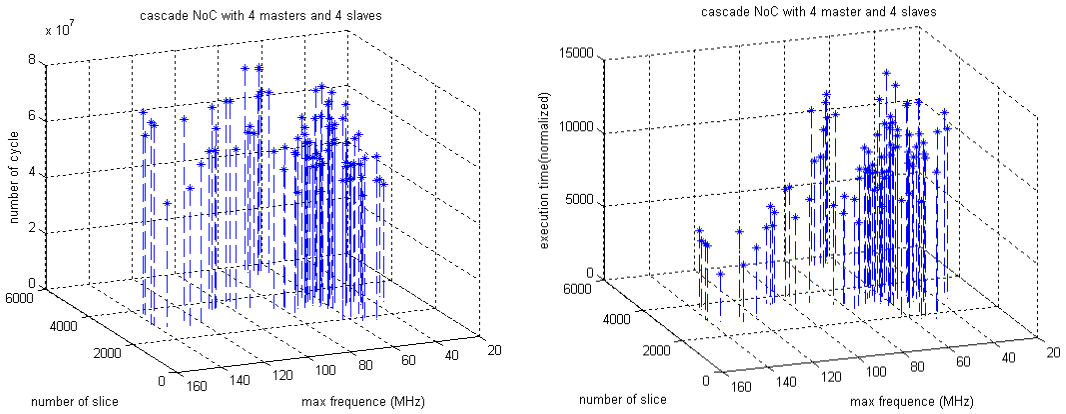


Figure 2.4 Cycles Based Results vs. Time Based Results

As it can be seen from the above figures time based and cycle based results of the same evaluation are different. In fact, systems of different configurations can not achieve the same working frequency. As the maximum frequency of system augment, the real execution time decreases quickly. In this way, we say that the way using the numbers of cycles as the measure of system performance is misleading and imprecise.

The impact of different options on the system's frequency must be taken into consideration, especially in the case of SoC design using ASIC devices, on which the maximum working frequency of system is sensible to the system configuration, PAR (place and rout) options and even the RTL language coding style.

2.2.2 OCP-IP NoC micro-Benchmarks

The OCP has released a comprehensive set of synthetic workloads as micro-benchmarks for the evaluation of network on chip. A section on performance benchmarking specifically describes requirements and features for application programs, synthetic micro-benchmarks, and abstract benchmark applications. It then proposes ways to measure reliability, fault tolerance, and testability of the on-chip communication fabric.

Although micro-benchmarks cannot represent a real application well they are complementary to application benchmark. OCP defines two classes of communication services: best effort and guaranteed services. The best effort is connection-less, delivering packets in a best-effort fashion. It has no establishment phase, and sources send packets without the awareness of states in destinations. The guaranteed service is connection-oriented providing certain bounds in latency and/or bandwidth. A connection is a unidirectional virtual circuit setting up from a

source NI to a destination NI via the network. The network reserves resources such as buffers and link bandwidth for connections.

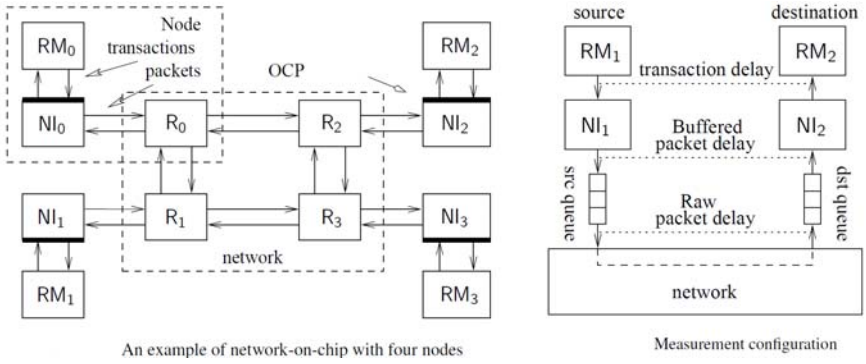


Figure 2.5 Measurement configuration

2.2.3 Academic and Commercial NOCs

This is a survey of nowadays Network on Chip design. NoC is expecting to be the future communication architecture for System on Chip. The design of NoC combines topology generation, core to router mapping and communication routing.

Table 2.3 characteristic of NoC design tools

	Aethereal [2]	Xpipe [3,4]	Danube [5, 6]	Arizona [28]	Nostrum [38]	TeraFlops [39]
Industry support	Phillips	iNoCs	Arteris	no	no	Intel
SystemC level	TLM flit-accurate	CA	TL0, CA	no	?	no
NoC topology	regular	custom	custom	custom	mesh	mesh
Switching policy	Circuit and WH	WH	WH	?	Circuit SAF	WH
Network interface	AXI OCP	OCP	AXI, OCP	no	OCP?	?
Evaluation method	TDMA model simulation	fast engine Simulation synthesis	Simulation ASIC FPGA	simulation	simulation	FPGA ASIC

In developing its Aethereal [2] Network-on-Silicon (NoS), Philips decided to use a combination of circuit and packing switching, asynchronous transfer mode, to enable communications between IP blocks. It is hardware architecture with a programming model based on the OSI reference model, which allows the structuring of communication complexity from the physical implementation up to the application in a number of layers. It use regular topology and support AXI network interface. A unique feature of aethereal [2] is the fast automatic performance verification: given the NOC hardware and configuration, the guaranteed minimum throughput, maximum latency, and minimum buffer sizes are analytically computed for all guaranteed connections. The guaranteed communication services of Æthereal are essential to achieve this. Any NOC instance and configuration can be verified, whether automatically or manually created.

Arteris NoC Solution [5, 6] consists of the Danube Intellectual Property Library and a suite of design tools for configuring and implementing the IP library as synthesizable RTL. It uses wormhole packet switching. The Danube Intellectual Property Library contains a set of configurable building blocks managing all on-chip communications between IP cores in SoC designs. The topology is totally customer defined. It supports AXI and OCP protocols. The simulation engine of NoCexplorer [5] computes the dependencies between the occupation time of each resource, arbitrates between transactions and allocates the resources to the transactions. The engine solves a set of equations analogous to description of fluid quanta moving in a network of pipes of various throughputs. The model is not event-driven, and clocks rates are taken into account by computing peak throughput of resources. The model is throughput-accurate and deals with latencies caused by conflicts in accessing resources, and tine spent by packets in buffers.

Xpipes Compiler [3, 4], is a tool for automatically instantiating an application-specific NoC for heterogeneous Multi-Processor SoCs. The Xpipes Compiler instantiates a network of building blocks from a library of parameterized soft macros (switches, network interfaces and links) described in SystemC at the cycle-accurate level. The network components are optimized for that particular network and support reliable, latency insensitive operation.

The group of Chatha [28, 12] develops their own model of NoC component. It's a VHDL based cycle accurate model for evaluating the latency, dynamic and leakage power of NoC based interconnection architecture. Comparing to Xpipe power model, which is obtained by

estimation of dynamic power due to switched capacitances, they get both power consumption due to controller and leakage power using SPICE and simulation. The problem is they don't support high level system simulation for quick verification.

Nostrum [38] is a mesh architecture NoC. It combines Virtual circuit and packet switching to supply both guaranteed and best effect (BE) services by mapping guaranteed service to virtual circuit and BE services to packets. No network interface support is reported and all the results are getting from simulation.

TeraFlop NoC [39] is a 10x8 2-D mesh architecture operating at 4 GHz. Wormhole switching policy is used. The max bandwidth between PEs gets to 80 Gbps. A router interface block (RIB) encapsulates packets between PE and switch.

2.2.4 Case study: Arteris Technology

Arteris [3,4] is developing innovative and patented technology to enable designers to efficiently connect and manage the on-chip traffic requirements among all the various elements required in today's SoC designs.

Arteris[1] SA is a start-up company based in Paris, France and founded in 2003 by a group of semiconductor industry veterans. The company's focus is on the next-generation of challenges associated with system on chip (SoC) design: on-chip communications, or Network-on-Chip (NoC). Arteris is developing innovative and patented technology to enable designers to efficiently connect and manage the on-chip traffic requirements among all the various elements required in today's SoC designs.

A possible topology is illustrated in the Fig.7, which only shows the request network to simplify the graphic. The NoC interfaces all devices through Network Interface Unite (NIU), which are also part of the Danube library. The proposed solution uses three lock domains that match the initiator/target operating frequencies. Transport units are implemented as configurable generators that are parameterized with Arteris NoCcompiler.

[1] Arteris is a registered trademark of Arteris S.A.

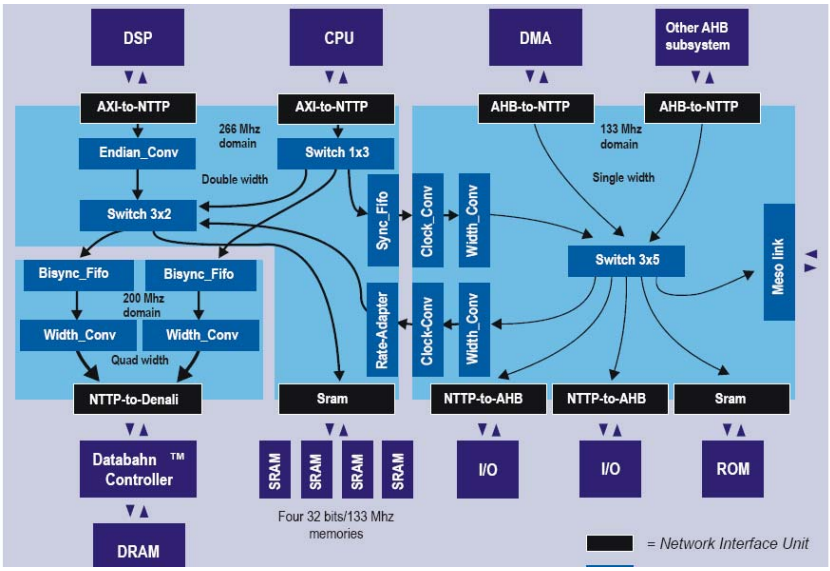


Figure 2.6 Example SOC Design

2.2.4.1 Arteris NoCexplorer

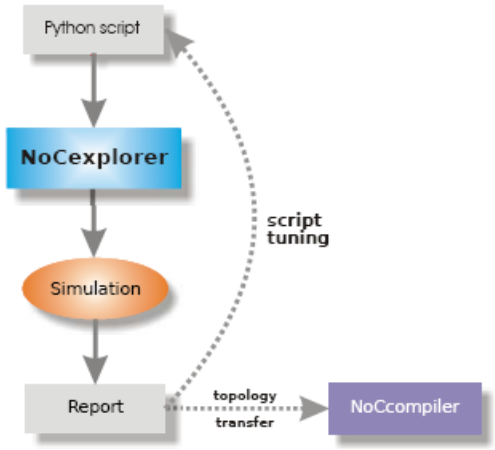


Figure 2.7 NoCexplorer workflow

NoCexplorer is the first link in the chain of Arteris NoC solution suite of Network on chip design tools. NoCexplorer is dedicated to the system architecture phase of the design process. More precisely, it is used to define the NoC interface, its quality of service (QoS) requirement, and elaborate a NoC topology compatible with these constraints.

Input to NoCexplorer is provided in the form of scripts, which relies on a subset of syntax and semantics derive from Python programming language. NoCexplorer simulations combine

IP interface specifications, traffic requirements and QoS constraints with NoC architecture specifications. NoCexplorer then reports the results in QoS pass-fail terms, traffic statistics and performance graphs. When simulation results match the requirements for each SoC use case, the resulting topology can be published into a PDD file, which can be then handed over to designers for actual implementation using NoCcompiler software and Arteris IP library.

2.2.4.2 Arteris NoCcompiler

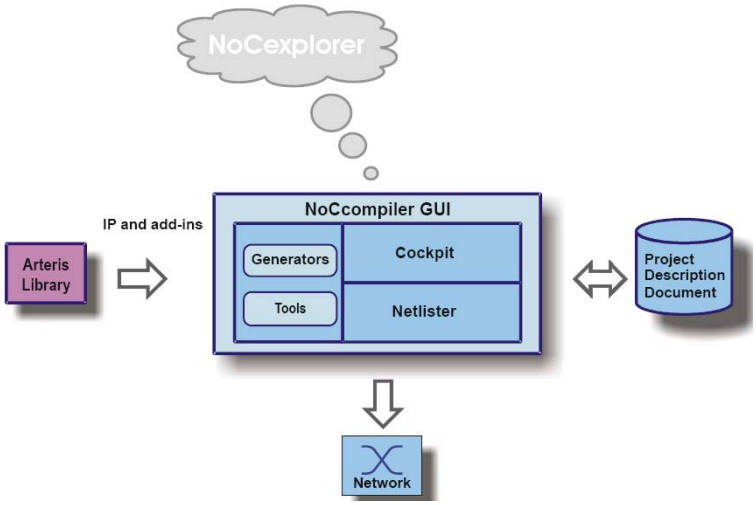


Figure 2.8 NoCexplorer workflow

NoCcompiler is the second stage in the flow of the Arteris NoC Solution suite of Network on Chip design tools. The architecture topology and NoC component specification can be used to implement the NoC using NoCcompiler. NoCcompiler allows to:

- select Arteris NoC IP generators and other add-in generators that are needed to evaluate the NoC design,
- create custom NoC modules into a hierarchical net-list,
- save the resulting design into a project description document (PDD)
- test the NoC in order to identify the connectivity problems, obtain the area estimations, and evaluate the performance in network simulation
- export project into a file compatible with several compatible industrial RTL standard (VHDL, Verilog, SystemC), for simulation or synthesis.

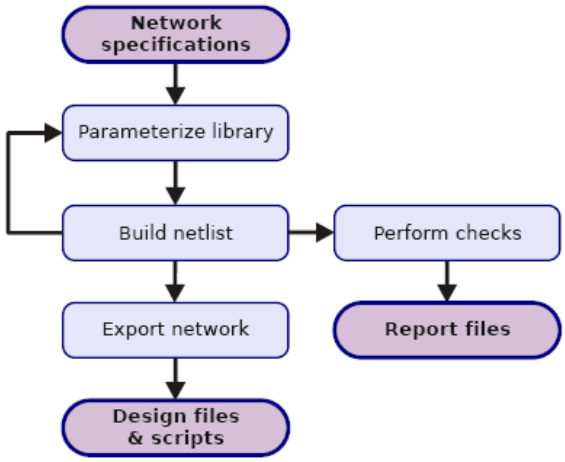


Figure 2.9 NoCcompiler Flow

When NoCcompiler is started, the program opens with an empty project pane. Project is built by three steps:

1. *parameterization*

The following objects should be added into the project:

- folders, to organize design intents, modules and generation
- modules, to instantiate the generation and other modules
- generations, based on one or more generators selected from the NoC IP library
- export option and socket types, also selected from the NoC IP library

When the project is valid, that is, when it contains at least one valid module, a protocol, an export generation, and any other generations required by design, it is ready to pass the next step.

2. *implementation*

Implementing a project involves opening a module in the netlist windows, instantiating the generation or modules required by design and connecting all the nets and ports correctly.

3. *export*

Once the project has been correctly implemented, that is, then the project status icon is green, the design can be exported by various export commands for RTL simulation and implementation. Supported RTL formats are VHDL,Verilog and SystemC.

2.2.4.3 Special Element

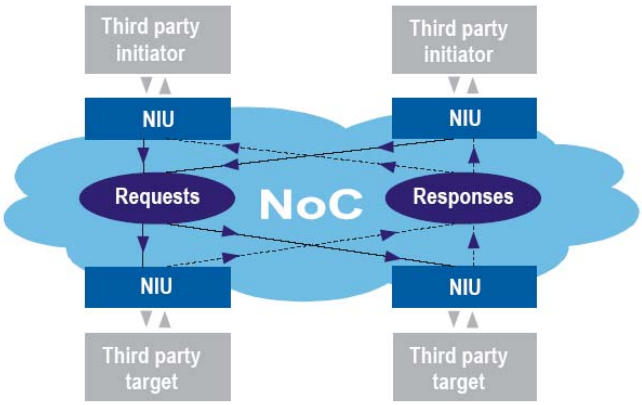


Figure 2.10 packet transport portion

More specifically, the Arteris Danube Library contains the Packet Transport Unit (PTU) generators which build the packet transport portion of the NoC which is comprised of a request network and a response network.

Inside the Arteris NoC has its own transport protocol and interface: NoC Transaction and Transport Protocol (NTTP) and Media Independent NoC Interface (MINI), which make sure its stability and applicability in different design application areas.

Table 2.4 Arteris Danube Library main components

PTU elements	Description
switch	key component that arbitrates and routes packets in the network
mux	arbitrates between several flows and multiplexes them over a single link
sync-fifo	provides buffering to avoid congestion
bisync-fifo	Performs resynchronization from asynchronous domains
Clock-conv	Connect links from distinct clock domains
Rate-adapter	Removes wait cycles that may arise when implementing units such as bisync-fifo
Bandwidth limiter	Prevent initiators from consuming too much bandwidth of a link or a target
Meso-link	Packet transport unit for long distance connections where the endpoints may be asynchronous in GALS systems

2.2.4.4 Arteris switch

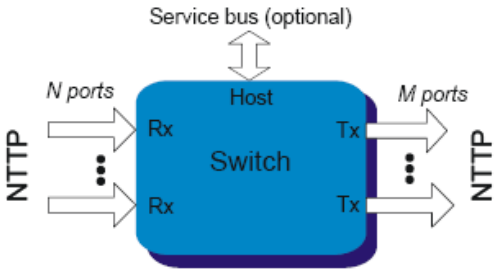


Figure 2.11 Arteris switch interface

Switch is the basic unit of all kinds of Network on Chip. The arteris Danube library includes the switch generator, which is an essential building block of the NoC interconnect system. The goal of this component is to accept NTTP packets got by input ports, and forward each packet to a specific output port.

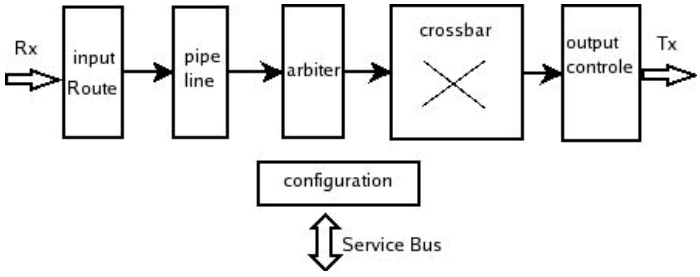


Figure 2.12 Switch internal architecture

The main features of Arteris switch are:

- fully synchronous operation
- internal full crossbar: up to one data word transfer per MINI port and per cycle
- Full throughput arbitration: up to one routing decision per input port and per cycle
- wormhole routing to reduce latency
- Freely cascading connection, supporting any loop-less network topology
- Pressure support for enhanced arbitration decision-making
- Lock support

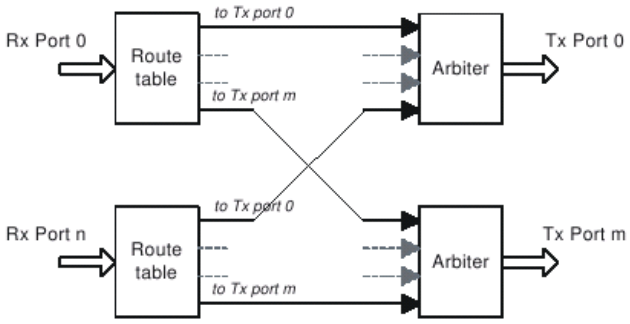


Figure 2.13 functional view of switch

A functional view of the switch potation is presented on the figure above: there is one route table per Rx port, and one arbiter per Tx port. Packet switching consists of four stages:

Stage 1: choosing the route

Using relevant information extracted from the packet received, the route table selects a target output port

Stage 2: Arbitrating

Since more than one single input port can request a given output port at a given time, an arbiter selects one requesting input port per output port. The arbiter maintains the input-output connection until the packet is transited in the switch.

Stage 3: Switching

Once routing and arbitration decisions have been made, the switch transports each word of the packet from input port to output port.

Stage 4: Arbiter release

Once the last word of a packet has been pipelined into the crossbar, the arbiter releases the output, making it available for other packets

The switch component architecture can be customized through several options described in the following table.

Table 2.5 Arteris Danube Library Switch component options

Switch options	**values**
Arbitration type	Round-Robin(0), LRU(1), Random(2), Fifo(3)
Input pipeline register	True(1), false(0)
Forwards pipeline register	True(1), false(0)
Backwards pipeline register	True(1), false(0)
Crossbar pipeline register	True(1), false(0)
Dual cycle arbitration	True(1), false(0)

2.2.4.5 Arteris arbiter library

In a switch, each output port has its own arbiter to decide between multiple packets destined to the same output. The arbitration algorithm is defined per output port. As listed in the table, there are four different arbitration algorithms:

1. *Random*

The RANDOM arbiter uses a 16-bits pseudo-random LFSR that is updated at each cycle. The arbiter's value is used to choose the input ports that are requesting the same output port simultaneously.

2. *Round-Robin*

The Round-Robin arbiter guarantees faire treatment among requests. One of the requests is given the highest priority until it is granted service, then the priority passes to the next request in round-robin order. In an example of 8-input arbiter whose order of priority is: 0,1,2,3,4,5,6,7, if input number 3 requests the output port, the priority order is updated when this input is granted service. The order changes to 4,5,6,7,0,1,2,3.

3. *LRU*

The LRU arbiter gives the highest priority to the least recently serviced request and gives the lowest priority to the most recently serviced one. Assuming an 8-input arbiter whose input priority order is: 0,1,2,3,4,5,6,7. If input number 3 requests the output port, the priority order is updated when the input is granted service and the order is now: 0,1,2,4,5,6,7,3.

4. *FIFO*

The FIFO arbiter takes into account the arrival order of requests and gives the highest priority to the least recently arrived one

2.3 Conclusion

MPSoC is increasingly dominating the landscape of new electronics products. It is different to distributed multicore or multiprocessor as all the processing elements are integrated onto one chip. MPSoC is most appropriate to MIMD architecture. The interconnection architecture of MPSoC greatly impacts the system performance. 6 commercial multiprocessors are compared. Three different interconnection methods have been used: bus, crossbar and network on chip (NoC). And NoC is proposed as the only solution for future large scale MPSoC design. A basic NoC consists of switches, links, and network interfaces. There are many metric to

evaluation NoC architecture such as: Quality of Service (QoS), bandwidth, operating frequency, error tolerance, wire length, latency jitter, or packet loss. We think the most important metrics for NoCs are application execution time, silicon area, power consumption, and latency. These metrics are used in our design space exploration for Pareto front series of solutions. We argue that execution time should be measured on real time rather than cycle numbers, which means system frequency must be at least measured for simulation results. Single transaction level simulation without RTL implementation is meaningless. OCP-IP NoC micro-benchmark is an initiative toward open Network-on-Chip (NoC) Benchmarking. 6 different NoC models and tools are discussed and finally the Arteris Danube NoC library is studied, which is used for our multiprocessor conception.

References

[1] L. Benini and G. De Micheli, Networks on chip: A new SoC paradigm., IEEE Computer, 2002

[2] Kees Goossens; John Dielissen; Om Prakash Gangwal; Santiago Gonzalez Pestana; Andrei Radulescu; Edwin Rijpkema, A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification, in Proc. of Desing, Automation and Test in Europe Conference and Exhibition, 2005

[3] A.Jalabert et. al, ×pipesCompiler: A Tool For Instantiating Application Specific Networks on Chips, Proc. DATE, 2004

[4] S. Stergiou et. al, xpipesLie: A Synthesis Oriented Design Library for Network on Chips, Proc. DATE, 2005

[5] NoCexplorer User's Guide, solution 1.4, Arteris S.A.

[6] NoCcompiler User's Guide, solution 1.4, Arteris S.A.

[7] S. Murali and G. De Micheli, SUNMAP: A tool for automatic topology selection and generation for NOCs. In Proc. of the Design Automation Conf., pages 914-919, June 2004

[8] S. Murali and G. De Micheli, Bandwidth-constrained Mapping of Cores onto NoC Architectures. In Proc. DATE04, June 2004

[9] U. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In Proc. Intl. Conf. on Computer Aided Design, November 2005

[10] U. Ogras and R. Marculescu. Energy- and performance-driven noc communication architecture synthesis using a decomposition approach. In Design, Automation and Test in Europe, March 2005

[11] J. Hu and Radu Marculescu, Exploiting the routing Flexibility for Energy/Performance Aware mapping of Regular NoC Architectures Proc. DATE, 2003

[12] K. Srinivasan; K. S. Chatha; G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 14(4):407–420, April 2006

[13] K. Srinivasan; K. S. Chatha; G. Konjevod, Application Specific Network-on-Chip Design with Guaranteed Quality Approximation Algorithms, Proc. ASP-DAC, 2007

[14] K. Srinivasan; K. S. Chatha, ISIS: A Genetic Algorithm based Technique for Custom On-Chip Interconnection Network Synthesis, Proc. VLSID, 2005

[15] K. Srinivasan; K. S. Chatha, A Methodology for Layout Aware Design and Optimization of Custom Networkon-Chip Architectures, Proc. ISQED, 2006

[16] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In Proceedings of the 35th International Symposium on Microarchitecture (MICRO), pages 294–305, November 2002

[17] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Transactions on Parallel and Distributed Systems 16(2):113–129, February 2005

[18] S. Murali; P Meloni; F. Angionlini et. al. Designing Application-Specific Networks on Chips with Floorplan Information, Proc. ICCAD, 2006
[19] J. G. Kim and Y. D. Kim, A linear programming based algorithm for floorplanning in VLSI design, IEEE *Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 22, no. 5, pp. 584–592, May 2003.
[20] M. Charikar and A. Karagiozova, On non-uniform multicommodity buy-at-bulk network design., In STOC '05: Proc. of the 37-th Ann. ACM Symp. on Theory of Computing, pages 176–182. ACM Press, 2005
[21] A. Pino; Luca Carloni; A. Sangiovanni-Vincentelli, Synthesis of Low Power NOC Topologies under Bandwidth Constraints, Technical Report No. UCB/EECS-2006-137, EECS University of California at Berkeley
[22] A. Hansson; Kees Goossens; A. Radulescu, A Unified Approach to Constrained Mapping and Routing on Network on Chip Architectures, Proc. CODES+ISSS, 2005
[23] G. De Micheli and L. Benini. Networks on chip. Morgan Kaufmann, 2006
[24] GLPK, www.gnu.org/software/glpk/
[25] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness.W. H. Freeman and Company, NY, 1979
[26] S. Murali, L. Benini, and G. De Micheli, Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees, in *ASP DAC*, vol. 1, pp. 27-32, 2005
[27] N. Sherwani, Algorithms for VLSI Physical Design Automation. Kluwer Academic Publishers, 1995
[28] N. Banerjee, P. Vellanki, and K. S. Chatha, A power and performance model for network-on-chip architectures, in *Proc. Des. Automat. Test Eur.*, 2004
[29] David E. Culler and Jaswinder P. Singh, Parallel Computer Architecture, Morgan Kaufmann Publishers, San Francisco,2005
[30] ARM 11 MPCore http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html
[31] MIPS32® 1004K™ Core http://www.mips.com/products/processors/32-64-bit-cores/mips32-1004k/
[32] S.Shibahara, M.Takada, T.Kamei, K. Hayase, Y.Yoshida, O. Nishii, T. Hattori, SH-X3: SuperH Multi-Core for Embedded Systems, Hot Chips 19th, Aug. 19-21 2007, Stanford, USA.
[33] Texas Instruments Multicore Fact Sheet SC-07175
[34] Texas Instruments TMS320C6474 Multicore DSP SPRS552 – Oct. 2008
[35] Texas Instruments TMS320VC5441 Fixed-Point DSP data manual SPRS122F – Oct. 2008
[36] QorIQ™ P4080 Communications Processor http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3bTdG25E4
[37] T.Miyamori, Venezia: a Scalable Multicore Subsystem for Multimedia Applications, 8th International Forum on Application-Specific Multi-Processor SoC 23 - 27 June 2008, Aachen, Germany http://www.mpsoc-forum.org/
[38] M. Millberg, R. T. E. Nilsson, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *DATE*, Feb. 2004, pp. 890–895.
[39] D. Bertozzi *et al.*, "NoC synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 2, pp. 113–129, Feb. 2005.
[40] M. Jersak and K. Richter and R. Ernst. "Performance Analysis for Complex Embedded Applications." In International Journal of Embedded Systems, Special Issue on Codesign for SoC, 2004.
[41] Simon Schliecker and Jonas Rox and Mircea Negrean and Kai Richter and Marek Jersak and Rolf Ernst. "System Level Performance Analysis for Real-Time Automotive Multi-Core and Network Architectures." In Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Volume 28, July 2009, pp. 979-992
[42] X.LI, O.HAMMAMI, "NOCDEX: Network on Chip Design Space Exploration through Direct Execution and Options Selection through Principal Component Analysis", Industrial Embedded Systems, 2006. IES '06. International Symposium on, page(s): 1-4

3. MPSOC Design Methodologies

Homogeneous and heterogeneous multiprocessors are two important and distinct branches of MPSoCs design. Inter-processor communications is very important for MPSoC design. Until now little MPSoC designs are NoC communication architecture based. A survey shows that until now there is no workflow for multiprocessor on chip design based on NoC technology larger than 32 cores. In this thesis, we propose the first industrial tools supported independent design workflow for large scale MPSoC design.

3.1 MPSOC Design and Synthesis

Beside of high performance, embedded on-chip multicore computing is real time restricted, which makes it different from multi-processor computing. Applications like multimedia applications require not only large computation but also strict timing deadline. Furthermore, these MPSoC systems is often designed under strict power and cost budgets.

MPSoCs design has two important and distinct branches: homogeneous and heterogeneous multiprocessors [36]. The homogeneous architecture is good candidate for data-parallel systems. Heterogeneous architectures are designed for heterogeneous applications with complex block diagrams that incorporate multiple algorithms, often using producer–consumer data transfers. MPSoC design targets for applications with multiple algorithms, which demand various computation and communication: types of operations, memory bandwidth and access patterns, activity profiles, etc. Such variations argue for heterogeneous architectures. Hardware and software architecture of MPSoC is tuned to meet the specific requirements of a particular set of applications while providing necessary levels of programmability. Because embedded system designers have specific goals and a better-defined set of benchmarks, they can apply optimization algorithms to solve many design problems.

Inter-processor communications is very important for MPSoC design. Buses are the traditional interconnection architecture for System on Chip (SoC) design. ARM AMBA and IBM CoreConnect buses are well known in the commercial processor design area. The OCP (Open Core Protocol) is proposed as an effective means to simplify the interconnection through the standardization of core interface protocol. As the number of processor in MPSoC grows exponentially, Network on Chip (NoC) is proposed as the only solution for the required communication bandwidth, design scalability and energy restrict. Choosing the right mix of standard buses, point to point communications, shared memory, and emerging network on chip is the one of the most important conception in MPSoC design.

3.1.1 Benchmarks

Benchmarks are so important both in the research and industrial area that they greatly motivate the progress of information technology and better formulation and algorithms. For example, the benchmarks SPEC [1], (Standard Performance Evaluation Corporation) and Mediabench[2] have a good contribution to the improvement of CPU design. In other hand, common benchmarks allow other researchers to redo and compare the results produced by their own with ours, making the communication between researchers more easily.

There are mainly two approaches to benchmarks.

Many researchers have attempted to developed free benchmarks, inclosing Task Graphs For Free [4] (TGFF), RAW benchmarks suit [5] and MiBench[6]. They are designed based on industry standard such as SPEC or EEMBC, more over they are open to download for all the researchers.

TGFF generates task graphs for scheduling and allocation research problems involving periodic or non-periodic task sets, in accordance with the user's parameterized graphs and resources. Users have parametric control over an arbitrary number of attributes for tasks, processors and communication resources. The capability of sharing parameter setting allows researchers to reproduce the examples used by others.

The RAW benchmarks suit is targeted to the reconfigurable computing system. It consists of twelve programs including sorting, matrix operations, and graph algorithms. The most

interesting point is they have developed a RAW Computation Structure which can be automatically compiled into large scale FPGAs without user's intervention.

MiBench is developed based on the industrial standard SPEC2000 and is adapted to the new instruction set of ARM. Following EEMBC, all 35 applications are divided into different suits including: Automotive and Industrial Control, Consumer Devices, Office Automation, Networking, Security, and Telecommunications. MiBench is available as stand C sources which are portable to any platform with compiler, but it contains many file operations which are not easy to realize in diverse real embedded systems without standard file components. And it will take time to allocate these applications into multi-processors as the developers have not considered the parallel computes.

The best choice is to use an industrial standard as our own benchmarks. There have been some efforts in the area of embedded system design, notably the suit developed by the EDN Embedded Microprocessor Benchmark Consortium [3] (EEMBC). Recognizing the difficulty of using just one suit to characterize such a diverse embedded application domain, they have instead produced 9 different suits targeting Automotive, Consumer, Digital Entertainment, Java, Networking, Office Automation, Telecom and an additional suite that allows users to observe the energy consumed by the processor when performing these algorithms and applications. The goal of its members is to make it an industry standard for evaluating the capability of embedded microprocessors, compilers and Java implementations. EEMBC also has MultiBench as a multicore-specific benchmarks that span multiple application areas

MultiBench 1.0 [37] is a suite of embedded benchmarks that allows processor and system designers to analyze, test, and improve multicore architectures and platforms. MultiBench uses standardized workloads and a test harness that provides compatibility with a wide variety of multicore embedded processors and operating systems. It extends the EEMBC scope to analyze multicore architectures, memory bottlenecks, OS scheduling support, synchronization efficiency, and other related system functions. It measures the impact of parallelization and scalability across both data processing and computationally-intensive tasks. It aims to provide an analytical tool for optimizing programs on a specific processor. This is first generation targets the evaluation and future development of scalable SMP architectures.

3.1.2 Design methods of MPSoC

Table 3.1 characteristic of current MPSoC design workflow

Method	description	Objective and algorithm	Processor Size	Architecture	Platform	applications
Simulink to RTL[39,40] (TIMA, Korea, Zhejiang Univ)	Gradual refinement flow from Simulink model to RTL hardware and software specification.	1. Performance of total cycle 2.Manuel mapping tuning	ARM7 Xtensa CKCore <=4	FIFO homogenous	Simulink SystemC TLM FPGA SMIC 0.13 μm	Motion-JEPG H.264
HOPES [41] (HA Korea)	multi-task signal processing application aiming to minimize the system cost while satisfying the real-time constraints.	1. Execution time and system cost 2. exclusive?	High level model <=4	Bus homogenous	SystemC	DSP appls
MAMPS [42] (Eindhoven)	methodology to generate multiprocessor systems in a systematic and fully automated way for *multiple use-cases*	1. Buffer-size and throughput 2. Exclusive for execution ; heuristic for partitioning	MicroBlaze 4	P2P FSL homogenous	FPGA	JPEG, H.263 Mobile
STARSoC [43] (UMR CNRS)	transaction-level modeling co-simulation methodology for embedded open architecture platform	1.Simulation time 2. no DSE	openRISC ISS 2	Bus homogenous	SystemC TLM RTL	OS Embedded application
Daedalus [44-48] (NL)	fully automatic tool-flow for design space exploration, system-level synthesis, application mapping, and system prototyping of MP-SoCs	1.Performance of total cycle and FPGA resource 2. SPEA2	MicroBlaze HW DCT IP PowerPC <=24	P2P CC CB heterogeneous	trace-driven cosimulation FPGA	multimedia
System-CoDesigner [49,50] (Erlanger U)	fast design space exploration and rapid prototyping of behavioral SystemC models synthesized with Forte Design Systems	1. latency, throughput and FPGA resource 2. Heuristic Multi–Objective PB Solver[49]	1 MB 19 HW IP by Forte	P2P FIFO heterogeneous	SystemC FPGA	Motion-JEPG
SoCDAL [51] (CHOI Seoul)	automatic convert the model to hierarchical SDF model and extend	1. Execution cycle and system cost 2. QEA extension	2 ARM7 4 HW models	Bus heterogeneous	SystemC	JPEG, H.263 H.264

	QEA algorithm to efficient application to-architecture mapping					
CoMPSoC [52, 53] (Philips)	integrated flow to automatically generate a highly configurable NoC-based MPSoC for FPGA instantiation.	1. execution time and cycle 2. no DSE	3 Silicon Hive VLIW	NoC homogenous	FPGA	JPEG filter

3.1.2.1 Simulink model based workflow of TIMA and Zhejiang University

A gradual refinement flow starting from Simulink model to a synthesizable and executable hardware is proposed by TIMA and Zhejiang University [39, 40]. The proposed methodology consists of five different abstract levels: (1)Simulink combined algorithm and architecture model (CAAM) for high-level algorithm and architecture specification, (2)virtual architecture (VA) model for early development and validation of the multithreaded application software, (3)transactional accurate architecture (TA) model for fast verification of hardware architecture and operating system (OS) library, (4)virtual prototype (VP) model for accurate system verification and performance estimation, and (5)RTL model for FPGA emulation and ASIC implementation.

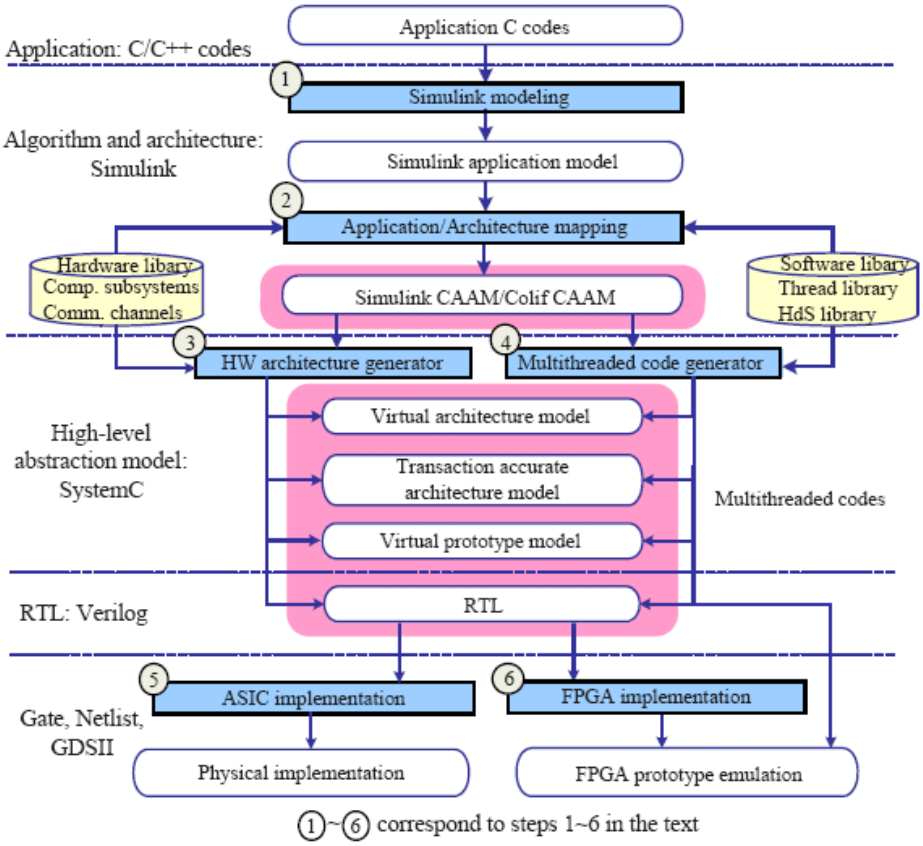


Figure 3.1 Design refinement from application codes to low-level implementation

Application specification is the starting point for the design process, and the mapping of an algorithm to architecture at CAAM level is an essential step from conception to implementation. According to feedback from the simulation at different abstraction levels, the architecture of the CAAM model can be modified until the design requirement is satisfied. But this exploration is reported as manual tuning step and no elite algorithm is used. Vperl enhances the automatic generation of Verilog code form CAAM level. FIFO is used for inter-processor communication, which is not scalable for large scale system and the final implemented system is limited to 4 ARM processors.

3.1.2.2 HOPSE design flow

HOPSE, a DSE based on Synchronous Data Flow (SDF) is described in [41]. It aims to find MPSoC architecture for multi-task signal processing application with single objective to minimize system cost while satisfying the real-time constraints. (1) With a library of HW PEs,

SDF model is used to select processor, map task and estimate performance before simulation; it prunes the exploration space and saves the time. (2) Then HW/SW cosimulation is performed to obtain memory traces. (3) These traces are used to for communication architecture selection. Again, estimation method prunes the design space before trace-driven simulation. (4) Global DES of these two steps is iterated to find the Pareto solutions between system performance and system cost.

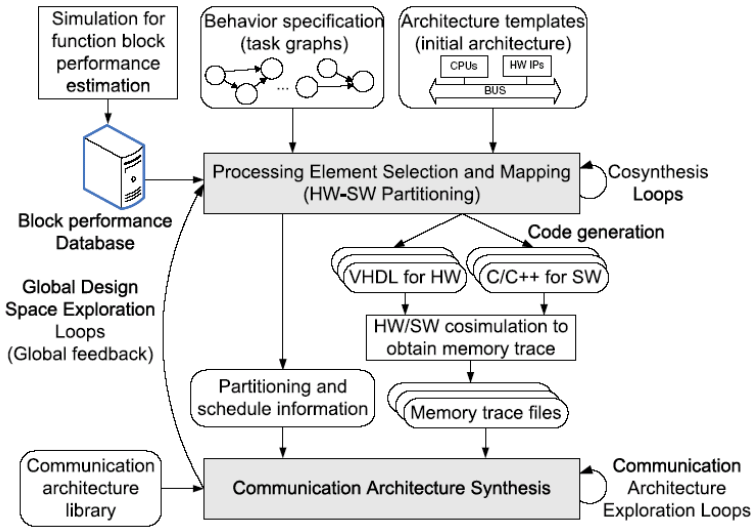


Figure 3.2 HOPSE design flow

This high level DSE separates the cosynthesis and communication synthesis, which in one way make easy the workflow, but in another way adds extensional exploration step. SDF model based estimation accelerates the exploration but is not precise. Bus is the basic communication architecture in design library, but NoC communication is hard to estimate, which make the work less scalable. Only one objective: system cost is considered under real-time constraints. More design objectives should be added to the workflow and combination of RTL level implementation will make the work practice.

3.1.2.3 MAMPS design workflow

MAMPS [42] is a design methodology to generate multiprocessor systems in a systematic and automated way for multiple use-case of multi application on a signal FPGA platform. Multiple use-cases are merged into one hardware design to minimize cost and design time. Heuristics to partition use-cases are also presented such that each partition can fit in an FPGA, and all use-cases can be catered for.

Applications are specified in the form of synchronous data-flow (SDF) graphs. From these application descriptions, a multiprocessor system is generated. The total number of processors in the final architecture corresponds to the maximum number of actors in any application. All the edges in an application are mapped on a unique FIFO channel. Since multiple applications running concurrently, there is often more than one link between some processors. In addition to the hardware topology, the software for each processor is also generated. A use-case represents a collection of multiple applications that are active simultaneously. To merge multiple use-cases into one design to save precious synthesis time and minimize hardware cost. (1) The algorithm iterates over all use-cases to compute their individual resource requirements. (2) Communication matrix is first constructed to generate the entire hardware for FPGA. (3) Software for each use-case is generated with communication matrix and compiled for direct execution on FPGA.

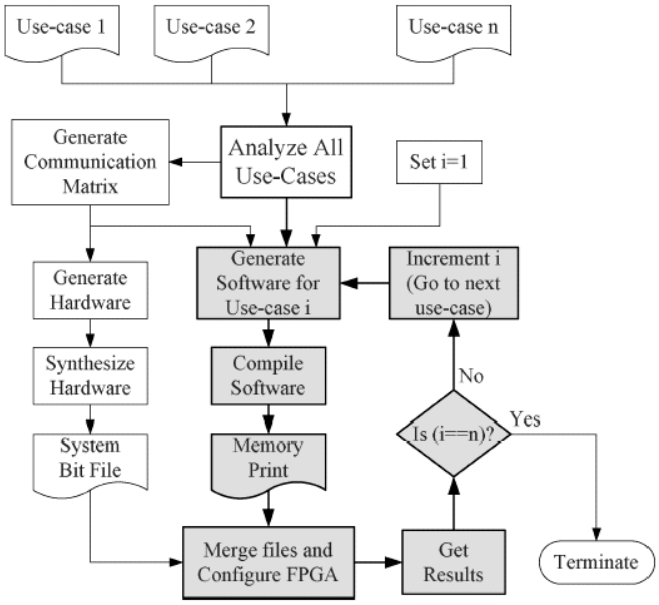


Figure 3.3 MAMPS design workflow for multiple use-case

As the number of use-cases to be supported increases, the minimal hardware design increases as well, and it often becomes difficult to fit all use-cases in a single hardware design. An approximation algorithm, called the greedy algorithm is means to have as few number of hardware partitions as possible. The largest feasible subset of use-cases is first selected and a hardware partition created for it. This is repeated with the remaining use-cases until all use-cases are covered. FPGA resources are estimated as hardware synthesis takes lots of time.

The hardware part is executed only once, whereas the software part is iterated until results for all the use-cases are obtained. This flow makes execution of multiple use-cases a lot faster, since hardware synthesis is no longer a bottleneck in system design and exploration. Lack of real time constraint is the biggest problem of this evaluation design flow. FIFO is feasible for this small scale system but is not scalable as the system getting larger. With a bigger FPGA and large scale MPSoC, applications can be mapped onto separate processing elements and there is no need for this merging technology. By contract, the mapping of multiple applications onto large scale MPSoC is totally different research.

3.1.2.4 STARSoC design workflow

STARSoC [43] is a SystemC Transaction-Level modeling co-simulation flow with integration of openRISC instruction set simulator OR1Ksim and the SystemC simulation components like wishbone bus and memories. The input description consists of a set of communicating parallel software and hardware processes described in C-code. (1) Number of processors and the list of peripheral input/output components connected to each processor are specified. (2)After hardware-software partitioning, the hardware part is synthesized into register-transfer level architecture, and the software part is distributed to the available processors.

Mixed language co-simulators generate a communication overhead between different simulators, often resulting in a significant degradation in the execution time. It is thus necessary to use the same language for modeling software and hardware, and simulating these models at system level in a unified systems design approach. The hardware and software partitions are defined by the user manually and no DSE is available.

3.1.2.5 Daedalus design workflow

Daedalus [44-48] offers a fully integrated tool-flow in which design space exploration (DSE), system-level synthesis, application mapping, and system prototyping of MPSoCs are highly automated. There are specifications at three different levels of abstraction in the flow, namely system level, RTL level and gate level. Application Specification describes an application as a KPN which is the workflow input. (1) SESAME, a design space exploration tool generates the platform and the mapping specifications automatically. These system level specifications are given as input to ESPAM. (2) First, ESPAM constructs a platform instance from the platform

specification and runs a consistency check on that instance. (3) Second, ESPAM refines the abstract platform model to an elaborate (detailed) parameterized RTL model that is ready for an implementation on a target physical platform. (4) Finally, ESPAM generates a program code for each processor in the multiprocessor platform in accordance with the application and mapping specifications. (5) With the hardware descriptions generated by ESPAM, a commercial synthesizer can convert the RTL level specification to the target platform gate level netlist.

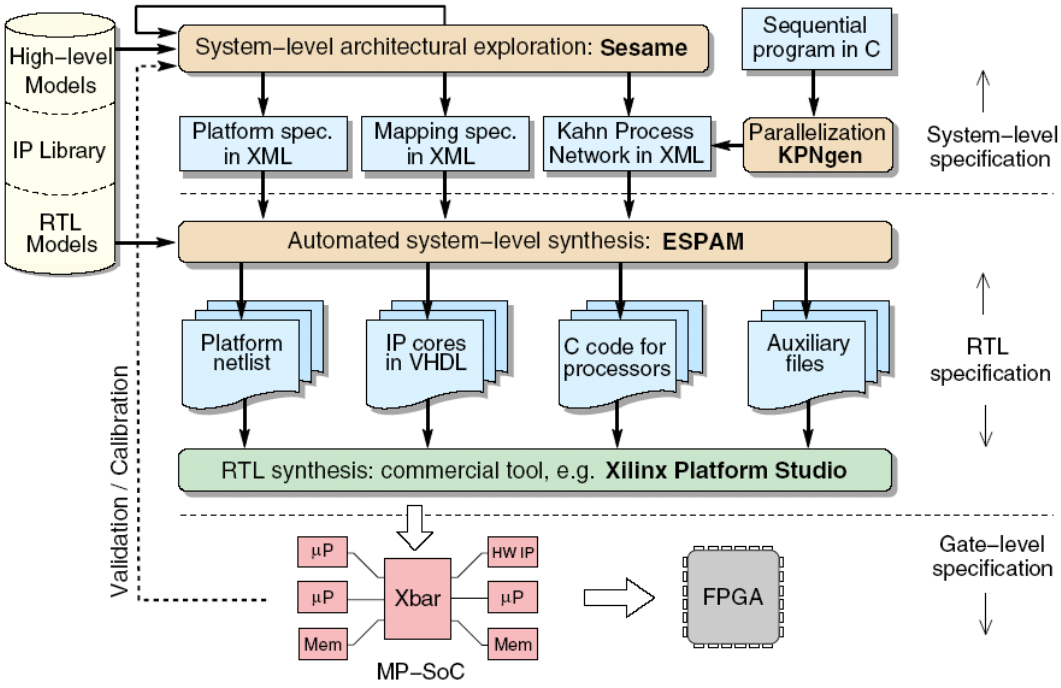


Figure 3.4 Daedalus design workflow with ESPAM and Sesame

The designers believe that a system should be specified at a higher level of abstraction called system level. There is an Implementation Gap between RTL and system level. The RTL system specification is very detailed and close to an implementation while the complexity of today's systems forces us to move to higher levels of abstraction. Base on Embedded System level Platform synthesis and Application Mapping (ESPAM), Daedalus implements a systematic and automated multiprocessor platform design, programming, and implementation in a very short time from system level to RTL level generation. DSE is based on simulation with SESAME tool to find the Pareto solution using SPEA2 genetic algorithm. Then these narrowed design space are implemented onto FPGA to get 100% accurate performance and

cost information. At most 24 cores (MB+DCT) are implemented en FPGA. The communication is FIFO based.

3.1.2.6 System-CoDesigner design workflow

System-CoDesigner [49,50] offers a fast design space exploration and rapid prototyping of behavioral SystemC models. Starting from a behavioral SystemC model, hardware accelerators can be generated automatically using Forte Cynthesizer and can be added to the design space. (1) The application is described in SystemC behavioral model using the SYSTEMOC library. (2) Each SYSTEMOC actor is transformed into both hardware accelerators and software modules. Whereas the latter one is achieved by simple code transformations, the hardware accelerators are built by the help of Forte Cynthesizer. VHDL file is generated and synthesized using Synplify Pro. From the synthesis reports the information required for automatic design space exploration like method execution delays, required look-up tables (LUT), flip flops (FF), and block RAMs (BRAM) are extracted. (3) Architecture template is specified by user as a graph which contains all possible hardware accelerators, processors, memories, buses as well as their interconnection. (4) Design space exploration by multi-objective optimization together with symbolic optimization techniques selects solutions which fulfill the user requirements in terms of overall throughput and chip size. (5) From this set of optimized solutions the designer selects a hardware/software implementation best suited for his needs. They are prototyped for the corresponding FPGA-based implementation. The program code for each microprocessor is generated. Finally, the entire platform is compiled into an FPGA bit stream.

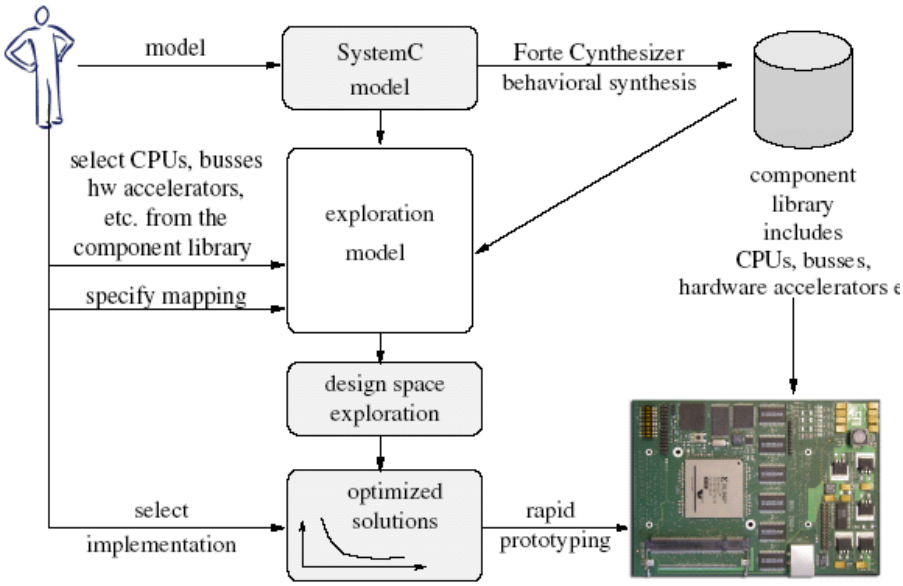


Figure 3.5 System-CoDesigner design workflow with behavioral SystemC model

Application domain is limited to multimedia and networking, i.e., streaming applications, SystemC model to be written using the SYSTEMOC library. In order to transform a SystemC application into a SYSTEMOC description, the input SystemC application is required to only communicate via SystemC FIFOs. The communication architecture is limited.

3.1.2.7 SoCDAL design workflow

SoCDAL [51] is a set of mostly automated tools covering system specification, HW/SW estimation, application-to-architecture mapping, simulation model generation, and system verification through simulation. (1) From a process network model in SystemC, c files and a hierarchical model consisting of SDFs and FSMs are generated using a conversion tool using the SUIF1 compiler. (2) The hierarchical model graph and C codes are the inputs for static SW/HW estimation. For SW estimation, the generated C codes are compiled to create a basic block graph through control flow analysis. The estimated results using ILP solver are annotated into the hierarchical model. (3) For HW estimation, a control dataflow graph (CDFG) is generated from C codes using SUIF1. Subsequently, we perform high-level synthesis on the generated CDFGs to obtain synthesized CDFGs. Estimation is done by ILP solver too. (4) The estimated results are also annotated into the hierarchical model and used to map the application to the target architecture. With the input, the system maps actors to the target architecture using an extended evolutionary algorithm based on QEA. (5)Finally, we

perform synthesis to generate codes to be executed on processors or hard-wired logics according to the mapping result, (6) and then we validate the system by simulation at various levels of abstraction, such as transaction level, bus-cycle-accurate level, etc.

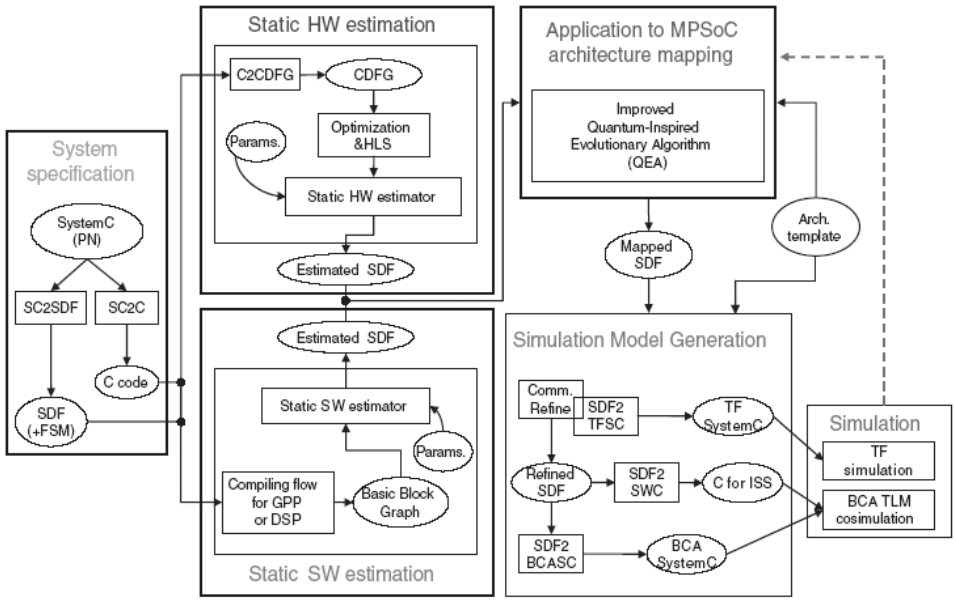


Figure 3.6 SoCDAL design workflow overall

A new approach is introduced to analyze a process network model statically, by automatically converting the model to a hierarchical SDF model and extending the QEA algorithm to apply it to efficient application-to-architecture mapping. It is very useful for hard real-time systems, but cannot support soft and non real-time systems. Dynamic analysis has to be done to estimate the performance of the system. Performance is limited by bus overhead communication. Algorithm is single objective and there is no hardware prototyping.

3.1.2.8 CoMPSoC design workflow

CoMPSoC [52, 53] is an integrated flow to automatically generate a highly configurable NoC-based MPSoC for FPGA instantiation. These are hardware and software flow separately. For hardware, (1) system description together with core description and peripheral description is used to generate RTL code of processor core, peripherals and NoC. The number of hardware resource is fixed. For software (2) core description file is used for compilation of codes. Communication by NoC is produced and linked with AEthereal Run-Time APIs. The

actual assignment to applications and the NoC configuration is reconfigurable to accommodate new or modified applications onto hardware platform.

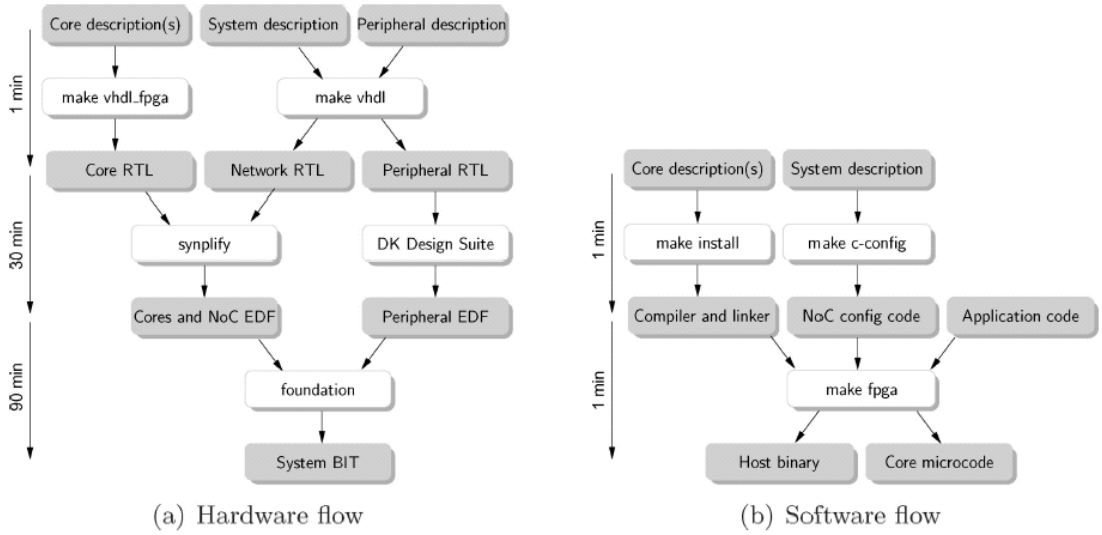


Figure 3.7 CoMPSoC design workflow overall

The architectural components are constructed to offer composability and runtime reconfiguration. The flow uses Æthereal NoC, and Silicon Hive processing cores, both configurable at design- and run-time. No exploration is reported.

3.2 NOC Design and Synthesis

The mapping of regular NoC problem can be formulated mathematically. The communication between the cores of the SoC is represented by the core graph:

The core graph is a directed graph, G(V,E) with each vertex $v_i \in V$ representing a core and the directed edge (v_i, v_j), denoted as $e_{i,j} \in E$, representing the communication between the cores v_i and v_j. The weight of the edge $e_{i,j}$, denoted by $comm_{i,j}$, represents the bandwidth of the communication from v_i to v_j.

The connectivity and link bandwidth of the NoC is represented by the NoC topology graph:

The NoC topology graph is a directed graph P(U, F) with each vertex $u_i \in U$ representing a node in the topology and the directed edge (u_i, u_j), denoted as $f_{i,j} \in F$ representing a direct communication between the vertices u_i and u_j. The weight of the edge $f_{i,j}$, denoted by $bw_{i,j}$,

represents the bandwidth available across the edge $f_{i,j}$.

The mapping of the core graph G(V,E) onto the topology graph P(U, F) is defined by the one-to-one mapping function map:

$$map: V \to U, s.t., map(v_i) = u_j, \forall v_i \in V, \exists u_j \in U$$

The mapping is defined when |V | ⩽ |U|. The communication between each pair of cores (i.e. each edge $e_{i,j} \in E$) is treated as a flow of single commodity, represented as d_k , k = 1, 2, · · · , |E|. The value of d_k represents the bandwidth of communication across the edge and is denoted by vl(dk). The set of all commodities is represented by D and is defined as:

$$D = \begin{Bmatrix} d_k : vl(d_k) = comm_{i,j}, k = 1, 2, ..., |E|, \forall e_{i,j} \in E, \\ source(d_k) = map(v_i), dest(d_k) = map(v_j) \end{Bmatrix}$$

Computation and communication design is now decoupled to enable cores to be designed and validated independently. Decoupling requires well defined communication services, because in many SoCs, on which the applications require real-time performance, service guarantees are essential. Quality-of-Service (QoS) guarantees enable independent design and validation of every part of the SoC by ensuring that real-time application requirements are met under all circumstances [2].

An important phase in the design of NoCs is choosing the most suitable NoC topology for a particular application and mapping of the application on to that topology. The challenges are in leveraging the intrinsic characteristics of on-chip communication to achieve both energy efficiency and high performance [22]. At the specification level, NOC design is made complex by the variety of the processing cores that can be integrated on a chip, (CPUs, micro-controllers, accelerators, memories,...) and the heterogeneity of bandwidth and latency requirements among them. [12, 17] At the implementation level, each target silicon technology offers a multitude of options to the NoC designers who, for instance, must decide the number and positions of network access points and routers as well as which metal layer to use for implementing each given channel.

In particular, choosing the network topology is challenging as the space of possible topologies is very large. [6, 16, 20] Hence, it is very difficult to guess the right communication topology only by experience, taking into account the heterogeneity of the requirements and the constraints imposed by the silicon technology. Consequently, the development an automatic

tool for optimal topology selection for on-chip networks is of great help to the NOC design paradigm.

3.2.1 Graph theory related material and GLPK

Table 3.2 characteristic of network analysis program

Library name	description	language	File format	OS	Algo support
igraph [28]	Handles very large graph and neat interface to python and R	C/R/Python	GraphML dimacs Pajek	Unix no gui	Little basic
Pajek [29]	Can handle Very Large Networks (Millions of Nodes)	C(?)/R	Pajek SVG EPS X3D VRML	Win, good gui	little Powerful visual
NetworkX [30]	Can handle very large networks (Millions of Nodes)	Python	Agraph dot vtk matplotlib	Unix win no gui	Basic Easy exchange
GraphViz [31]	Graph Layouts	C	Agraph dot dotty	Unix win Mac	Graph drawing interface
Boost [32]	Good reputation Traversals, Spanning Tree...	C++	Agraph exrensible	All no gui	Lots typical extensible
Jgraph [33]	Visualization and basic algos with nice interactive plcmt	Java	UML XML strings	All swing	Basic extensible
Goblin [34]	Standard Textbook Graph Optimization Pbs	C++/Tcl	Goblin dimacs tsplib steinlib	Unix win gui	Rich extensible
LEDA [35]	Good commercial library	C++	?	Win unix gui/pakcet	Rich extensible

Graph theory is the study of graphs, mathematical structures used to model pair wise relations between objects from a certain collection. A "graph" in this context refers to a collection of vertices and a collection of edges that connect pairs of vertices. A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pairwise connections have some numerical values. For example if a graph represents a road network, the weights could represent the length of each road. A digraph with weighted edges in the context of graph theory is called a network. And this is exactly the study domain of NoC.

There are many program tools to analyze and visualize graphs as listed in the table II.

The power consumption problem of NoC design is widely studied. [12, 13, 16, 20, 21] Same of them take the power consumption of NoC as the linear combination of switch and physical link [12, 13, 21] which is based on the equation is based on experimental results.

Linear programming (LP) problems involve the optimization of a linear objective function, subject to linear equality and inequality constraints. There are many mature tools for solving LP problems like GLPK, Xpress-MP and other open source or commercial tools.

The GLPK [24] (GNU Linear Programming Kit) package is intended for solving large-scale linear programming (LP), mixed integer programming (MIP), and other related problems. It is a set of routines written in ANSI C and organized in the form of a callable library.

GLPK supports the GNU MathProg language, which is a subset of the AMPL language.

The GLPK package includes the following main components:

- Revised simplex method.
- Primal-dual interior point method.
- Branch-and-bound method.
- Translator for GNU MathProg.
- Application program interface (API).
- Stand-alone LP/MIP solver.

3.2.2 Workflow for regular topology

The mapping of regular NoC problem can be formulated mathematically. The

communication between the cores of the SoC is represented by the core graph:

Table 3.3 characteristic of regular noc workflow

method	topo	object	mapping	routing	floor planning
Hu [11]	Tile based	Min energy	Branch and bound algo	Static xy routing	no
Ogras [9]	Tile based mesh	Min latency	Supposed done	long-rang link Iterative algo	no
SUNMAP [7,8]	Mesh torus … Other regular NoC	Min latency area energy	[7]Heuristic greedy pair wise swapping [8]robust tabu search	minimum path Dijkstra's shortest path algo	area -power model; floorplanafter mapped
NMAP [8]	Mesh torus	Min latency	Heuristic greedy	minimum path Dijkstra/ split traffice LP solve	no
Aethereal [2]	mesh	guaranteed ser/ best effort	Core clustering based on balance	Static xy routing	Phillips power measure/0.13 um proc
UMARS [22]	mesh	Mini latency area energy	Mapping during routing	Heuristic iterative combine mapping scheduling	Phillips power measure/ 0.13 um proc

Creating a NoC-based system with guaranteed services requires efficient mapping of cores and distribution of NoC resources. Design choices include mapping core port to network port, routing of communication between cores and schedulng of network channel capacity over time. These choices have significant impact on energy, area and performance metrics of the system.

Many existing solutions decouple the mapping and routing on two separate steps [2, 7, 11, 13] whose objectives do hereby not necessarily coincide. The routing phase must adhere to decisions taken in the mapping phase which invariably limits the routing solution space. Mapping therefore significantly impacts energy and performance metrics of the system. In the table III the routing and mapping are listed separately, but it should be clear that routing and corrective traffic arbitration is closely bound to the first step of mapping.

In [11] a branch-and-bound algorithm is used to map cores onto a tile-based architecture, aiming to minimize energy while bandwidth constraints are satisfied while balancing network load. Static *xy* routing is used in this work. And work of ogras[9] reduces the packet latency by inserting long-rang link, which is a fresh ideal comparing to the traditional design methods.

In [7, 8, 9] a heuristic improvement method is used. An initial mapping is derived with objectives such as minimizing communication delay, area or power dissipation. This is succeeded by routing according to a predefined routing function. Routing and evaluation is repeated for pair-wise swaps of nodes in the topology, thereby exploring the design space in search for an efficient mapping. In [9] the algorithm integrates physical planning and QoS guarantees. Design space exploration is improved with a robust tabu search.

In all these approaches [11, 7, 8, 9], multiple mapping and routing solutions are evaluated iteratively to mitigate the negative effects mapping decisions may have on routing. A greedy non-iterative algorithm is presented in [2]. Mapping is done based on core clustering whereafter communication is routed using static *xy* routing. A guaranteed performance is essential to replace time-consuming simulation by fast analytical performance validation.

UMARS [22] unified single objective algorithm unifies three steps: spatial mapping of cores, spatial routing of communication and scheduling of traffics. The fundamental difference if that mapping is no longer done prior to routing but instead during it. Comparing to early aethereal [2] method, UMARS reduces area and power by more than 33% and wore-case latency by a factor four in an MPEG decoder SoC.

As a conclusion, regular topology NoC design problem can be modeled as mapping the weighed communication core graph onto different NoC networks, in this section the network topology is regular and in the next section irregular. The general problem of embedding one graph onto another is NP-complex and is a special case of the Quadratic Assignment Problem (QAP) [25].

Of all the algorithms talked above, only SUNMAP and related works [7, 8, 9] take floorplan information as input of design workflow. They use ORION [16] to set up the xpipe power model and specific LP floorplanning tools [19, 27] to get area model and physical placement of IP cores and switches.

3.2.3 Workflow for application specific topology

Table 3.4 comparison of custom NoC workflow

method	algo	workflow	Floorplan/power	remark
NetChip [17]	Minimum path mapping/ Heuristic swapping	1] topology mapping 2] topology selection 3] topology generation	LP-based floorplanner [27]/ ORION[16]	1] fixed topo library 2] manual custom NoC by GUI
INI-TOP [18]	Min-cut partition/ clustering heuristic	1] min-cut partition of core graph 2] switch cost graph with prohibited turn set 3] path_compute to minimize power 4] floorplan evaluate wire power	PARQUET[19] or Cadence SoC encounter/ Synopsys PrimePower	1] Chaco for graph partition 2] PTS to avoid dead-lock 3]floorplan info is not used in min-cut
ANOC [12]	ILP with Xpress-MP/ clustering heuristic	1] system level floorplannig with an objective of minimizing the power consump of the NoC with layout constraints 2] router selection and NoC topology 3] map and route minimizing power with perf constraints	Customized PARQUET[19] ?/ Arizona SPICE experiment	1] floorplan before topology generation 2] a linear combination of the power-latency function and area of layout
GQA [13]	1] max flow min cut: Rush-Relabel algo 2]2approximation	1] core to router mapping with floorplan information 2]routing and topology generation	Same to ANOC	1] custom floorplan 2] mapping separated from routing
ISIS [14]	GA	1] hierarchical representation of solution 2] GA evaluation	? Arizona	1] hierarchi representation of NoC solution
BAB	buy-at-bulk algo	1] placement of cores	PARQUET[19]/	1] place without

[21]		with pre-allocation of area for NoC 2] discretize NoC area for graph PLT 3] map and route minimize power	ORION [16]	optimal 2] variable NoC area 3] variable router position

NoC architectures can be designed with both regular and irregular topologies. The primary advantage of a regular NoC architecture is topology reuse and reduced design time. They are suitable for general purpose architectures, such as the RAW processor that include homogeneous cores. Regular topologies assume that every core has equal communication bandwidth with every other core which does not hold in custom SoCs. Application-specific SoC architectures consist of heterogeneous cores and memory elements which have vastly different sizes. Consequently, even if the system-level topology is regular, it does not remain regular after the final floorplanning stage. The alternative option of regular layout results in a large amount of area overhead.

The custom NoC architecture is superior to regular architecture in terms of power and area consumption under identical performance requirements. [12, 17, 18] In custom topologies, the router architecture itself is regular and can be easily parameterized for reuse.

In nanoscale technologies, the link energy consumption will constitute a considerable part of the total communication energy. Therefore, the total energy consumption of the NoC is strongly influenced by system-level floorplan. All of the application specific NoC design [12, 18, 21] have computed the power consumption by both the physical links and routers.

In the NetChip [17] design flow, the topology is selected by an heuristic algorithm that tries many fixed topologies described in a library and selects the best one according to the user objective function. The designer can use xpipecompiler to generate a custom NoC using the GUI. A two-state Markov Models as stochastic traffic generators model the bursty nature of the application traffic, with average communication bandwidth matching the real application's. The general solution to the floorplanning problem has two basic steps: first is finding out the relater position of modules and the second is finding the exact position, area and size of the modules [37]. In NetChip situation, the relative positions of modules are known. Thus, a simple LP-based floorplanner existing in literature [38] is used.

The work INI-TOP [18] of Murali et. al. minimizes the power consumption of switches considering the activities of switches in their algorithm PATH_COMPUTE, which maps the inter-partition communication flows to physical paths as less as possible while guaranteeing deadlock freedom and communication bandwidth constraints. The generated NoC is synthesized by floorplanner, which compute the design area and wire length minimizing a dual-objective function of area and wire length. From the obtained wire length the power across the wires is calculated. So the optimization of router and wire power consumption is separated in this workflow. As discussed in the work of Chatha's[12, 13], the various sizes of different cores impact the floorplanning of IP cores and routers greatly and consequently impact the mapping of IP core to router, which is done before floorplanning in Murali's work. That is to say the min-cut partition dos not take into account of the floorplan information, which is not a good practice.

As the power consumption of wire is getting larger part of whole system, the work of K. Srinivasan and K. Chatha [12,13,14] is a good practice to combine the router and wire power consumption of NoC together with physical floorplan placement information. They set up the ILP objective function to minimize power consumption and propose different heuristic algorithms to accelerate the execution time like: clustering in [12], Rush-Relabel algorithm and 2-approximation in [13] and GA algorithm in [14]. The limitation is that their work is built on the assumption that the area of router is much less than IP cores.

A. Pinto et. al. [21] adopt the same workflow as K. Chatha's [12]: floorplan, generation of admissible router position, optimal routing. They rely on the modified buy-at-bulk approximation algorithm to exploit the large design space more efficiently. They consider the size of router and give a changeable area for the routers and wires. Router is placed around the IP cores instead of the four corners in Chatha's work [12]. They simplified the physical placement of IP cores without optimization functions.

3.3 Conclusion

Real time constrains must be considered during MPSoC design. Homogeneous and heterogeneous multiprocessors are two important and distinct branches of MPSoCs design. Inter-processor communications is very important for MPSoC design. Until now little MPSoC designs are NoC communication architecture based. A survey of design benchmarks and

MPSoC design methodology is done. Analysis and comparison help to better understand different design approach and overcame their shortcomings.

MAMPS is lack of real time constraint is the biggest problem of this evaluation design flow. FIFO is feasible for this small scale system but is not scalable as the system getting larger. With a bigger FPGA and large scale MPSoC, applications can be mapped onto separate processing elements and there is no need for this merging technology.

Daedalus communication is FIFO based. At most 24 cores (MB+DCT) are implemented en FPGA. the scale of MPSoC is still limited.

For System-CoDesigner, in order to transform a SystemC application into a SYSTEMOC description, the input SystemC application is required to only communicate via SystemC FIFOs. The communication architecture is limited.

To our best knowledge, until now there is no workflow for large scale multiprocessor on chip design based on NoC technology. In this thesis, we propose the first industrial tools supported independent design workflow for large scale MPSoC design.

Floorplan constrained NoC synthesis is getting suitable for real application specific SoC design. Custom NoC with its better performance and less power consumption comparing to regular topologies, is currently studied. The mapping of core graph to NoC topologies is well known NP-Hard, only heuristic algorithms can be used to get the optimal solution according to different objective functions. The ILP formulation of NoC power consumption is a good practice. Different approximation algorithms are proposed to reduce the execution time of IPL problem.

References

[43] L. Benini and G. De Micheli, Networks on chip: A new SoC paradigm., IEEE Computer, 2002

[44] Kees Goossens; John Dielissen; Om Prakash Gangwal; Santiago Gonzalez Pestana; Andrei Radulescu; Edwin Rijpkema, A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification, in Proc. of Desing, Automation and Test in Europe Conference and Exhibition, 2005

[45] A.Jalabert et. al, ×pipesCompiler: A Tool For Instantiating Application Specific Networks on Chips, Proc. DATE, 2004

[46] S. Stergiou et. al, xpipesLie: A Synthesis Oriented Design Library for Network on Chips, Proc. DATE, 2005

[47] NoCexplorer User's Guide, solution 1.4, Arteris S.A.

[48] NoCcompiler User's Guide, solution 1.4, Arteris S.A.

[49] S. Murali and G. De Micheli, SUNMAP: A tool for automatic topology selection and generation for NOCs. In Proc. of the Design Automation Conf., pages 914-919, June 2004

[50] S. Murali and G. De Micheli, Bandwidth-constrained Mapping of Cores onto NoC Architectures. In Proc. DATE04, June 2004
[51] U. Ogras and R. Marculescu. Application-specific network-on-chip architecture customization via long-range link insertion. In Proc. Intl. Conf. on Computer Aided Design, November 2005
[52] U. Ogras and R. Marculescu. Energy- and performance-driven noc communication architecture synthesis using a decomposition approach. In Design, Automation and Test in Europe, March 2005
[53] J. Hu and Radu Marculescu, Exploiting the routing Flexibility for Energy/Performance Aware mapping of Regular NoC Architectures Proc. DATE, 2003
[54] K. Srinivasan; K. S. Chatha; G. Konjevod. Linear-programming-based techniques for synthesis of network-on-chip architectures. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 14(4):407–420, April 2006
[55] K. Srinivasan; K. S. Chatha; G. Konjevod, Application Specific Network-on-Chip Design with Guaranteed Quality Approximation Algorithms, Proc. ASP-DAC, 2007
[56] K. Srinivasan; K. S. Chatha, ISIS: A Genetic Algorithm based Technique for Custom On-Chip Interconnection Network Synthesis, Proc. VLSID, 2005
[57] K. Srinivasan; K. S. Chatha, A Methodology for Layout Aware Design and Optimization of Custom Networkon-Chip Architectures, Proc. ISQED, 2006
[58] H. S. Wang, X. Zhu, L. S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In Proceedings of the 35th International Symposium on Microarchitecture (MICRO), pages 294–305, November 2002
[59] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. IEEE Transactions on Parallel and Distributed Systems 16(2):113–129, February 2005
[60] S. Murali; P Meloni; F. Angionlini et. al. Designing Application-Specific Networks on Chips with Floorplan Information, Proc. ICCAD, 2006
[61] J. G. Kim and Y. D. Kim, A linear programming based algorithm for floorplanning in VLSI design, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 22, no. 5, pp. 584–592, May 2003.
[62] M. Charikar and A. Karagiozova, On non-uniform multicommodity buy-at-bulk network design., In STOC '05: Proc. of the 37-th Ann. ACM Symp. on Theory of Computing, pages 176–182. ACM Press, 2005
[63] A. Pino; Luca Carloni; A. Sangiovanni-Vincentelli, Synthesis of Low Power NOC Topologies under Bandwidth Constraints, Technical Report No. UCB/EECS-2006-137, EECS University of California at Berkeley
[64] A. Hansson; Kees Goossens; A. Radulescu, A Unified Approach to Constrained Mapping and Routing on Network on Chip Architectures, Proc. CODES+ISSS, 2005
[65] G. De Micheli and L. Benini. Networks on chip. Morgan Kaufmann, 2006
[66] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness.W. H. Freeman and Company, NY, 1979
[67] S. Murali, L. Benini, and G. De Micheli, Mapping and Physical Planning of Networks-on-Chip Architectures with Quality-of-Service Guarantees, in ASP DAC, vol. 1, pp. 27-32, 2005
[68] N. Sherwani, Algorithms for VLSI Physical Design Automation. Kluwer Academic Publishers, 1995
[69] Igraph, http://cneurocvs.rmki.kfki.hu/igraph/
[70] Pajet, http://vlado.fmf.uni-lj.si/pub/networks/pajek/
[71] NetworkX, https://networkx.lanl.gov/wiki
[72] GraphViz, http://www.graphviz.org/
[73] Boost, http://www.boost.org/libs/graph/doc/index.html
[74] Jgraph, http://jgrapht.sourceforge.net/
[75] Goblin, http://www.math.uni-augsburg.de/~fremuth/goblin.html
[76] LEDA, http://www.algorithmic-solutions.com/enleda.htm
[77] N. Banerjee, P. Vellanki, and K. S. Chatha, A power and performance model for network-on-chip architectures, in Proc. Des. Automat. Test Eur., 2004
[78] Wayne Wolf, Ahmed A. Jerraya and Grant Martin, Multiprocessor System-on-Chip (MPSoC) Technology, IEEE trans. On CAD of integrated circuits and systems. VOL. 27, NO. 10, Oct. 2008
[79] EEMBC, MultiBench™ 1.0 Multicore Benchmark Software , http://www.eembc.org/benchmark/multi_sl.php
[80] Shay Gal-On, Markus Levy, "Measuring Multicore Performance," Computer, vol. 41, no. 11, pp. 99-102, Nov. 2008

[81] Sang-II Han et al. , Simulinks-based heterogeneous multiprocessor SoC design flow for mixed hardware/software refinement and simulation , INTEGRATION,the VLSI journal, Volume 42, Issue 2, February 2009, Pages 227-245
[82] Kai HUANG et al, Gradual refinement for application-specific MPSoC design from Simulink model to RTL implementation , Journal of Zhejiang University SCIENCE A, Volume 10, Number 2 / February, 2009
[83] Choonseung Lee, Sungchan Kim and Soonhoi Ha , A Systematic Design Space Exploration of MPSoC Based on Synchronous Data Flow Specification , Journal of Signal Processing Systems, March 11, 2009
[84] AKASH KUMAR et al, Multiprocessor Systems Synthesis for Multiple Use-Cases of Multiple Applications on FPGA , ACM Transactions on Design Automation of Electronic Systems, Volume 13 , Issue 3 ,July 2008
[85] Sami Boukhechem and El-Bay Bourennane , SystemC Transaction-LevelModeling of an MPSoC PlatformBased on an Open Source ISS by Using Interprocess Communication , International Journal of Reconfigurable Computing, Volume 2008 (2008),
[86] Pimentel, A.D. , Erbas, C. and Polstra, S. , A systematic approach to exploring embedded system architectures at multiple abstraction levels, IEEE Transactions on Computers, Volume: 55, Issue: 2, Feb. 2006
[87] Andy Pimentel et al. , "Tool Integration and Interoperability Challenges of a System-level Design Flow: a Case Study", Invited paper In Proc. "8th Int. Sumposium on Systems, Architectures, MOdeling, and Simulation (SAMOS'08)", LNCS 5114, pp. 167-176, Samos, Greece, July 21-24, 2008.
[88] Hristo Nikolov, Todor Stefanov, and Ed Deprettere, Automated Integration of Dedicated Hardwired IP Cores in Heterogeneous MPSoCs Designed with ESPAM, EURASIP Journal on Embedded Systems Volume 2008
[89] Pimentel, A.D., Erbas, C., Polstra, S.: A systematic approach to exploring embedded system architectures at multiple abstraction levels. IEEE Trans. on Computers 55, 99–112 (2006)
[90] Erbas, C., Pimentel, A.D., Thompson, M., Polstra, S.: A framework for system-level modeling and simulation of embedded systems architectures. EURASIP Journal on Embedded Systems (2007)
[91] Martin Lukasiewycz, Michael Glaß, Christian Haubelt, and Jürgen Teich. Efficient Symbolic Multi-Objective Design Space Exploration. In Proceedings of the 13th Asia and South Pacific Design Automation Conference (ASP-DAC 2008). pp. 691-696, Seoul, Korea, January 2008
[92] Christian Haubelt, Mike Meredith, Thomas Schlichter, and Joachim Keinert. SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models. In Proceedings of the 2008 ACM/EDAC/IEEE Design Automation Conference (DAC 2008). pp. 580-585, Anaheim, CA, U.S.A., June 8-13 2008.
[93] YONGJIN AHN et al. , SoCDAL: System-on-chip design AcceLerator, ACM Transactions on Design Automation of Electronic Systems, Volume 13 , Issue 1 ,January 2008
[94] Andreas Hansson, Kees Goossens, Marco Bekooij and Jos Huisken, CoMPSoC: A template for composable and predictable multi-processor system on chips, ACM Transactions on Design Automation of Electronic Systems, Volume 14 , Issue 1 January 2009
[95] Akash Kumar, Andreas Hansson, Jos Huisken and Henk Corporaal. An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE), April 2007.

4. Multi-objective TLM Level NOC Design Space Exploration

A multi-objective design space exploration of NoC at TLM level is proposed at the first step. Automatic exploration is needed in order to guarantee the evaluation of all potential solutions. Although some work has been achieved in this area, the proposed design space explorations are based on different levels of abstraction. SystemC TLM hides many of the implementation details. It allows fast simulation of complex systems however at the price of less accuracy. How to manage design space exploration with this lack of accuracy is the objective of this work. Design space exploration becomes a challenge with multi-objective evaluation since area and timing are subjects to be extracted from lower levels of abstraction. The TLM representation of NoC requires deep insight and implementation experience in order to interpret correctly the embedded semantics at this level of abstraction. We propose an automatic approach to this problem with three instances: (1) best-effort optimization (2) latency constrained and (3) area and latency constrained. Application of our methodology on a 16 processors MPSOC validates our approach

4.1 NOC SystemC TLM Modelling and Traffic Generators

Network on chip based Multiprocessors system on chip are increasingly complex to design and tune for emerging applications [1-3]. Ad-hoc techniques can not meet the requirements of the increasingly complex traffic flows in system on chip. Automatic design space exploration is needed in order to guarantee the evaluation of all potential solutions. Although some work have been achieved in this area the design space exploration differs depending on the level of abstraction. We address in this chapter the design space exploration of network on chip at SystemC TLM level. SystemC TLM hides many of the implementation details and as such

allows fast simulation of complex systems however at the price of less accuracy. How to manage design space exploration with this lack of accuracy is the objective of this work.

4.1.1 SystemC TLM

SystemC TLM 2.0 has been released recently [19] and it allows high level modeling of system on chip with various complexities. Since SystemC TLM is transaction based, communication between IPs is considered to be conducted at transaction level. The use cases and coding styles are described in Figure 4.1 and architectural analysis may use the three available coding styles that are unlimited, loosely-timed and approximately-timed. It is clear that application specific design space exploration of network on chip based multiprocessor requires the highest possible accuracy in order to provide the right amount of supporting resources to meet timing constraints (i.e. latency) and especially if area constraints are added. Although TLM abstraction level introduces some amount of inaccuracy [17] lowering this amount is paramount.

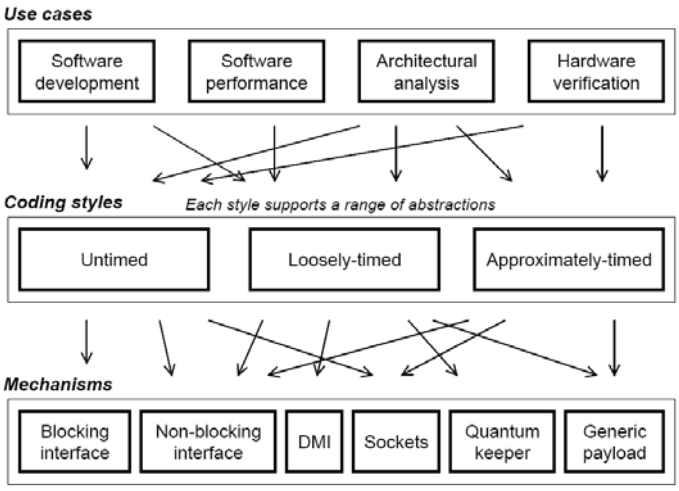


Figure 4.1 TLM, Use Cases and mapping

In this hypothesis packet based, wormhole based network on chip represents a challenge.

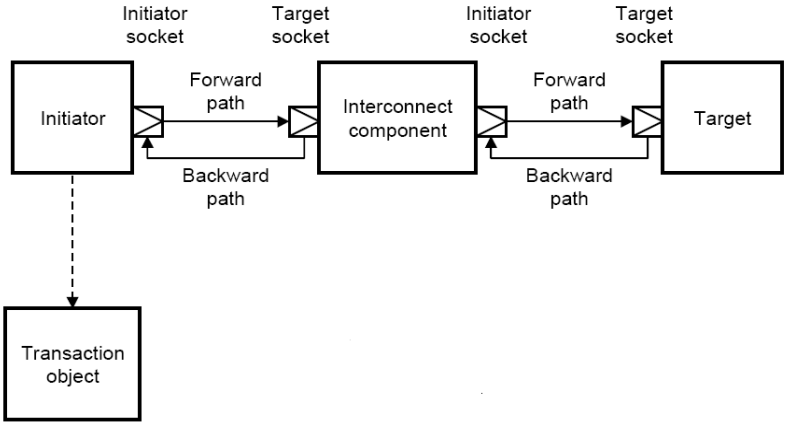


Figure 4.2 TLM Communication

The TLM communication starts with a transaction from an initiator to a target through some interconnect component which in our case represents the network on chip and cannot be as accurate as cycle level switches. The concepts used in this study are the queue which represents a line of ordered transactions usually associated with an initiator socket. A link is a physical medium characterized by its clock and a width with possible FIFO buffering capacity. Various arbiters can be used for links and defining an architecture consists in describing routing across a network of links. The connectivity map describes the addressing space of each initiator.

4.1.2 TLM NoC modeling with commercial tools

We used in this study an industrial design tools: the Arteris NoC Solution [20-23]. As seen from Figure 4.3, Arteris NoC solution contains two EDA tools: NoCexplorer [22] and NoCcompiler [21], focusing on different levels of simulation and implementation. NoCexplorer is a NoC generation and simulation tools using SystemC TLM language. Arteris' cycle based model accelerates the simulation speed. NoCcompiler can be used to generate VHDL or SystemC RTL source of NoC.

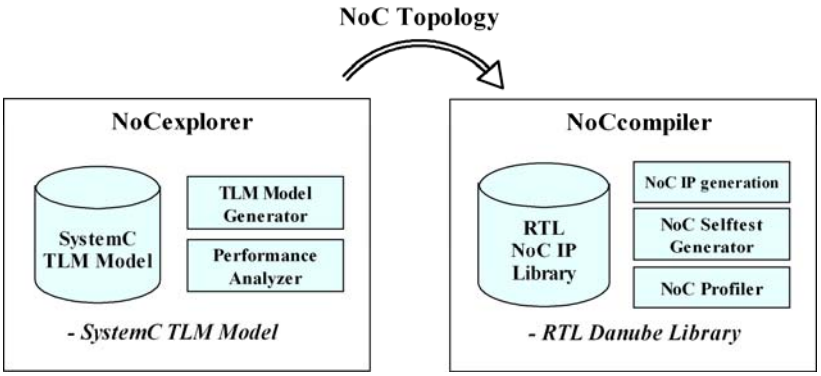


Figure 4.3 Aretris NoC design tools.

At this TLM level, the topology of NoC is represented by 'links'. The basic component of NoC architecture using NoCexplorer as showed in the Figure 4.4. A link is characterized by its clock and width, and possibly associated with a FIFO buffer capacity. The links carry transaction request and response packets between each initiator-target pair. Any topologies, regular or irregular, can easily be described using 'links'. For each pair of source and destination, we describe the link sequence, in which the communication passes through, as its routing. To define the architecture of NoC, a NoCexplorer script essentially describes the routing across a network of links instead of switches. The merging point of different links is like a switch at RTL level but which is not a module at TLM level. The NoC and master-slave models are described in a script file as the input of NoCexplorer.

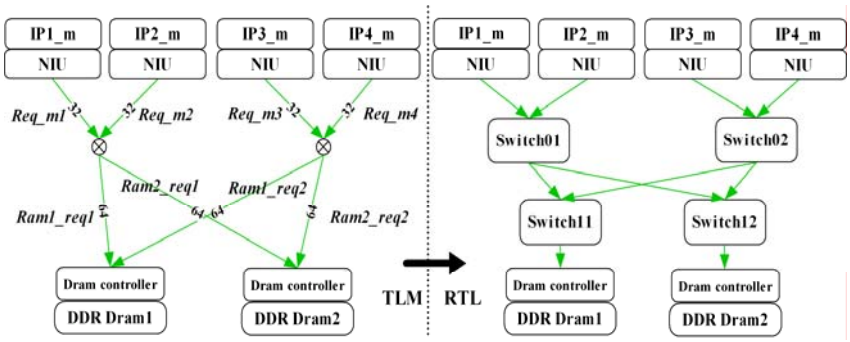


Figure 4.4 Example of NoC modeling with NoCexplorer and NoCcompiler.

After the high speed TLM level SystemC simulation, NoCexplorer will report the system's performance and we can transfer the NoC topology to NoCcompiler, for further RTL level simulation and implementation. Performances including latency and throughput are taken as fitness into our design space exploration. Previously the described design space methodology has been applied to a significant size case study.

4.1.2.1 Area models:

We construct the area models of links and switches using NoCcompiler area estimation at RTL level. Area is computed in unit of NAND2 gate.

The TLM level NoC models hide many implementation details comparing to the corresponding RTL level models to get a high simulation speed. But still we can construct a TLM level area models for area estimation. The RTL component is set to the same configurations of the TLM model, if these options are presented at TLM level. If not, the options of RTL component rest as default. In the study case, the depth of links and the IO number of switches change between different individuals of evaluation exploration. Correspondingly we change the buffer capacity of FIFO component and the IO number of switch to find the relations between area consumption and these variables.

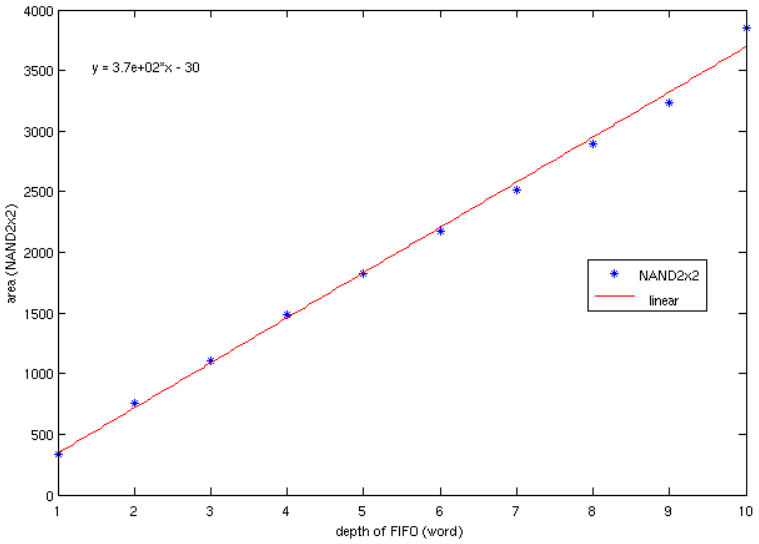


Figure 4.5 FIFO area estimation of the TLM link model with different depth.

According to the area estimation data of NoCcompiler, we can see that the proportion between the FIFO capacity and area consumption is linear, which can be presented as:

$$gates = \begin{cases} 0, depth = 0 \\ 372 * depth - 30, depth >= 1 \end{cases} \quad (1)$$

where the gates means the area of links and the depth is the buffer capacity.

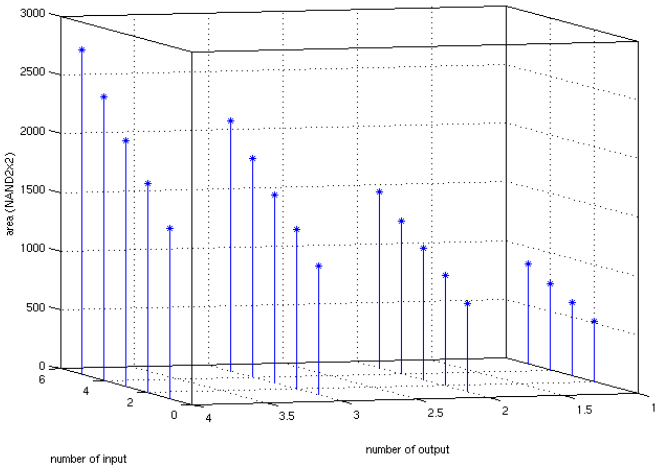


Figure 4.6 Switch area estimation with different number of IO.

The relation between area and the IO number of switch is more complex as showed in Figure 4.6. Switch area is proportional linearly to the number of input (or output), when the number of output (or input) is fixed. With numerical analysis of data, the relation between area of switch and the IO number is:

$$Gates = 72*X*Y+273*Y+39*X+18 \quad (2)$$

where the gates means the area of switch (merging point at TLM level), X the number of input and Y the number of output.

4.1.2.2 Different Traffic Model

4.1.2.2.1 RTLL traffic model:

We model processors are treated as Real-Time Low Latency (RTLL) traffic generator. They are representative of CPU traffics. Sometimes it is acceptable to have brief intervals of longer latency as long as the average latency is low. Cache miss is a typical example of RTLL service. Delays incurred in retrieving data from DRAM can significantly degrade software performance as CPUs stall until data become available. Assume a copy-back cache is employed. Misses to this kind of cache can generate three types of DRAM transactions:

- read miss: The existing content of the target cache line is identical to the data in DRAM. Therefore it can be safely overwritten.
- read with eviction: The target cache line cannot simply be overwritten since its content is different from the corresponding locations in DRAM. The existing data in the cache must be

written to DRAM to avoid loss. To minimize the CPU's wait for the data, the actual order of operation is to read the cache line first and then write the old cache line to DRAM.

- write miss: like read with eviction, the target cache line is inconsistent with data in DRAM. However, there is no need to write the line to DRAM. The target cache line must first be read from DRAM then the cache merges the new CPU write data with this existing cache line content.

The following process implements the copy-back cache traffic flows.

```
Process('CPU_flow',procedure=( Choose ( {
Load(64, ExplicitDequeue)                + Dequeue(64) : 7 # Read misee
Load(64, ExplicitDequeue) + Store(64) + Dequeue(64) : 2 # Read with evictice
Load(64, ExplicitDequeue)                + Dequeue(64) : 1 # Read misee
)} (Dram_addr_Ran ) / 75.0 **MBps ^ Queue('GT1_Q', depth=128, initiator= GT1_m, urgencyThresholds =[64,96])) )
```

The Choose operator is used to alternate between types of cache transactions. In the preceding script, read miss is chosen with a frequency of 70 percent, read miss with eviction at 20 percent and write miss at 10 percent.

The cache traffic is directed to random DRAM address (Dram_Addr_Ran address generator). The transaction are randomly spaced but with a mean spacing of 75 MBps average bandwidth. The depth of the initiator FIFO is one transaction: 64 bytes. Effectively, the FIFO acts like the cache register. Whenever the FIFO is full, the CPU will stall. If the CPU sends another cache miss transaction before the previous one completes, the full FIFO blocks the CPU and performance drops.

4.1.2.2.2 GT traffic model:

Guaranteed Throughput (GT) model must maintain its throughput over a relatively long time. A 3-level dynamic pressure scheme is implemented.

```
Process('GT1_flow',procedure=(Load(32) (Dram_addr_Ran ) / 50.0 **MBps >> Queue('GT1_Q', depth=128, initiator= GT1_m, urgencyThresholds =[64,96])) )
```

In the process, 2 urgency thresholds are entered. Thus, if initiator GT1_m issues up to 2 unacknowledged transactions at 75 MB/s, then the lowest pressure is used. If the NoC and/or Dram are busy, a 3rd transaction might be issued before the first two are acknowledged. In

this case it is assigned intermediate pressure which will help it to compete with other traffic and attain higher throughput, thus draining the fifo. If at some point a 4th unacknowledged transaction fills the fifo beyond 96 bytes, this transaction will be assigned the highest pressure, enabling it to win extra bandwidth even at the expense of the CPU flow.

4.1.2.2.3 BE traffic model:

Best Effort (BE) does not require guaranteed latency or throughput but to which the principle of fairness in treatment is applied. The BE flows always have the lowest pressure. This is indicated in the urgency Threshold by assigning a NULL threshold.

```
Process('BE1_flow', procedure=(Store(32)(Dram_Addr_Ran) / 70.0**MBps >> Queue('BE1_Q',depth = 128, initiator= BE1_m,urgencyThresholds=[])))
```

Whenever a BE flow competes at a switch with a higher pressure flow, it is blocked until that flow clears. If it competes with other BE flows, it is treated fairly and wins arbitration alternately with them.

4.1.2.3 DDR2 model:

The data bus width of DDR2 is 32bits. We have assigned the 11 MSBits to the page address and 2bits low as bank select bits. Because low bits of random address change more frequently than higher bits, the probability that consecutive transactions target different banks is maximized. Each DDR memory has a controller with 3 schedulers: bank optimization scheduler, priority scheduler and read-write turn scheduler. For most addressing patterns, bank optimization will minimize performance loss caused by the bank miss big penalty. When the arbiter can not avoid bank penalty, it chooses the pending transaction with highest priority. Finally, if it cannot avoid bank penalty and there are no high priority pending transaction, the arbiter will try to select a like transaction (load-load or store-store) to avoid a read-write turn penalty. According to our hardware platform technique documents, the bank miss delay is set as 10 cycles and the write-read turnaround delay is set as 5 cycles.

Both the traffic generator and DDR2 memory are specified as OCP sockets which respect the protocol.

4.2 NoC Multi-objective Optimization: NSGA-II

4.2.1 Multi-objective modeling formulation

In order to evaluate various NoC configurations we propose a TLM multi-objective based design space exploration.

The multi-objective optimization problem is the problem of simultaneously minimizing the *n* components (e.g. area, number of execution cycles, energy consumption), f_k , k = 1, …, n of a possibly non linear function f of a general decision variable x in a universe U where

$$f(x) = (f_1(x), f_2(x), \ldots, f_n(x))$$

The problem has usually no unique optimal solution but a set of non dominated alternative solutions known as the Pareto-optimal set. The solution of a practical problem such as embedded multiprocessor design may be constrained by a number of restrictions imposed on a decision variable. Constraints may express the domain of definition of the objective function or alternatively impose further restrictions on the solution of the problem according to knowledge at a higher level. The constrained optimization problem is that of minimizing a multi-objective function ($f_1,\ldots,f_k$) of some generic decision variable x in a universe U subject to a positive number n-k of conditions involving x and eventually expressed as a functional vector inequality of the type

$$(f_{k+1}(x),\ldots f_n(x)) < (g_{k+1}, \ldots,g_n)$$

where the inequality applies component-wise. It is implicitly assumed that there is at least one point in U which satisfies all constraints although in practice that cannot always be guaranteed.

4.2.2 Multi-objective Evolutionary Algorithm

Multi-objective Evolutionary algorithms (MOEA) are more appropriate to solve optimization problems with concurrent conflicting objectives and are particularly suited for producing Pareto-optimal solutions. The NSGA-II is an MOEA considered to outperform other MOEA. The NSGA-II algorithm runs in time $O(GN\log^{M-1}N)$ where G is the number of generations, M is the number of objectives and N is the population size. In addition, our previous experience on multi-objective optimization of multiprocessor [24] emphasizes this choice.

A typical genetic algorithm requires:

- a genetic representation of the solution domain,

- a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits which is called chromosome. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations. Variable length representations may also be used, but crossover implementation is more complex in this case. Tree-like representations are explored in genetic programming and graph-form representations are explored in evolutionary programming. In NSGAII, the encoding of chromosome is binary string.

The crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by these two operators. Am example of two-cut string crossover is showed in fig. 3.

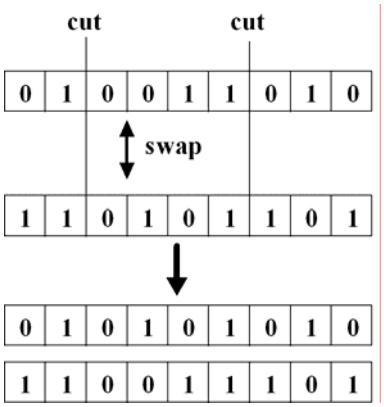


Figure 4.7 two-cut string crossover operation.

Initially, a random parent population P0 is created. Each individual of this population is affected to an adequate Pareto rank. From the population P0, we apply the genetics operators (selection, mutation, and crossover) to generate the population child Q0 of size N. The elitism is ensured by the comparison between the current population P_t and the preceding population P_{t-1}. The NSGA-II procedure follows (see Algorithm 1).

$R_t = P_t U Q_t$ # combine parent and children population
F = fast-nondominated-sort (R_t) # F all nondominated fronts sets
$P_{t+1} = \emptyset$ and $i = 1$ # initialization
until $|P_{t+1}| + |F_i| \leq N$ # till parent pop is filled
  Crowding-distance-assignment (F_i) # compute distance in Fi
  $P_{t+1} = P_{t+1} U F_i$ # include ith nondominated front in the parent pop
  $i = i + 1$ # check the next front for inclusion
Sort $(F_i, <_n)$ # Sort in descending order using $<_n$
$P_{t+1} = P_{t+1} U F[1 : (N - |P_{t+1}|)]$ # Choose the first $(N - |P_{t+1}|)$ elements
Q_{t+1} = make-new-pop (P_{t+1}) # apply genetic operators to create new pop Q_{t+1}
$T = t + 1$ # increment to next generation

Algorithm 1: NSGA-II.

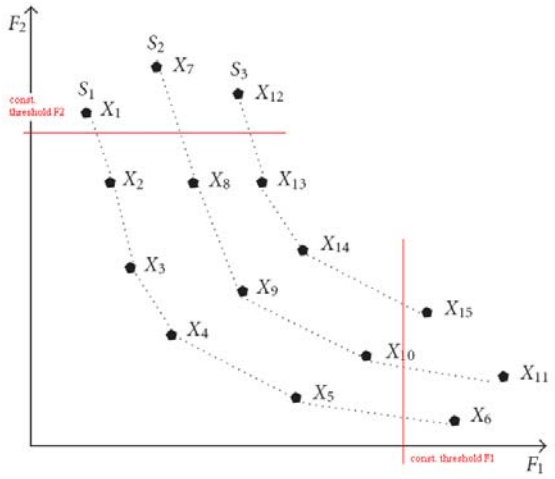


Figure 4.8 Classification of individuals in fronts with constraints threshold.

4.3 NOC Multi-objective Optimization under Constraints

The case study of multi-objective optimization addressed in this chapter is the minimization of whatever individual high level parameters in a network of chip for a base architecture as described in the following figure.

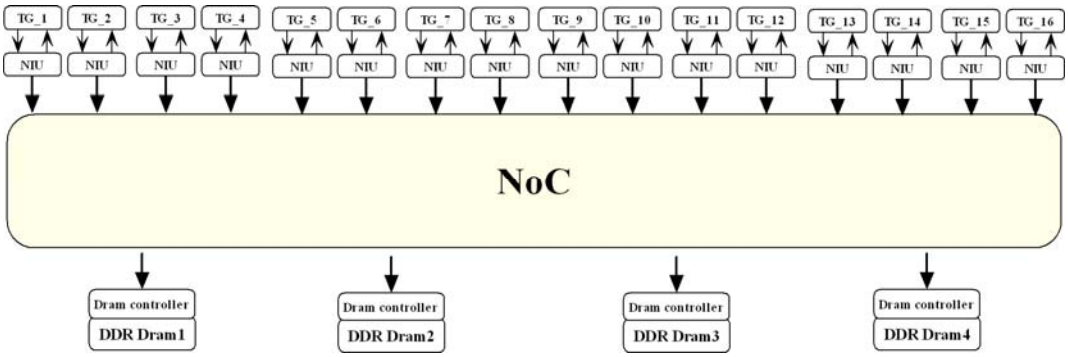


Figure 4.9 Base Architecture

The NOC topology needs to be synthesized at TLM level under constraints. We selected the case of a 16 initiators and 4 targets architecture. The 4 targets are DDR2 DRAM which are fully specified in an FPGA based target board described in Figure 4.10.

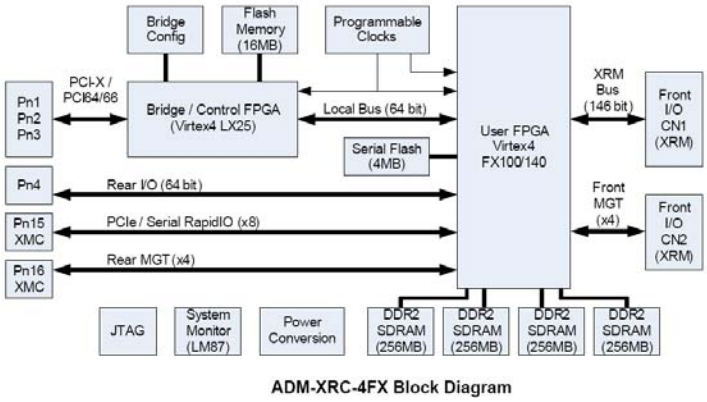


Figure 4.10 Target Board

4.3.1 Design Flow

The design flow takes platform as input and parameters values as constraints.

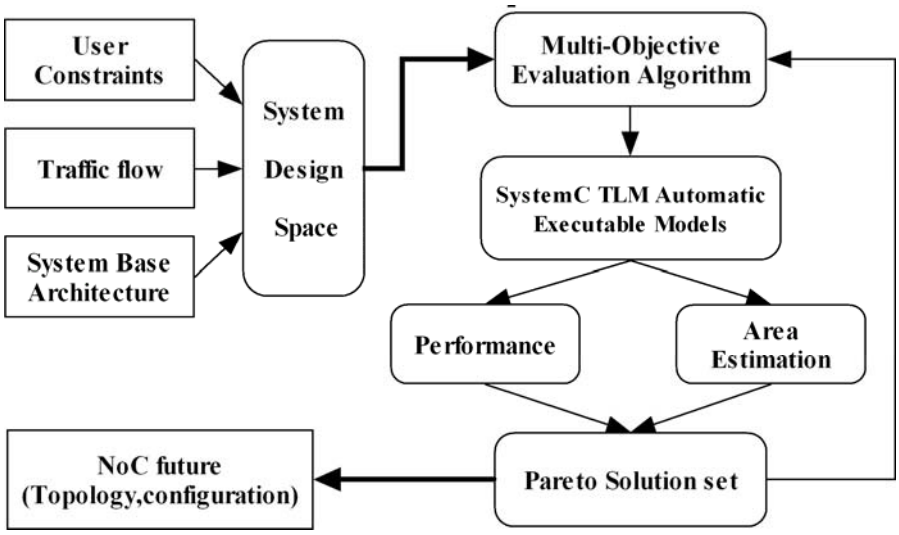


Figure 4.11 TLM Network-on-chip design flow.

A multi-objective optimization technique is applied on this platform by taking into account: (1) user constraints, (2) core graph (3) traffic generators in order to deliver a Pareto set for each case.

4.3.2 Multi-objective NOC TLM DSE

Our proposed flow NOCDEX2 is described below.

NOCDEX 2

```
Generate random NOC configurations population  through modification of scenario
while termination criteria not met
   for all NOC configurations
      simulate at TLM level and  record  all performances parameters,
      Estimate area
      rank the solution
generate new population of NOC
Analyze   final Pareto front
```

This approach provides an automatic exploration of the design space exploration at TLM level which provides at the end a Pareto set of NOC architectures.

NoCDEX 3 design flow

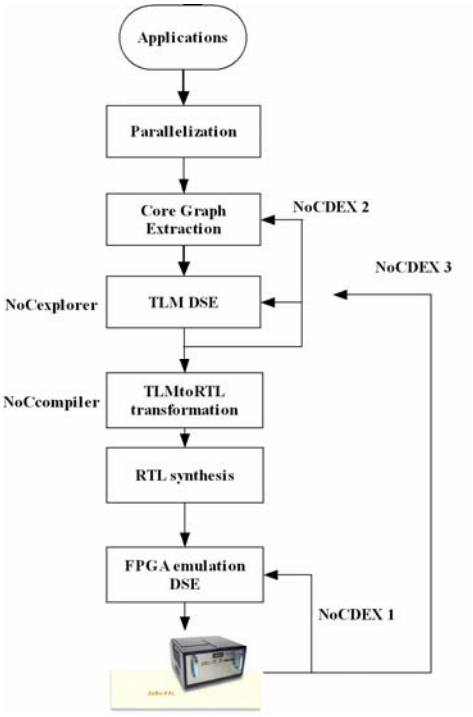


Figure 4.12 NoCDEX 3 design exploration flow

Combining our TLM level exploration NoCDEX 2 and FPGA emulation based exploration NoCDEX, a full level design space exploration flow NoCDEX3 is shown in Figure 4.12. After parallelization, core graph is extracted from real applications. NoCDEX 2 TLM level fast exploration are used to find first Pareto solutions, which are used for FPGA based accurate emulation exploration NoCDEX 1. If the results don't meet the design target, results are feed back to high level simulation for new application parallelization.

4.3.3 Chromosome:

The input of multi-objective genetic evaluation algorithm is the chromosome, which represents the topology and configuration of NoC architecture.

In the study case, a special NoC topology is proposed as showed in the following figure, which is well used in the industry case of Arteris. The NoC is divided into two parts: the request paths side and the response paths side, which can avoid the deadlock problem. The request side of NoC is in the Fig. 7: initiators are at the top and targets at the bottom. The initiators and targets are grouped separately and connected by links to different merging points, which represent switches at RTL level. According to different initiator-target pair transactions, these merging points are connected by links for the transaction routing. The IO

number of switch (merging point at TLM level) changes with different grouping of initiators and targets.

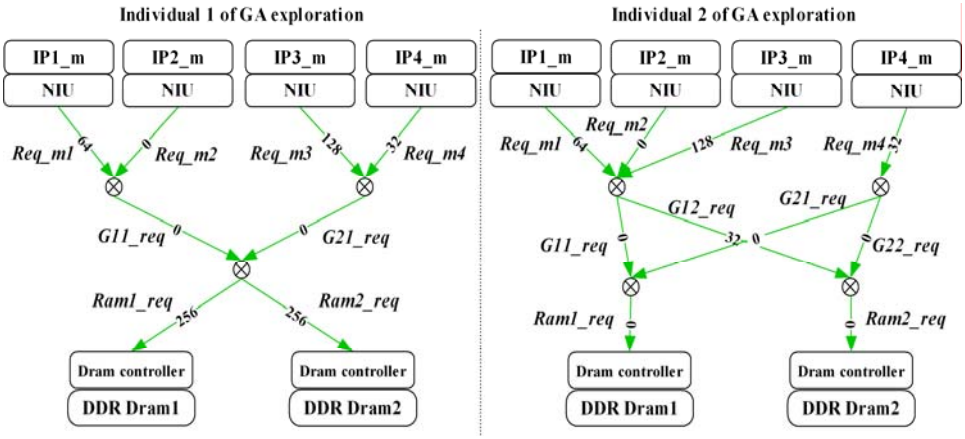


Figure 4.13 Examples of different individuals in GA exploration.

The NoC architecture changes with different grouping of initiators and targets, and with the FIFO capacity of links, which has important impact on the performance and area of NoC.

There are two kinds of genes in the chromosome of NSGAII in this case: they can represent separately the grouping of initiators or targets at the request or response side, or they present the FIFO capacity of each links in the architecture.

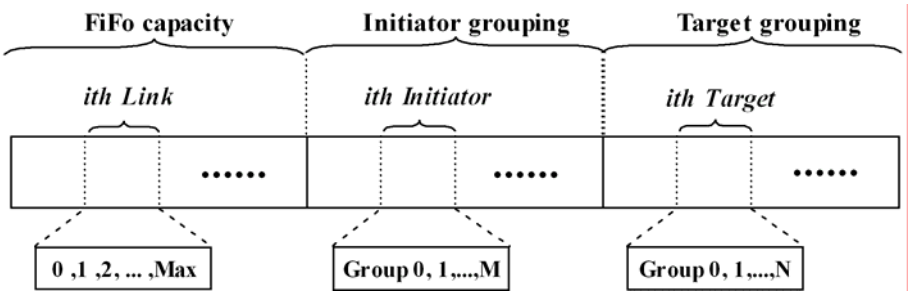


Figure 4.14 Chromosome representation of DES for NSGAII.

The FIFO capacity of each link can change form o to Max. Each initiator connects to one of all M merging points at the top side of NoC architecture, and each target is connected to one of all N merging point at the bottom side. Remember there are request and response path side in our NoC architecture.

ith Initiator ith Target

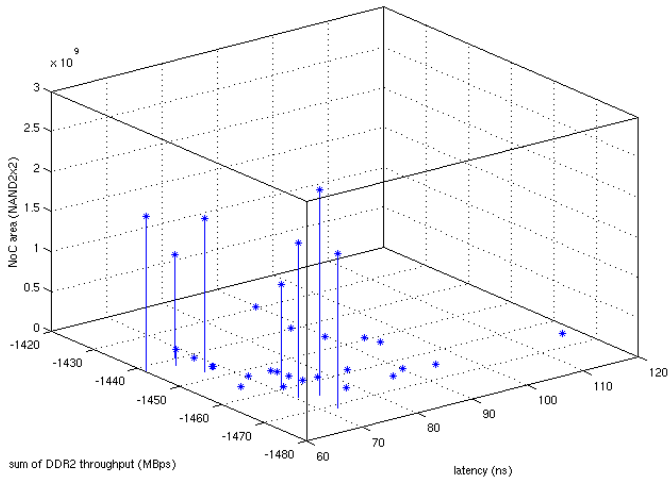
4.4 Performance Evaluation and Comparison

The throughput and average latency of NoC is two performances which can not be satisfied simultaneously. The NoC designers have to find a trade-off between them. We choose 3 different configurations at the front of Pareto curves for further architecture analysis. Depth is one option of 'link' in NoCexplorer exploration represents the depth of FIFO on this link. Here are the configurations of 16 links connected to the 16 processors, which varies in entity of {0, 4, 16, 32}. If the column is invisible in the figure, the value of that link is 0. We can see that for each configuration, the depths of 16 links are different. For each link, the value of depth changes in different configuration solutions.

In Busy is a measurement for link busy status. It represents the percentage of time when the link is at busy state. We can see that the busy state of 16 links connected to 16 processors varies in each configuration solution. For each link, its busy state changes a lot in different configuration solutions.

4.4.1 Best effort exploration

Our NOCDEX2 workflow is generic for any type of NoC topology. Here we use a 16 multi-processor based on NoC as an example to validate our approach. The NOCDEX2 algorithm is set to have 30 individuals and executes for 32 generations based on previous experience. The exploration results are given in the figure below in decimal and logarithm scale.



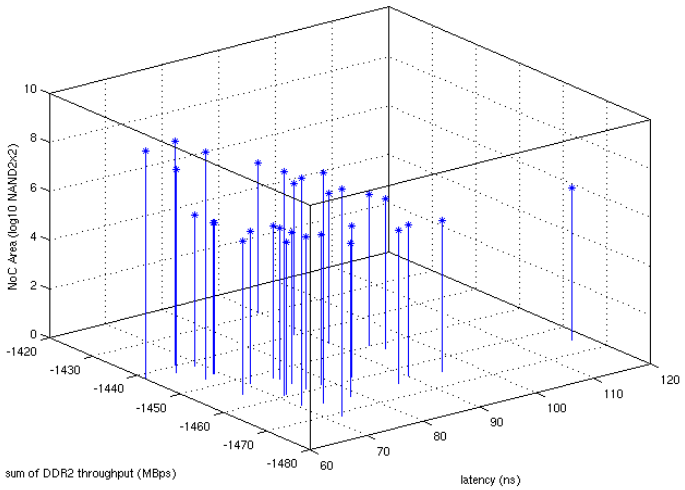


Figure 4.15 Results of best effort exploration

The exploration have stretched NoC configurations ranging from low high latency medium NoC area to low latency and high NoC area with varying DDR2 busyness. The impact of the type of performance metric is important. Many other types of NoC performance parameters could have been used: individual link performance, width, frequency. However, our TLM case study specified a very practical design constraint which was the one imposed by our target board environment that is a limited number of DDR2. This aspect if common for MPSOC since chip pins are scarce and it is difficult to afford even a medium number of DDR2 banks. In our case 4 DDR2 banks are the strong constraint imposed on the design. It is therefore essential that the DDR2 be used to their maximum capacity through a carefully designed network on chip and not wastes this important memory bandwidth resource due to network on chip bottlenecks.

This SOC constraint makes the difference with traditional parallel architectures designs where interconnection networks have access to large number of memory banks. The figures also point the important chip area savings potential when 2 configurations are close in their performance but with significant difference in their area. The designer will naturally select the best NOC area-DDR2 use -latency tradeoff.

4.4.2 Experiment with different constraints

The whole system is set to work at 266 MHz. The data bus width of processors and DDR2 memories is 32bits. Every processor can access the 4 DDR2 memories, each 256MB.

Table 4.1 NOCDEX2 exploration with different constraints

Constraint	Traffic (16 RTLLs and 4 DDRs)
Best-effort	No constraint
Latency constraint	Latency threshold: 24 ns
Latency and area constraint	Latency threshold: 24 ns Area measurement: number of link

The NOCDEX2 can work with different user constraints. In this experiment, we use 3 different configurations:

1) Best effort: without any constraints.

2) Latency constraint: the average memory operation latency threshold is set to 24 ns. If the average memory latency of any of these 16 processors exceeds 24 ns, the current NoC will not be accepted as a solution.

3) Latency and area constraints: with the same average latency threshold as instant (2), we add the number of used links as a fitness of NOCDEX2 algorithm to find out optimal area solution. As at the level of TLM, there is not any area information, we suppose that the area of NoC is proportional to the number of used links. And we take the number of used links as area measurement.

Here we use a Real-Time-Low-Latency (RTLL) model provided by the tool as the traffic generator. They are representative of CPU traffics. Sometimes it is acceptable to have brief intervals of longer latency as long as the average latency is low. Care must be taken to avoid starving other traffic flows as a side effect of pursuing low latency. In this experiment, the RTLL model has a frequency of 80% to read DDR2 memory and 20% time for write operation. The traffic is to random DDR2 addresses in an average bandwidth of 75 MB/s over time.

The data bus width of DDR2 is 32bits. We have assigned the 11 MSBits to the page address and 2bits low as bank select bits. According to our hardware platform technique documents, the bank miss delay is set as 10 cycles and the write-read turnaround delay is set as 5 cycles.

4.4.3 Results of 16 processors with NoC

The results of 16 processors and 4 DDRs are presented in this section with a detailed analysis and comparison of the 3 different configurations.

4.4.3.1 Best effort exploration

The best effort exploration has no constraints on the NoC system performance. So the NOCDEX2 algorithm evolves with its 2 objectives function: (1) average latency and (2) global DDR state. The maximum of the 16 processors' average NoC latency is used as the fitness 1. NoCexplorer gives the analysis of DDR memory status: for example the percentage of time when DDR is at the status of busy. The reciprocal of the sum of 4 DDR busy status percentages is used as the fitness 2 of NOCDEX2. The results of last generation are presented in the Figure 4.16.

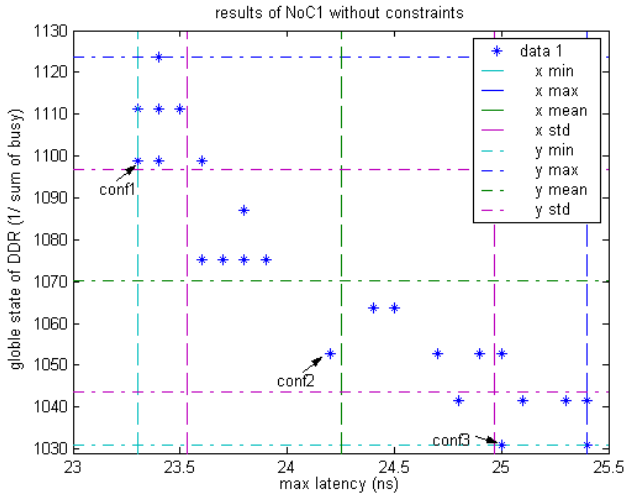


Figure 4.16 Best Effort Exploration Results.

In the Figure 4.16 are the 30 different NoC configurations, which are clearly ranked as a curve of Pareto. The global status of DDR is a measurement of memory busy status, which in consequence is a measurement of NoC throughput.

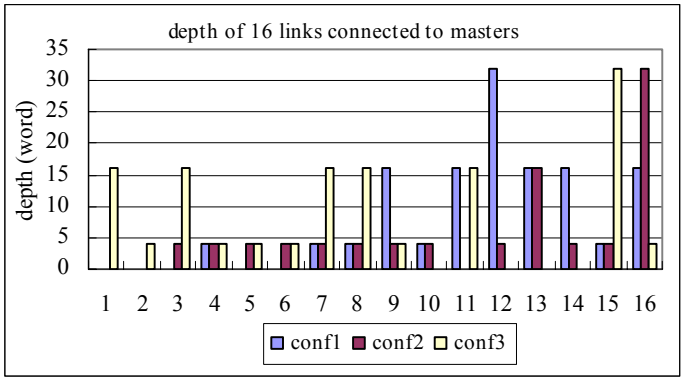


Figure 4.17 Depth of Master links in Best Effort Exploration.

The throughput and average latency of NoC is two performances which can not be satisfied simultaneously. The NoC designers have to find a trade-off between them. We choose 3 different configurations at the front of Pareto curves for further architecture analysis. Depth is one option of 'link' in NoCexplorer exploration represents the depth of FIFO on this link. Here are the configurations of 16 links connected to the 16 processors, which varies in entity of {0, 4, 16, 32}. If the column is invisible in the figure, the value of that link is 0. We can see that for each configuration, the depths of 16 links are different. For each link, the value of depth changes in different configuration solutions.

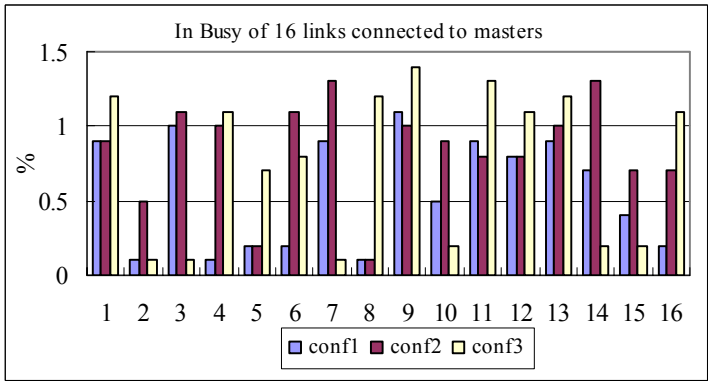


Figure 4.18 Busy Status of Master Links in Best Effort Exploration.

In Busy is a measurement for link busy status. It represents the percentage of time when the link is at busy state. We can see that the busy state of 16 links connected to 16 processors varies in each configuration solution. For each link, its busy state changes a lot in different configuration solutions.

4.4.3.2 Latency constraint exploration

Latency constraint is added to NoCDEX2 exploration to get the solutions under the latency threshold of 24 ns. We take the same 2 fitness as best effort exploration as NOCDEX2's fitness, but those solutions which exceed the 24 ns latency threshold will be punished by multiplying their NOCDEX2 fitness with 1000. in this way NoCDEX2 will find all the solutions under the threshold finally. Here are the results of last generation in Figure 4.19.

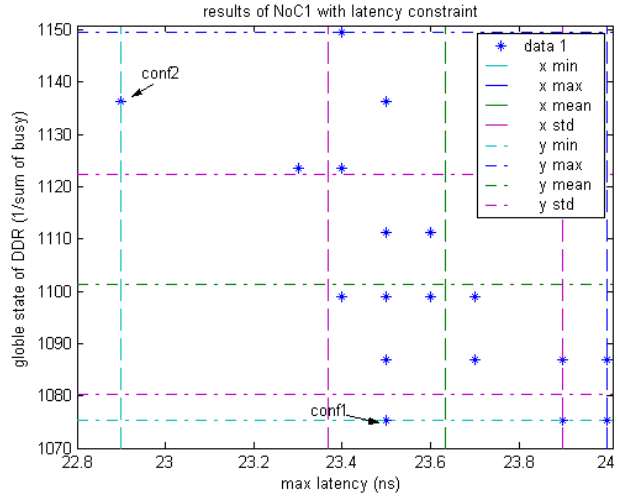


Figure 4.19 Latency Constraint Exploration Results.

As we have added a threshold in NSGAII algorithm, NoCDEX2 give all the configuration solutions which are all under the 24 ns threshold in the last generations. Comparing to the best effort exploration, we can see form the minimum value of average latency and global DDR status that NoCDEX2 has found the solutions which have less average latency in average latency constraint exploration, for example the point marked as conf2 in the figure. But we have to note that in latency constraint exploration, the throughput of NoC is sacrificed: we can not find the solutions which have as large throughput as in best effort exploration.

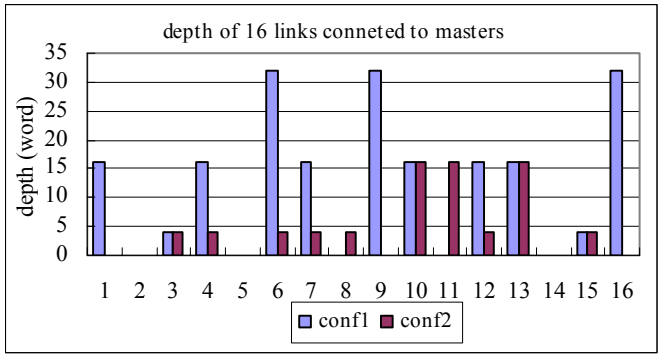


Figure 4.20 Depth of Master Links in Latency Constraint Exploration.

Of the results, we use two configurations to analyze the NoC architecture. Comparing the best average latency solutions: conf2 in latency exploration and the conf1 in best effort exploration, conf2 use less FIFO than conf1. in this way, conf2 has less latency than conf1, but the throughput is not good as conf1.

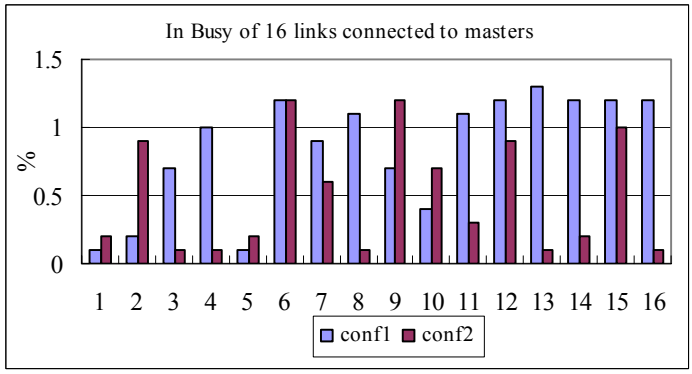


Figure 4.21 Busy Status of Master Links in Latency Constraint Exploration.

At the side of link busy state, we ca see that they fluctuate a lot as in the best effort exploration.

4.4.3.3 Latency and area constraints exploration

Based on the latency exploration, the area measurement is added to NSGAII algorithm as fitness. At the level of TLM, we do not have any area information, so the number of links is taken as the area measurement. The results of last NOCDEX2 generation are showed in the Figure 4.22. The trade-off between NoC latency and area is presented in this figure. If there is more links used in the NoC, there will be less communication conflicts, so the average latency will be smaller.

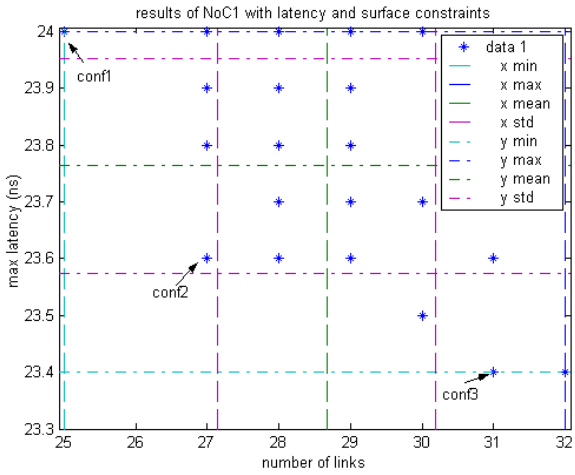


Figure 4.22 Latency and Area Constraint Exploration Results.

It is a good explication to the results in the figure. The latency and area constraint exploration is the first step to evaluate the trade-off between NoC performance and area. It can give a guideline for the further RTL level exploration.

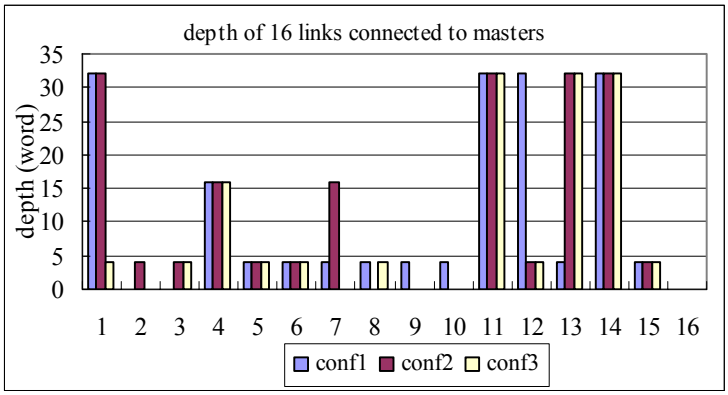


Figure 4.23 Depth of Master Links in Latency and Area Constraint Exploration.

As we add the number of links as fitness of NSGAII algorithm, the depth of links in different configuration solutions does not fluctuate as much as it does the other two explorations, which is really an interesting phenomenon.

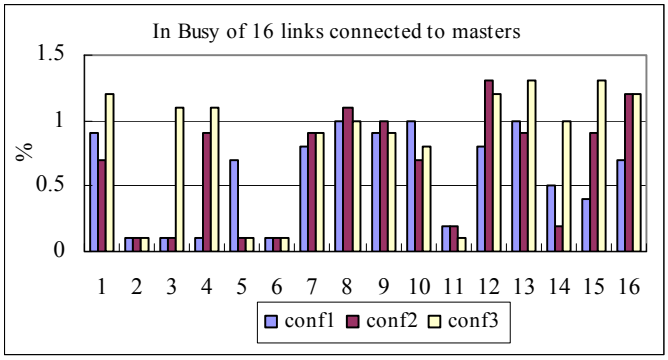


Figure 4.24 Busy status of master links in latency and area constraint exploration.

Since the links depth of different solutions does not change a lot, it is logic to find out that the busy state of the links does not change too much consequently.

4.4.4 Architecture of exploration results

The 3 NoC architectures of 3 different explorations can be found herewith: solution conf1 of no constraint exploration in Figure 4.25; solution conf2 of latency exploration in Figure 4.26 and solution conf1 of latency and area exploration in Figure 4.27.

At the top of the figures, 16 traffic generators (TG) and their NIU (Network Interface Unit) are presented as green rectangles. The 4 DDR memories and Dram controllers are found at the bottom. The NoC architecture has two levels of links. The first level of 16 links is connected to each traffic generator's NIU separately as black lines. The figures written in the middle of lines are the FIFO depths of each link, same for the red lines, which represented the links of second level. The merging circle indicates there will be a competition of link if the different communications conflict there. In our experiment, the second level of links is optimal: the traffic can get to the DDR memory through only the first level of link without through the second one. The dashed green lines designate the routing of communications from generators to memories, and do not exist during the NoCexplorer TLM simulation. All the communications through only the first level of links are not showed in the pictures to get a legible image: they go directly from the black links to the Dram controllers.

The NoC architecture is not quite equivalent for each traffic generator. It is a further work to find a good scenario of the optimization technique to balance the architecture for each generator.

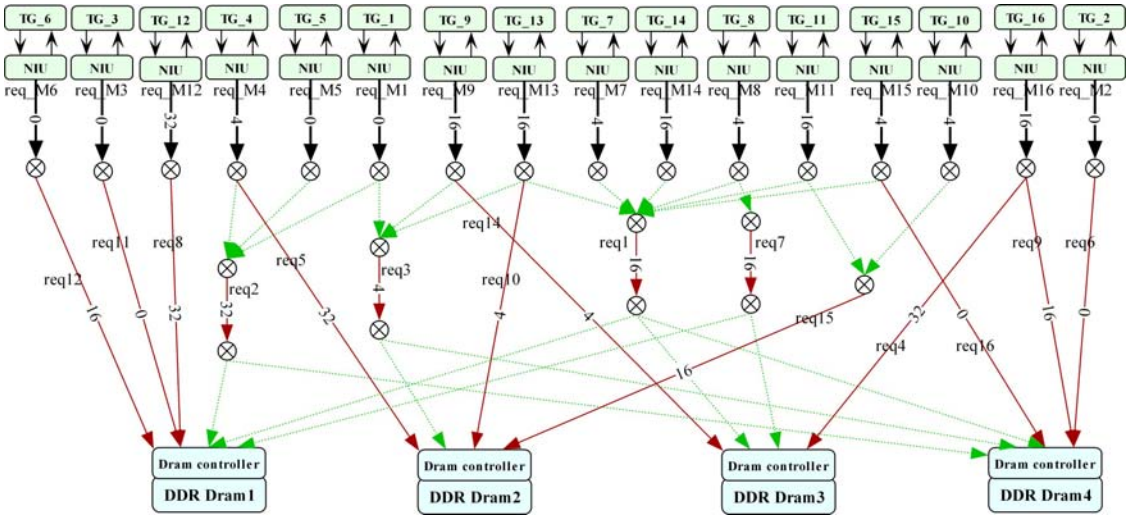


Figure 4.25 TLM level NoC architecture of no constraint exploration's conf1 solution.

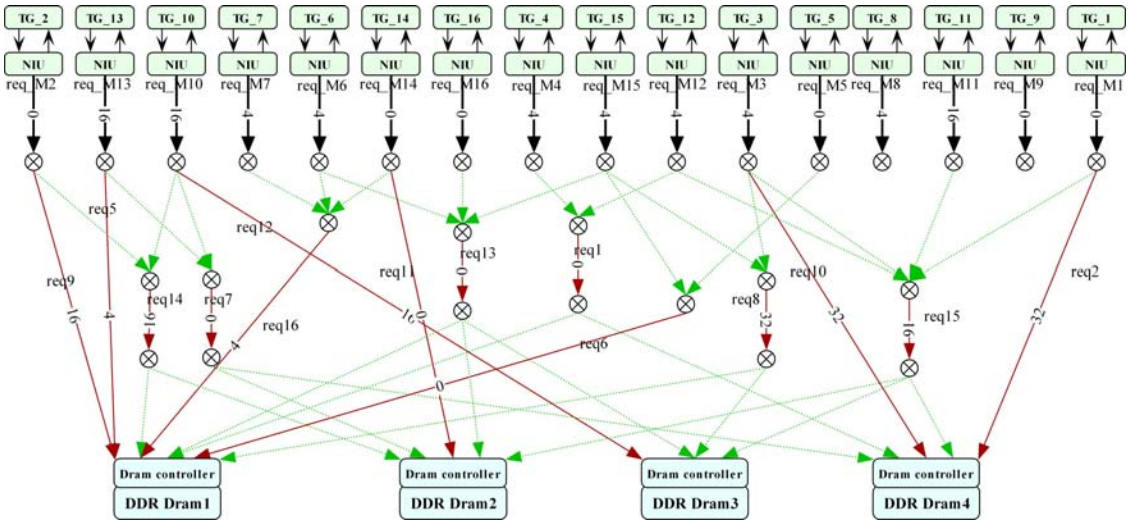


Figure 4.26 TLM level NoC architecture of latency constraint exploration's conf2 solution.

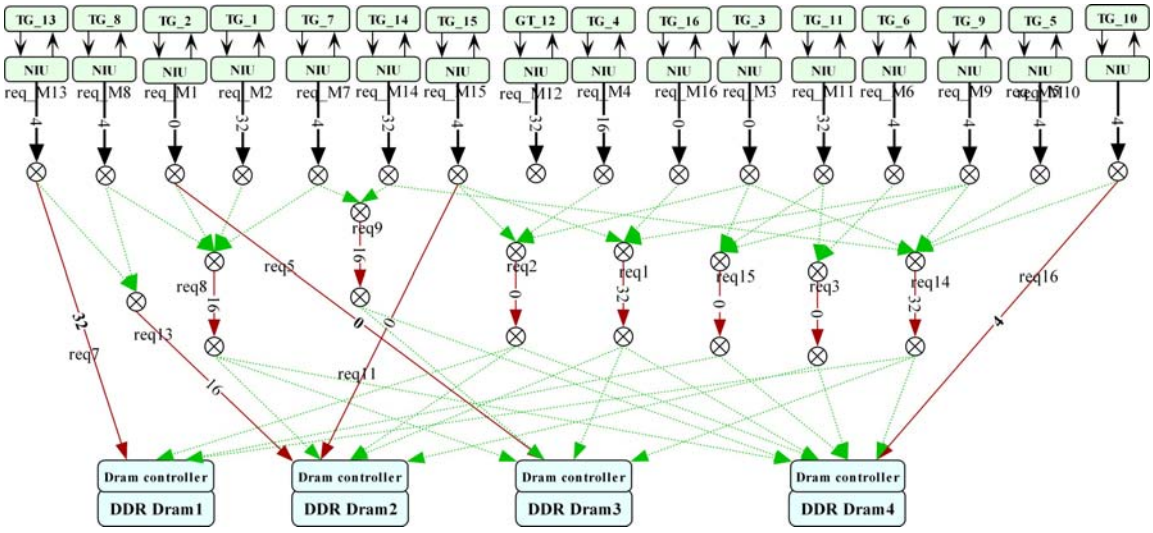


Figure 4.27 TLM level NoC architecture of latency and area constraint exploration's conf3 solution.

4.5 Conclusion

Design space exploration of network-on-chip can be conducted at multiple levels of abstraction from transaction level modeling down to emulation. A fully automatic multi-objective design workflow is proposed for network on chip at TLM level. The timing and area criteria are explored but not limited using the TLM NoC models of NoCexplorer. Further work is to build the energy consumption model library, which can be integrated into our workflow for power-area- performance multi-criteria system on chip design exploration. Combining this flow with our previous work at RTL level will allow a fully integrated solution.

Reference

[1] Benini, L.; De Micheli, G.; Networks on chips: a new SoC paradigm, Computer, Volume 35, Issue 1, Jan. 2002 Page(s):70 – 78

[2] Tobias Bjerregaard, Shankar Mahadevan, A survey of research and practices of Network-on-chip, ACM Computing Surveys (CSUR), Volume 38 Issue 1, June 2006

[3] Owens, J.D.; Dally, W.J.; Ho, R.; Jayasimha, D.N.; Keckler, S.W.; Li-Shiuan Peh; Research Challenges for On-Chip Interconnection Networks, IEEE Micro, Volume 27, Issue 5, Sept.-Oct. 2007 Page(s):96 - 108

[4] Li, X.; Hammami, O.;, NOCDEX: Network on Chip Design Space Exploration Through Direct Execution and Options Selection Through Principal Component Analysis, Industrial Embedded Systems, 2006. IES '06. International Symposium on 18-20 Oct. 2006 Page(s):1 - 4

[5] Partha Pratim Pande; Grecu, C.; Jones, M.; Ivanov, A.; Saleh, R.;, Performance evaluation and design trade-offs for network-on-chip interconnect architectures, Computers, IEEE Transactions on Volume 54, Issue 8, Aug. 2005 Page(s):1025 – 1040
[6] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, Radu Marculescu On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 12 Issue 3, August 2007
[7] Jingcao Hu; Marculescu, R.;, Energy- and performance-aware mapping for regular NoC architectures, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 24, Issue 4, April 2005 Page(s):551 – 562
[8] Xinping Zhu, Sharad Malik, A hierarchical modeling framework for on-chip communication architectures of multiprocessing SoCs, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 12 Issue 1 , January 2007
[9] Lahiri, K.; Raghunathan, A.; Dey, S.;, Design space exploration for optimizing on-chip communication architectures, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 23, Issue 6, June 2004 Page(s):952 – 961
[10] Bertozzi, D.; Jalabert, A.; Srinivasan Murali; Tamhankar, R.; Stergiou, S.; Benini, L.; De Micheli, G.; NoC synthesis flow for customized domain specific multiprocessor systems-on-chip, Parallel and Distributed Systems, IEEE Transactions on Volume 16, Issue 2, Feb 2005 Page(s):113 – 129
[11] Jiang Xu, Wayne Wolf, Joerg Henkel, Srimat Chakradhar, A design methodology for application-specific networks-on-chip, ACM Transactions on Embedded Computing Systems (TECS), Volume 5 Issue 2, May 2006
[12] Murali, S.; Benini, L.; De Micheli, G.;, An Application-Specific Design Methodology for On-Chip Crossbar Generation, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 26, Issue 7, July 2007 Page(s):1283 – 1296
[13] Brandolese, C.; Fornaciari, W.; Pomante, L.; Salice, F.; Sciuto, D.; Affinity-driven system design exploration for heterogeneous multiprocessor SoC, Computers, IEEE Transactions on Volume 55, Issue 5, May 2006 Page(s):508 - 519
[14] Murali, S.; Atienza, D.; Meloni, P.; Carta, S.; Benini, L.; De Micheli, G.; Raffo, L.; Synthesis of Predictable Networks-on-Chip-Based Interconnect Architectures for Chip Multiprocessors, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Volume 15, Issue 8, Aug. 2007 Page(s):869 – 880
[15] Angiolini, F.; Meloni, P.; Carta, S.M.; Raffo, L.; Benini, L.;, A Layout-Aware Analysis of Networks-on-Chip and Traditional Interconnects for MPSoCs, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 26, Issue 3, March 2007 Page(s):421 – 434
[16] T.Kogel, R.Laupers and H.Meyr, "Integrated System-Level Modeling of Network-on-chip enabled Multi-Processor Platforms", Springer 2006.
[17] F.Ghenassia, "Transaction-level Modeling with SystemC - TLM Concepts and Applications for Embedded Systems", Springer 2005.
[18] W.Muller, W.Rosenstiel nd J.Ruf, " SystemC Methodologies nd pplictions", Kluwer Academic Publishers 2003.
[19] OSCI TLM 2 User Manual Doc. Version 1.0.0. Nov. 2007
[20] Arteris S.A. http://www.arteris.com
[21] NoC Solution 1.9, NoC Compiler user's Guide, o918v5, Sep. 2007, Arteris
[22] NoC Solution 1.9, NoCexplorer user's Guide, o3088v5, Sep. 2007, Arteris
[23] Danube 1.9, Packet Transport Units Technical reference, o4277v9, Oct. 2007, Arteris
[24] R. Ben Mouhoub and O. Hammami, "MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution," EURASIP Journal on Embedded Systems, vol. 2006, Article ID 54074, 14 pages, 2006.

5. Reconfigurable NoC on eFPGA

Multiprocessor system on chip (MPSoC) is expected to be used for multiple applications which might exhibit distinct communication patterns. As number of IPs is increasing exponentially, the most important issue in communication is to guarantee the quality of service. Network on chip provides a proven solution for the communication of complex System on Chip and several designs have been made. However, few studies have addressed the requirements of network on chip design for multiple applications. Finding a common efficient network on chip for these multiple applications might be simply impossible due to the diverging requirements.

Reconfigurable network on chip is a potential solution, in which the network is reconfigured before the execution of applications in order to match the applications' specific requirements. Implementation of this reconfigurability might be done using eFPGA. We propose a methodology to specify the area dimension of reconfigurable eFPGA for NoC (Network on Chip). Various objective functions are used to drive our study. The experiment results show the efficiency of our approach.

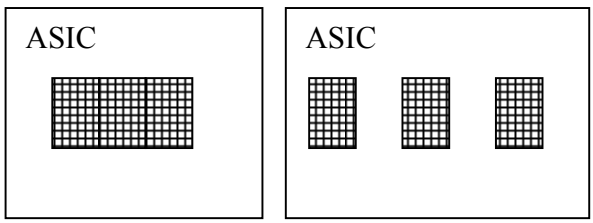


Figure 5.1 ASIC with eFPGA (a) centralized (2) scattered

Although this is possible to be done on FPGA devices by exploiting the reconfigurability of FPGA low power, high performance and area efficiency requirements of SoC push for an ASIC solution with eFPGA. This eFPGA have been the focus of various studies (e.g. [5]) and

represents the only solution so far to introduce reconfigurability in non-FPGA environment that is ASIC.

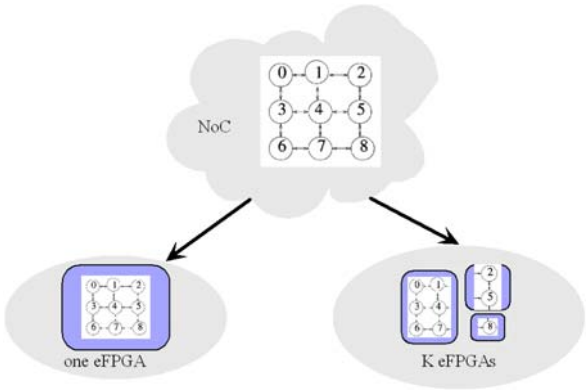


Figure 5.2 Placement of NoC onto eFPGAs

In order to efficiently use this provided resource it is essential to characterize the various designs to be mapped on it. Proper selection among area-performance allows such a choice.

5.1 eFPGA and Reconfigurable NoC State of Art

5.1.1 eFPGA technology

The ability to make post fabrication changes in System-on-Chip (SoC) designs is the major advantage of eFPGA cores. Its reconfigurability makes eFPGA suitable for on-chip logics such as hardware accelerators for processors to speed up embedded applications, data encryption units in wireless devices that need to be changed from time to time, I/O interfaces for data communication, configuration and packet routing of Network-on-Chip to adapt dynamic traffic. The advantages of this approach make possible to supply multiple customers with a single programmable chip and accommodate changes in standards or design specifications.

There are two French companies accelerate the embedded FPGA technology implementation on chips. They are Menta [14] and M2000 (change to Abound) [15]. They develop eFPGA cores in the form of soft IP. Chip designers would use their eFPGA core and integrate it on their design to get the live capability of reconfiguration..

Menta licenses world's first pure soft FPGA IP core [14]. As a soft IP, it is synthesized with the standard HDL design flow, makes very easy its integration onto a SoC. The Menta's

soft core eFPGA is technology independent. It can be integrated with any process technology, which gives a lot of easiness for SoC manufacturing.

In addition of being soft core, Menta's eFPGA [14] is highly customizable so a domain specific-FPGA (dsFPGA) [16] can be used in SoC which provides extensive features for the target applications giving enormous benefits in terms of Area, Power consumption and Speed. Current, eFPGA core-II architecture for general purpose logic mapping without using any hard macros can achieve a density of around 5,000 equivalent gates per mm2 for 90 nm CMOS process. Menta forecast for next generation having more than 8,000 equivalent gates per mm2 for 90 nm CMOS process. For critical and high volume users, target technology achieves more than 15,000 equivalent gates per mm2 (estimates on 90 nm) while still keeping much of the flexibility of soft IP core.

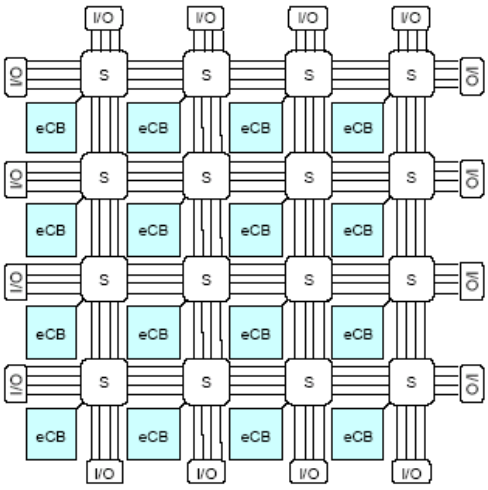


Figure 5.3 Abstract view of Menta eFPGA architecture

As it is soft IP there are no SRAM cells on Menta eFPGA [16], Pass transistor or tri state buffer switches. The configuration element is a Flip Flop and switching element is a Multiplexer. The core is highly configurable. We can select all the fundamental parameters of the eFPGA like LUT size, Cluster size (number of LUT in a eCB), Routing channel size and array size (number of eCBs) etc.

M2000 is another French startup working of embedded FPGA technology, which has become as Abound Logic. They claim to have the largest FPGA on earth. The technology details are given in the next section. Comparing to Menta, M2000 has more commercial success. They are involved in several European research projects such as MORPHEUS

project [23], which targets at a modular heterogeneous SOC platform. Hardware accelerators are implemented onto M2000 eFPGA which are connected by STNoC. STMicroelectronics uses their technology for several ASAP+eFPGA chips. For example, Xtensa processor connects an eFPGA [18] onto which hardware accelerators are implemented. And in [19], three eFPGAs are connected by an eight port NoC with the main ARM processor.

Stretch S6000 family of software configurable processors [20] combines Tensilica's Xtensa® LX Processor together with their own proprietary programmable fabric, which they call Stretch Instruction Set Extension Fabric (ISEF). ISEF is used to optimize the instruction set for specific applications in real time. This is probably the first commercial CPU with embedded FPGA. Both the S6100 and the S6105 contain an integrated Processor Array (PA) interface to connect multiple Stretch processors into hyper-cubes.

Selected C-functions are converted by Stretch compiler into programmable logics automatically inside ISEF and these new instructions execute in a single cycle on ISEF. The FPGA-like ISEF logic is designed specifically for implementing variable-sized ALUs, multipliers, and shifters — all data path extensions to the processor. The ISEF computes complex functions "in parallel" but tailored by system designers to meet the needs of their own compute-intensive applications.

Warp processor [21] consists of a main processor ARM7, an efficient on-chip profiler, warp oriented FPGA (W-FPGA), and an on-chip computer-aided design module (OCM). Another ASAP+eFPGA solution proposed by RWTH Aachen University [22], uses MIPS-IV instruction set and an ARM940T separately as main processor, and their arithmetic oriented eFPGA as coprocessor. LEON3 +Menta eFPGA [16] proposes to use the eFPGA logics as coprocessor or integrate the eFPGA logic as pipeline into LEON3 main processor as Stretch solution. Performances are obtained by simulation to show the efficacy of reconfigurable eFPGA solution.

5.1.2 Reconfigurable NoC implementation with FPGA

A heterogeneous reconfigurable hardware platform composed of two ASIC and two FPGA for Cognitive Radio is used for reconfigurable SDR system emulation [24]. NoC FAUST of wormhole switching mode is extended to a FPGA and Functional units are connected to the network through a network interface (NI). The amount of credit and data

inside each NI is periodically reconfigured using Xilinx's PR (Partial Reconfiguration) technology.

Dynamic Reconfigurable NoC (DRNoC) Emulation Platform [25] is an FPGA emulation-based fast Network on Chip (NoC) prototyping framework. Again, partial reconfiguration technology is applied to avoid whole system re-synthesis during the design exploration for best NoC configuration. Comparing to previous partial reconfiguration works [26, 27] for dynamical insertion and removal of routers and cores on a MESH based NoC, DRNoC is a complete fast NoC emulation framework based on different types of partial reconfiguration. After all configurations have been setup, each DRNoC configuration is emulated FPGA platform. All configurations are emulated consecutively, and the results related to each hardware measuring point are saved. The workflow is not automatic.

An overview of existing reconfigurable NoC work is given is [29]. A description of a dynamic reconfiguration NoC model is introduced. Three research issues: dynamic reconfiguration administration, network infrastructure reconfiguration and network protocols reconfiguration are listed. Existing approaches and open Issue are discussed. a controlled tradeoff between quality of service and overhead at the three levels of decision is focused on.

5.2 eFPGA: M2000 Case Study

5.2.1 M2000 eFPGA technology

M2000 was created in 1996 on France by experts on configurable logic. Started with Meta Systems they developed the industry's first emulation system based on custom FPGAs (Field Programmable Gate Arrays.) The founding team holds numerous patents in the field of configurable logic and its applications for electronic system testing. M2000's current focus is the design and development of state of the art configurable logic technology for the rapidly growing reconfigurable SoC (System on a Chip) market. They have changed the name as Abound and moved the headquarter to California USA. FlexEOS is the last technique supplied by M2000, and the new FPGA technology is called as Raptor™ FPGA.

M2000's 90nm FlexEOS embedded FPGA macros in 90 nanometer CMOS technology breaks the density barrier of 1,000 reprogrammable Look-Up Tables (LUTs) per mm2 in 2005, with an intrinsic technology performance capable of 2.7 GHz. The FlexEOS [15] range of embedded FPGA cores are SRAM based, and can be dynamically reconfigured to change

the functionality of ASIC and SoC circuits after silicon processing and packaging. FlexEOS macros are suitable for a wide range of applications, and can be supplied for any silicon foundry technology. Each macro is delivered with a comprehensive software tool suite for the compilation of the applications which are to run on the macro. Mentor Graphics is the supplier of the front-end synthesis technology for Abound Logic's products.

Based on a dense, hierarchical routing structure, Raptor™ FPGAs [15] deliver an unprecedented level of density, far beyond that of any other FPGA. Providing more than 750,000 LUTs and an equal number of flip-flops to the designer, Raptor FPGAs are an attractive alternative to ASICs in many applications.

At the heart of the Raptor FPGA is the multifunction cell (MFC) composed of a 4-input LUT plus a D-type flip-flop, offering invertible clock polarity, programmable enable and asynchronous/synchronous reset.

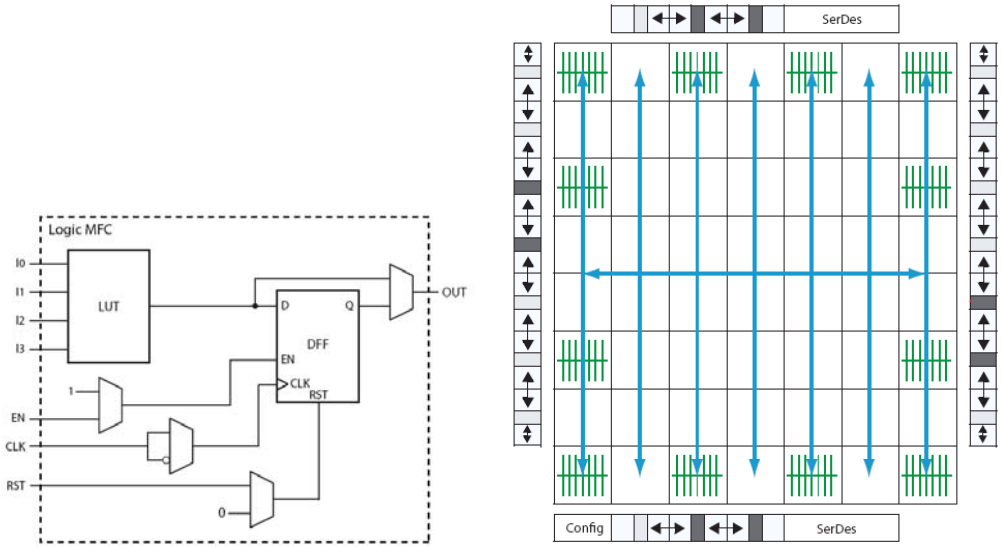


Figure 5.4 Abound Multi-function cell and layout of Raptor FPGA

The Raptor architecture includes three types of MFCs to provide access to other resources: logic, memory and arithmetic. Logic MFCs contain a single LUT and D-type flip-flop; memory MFCs add access to an embedded 32 × 18 register file; arithmetic MFCs provide access to 4-bit adders. More information can be found in [17].

5.2.2 Switch Area and frequency characterization

We use the Danube IP Library of company Arteris to generate the NoC. The switch component architecture can be customized through several options described in the following table.

Table 5.1 Switch Options

Switch options	values
1.Arbitration type	Round-Robin(1), LRU(2), Random(3), Fifo(4)
2.Input pipeline register	True(1), false(0)
3.Forwards pipeline register	True(1), false(0)
4.Backwards pipeline register	True(1), false(0)
5.Crossbar pipeline register	True(1), false(0)
6.Dual cycle arbitration	True(1), false(0)

Because of the constraints between different options, there are 80 different configurations for each switch. There is one example of a switch with 4 inputs and 4 outputs synthesized on the eFPGA of M2000 using 80 configurations with 90nm FlexEOS embedded FPGA library.

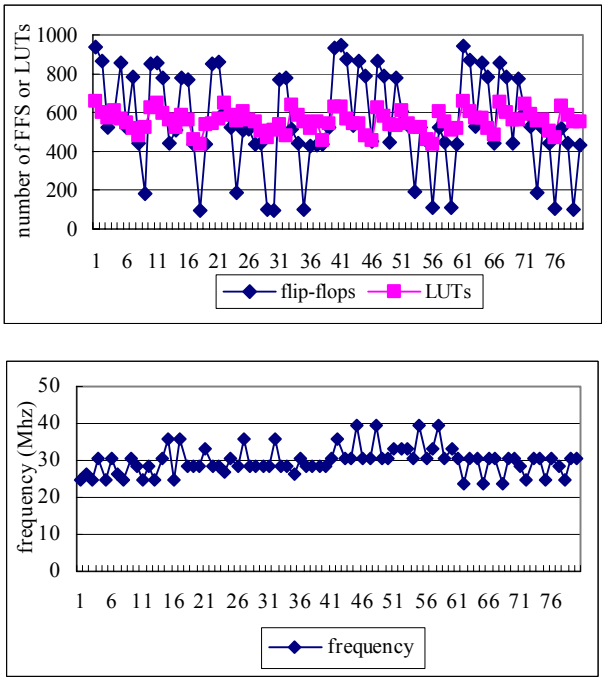


Figure 5.5 Area and Frequency of Switches

The number of flip-flops and LUTs (Look Up Table) are taken as the switch area measurement. With different configurations, the switch area changes a lot. The switch

frequency fluctuates from 25 MHz to 40 MHz. And it should e noticed that the frequency does not change proportionally to the area.

5.2.3 Reconfigurable NoC problem definition

We have the area and frequency library of each switch configuration. The objective is to find out a high performance heterogynous NoC at the constraints of eFPGA area.

Input: (1) a NoC with N switches; (2) the area constraint of one or numerous eFPGAs disposable.

Output: the choice of configuration for each switch and the placement of each switch if more than one eFPGA are used.

Constraints: the total area of switches can not exceed the whole area constraint of eFPGA where these switches are placed.

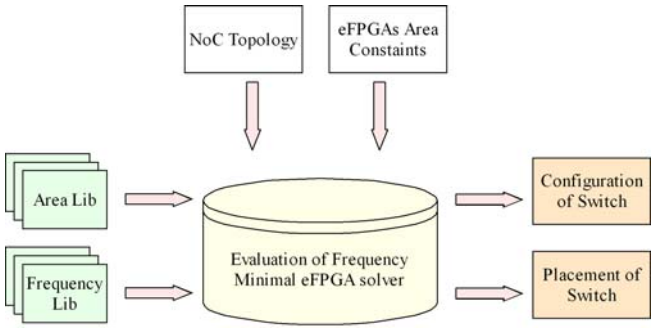


Figure 5.6 Design flow of NoC on eFPGA

5.3 NoC Optimization on eFPGA with Place and Route Constraints

In this section we present our ILP formulation for the reconfigurable NoC design problem. Depending on the number of eFPGAs, we have two different formulations: (1) reconfigurable NoC is placed on just one eFPGA, the configuration of each switch can be chosen with the area constraint; (2) NoC is placed on more than one eFPGAs, not only the configuration should be selected but also the placement of each switch on eFPGA is decided. At the base the formulation (2), we extend to find a solution which uses the minimal number of eFPGAs among all the possible ones. And the workflow is described.

5.3.1 Reconfigurable NoC on one eFPGA

At first situation there is just one eFPGA to place the NoC with N switches of which each has M different possible configurations. This eFPGA has LUT_{max} luts and RAM_{max} rams.

We introduce a binary variable $x_{i,j}$ to represent the choice of configuration of switch i:

$x_{i,j} \in \{0,1\}$ for $i = 1,\ldots,N$ and $j = 1,\ldots,M$.

$x_{i,j} = 1$ if switch i is set to its configuration j, and $x_{i,j} = 0$ otherwise.

In out area library we have all the information of each switch configuration. Let $lut_{i,j}$ and $ram_{i,j}$ be the number of luts and rams of switch i if it's set to configuration j. And $f_{i,j}$ represents the frequency of switch i set to configuration j. The ILP formulation A of this problem is:

$$\text{Max:} \quad \sum_i \sum_j f_{i,j} \cdot x_{i,j} \tag{1}$$

$$\text{s.t.} \quad \sum_j x_{i,j} = 1, \forall i \tag{2}$$

$$\sum_i \sum_j lut_{i,j} \cdot x_{i,j} \le LUT_{\max} \tag{3}$$

$$\sum_i \sum_j ram_{i,j} \cdot x_{i,j} \le RAM_{\max} \tag{4}$$

The objective is to maximize the frequency of all switches as we suppose to construct a heterogeneous NoC with each switch different frequency as in (1). The constraint (2) makes sure that each switch is set to only one of its configurations. In constraint (3), we ensure that the sum of all switches' luts will not exceed the whole LUT_{max} luts of eFPGA. And in constraint (4), we ensure the sum of all switches rams will not exceed the whole RAM_{max} rams of eFPGA.

5.3.2 Reconfigurable NoC on numerous eFPGAs

We consider the situation there is K eFPGAs to place the NoC with N switches. Each eFPGA has LUT_k luts and RAM_k rams where $k = 1,\ldots,K$. The switch is free to be placed in any of these eFPGAs. The configuration of each switch is to be set, but also their placement.

In this case, we introduce a binary variable $x_{i,j,k}$ to represent the choice of configuration and the placement of switch i:

$X_{i,j,k} \in \{0,1\}$ for $i = 1,\ldots,N$; $j = 1,\ldots,M$ and $k = 1,\ldots,K$.

$x_{i,j,k}$ = 1 if switch i is set to its configuration j and is placed on the eFPGA k, and $x_{i,j,k}$ = 0 otherwise.

Let $lut_{i,j,k}$ and $ram_{i,j,k}$ be the number of luts and rams of switch i if it's set to configuration j and placed on eFPGA k. And $f_{i,j,k}$ represents the frequency of switch i set to configuration j and placed on eFPGA k. The ILP formulation B of this problem is:

Max: $$\sum_i \sum_j f_{i,j,k} \cdot x_{i,j,k} \qquad (5)$$

s.t. $$\sum_j \sum_k x_{i,j,k} = 1, \forall i \qquad (6)$$

$$\sum_i \sum_j lut_{i,j,k} \cdot x_{i,j,k} \le LUT_k, \forall k \qquad (7)$$

$$\sum_i \sum_j ram_{i,j,k} \cdot x_{i,j,k} \le RAM_k, \forall k \qquad (8)$$

The objective is still to maximize the frequency of all switches in (1). The constraint (2) makes sure that each switch is set to only one of its configurations and is placed to only one eFPGA. In constraint (3), we ensure that on eFPGA k the sum of all switches' luts will not exceed the whole LUT_k. And in constraint (4), we ensure on eFPGA k the sum of all switches' rams will not exceed the whole RAM_k.

5.3.3 Reconfigurable NoC on minimal eFPGAs

In the case of numerous eFPGAs, not all of them are need to place the NoC. So we want to minimize the number of used eFPGAs. We use the algorithm based on formulation B to solve this problem.

The major idea is to test the feasibility of placing the NoC on form one eFPGA to K-1 eFPGAs. If the objective function value of the n (less than K) eFPGAs solution is equal to the K eFPGAs solution's F_{max} and the other m (less then n) eFPGAs solutions' objective function value is less than F_{max}, then we find a minimal solution for the NoC on numerous eFPGAs problem.

In this algorithm 1, a list of combination of K eFPGAs is built for the i eFPGAs problem. A combination example of 3 eFPGAs (1,2,3) is:

(1); (2); (3); (1,2); (1,3); (2,3); (1,2,3)

Algorithm 1 NoC on Minimal eFPGAs Algorithm

1: calculate ILP value F_{max} of K eFPGAs problem
2: build list of all the combination of K eFPGAs
3: **for** i =1 to K **do**
4: calculate ILP value Fi of i eFPGAs problem
5: **if** $F_i >= F_{max}$ **then**
6: **break**
7: **end if**
8: **end for**
9: output F_i, i , used eFPGAs and switch configurations

5.4 Performance Evaluation and Results

5.4.1 Experimental setup

We generate our area and frequency library by synthesis on the eFPGA using the M2000 techniques and Mentor Graphics Precision Synthesis. A NoC of 16 inputs and 4 outputs is used as test bench for out NoC on minimal eFPGA algorithm. GNU glpk library is used to solve the ILP problems. The function 'lpx_intopt' with branch-and-bound method is effective to solver our IPL formulations in seconds. We obtain the results of NoC on one eFPGA and on numerous eFPGAs. All results were obtained on a 2.6GHz Xeon station with 4GB DDR memories.

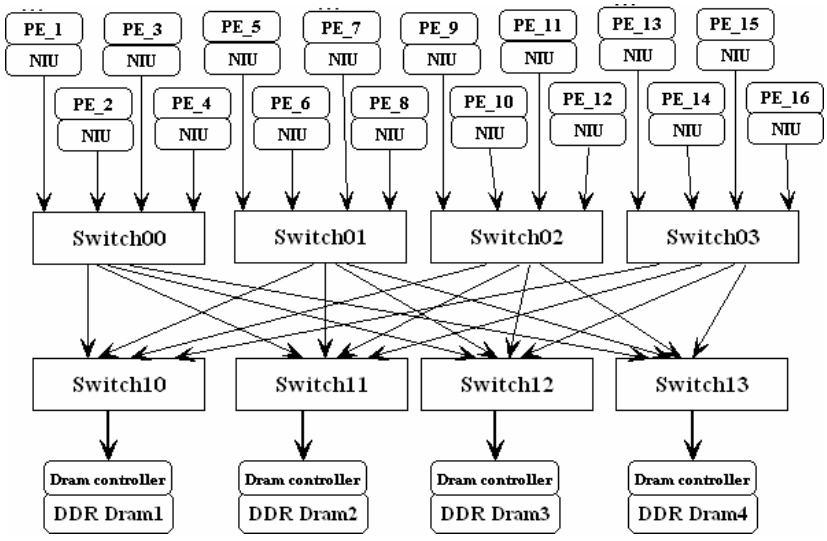


Figure 5.7 NoC testbench architecture

5.4.2 Results of NoC on one eFPGA

Here we compare our optimal results with the original NoC implementation on the eFPGA of M2000 technique. The original NoC is synthesized on one M2000 eFGPA f4-16000-90nm_fast without optimization. Then our ILP solver is used to get the optimal result and the configurations of switches. The new NoC with optimal configurations is re-synthesized on the same eFGPAs. The area of eFPGA is set to 7000 flip-flops and 5000 LUTs.

Table 5.2 results of NoC on one eFPGA

switch	Frequency (MHz)	options					
		1	2	3	4	5	6
Switch00	26	2	0	1	0	1	0
Switch01	26	2	0	1	0	1	0
Switch02	26	2	1	0	1	1	0
Switch03	26	2	0	1	0	1	0
Switch10	30	2	0	1	0	1	0
Switch11	30	2	1	1	0	1	0
Switch12	30	2	1	1	1	1	0
Switch13	30	2	0	1	0	1	0

5.4.3 Results of NoC on numerous eFPGAs

Table 5.3 area constraint of 4 eFPGAs

eFPGA	Flip-flops	LUTs
0	5000	4000
1	4000	4000
2	3000	4000
3	3000	5000

We try to place the NoC on 4 eFPGAs whose size is not very large. At first all the 4 eFPGAs are used for placement and the minimal eFPGAs algorithm is used to get the optimal solution.

Table 5.4 results of NoC on 4 eFPGAs

switch	eFPGA	Frequency (MHz)	options					
			1	2	3	4	5	6
Switch00	0	26	2	1	1	0	1	0
Switch01	0	26	2	1	0	1	1	0
Switch02	2	26	2	1	0	1	1	0
Switch03	3	26	2	0	0	1	1	0
Switch10	0	30	2	1	1	1	1	0
Switch11	3	30	2	1	1	1	1	0
Switch12	2	30	2	1	1	1	1	0
Switch13	1	33	2	1	1	1	1	0

Table 5.5 results of NoC on minimal of 4 eFPGAs

switch	eFPGA	Frequency (MHz)	options					
			1	2	3	4	5	6
Switch00	0	26	2	0	1	0	1	0
Switch01	0	26	2	0	0	1	1	0
Switch02	0	26	2	1	1	0	1	0
Switch03	1	26	2	1	1	0	1	0
Switch10	1	30	2	0	1	0	1	0
Switch11	1	33	2	0	1	0	1	0
Switch12	1	30	2	1	1	0	1	0
Switch13	1	30	2	1	1	1	1	0

The 8 switches are placed on all the 4 eFPGAs, if the minimal eFPGAs Algorithm is not used. Then an optimal solution with only 2 used eFPGAs is found by the NoC on minimal eFPGAs Algorithm. The switches are placed on the eFPGA 0 and 1 with different configurations.

5.5 Conclusion

Reconfigurable network on chips require efficient reconfigurable hardware support in ASIC environment. The emerging eFPGA IPs allow the integration of reconfigurable area in ASIC devices. The organization and dimensioning of this area is an important issue to be tackled in order to maximize the efficiency of network on chip mapping. The proposed linear programming methodology provides a solution to this problem. Several use cases have been studied which proves the efficiency of our approach. As the eFPGA process technology is reaching 45 nm, the implementation of our ILP solution onto the newest technology will be the future work.

Reference

[1] A.A.Jerraya and W.Wolf, “Multiprocessor Systems-on-Chips”, Morgan Kaufman Pub., 2004

[2] Tobias Bjerregaard, Shankar Mahadevan, A survey of research and practices of Network-on-chip, ACM Computing Surveys (CSUR), Volume 38 Issue 1, June 2006

[3] Owens, J.D.; Dally, W.J.; Ho, R.; Jayasimha, D.N.; Keckler, S.W.; Li-Shiuan Peh; Research Challenges for On-Chip Interconnection Networks,IEEEMicro Volume 27, 5, Sept.-Oct. 2007 pp. 96 – 108

[4] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma and R. Guerrieri, “XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module,” IEEE Journal of slid-state circuits. vol. 41, NO.1, pp. 85–96, January 2006.

[5] S. Murali, M. Coenen, A. Radulescu, K. Goossens and G. De Micheli, “A Methedology for Mapping Multiple Use-Cases onto Networks on Chips,” Desing, Automationand Test in Europe,2006, prceedings. vol: 1, pp. 1- 6, March 2006 .

[6] A. Kumar, A. Hansson, J. Huisken and H. Corporaal, “An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip,” Desing, Automationand Test in Europe,2007, prceedings. vol: 1, pp. 16- 20, April 2007.

[7] V. Aken'Ova and R. saleh, "A 'Soft++' eFPGA Physical Design Approach with Case Studies in 180nm and 90nm," Proceeding of the 2006 Emerging VLSI Technologies and Architectures, vol:1, pp103-108. March 2006 .

[8] V. Nollet, T. Marescaux, D. Verkest and J-Y. Mignolet, "Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles," Desing, Automationand Test in Europe,2005, prceedings. vol: 1, pp. 234- 239, March 2005.

[9] V. Aken'Ova, G. Lemieux, and R. Saleh, "An Improved 'Soft' eFPGA Design and Implementation Strategy," IEEE Custom Integrated Circuits Conference, 2005 proceeding, pp. 179–182, September 2005.

[10] M2000 http://www.m2000.com

[11] GNU Linear Programming Kit http://www.gnu.org/software/glpk/

[12] Schrijver, Theory of Linear and Integer Programming,

[13] Li, X.; Hammami, O.;, NOCDEX: Network on Chip Design Space Exploration Through Direct Execution and Options Selection Through Principal Component Analysis, Industrial Embedded Systems, 2006. IES '06. International Symposium on 18-20 Oct. 2006 Page(s):1-4

[14] Menta company http://www.efpga.com/

[15] Abound company http://www.aboundlogic.com/

[16] Ahmed, Syed Zahid et al. , Exploration of power reduction and performance enhancement in LEON3 processor with ESL reprogrammable eFPGA in processor pipeline and as a co-processor, Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09. page(s): 184 – 189

[17] Raptor™ FPGA Product Information Brief, PB001 (V1.0) May 2009, http://www.aboundlogic.com/

[18] Borgatti, M.; Lertora, F.; Foret, B.; Cali, L., A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA, and customizable I/O, Solid-State Circuits, IEEE Journal of, Volume 38, Issue 3, Mar 2003 Page(s): 521 – 529

[19] Andrea Lodi et al. , XiSystem: A XiRisc-Based SoC With Reconfigurable IO Module, IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 41, NO. 1, JANUARY 2006

[20] Strech, http://www.stretchinc.com

[21] Roman Lysecky, Scalability and Parallel Execution of Warp Processing: Dynamic Hardware/Software Partitioning, International Journal of Parallel Programming, Volume 36, Number 5 / October, 2008

[22] B. Neumann, T. von Sydow, H. Blume, T. G. Noll , Application Domain Specific Embedded FPGAs for Flexible ISA-Extension of ASIPs, Journal of Signal Processing Systems, Volume 53 , Issue 1-2 (November 2008)

[23] Deledda, A et al. , Design of a HW/SW Communication Infrastructure for a Heterogeneous Reconfigurable Processor, Design, Automation and Test in Europe, 2008. DATE, Page(s):1352 – 1357

[24] Delorme, J. Martin, J. Nafkha, A. Moy, C. Clermidy, F. Leray, P. Palicot, J. A FPGA partial reconfiguration design approach for cognitive radio based on NoC architecture, Circuits and Systems and TAISA Conference, 2008. NEWCAS-TAISA 2008

[25] Krasteva, Y.E. Criado, F. de la Torre, E. Riesgo, T. , A Fast Emulation-based NoC Prototyping Framework, Reconfigurable Computing and FPGAs, 2008. ReConFig '08, page(s): 211-216

[26] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. van der Veen, "Dynoc: A dynamic infrastructure for communication in dynamically reconfigurable devices," in FPL, 2005, pp. 153–158.

[27] T. Pionteck, R. Koch, and C. Albrecht, "Applying partial reconfiguration to networks-on-chips." in FPL. IEEE, 2006, pp. 1–6.

[28] R.Benmouhoub and O.Hammami, "Noc monitoring harwdare support for fast noc design space exploration and potential noc partial dynamic reconfiguration," in IEEE IES, 18-20 oct 2006.

[29] Dafali, R. Diguet, J.-P. Sevaux, M. , Key Research Issues for Reconfigurable Network-on-Chip, Reconfigurable Computing and FPGAs, 2008. ReConFig '08, page(s): 181-186

6. SSM IP: Small Scale Multiprocessor IP

Future generation multiprocessor system on chip (MPSOC) will be based on hundreds of processors connected through network on chips. One of the challenges is to tackle the design productivity required to reach this goal. We propose a NoC based small scale multiprocessor IP (SSM IP) as a building block for large scale multiprocessor. In this chapter we describe the architecture of such an SSM IP as well as the prototyping results on a single chip FPGA. Image processing applications are used as preliminary parallel software evaluation and demonstrate the potential of design space exploration at small scale multiprocessor.

6.1 Cluster MPSOC and NoC design

ITRS Semiconductor roadmap [1] projects that hundreds of processors will be needed for future generation multiprocessor system on chip (MPSOC) designs. Among the various challenges of MPSOC [2] design productivity is paramount. Design productivity of system on chip at large is by itself a declared design technology major challenge [1]. The design productivity gap represents the fact that Moore's law regular progress generates a number of available transistors which grows faster than the ability to use them in a meaningful way. This gap puts at risk the entire semiconductor investment cycle. The design productivity gap is the result of the combined silicon complexity and system complexity which translates into super exponentially increasing complexity. Silicon complexity is the result of physical properties (non ideal scaling of device parasitic and supply threshold/voltages, coupled high-frequency devices and interconnects, manufacturing variability, complexity of manufacturing handoff, process variability, decreased reliability) and scaling of global interconnect performance relative to device performance. System complexity associated challenges are reuse, verification and test, cost-driven design optimization, embedded software design, reliable implementation platforms, and design process management.

Although design reuse, platform and IP based design are powerful concepts [1] in order to speed up the design process the basic IPs are still too elementary to quickly build large scale multiprocessors. Indeed, reuse productivity of design, verification and test must scale at more than twice per technology generation it is then necessary to raise the size and complexity of IPs to small scale multiprocessor IPs in order to speed up the process. Designing large scale multiprocessors based on small scale multiprocessors allows quickly duplicating building elements and building a large scale multiprocessor in reasonable design time. Small scale multiprocessor IP can be soft or hard IP. Hard IP alleviates many design efforts by providing quick solution. However, soft IP allows design space exploration and efficient tuning of resources to match multi-objective requirements. In addition, reliable and predictable silicon implementation fabrics that support ever-high level electronic system design handoff are needed. We propose a small scale multiprocessor soft IP fully based on more elementary soft IP and analyze the potentials of design space exploration on this IP. The effects of silicon complexity and system complexity are partially addressed as the first step towards a more general and extended flow.

Design productivity with regard to multiprocessor system on chip is a relatively new research issue and there are still many open research issues. Modelling of multiprocessors and adequate level of abstraction (TLM , RTL), performance evaluation and design space exploration, verification and test trough simulation or emulation are current topics of debates. Multiprocessor on chip performance evaluation requires billion of cycles of validation which cannot be afforded by traditional simulation techniques. Multiprocessor designers face prohibitive simulation times which have been coined as the simulation wall [3]. In addition, simulation is not able to take into account implementation issues: processors have technology dependent frequency and area, network on chip are layout sensitive and DDR modules have very precise timing models and operations mode. Emulation and FPGA-based emulator have been recognized as efficient techniques for the performance evaluation and validation of multiprocessors. In [5] design space exploration of multi-processor on multi-FPGA platform [11] have been conducted with masters on one chip, network on chip on a second chip and slaves on a third chip. Automatic design space exploration of multiprocessor on chip on a single large scale FPGA chip have been conducted [13] with automatic tuning and test of the multiprocessor. The combined effects of compiler, architecture and place and route have been

addressed for a single chip in [8]. A complete network on chip emulation platform has been proposed in [15] and a multiprocessor platform in [16,17]. Finally large scale multiprocessor project have emerged in [18]. However to the best of our knowledge, the issue of design productivity and the use of small scale multiprocessor IP as building blocks for large scale multiprocessor have not been addressed.

We address the issue of building a small scale multiprocessor as a basic building block for large scale multiprocessors and evaluate through actual synthesis, place and route and execution the potentials of such IP.

6.2 SSM IP FPGA Design and Implementation

6.2.1 Small Scale Multiprocessor Soft IP

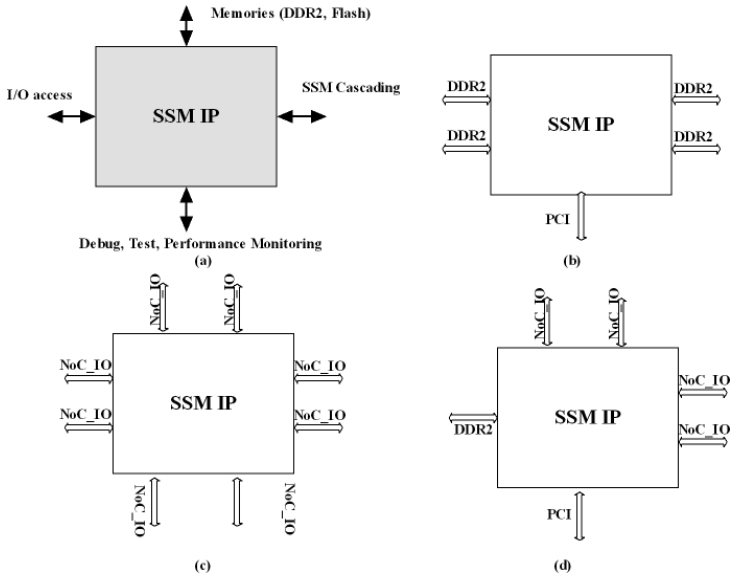


Figure 6.1 Small Scale Multiprocessor IP Interfaces

A small scale multiprocessor soft IP can be seen as modulable IP where external connections are more or less modified. Figure 1 shows possible variations (b,c,d) of SSM IP from a basic configuration a. The basic configuration includes external connections for: (1) memories (DDR2, flash) (2) IO access (3) debug, test and performance monitoring and (4) SSM cascading. SSM cascading represents the possibility to reuse the SSM IP to design a larger scale multiprocessor system on chip. This extension for a NOC based SSM goes through extended NOC connections (NOC_IO) as seen in Figure 6.1 for variations c and d.

Figure 6.2 shows a general view of this principle through a general network on chip in (a) or a mesh oriented network on chip in (b).

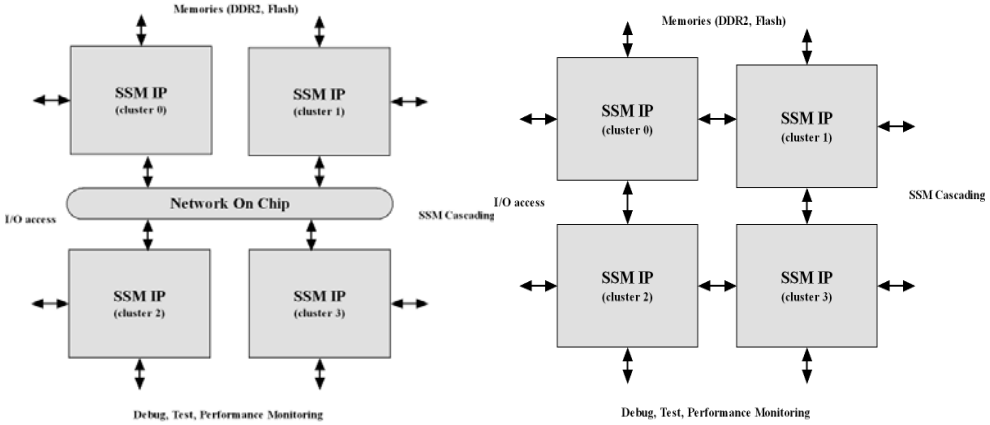


Figure 6.2 Small Scale Multiprocessor IP Composition (a) NOC based general case (b) NOC Mesh organized

The SSM IP should be designed from start by taking into account the extensibility and design reuse of the SSM IP. A natural NOC for the connections of SSM IP is a mesh network on chip under the condition that the SSM IP is organized internally as a cluster in order to reduce the size of the network (average number of hops) and maximizes the area dedicated to computation over communication.

6.2.2 Architecture

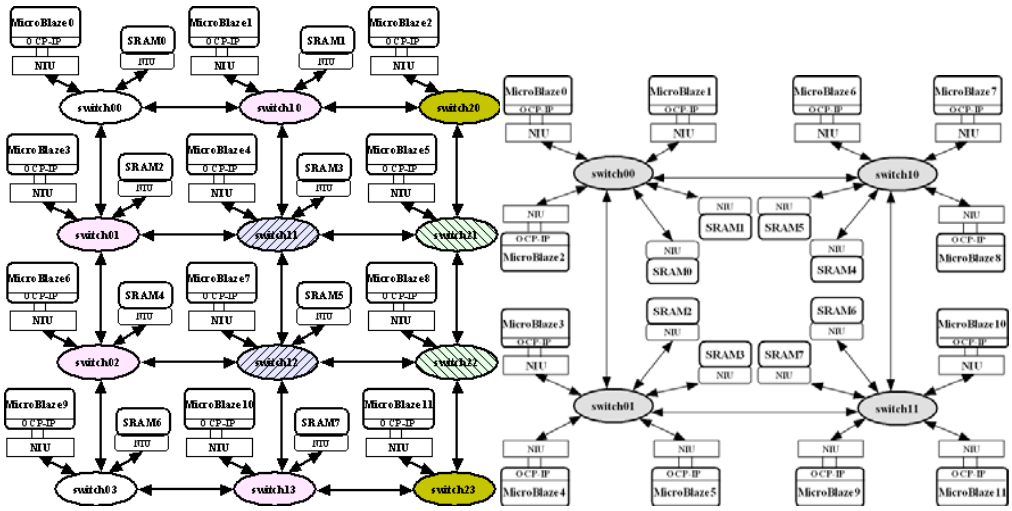


Figure 6.3 Small Scale Multiprocessor IP architecture (a) Full mesh (b) Cluster based

The SSM IP architecture is a mesh of clusters. The SSM IP is composed of 12 processors connected through a 2x2 mesh with 3 Xilinx MicroBlaze processors and 2 SRAM on chip memories per switch. The network on chip topology is mesh for its scalability properties and easy extensibility [3,4,5]. In addition, in system on chip where layout regularity and interconnection delays are paramount the mesh topology provides short links and easier place and route. With regards to silicon complexity scaling of global interconnect performance relative to device performance is better managed through short links. Several recent system on chip projects and devices use the mesh topology. The TRIPS project [7] use a mesh topology while industrial chips such as TILE64 form Tilera [6] and Intel TERAFLOPS [8] use mesh topology as well.

However, our proposed cluster based mesh architecture design has been preferred over a full mesh in order to fully exploit the data locality processing of image and multimedia applications. Images can be distributed equally among the shared memories of each cluster so that processors belonging to a cluster can operate on the image portion associated to a cluster.

The full mesh architecture example of our SSM IP is shown in Figure 6.3 (a), and our proposed cluster based mesh architecture is presented in Figure 6.3 (b) for comparison. Our SSM IP architecture is a mesh of clusters. It is composed of 12 processors and 8 SRAMs connected through a 2x2 mesh with 3 Xilinx MicroBlaze processors and 2 SRAM on chip memories per switch. Meanwhile full mesh architecture use 12 switches or even more dimension if each switch is restricted to just one processor or SRAM, see in Figure 6.3 (a).
Since the SSM IP is a soft IP it is composed of various soft IP described in following table.

Table 6.1 IPs of SSM multi-processor.

IP component	description	source	version	Qty
processor	Soft core IP	Xilinx MicroBlaze Soft core IP	6.00 b	12
memory	Soft core IP	Xilinx Coregen 96KB	v.2.4.	8
Network on chip switch	Soft core IP	VHDL Arteris Danube library	1.10	4

Our design of the network on chip is based on Arteris Danube Library. The Arteris Danube library [20] includes the switch generator which is an essential building block of the NoC interconnect system. NoCcompiler [21] tools estimates switch area using NAND2 gate as unit. The relation between switch area and IO number is complex as showed in Figure 6.4.

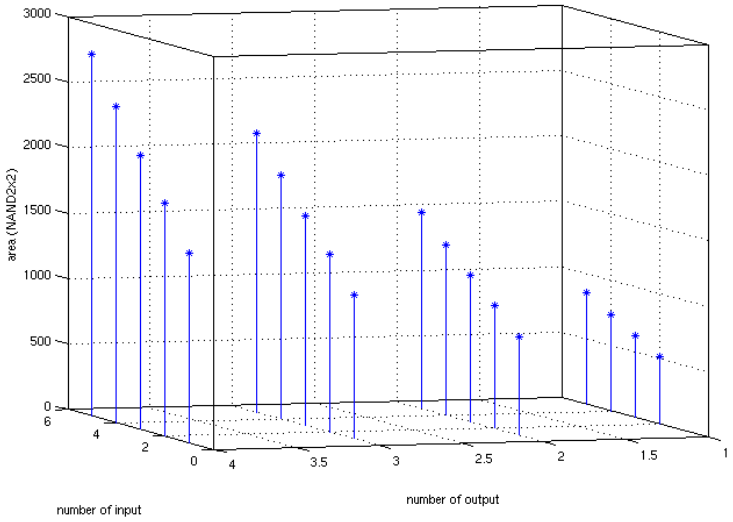


Figure 6.4 Switch Area Variations as Function of Number of Inputs-Outputs

Switch area is proportional linearly to the number of input (or output), when the number of output (or input) is fixed. With numerical analysis of data, the relation between area of switch and the IO number is:

$$F(X, Y) = 72*X*Y+273*Y+39*X+18$$

where the F(X,Y) means the area of switch in unit of NAND2x2, X the number of input and Y the number of output.

Processor to switch is taken as input and switch to SRAM as output, while interconnection between switches is 2 directions. In cluster based mesh, for each switch X=5 and Y=4. Total area is 4*F(5,4) = 10980 NAND2x2. While in full mesh of figure 3(a), there are 5 types of switch regarding to IO numbers. And total area of this full mesh is 4*F(4,4)+2*F(3,3)+2*F(3,2)+2*F(5,5)+2*F(4,3) = 25572 NAND2x2.

Besides shorter communication link, our cluster based mesh architecture consumes much less area than full mesh implementation. This clustered design increases the size of each switch but reduces the number of switches.

6.2.3 Design Automation Flow

EDA tools of Xilinx and Arteris companies are combined together for our SSM IP design automation work flow in Figure 6.5.

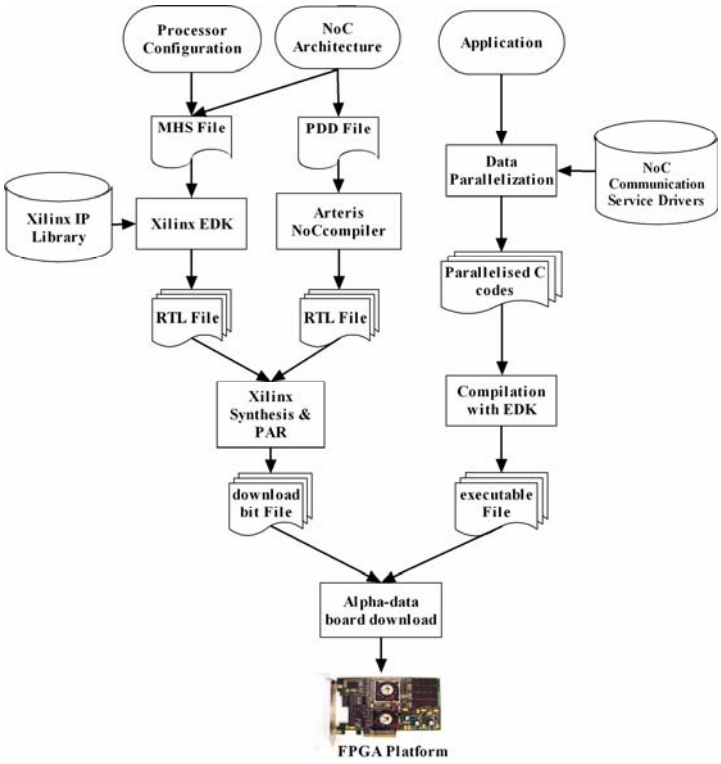


Figure 6.5 SSM Design Automation Work Flow

The Xilinx EDK tool is used to generate our SSM (Small Scale Multiprocessor) processor elements according to the configuration input. Once the RTL files of SSM are generated, they are reused for the multi-FPGA large scale multiprocessor synthesis, which can largely reduce system design time. Arteris NoCcompiler tool is used to generate NoC RTL files with the input of NoC architecture. Then Xilinx synthesis and PAR (place and route) tools are used to generate the download bit files of FPGA from these RTL files. Sequential C code of application is parallelized to each processor and NoC communication service functions are used for data communication and synchronization. These parallelized codes are compiled with Xilinx EDK tools to generate execution file for each processor. Finally system bit file and executable files are downloaded to Alpha-data FPGA platform and application is executed.

6.2.4 Processor Element and NoC communication service

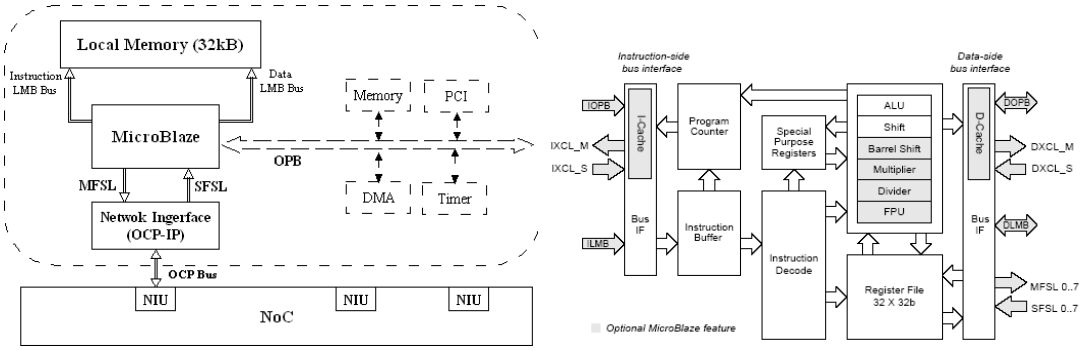


Figure 6.6 (a)Processor Tile (b) MicroBlaze core block diagram

MicroBlaze soft IP [24] is used as basic processor element in SSM IP. It is a 32-bit 3-stage single issue pipelined Harvard style embedded processor architecture provided by Xilinx as part of their embedded design tool kit (EDK), in Figure 6.6 (b). The MicroBlaze processor is flexible, and gives the user control of a number of features such as the cache sizes, interfaces, and execution units like: selectable barrel shifter (BS), hardware multiplier (HWM), hardware divider (HWD), and floating point unit (FPU).

Our SSM design is OCP-IP [26] compliant which implies that we can change processor IP by any other OCP-IP compliant processor IP while leaving the overall design identical. The OCP-IP protocol is used for the communication between the processors and Network on Chip. Arteris supplies NIU (Network Interface Unit) in NoC to support OCP-IP protocol. A FSL-OCP network interface is designed in order to make MicroBlaze processor compatible to OCP-IP protocol as shown in Figure 6.6 (a). This interface gets data from MicroBlaze processor through FSL link [23] and transfers data to SRAM through Network on Chip under OCP-IP protocol. It encapsulates and decapsulates data between FSL links and OCP-IP bus interface.

A NoC communication service library of driver is written in C for the FSL-OCP network interface to support OCP-IP basic and extended communication modes as in figure 5. As FSL link works like 32 bits FIFO interface, the address and data are transferred in two phases between MicroBlaze processor and FSL using basic micro C code ‘putfsl()’ and ‘getfsl()’ in EDK library. In the service library, 5 OCP-IP compliant communication modes [26] are supported: Write, Read, Read Exclusive, Write Non-Post and Write Conditional. Parallelized

application codes use these drivers for data communication and synchronization between processors and SRAMs.

6.2.5 Implementation

The implementation of the small scale multiprocessor has been realized by targeting the largest Xilinx Virtex-4 FPGA chip the FX140. Design has been realized using the Xilinx tools (EDK, ISE) with the Xilinx library of IPs. The objective of the implementation was to design a multiprocessor of sufficient scale to be significant while leaving some chip area and resources for design space exploration. The implementation details are given in Table 6.2.

Table 6.2 Implementation results

FPGA Resource	Utilization	%
Number of DSP48s	36/192	**18%**
Number of RAMB16s	544/552	**98%**
Number of Slices	25261/63168	**39%**
Number of SLICEMs	2795/31584	**8%**

The layout has been left intentionally un-optimized to leave enough area to extend the processor and network on chip features through design space exploration. The network on chip is based on Arteris network on chip technology [19-21]. The Arteris Danube library [21] provides a wide range of network on chip components allowing the design of any kind of packet based wormhole routing network. Implementation is achieved through the NoCcompiler design tool. It should be noted that all the network on chip component are parametrical components which allows design space exploration at all network on chip levels. This automatic exploration has already been achieved on Arteris Danube library components for small scale multiprocessor with NOCDEX design flow [11].

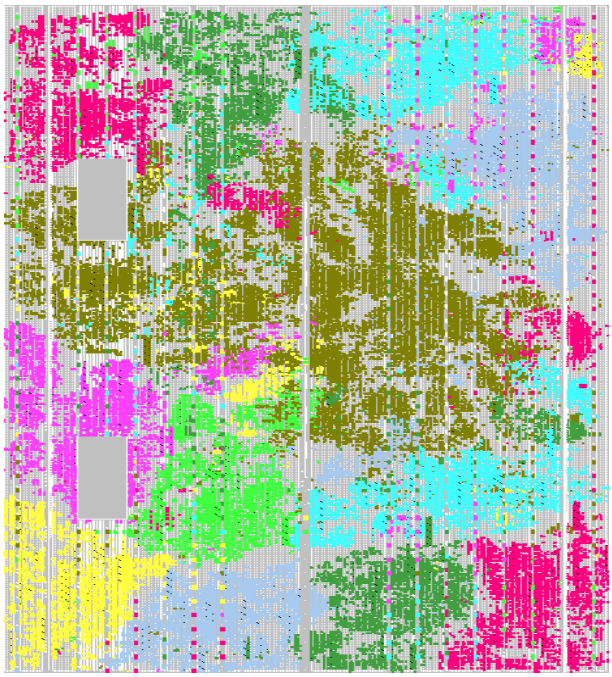


Figure 6.7 Multiprocessor Xilinx FX140 floorplan (un-optimized)

We used the Alpha-data board ADM-XRC for the validation and test of the small scale multiprocessor with debugging from a host through PCI.

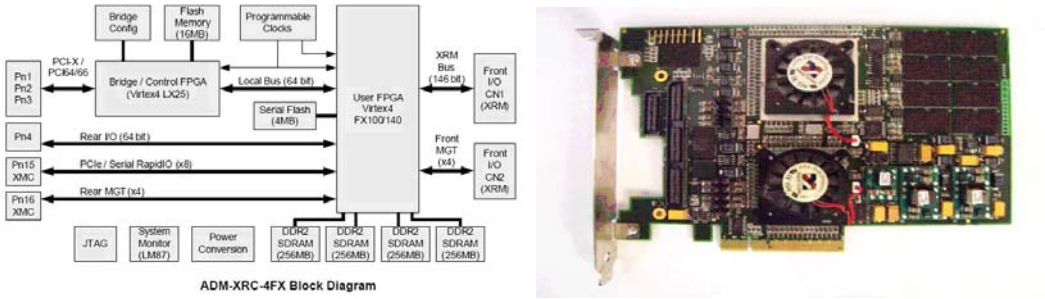


Figure 6.8 Alpha-Data Board ADM-XRC (a) board architecture (b) board

This validation guarantees that the small scale multiprocessor is verified and allows design space exploration of SSM IP.

6.3 Design Space Exploration of SSM IP

6.3.1 SSM Soft IP potential variations

The SSM IP is a soft IP. It is fully configurable for all its components: (1) embedded processor (2) NOC elements (3) memories. Therefore, it offers numerous opportunities to

tailor the SSM under resource and performance constraints. In MicroBlaze architecture, barrel shifter (BS), hardware multiplier (HWM), hardware divider (HWD), and floating point unit (FPU) are all selectable.

The Arteris Danube IP Library switch component architecture can be customized through several options described in the following table. It is possible to select the arbitration type of the switch among 4 possible values Round-Robin, LRU, Random, Fifo with default value round robin. Several optional registers can be added in order to pipeline the switch.

Table 6.3 Switch options.

Switch options	values
Arbitration type	Round-Robin(1), LRU(2), Random(3), Fifo(4)
Input pipeline register	True(1), false(0)
Forwards pipeline register	True(1), false(0)
Backwards pipeline register	True(1), false(0)
Crossbar pipeline register	True(1), false(0)
Dual cycle arbitration	True(1), false(0)

Because of the constraints between different options, there are 80 different configurations for each switch. Here we show four examples of possible system architecture in the design exploration space. The 4 examples modify the embedded processor features as well as the switch features. Although this design space exploration is not large as in [9, 11] it illustrates the powerful variations of SSM soft IP.

Table 6.4 the 4 versions of architecture.

	NOC	**MicroBlaze**
Arch. V1	Fwdpipe	Multiplier
Arch. V2	Fwdpipe+Bwdpipe+Pipe	Multiplier
Arch. V3	Fwdpipe	Multiplier+FPU
Arch. V4	Fwdpipe+Bwdpipe+Pipe	Multiplier+FPU

6.3.2 Synthesis, Place and Route Time and Target frequency

In this section we provide information regarding the time required for respectively: (1) the synthesis (2) place and route (PAR) of each potential architecture. Most time of hardware platform design space exploration is spent on the system synthesis and place and rout (PAR).

Figure 6.9 shows the time used for the 4 examples in Table 6.4 and the maximum frequency on which the system can be run with these different configurations.

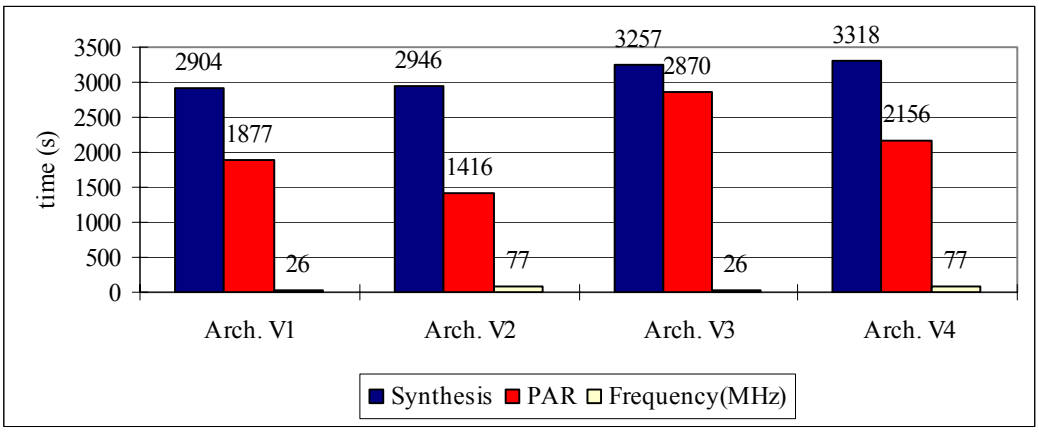


Figure 6.9 DSE Execution Time

The design space exploration translates to varying frequencies for each SSM IP configuration but obviously also by varying areas. Table 5 describes the resulting SLICEs and SLICEMs for the 4 configurations and shows a 54% variation in area between Arch V1 and Arch V4.

Table 6.5 Area of 4 version architectures

	SLICEs	**SLICEMs**
Arch. V1	24888	2795
Arch. V2	27040	2795
Arch. V3	36067	2987
Arch. V4	38183	2987

With regard to the design space exploration process, we can observe that changing the SSM soft IP change synthesis and place and route execution time as much as twice on place and route and up to 1,14 on synthesis.

6.4 Application Performance Evaluation and Comparison

The optimization of SSM multi-processor is multi-objective by nature and is focused on the system frequency, performance and area. image processing application NL-means (Non-Local means) filter and various basic image applications are used as preliminary parallel software evaluations on our SSM. A performance evaluation of the various configuration architectures demonstrates the potential of design space exploration for small scale multiprocessor.

6.4.1 NL-means image filter

For 2D natural images, the NL-means image filter outperforms state-of-the-art denoising methods such as total variation minimization scheme, anisotropic diffusion or translation invariant wavelet thresholding [27]. Nevertheless, the main drawback of the NL-means filter is the computational burden due to its complexity. Let N^2 denote the size of the 2D image, then the complexity of the filter is in the order of O ($N \times m^2 \times d^2$). [28] finds a computational time of 58.1 seconds for a 512 x 512 image, with parameters m=15 and d=7 running on a 2.0GHz CPU.

Given a discrete noisy image z=z(x), the estimated value 'NL z(x)' for a pixel x is computed as a weighted average of all the pixels in the image,

$$\text{NL z(x)} = \frac{1}{C(x)} \sum_{y \in \Omega} w(x,y) z(y)$$

where the weights $\{w(x,y)\}_y$ depend on the similarity between the pixel x and y. and C(x) is the normalizing factor.

$$\text{C(x)} = \sum_{y \in \Omega} w(x,y)$$

The similarity between two pixels x and y depends on the similarity of the intensity gray level vectors, the similarity window, of fixed size d x d pixels and centered at a pixel k. This similarity is measured as a decreasing function of the weighted Euclidian distance.

The weights associated with the quadratic distance are defined by

$$\text{w(x,y)} = e^{-\frac{\|z(x)-z(y)\|_{2,a}^2}{h^2}}$$

where the parameter h controls the decay of the exponential function and therefore the decay of the weights.

6.4.2 Implementation and Results of NL-means filter

The NL-means algorithm is written in C for each MicroBlaze processor in SSM IP. The gray image of 64x48 is divided into 12 blocks with the same size and dimension of 16x16 and. Each line of 3 blocks is mapped to one SRAM of each cluster as shown in Figure 6.10. Each processor get one block of image from local memory and after the NL-means denoising, the results are sent and stock in the other local memory at the cluster. Finally execution time is recorded.

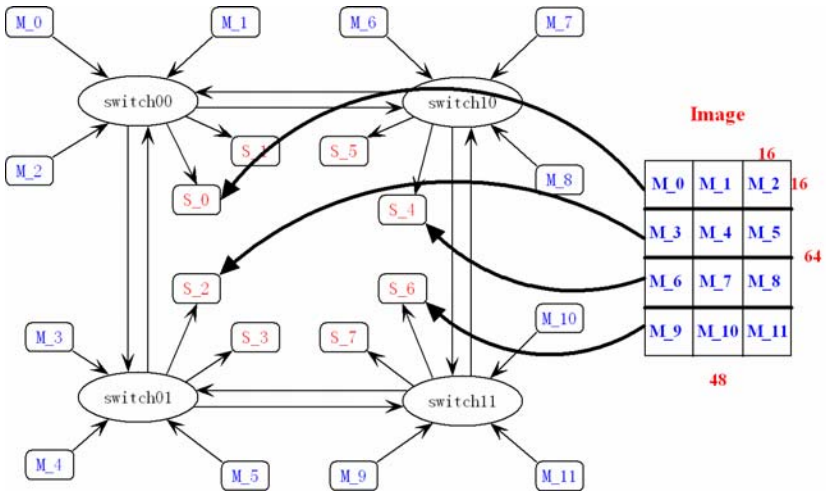


Figure 6.10 Image mapping to SRAMs for NLMeans filter application

Figure 6.11 shows the performances of NL-means image filter application on the 4 different architectures described in section 5. In order to better analyze the correlation between execution time and the area consumption, both information are provided in the same figure. The right Y axis provides area (slices) information while the left Y axis provides execution time information.

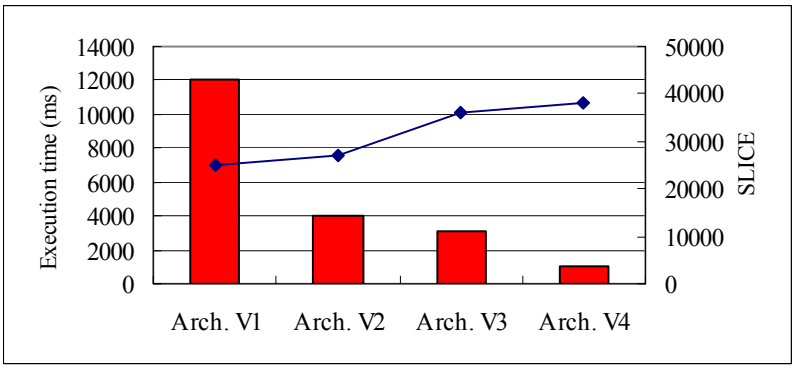


Figure 6.11 Execution results of NLMeans filter

6.4.3 Implementation and Results of other basic applications

Three basic applications are used to test the system performance: dot product, matrix multiplication and conservative image filter. Each MicroBlaze calculates a dot product of dimension 800*800; a matrix multiplication of dimension 80*80 and a conservative filter of dimension 80*80. All the variables are of floating point types. MicroBlaze0 in the 12 MicroBlazes of SSM is used as a system pilot; it checks the flag variables of the other 11 MicroBlazes to find out whether all the MicroBlaze have finished the computation and report the total number of cycle as the system performance.

We compared the NoC based SSM system with a simple system composed of just one MicroBlaze with a timer to see the error of performance caused by the verification mechanism of flag variable in SSM system. Only one of the MicroBlazes in SSM is used for the computation.

Table 6.6 difference between NoC based system and bus-based system.

	NoC (cycle)	**NoC (us)**	**Bus (cycle)**	**Bus (us)**
Dot product	286608	10891.1	286511	2865.11
Matrix multiplication	4235322	160942.2	4235247	42352.47
Filter conservative	15965011	606670.4	15953716	159537.16

We can see from the table 4 that the error of total number of cycle between NoC based system and bus-based system is less than 0.1%.

All the variables are of floating point types to show to impact of FPU in MicroBlaze processor. As the image mapping of NL-means filter, matrix A is stocked in one local memory of cluster and matrix B in each processor's local memory. The final results are sent to the other local memory of cluster.

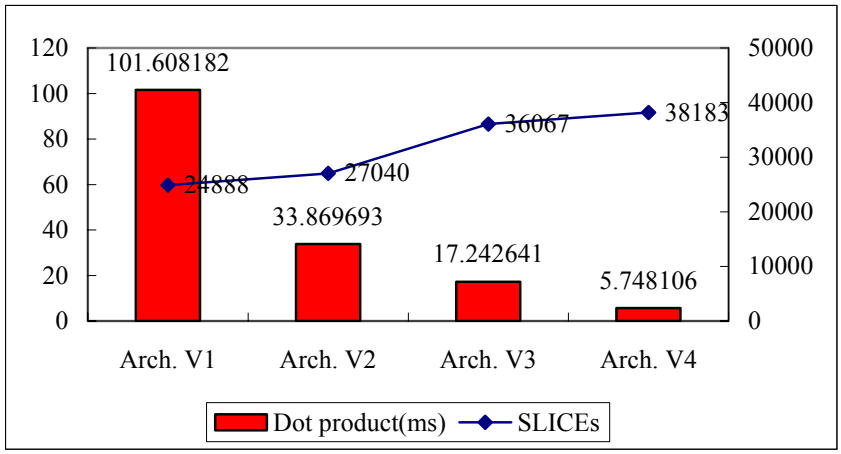


Figure 6.12 Execution time of dot product

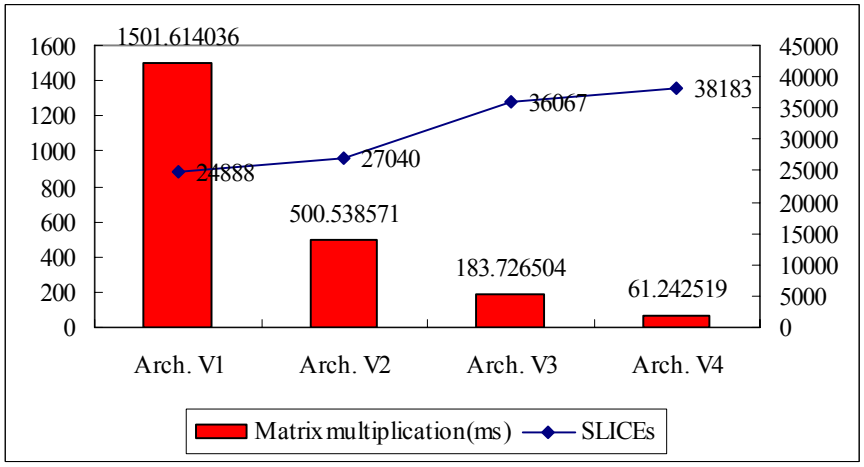


Figure 6.13 Execution time of matrix multiplication

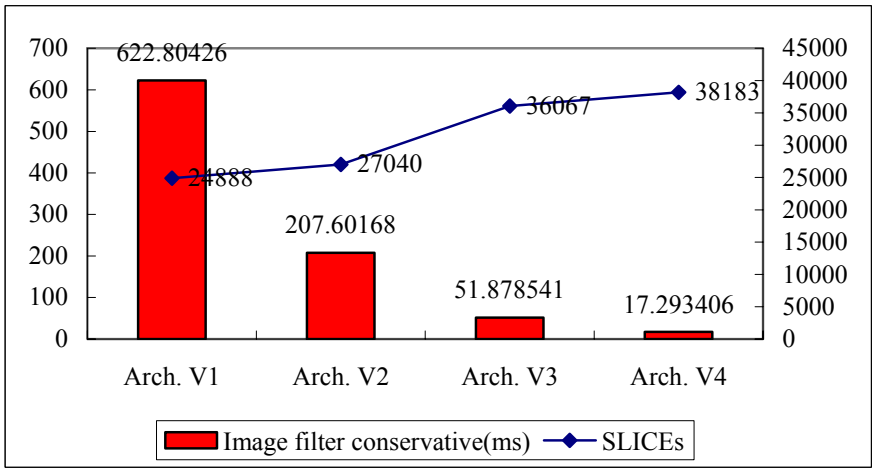


Figure 6.14 Execution time of filter conservative

6.4.4 Results Analysis

Clearly increasing the area reduces the execution time in all cases. However, the gain is not linear. For example between Arch. V4 and V3 or between Arch. V2 and V1, the area variation is a mere 5 % but the execution time is reduced by more than half in both examples, as the frequency of system is improved greatly using more pipelines in NoC with little usage of Slices . This point emphasizes the importance of a SSM soft IP over a SSM hard IP in order to better exploit area and customization. The floating point unit (FPU) can greatly improve the floating point calculation performance of MicroBlaze processor. This impact is more obviously in matrix multiplication, as almost 100% of instructions are floating-point calculation in matrix multiplication case, while only 33% of instructions in NL-means filter application. The system performance can be improved by one order of magnitude between the Arch. V4 and V1.

6.5 Conclusion

Next generation multiprocessor on chip will be based on hundreds of processors. Multiprocessor on chip design is very complex and in order to reach efficient working silicon in reasonable time we propose a small scale multiprocessor design as a building block (soft IP) for large scale multiprocessor. Regularity and data locality of image processing and multimedia applications favours both regular NOC like mesh and computation clustering. We proposed a Mesh NOC based small scale multiprocessor IP which have been fully prototyped on a large scale FPGA chip. Architectural variations among 4 selected architectures demonstrate the area saving and performance potential of soft IP. In addition reasonable synthesis, place and route execution time and achieved target frequencies justify the design effort. Building large scale multiprocessors from the proposed SSM IP can be fast as the main design effort resides in the connection and adaptation of NOC addressing. We plan to extend this work to a larger family of SSM IP. The effects of silicon complexity and system complexity are partially addressed.

References

[1] ITRS http://www.itrs.net/

[2] A.A.Jerraya and W.Wolf, “Multiprocessor Systems-on-Chips”, Morgan Kaufman Pub., 2004.

[3] Benini, L.; De Micheli, G.; Networks on chips: a new SoC paradigm, Computer, Volume 35, Issue 1, Jan. 2002 Page(s):70 – 78

[4] T.Bjerregaard, S.Mahadevan, “A survey of research and practices of Network-on-chip”, ACM Computing Surveys (CSUR), Volume 38 Issue 1, June 2006

[5] Owens, J.D.; Dally, W.J.; Ho, R.; Jayasimha, D.N.; Keckler, S.W.; Li-Shiuan Peh, “Research Challenges for On-Chip Interconnection Networks”, IEEE Micro, Volume 27, Issue 5, Sept.-Oct. 2007 Page(s):96 – 108

[6] Bell, S.; Edwards, B.; Amann, J.; Conlin, R.; Joyce, K.; Leung, V.; MacKay, J.; Reif, M.; Liewei Bao; Brown, J.; Mattina, M.; Chyi-Chang Miao; Ramey, C.; Wentzlaff, D.; Anderson, W.; Berger, E.; Fairbanks, N.; Khan, D.; Montenegro, F.; Stickney, J.; Zook, J. , TILE64 Processor: A 64-Core SoC with Mesh Interconnect , Tilera Corp. Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International.

[7] P.Gratz and al. “On-chip Interconnection Networks of the TRIPS Chip”, IEEE Micro, pp. 41-50, Sept. – Oct. 2007.

[8] Intel Teraops Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International

[9] Mouhoub, R.B.; Hammami, O.;Multiprocessor on chip: beating the simulation wall through multiobjective design space exploration with direct execution PDPS 2006, 25-29 April 2006 Page(s):8 pp.

[10] S.Hauck and A.DeHon, "Reconfigurable Computing The Theory and Practice of FPGA-Based Computation", Morgan Kaufmann 2007.

[11] Li, X.; Hammami, O.;, NOCDEX: Network on Chip Design Space Exploration Through Direct Execution and Options Selection Through Principal Component Analysis, Industrial Embedded Systems, 2006. IES '06. International Symposium on 18-20 Oct. 2006 Page(s):1 – 4

[12] CHIPit Platinum Edition – ASIC Emulation and Rapid Prototyping System – v.2.0., 2004, ProDesign www.uchipit.com

[13] R. Ben Mouhoub and O. Hammami, “MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution,” EURASIP Journal on Embedded Systems, vol. 2006, Article ID 54074, 14 pages, 2006.

[14] O.Hammami, "Heterogeneous Multiprocessor on chip Compiler, Architecture, Place and Route Design Space Exploration", in IEEE MELECON, May 5-7, 2008, Ajaccio, France

[15] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, F. Catthoor, A Complete Network-On-Chip Emulation Framework, Proceedings of the conference on Design, Automation and Test in Europe Volume 1 DATE '05, March 2005

[16] Nava, M.D.; Blouet, P.; Teninge, P.; Coppola, M.; Ben-Ismail, T.; Picchiottino, S.; Wilson, R.; An open platform for developing multiprocessor SoCs, Computer, Volume 38, Issue 7, July 2005 Page(s):60 – 67

[17] E.S. Chung, E. Nurvitadhi, J.C. Hoe, B.Falsafi, K.Mai Virtualized Full-System Emulation of Multiprocessors using FPGAs, 2nd Workshop on Architectural Research Prototyping WARP-2007

[18] K. Asonvic, RAMP: Research Accelerator for Multiprocessors, 2nd Workshop on Architectural Research Prototyping, WARP-2006

[19] Arteris S.A. http://www.arteris.com

[20] NoC Solution 1.10, NoC Compiler user's Guide, o918v2rs4, Dec. 2008, Arteris www.arteris.com

[21] Danube 1.10 – Packet Transport Units Technical reference – 04277v3rs6 – March 2008, Arteris. www.arteris.com

[22] Xilinx. Embedded system tools guide. Available on: http://www.xilinx.com/ise/embedded/edk_docs.htm.

[23] Xilinx. Xilinx fast simplex link IP. Available on: http://www.xilinx.com/bvdocs/ipcenter/data_sheet/FSL_V20.pdf.

[24] Xilinx. Xilinx microblaze soft core processor. Available on: http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

[25] Alpha-Data. ADM-XRC-4 PCI mezzanine card. Available on: http://www.alpha-data.com/.

[26] OCP-IP Open Core Protocol Specification 2.2.pdf http://www.ocpip.org/, 2008

[27] A. Buades, B. Coll, and J.M. Morel, A review of image denoising algorithms, with a new one, Multiscale Modeling & Simulation, vol. 4, no. 2, pp. 490-530, 2005.

[28] C. Kervrann, J. Boulanger, and P. Coupé, Bayesian non-local means filter, image redundancy and adaptive dictionaries for noise removal, in Proceedings of the 1st International Conference on Scale Space and Variational Methods in Computer Vision (SSVM’07), pp. 520-532, 2007 May-June.

[29] EVE TEAM Zebu-UF Emulation platform. www.eve-team.com

7. Large Scale Multiprocessor (LSM) DSE

Design productivity is one the most important challenge facing future generation multiprocessor system on chip (MPSOC). The modeling of dozens of interconnected IPs with distributed memories implies intensive manual EDA based design activity. We propose to improve design productivity by raising IP reuse to small scale multiprocessor IP combined with fast extension techniques for system level design automation in the framework of multi-FPGA based emulator. A design case study of a 48-processors multiprocessor on 4 large scale FPGA based industry class emulator validates our approach.

7.1 Flow Methodology from SSM to LSM

The adopted flow methodology follows the analysis of the previous section that is the combination of a tile design space exploration flow with a multi-tile design space exploration.

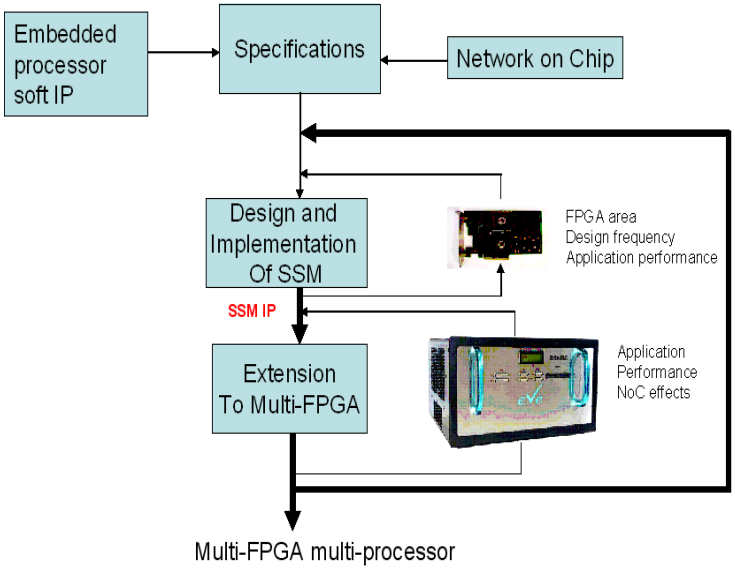


Figure 7.1 **MPSOC Design Space Exploration**

The resulting overall exploration flow is depicted in the following figure. Clearly steps 3 and 4 , heterogeneous MPSOC exploration and synthesis, place and route exploration are

performed on a single tile which in turn is duplicated and extended to multi-tile MPSOC for the exploration of the upper part of the design space exploration flow. This allows the analysis of application performance and NOC effects.

7.2 Extension of SSM IP to LSM

The Arteris switch is a wormhole routing synchronous switch, which can be used to construct a mesh connection of network on chip. XY routing is used for both the intra and inter FPGA communication. As shown in Figure 7.2, the mesh NoC can be easily extended to a multi-SSM multi-processor.

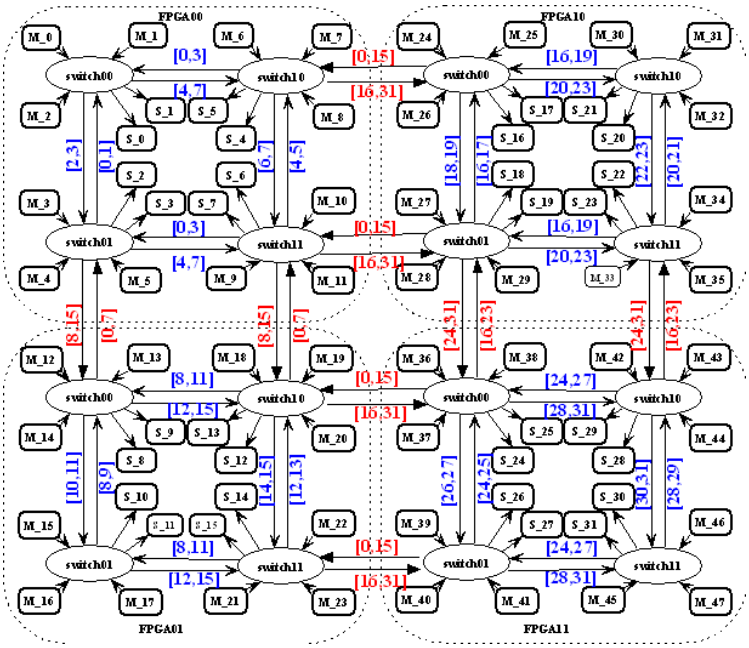


Figure 7.2 routing example of 2x2 multi-SSM

A 2x2 inter-FPGA multi-processor is planned to be implemented on the EVE Zebu-UF emulation platform [39]. All the configuration files of SSM IP are reused for the extension, which can largely minimize the synthesis time of this large scale multi-processor system. The SSM IP is reused to accelerate multi-FPGA multiprocessor design. Duplicated onto 4 FPGAs, all the SSM components will not be changed except the NoC adapted for the new 4x4 mesh topology.

The MicroBlaze processors still use their 32 bits address, but the SRAM's address is changed since there are 32 SRAMs in total. The first 5 MSB of the 32bits address are used as

the identification of SRAM as shown in the figure. The index of MicroBlaze in each FPGA is equal to (original index in SSM + index of FPGA * 12).

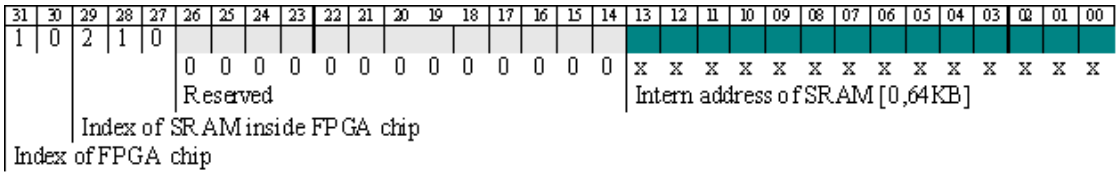


Figure 7.3 Multi-FPGA SRAM Addresses

The Arteris switch is a wormhole routing synchronous switch, which can be used to construct a mesh connection of network on chip. XY routing is used for both the intra and inter FPGA communication.

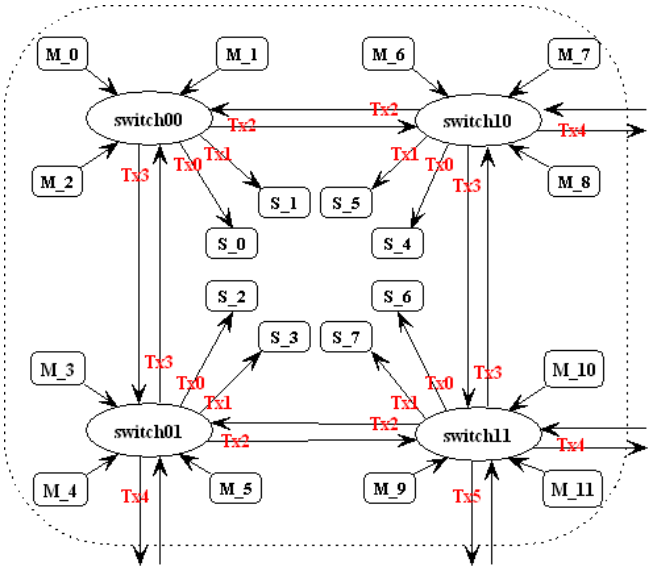


Figure 7.4 Tx output of FPGA00

A generic route-table is generated for each switch. Each output port is allocated with one or more packet address segments. For example, the switch00 in the Figure 7.4 can be configured as:address 0 to tx0; address 1to tx1; address [4,7], [16,31] to tx2; address [2,3], [8,15] to tx3. And the output ports to the top and left of switch00 are not used in 2x2 LSM.

In a generic rout-table, each Tx output port, except the port with the highest index, is allocated with one or more packet address segments. Therefore, when one packet enter the switch, the destination address is compared to all the segments, and in case of a match, the allocated Tx port is selected, from which the packet will be sent out. When no segment matches the destination address, the port with the highest index is selected. In the case where

segments form several ports match simultaneously the destination address, the rout-table selects the port with the lowest index among the candidates. A segment requires two parameters: *begin* is the first address of the segment; *end* is the last address of the segment. When the parameter *end* is set to *None*, the segment has a signal address that is defined by the parameter *begin*.

The route-table of the 4 switches implemented on the top left FPGA of the 2x2 mesh multi-FPGA platform is shown in Table 7.1.

Table 7.1 Generic Rout-Table of FPGA00

Switch	Tx	Segment	Begin	End
00	0	0	0	None
	1	0	1	None
	2	0	4	7
		1	16	31
01	0	0	2	None
	1	0	3	None
	2	0	4	7
		1	16	31
	3	0	0	1
10	0	0	4	None
	1	0	5	None
	2	0	0	3
	3	0	6	15
11	0	0	6	None
	1	0	7	None
	2	0	0	3
	3	0	4	5
	4	0	16	31

For example, according to the rout-table of switch00, packets of destination address 0 are sent to Sram0 form Tx0 port and all the packets of destination address from 4 to 7 and from 16 to 31 are all sent out from Tx2 port.

7.3 Automatic EDA Support

7.3.1 SSM IP Reuse and Automatic Composition

In order to achieve fast design productivity for this target architecture we need: (1) to raise the level IP design and reuse to Small Scale Multiprocessor (SSM IP) and automatically duplicate and adjust NOC characteristics to reach the desired size (2) fully integrate all EDA

tools involved in the design (3) due to its large size and prohibitive simulation time at RTL level, we need emulation for the validation, test and performance evaluation of this multiprocessor architecture. In addition emulation requires synthesis place and route which provides accurate area and maximum operating frequencies data. Our methodology will exploit the concept of FPGA IP which is the maximum size modular IP which can fit in a single FPGA device of the emulation platform and which can be duplicated. This requires a prior analysis of the emulation platform and the FPGA devices used in it.

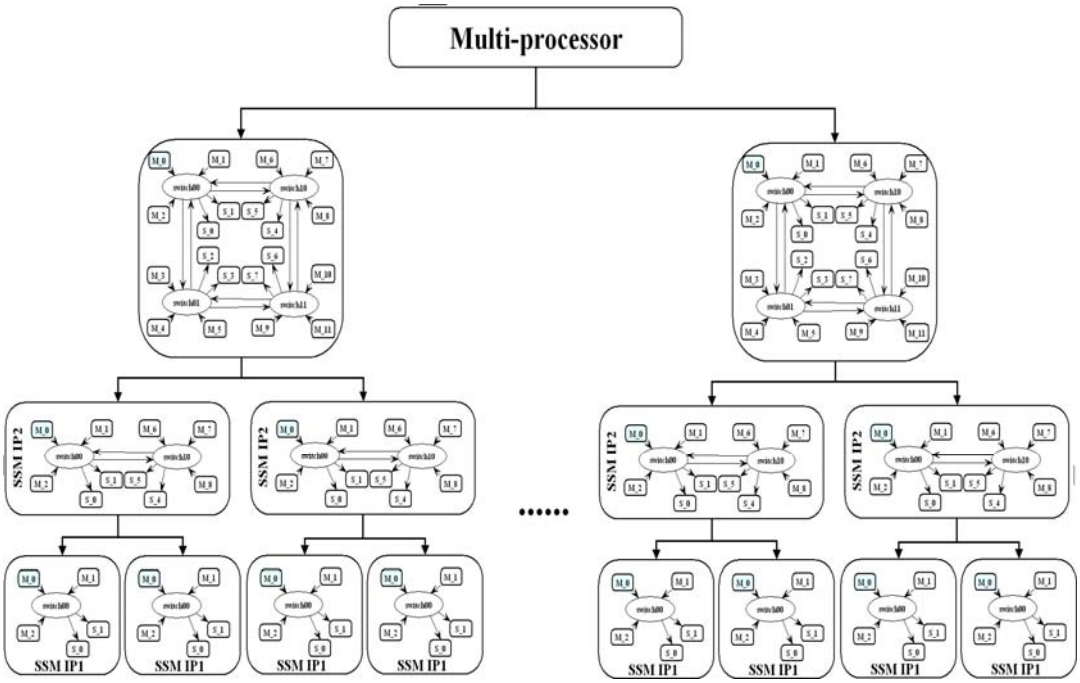


Figure 7.5 Small Scale Multiprocessor IP Reuse and Automatic Composition

7.3.2 Eve Zebu-UF Platform

The ZeBu-UF4 [39] emulator platform is based on 4 Xilinx Virtex-4 LX200 [35] devices placed on an extended PCI card via a motherboard-daughter card approach.

Table 7.2 EVE Zebu-uf4 platform details

Modules	**Descriptions**
FPGA	4 Virtex-4 LX200
DRAM	512 MBytes
SSRAM	64 MBytes
ICE	Smart and Direct

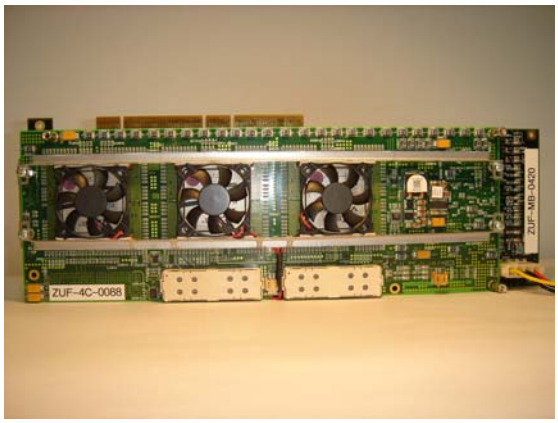


Figure 7.6 Eve Zebu-UF4 Platform.

The 4 FPGA based system can emulate the equivalent of up to 6 million ASIC gates in a single system. ZeBu-UF4 also includes on-board memory capacity based of 64 MBytes of SSRAM and 512 MBytes of DRAM memory chips via an additional memory board, which plugs into the PCI motherboard. The ZeBu-UF4 emulation system can be used in various ways such as co-emulation with commercial HDL simulator, co-emulation with both transaction level and signal-level SystemC and with synthesizable test bench. Performance ranges for these various uses are given in Table 7.3.

Table 7.3 EVE Zebu-uf4 operating mode and performance

Operating Mode	Performance Range
Max capacity in ASIC gates	6M
Co-emulation with commercial HDL simulator	5K-100KHz
Co-emulation with signal-level C/C++/SystemC	100K-500KHz
Co-emulation with transaction-level C/C++/SystemC/SystemVerilog	500K-20MHz
Test vectors	100K-500KHz
Emulation with synthesizable test bench	<=20MHz
In-circuit emulation, connected to target system	<=20MHz
Emulation with SW debuggers via JTAG interface	<=20MHz

This main approach requires EDA tools combination and integration. We first introduce Eve Zebu design flow.

7.3.3 Eve Zebu Design Flow

The Design Under Test (DUT) is mapped onto one or several FPGAs and memory chips. The mapping is carried out through any one of the most popular commercial ASIC/FPGA RTL synthesis tools plus the ZeBu software compilation package to deal with the DUT gate-level clustering, and clock and memory modeling. The Zebu design flow is given in figure 6. All the system EDIF files generated by synthesis are used by Zebu compiler for the implementation on FPGAs. The compilation is incremental but the Xilinx ISE P&R phase can be parallelized to reduce the turnaround time.

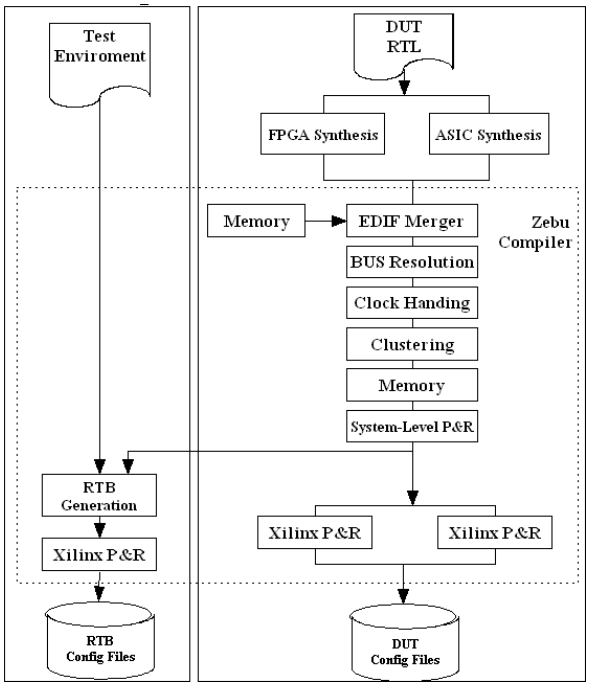


Figure 7.7 ZeBu Compilation Flow Overview.

Once the design and the test environment have been mapped, ZeBu provides a comprehensive, efficient and high-performance hardware or software test environment for the emulated DUT.

7.3.4 EDA Tools Integration and workflow

Design automation tools of 3 commercial companies are combined together to generate our multi-FPGA MPSoC. Figure 7.8 describes the workflow. The Xilinx EDK [36] tool is used to generate our SSM multiprocessor using Xilinx IPs. Once the RTL files of SSM are generated, they are reused for the multi-FPGA large scale multiprocessor synthesis, which can largely reduce system design time. Different NoC files are synthesized for each SSM on different

FPGA chips of Zebu platform by changing the generic route-table according to the XY routing algorithm and the SRAM addresses on each FPGA. These NoC RTL files are generated by Arteris NoCcompiler tool [33], which allows the export of NoC using the Arteris Danube Library [34]. Eve Zebu compiler [39] takes the EDIF files converted by Xilinx synthesis tools for the implementation. Different SSM IPs are analyzed and distributed onto FPGAs. User constraints can be used to accelerate this incremental process. Finally Xilinx place and rout tools are used to generate the download bit files of FPGA. This phase can be parallelized to reduce the turnaround time. Area and performance results are obtained.

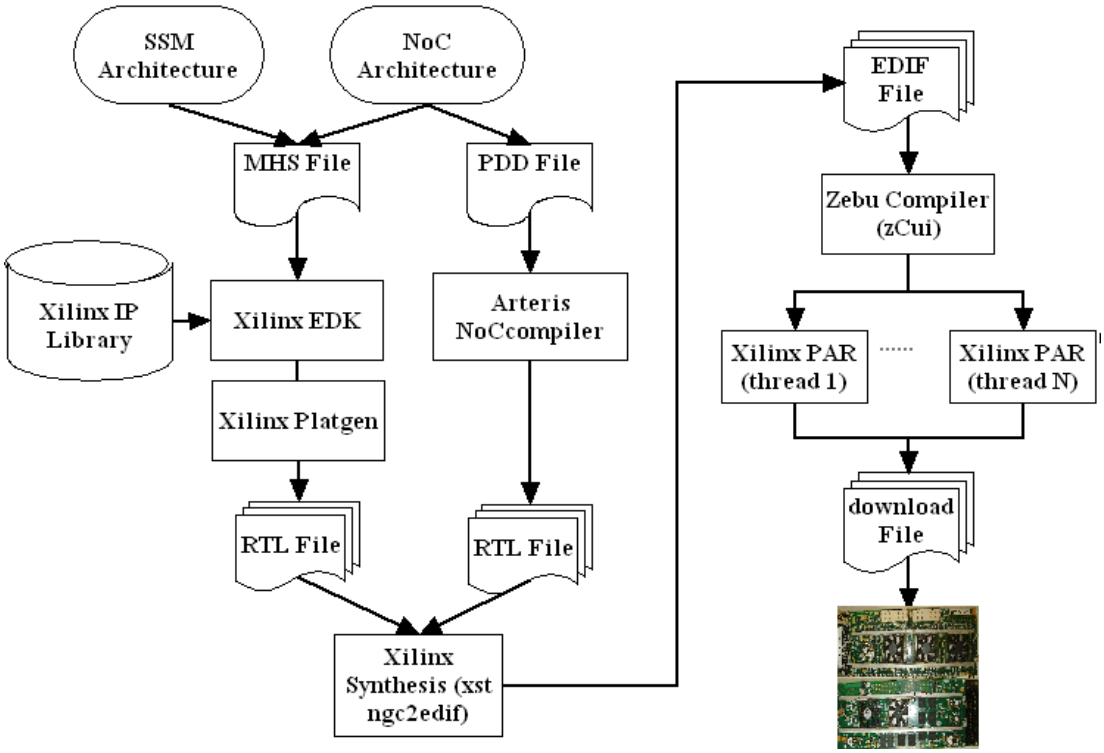


Figure 7.8 Workflow of Multi-FPGA MPSoC

7.4 Performance Evaluation and Comparison on synchronization

As the scale of our multi-FPGA multiprocessor gets large, the processor synchronization becomes complicated. OCP [31] blocking synchronization can easily realized based on our architecture and network on chip, without adding any extra resource. The blocking synchronization (read-modify-write) used by most processors is implemented by the OCP Read-Exclusive master command, and the Write or Write-Non-Posted slave commands. The Read-Exclusive command sets a lock on a memory location, and the corresponding Write or

Write-Non-Posted command to that memory location releases the lock. The Arteris NoC fully supports the OCP protocol, including the read-modify-write synchronization. The OCP Network Interface Unit (NIU) of NoC can decode the Read-Exclusive command and transfers it through the network to the target NIU, and then this slave NIU is blocked until the corresponding write command release the lock. Barrier synchronization performance is tested using our multi-FPGA platform.

Barrier synchronization is a common synchronization operation in programs with parallel loops. It allows the multiprocessors to wait until a certain number of processors have reached a 'barrier'. When the barrier number is enough, all the waiting processors can continue. Each processor tries to read the synchronization variable '*count*' by the OCP command Read Exclusive. When it successes, the variable is locked until the processor updates it by adding 1 and write it back. Then the processor will check the condition whether variable 'count' is equal to the number of processors. When the barrier is reached, the total execution time is record by one of the processors.

There are 32 SRAM in our multi-FPGA platform. The choice of SRAM where the synchronization variable 'count' is stored will impact the performance as shown in figure 10. Required number of cycles for synchronization is provided for both processors 0 and 36 to illustrate the impact of synchronization RAM. The SRAM located in the middle of the large scale MPSOC 4x4 mesh network is preferred: like SRAM6 connected to switch11 on FPGA00 shown in the architecture figure.

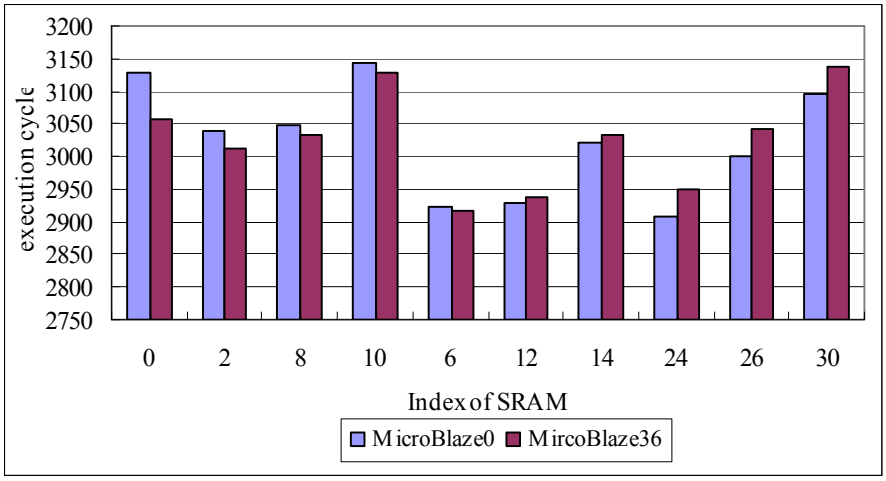


Figure 7.9 Execution cycle of synchronization using different pilot and SRAM.

The execution time of processor is different to each other according to the processor's position in the communication network. So the positions of synchronization variable and processors will impact the performance, which should be taken into account when programming the system. In order to evaluate the architecture effect we designed 4 variations of SSM IP described in the following figure where the switch uses pipelining or not and MicroBlaze have multiplier and FPU or not.

Table 7.4 Different Versions of SSM IP Architecture

	NOC	MicroBlaze
Arch. V1	Fwdpipe	Multiplier
Arch. V2	Fwdpipe+Bwdpipe+Pipe	Multiplier

The synchronization performance of two architectures: Arch. V1 and Arch. V2 are compared. The execution time of MicroBlaze36 is showed in the figure 11. As the timing of Arch. V2 is improved by 3 stages of pipelines, the real execution of Arch. V2 is much faster than Arch. V1, although it takes more number of cycles.

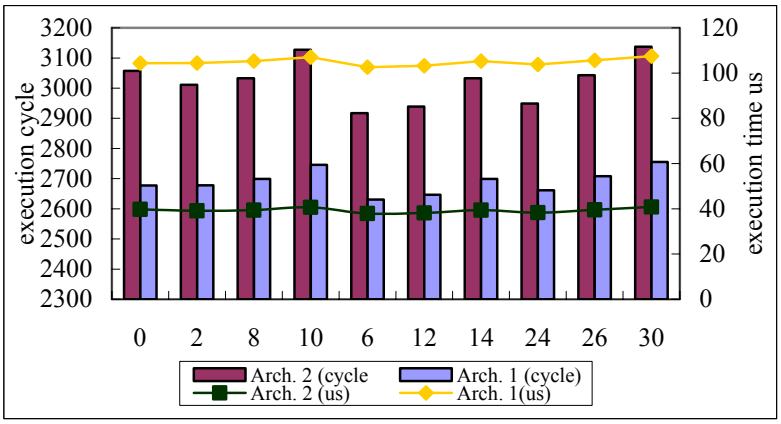


Figure 7.10 Execution time of synchronization with different architecture.

In the first test, all 48 processors use just one global variable for synchronization. Another block-by-block method is used to reduce the execution time. Four synchronization variables are used instead of just one global. Block-by-block means processors on the same FPGA block synchronize using one local variable which is stored on one SRAM on the same FPGA block. After the local synchronization, processors will check the other 3 variable to find out whether all the synchronizations are done. According to the above results, the SRAMs in the middle of whole 4x4 network are chosen to store the four variables. One example is shown in the following figure.

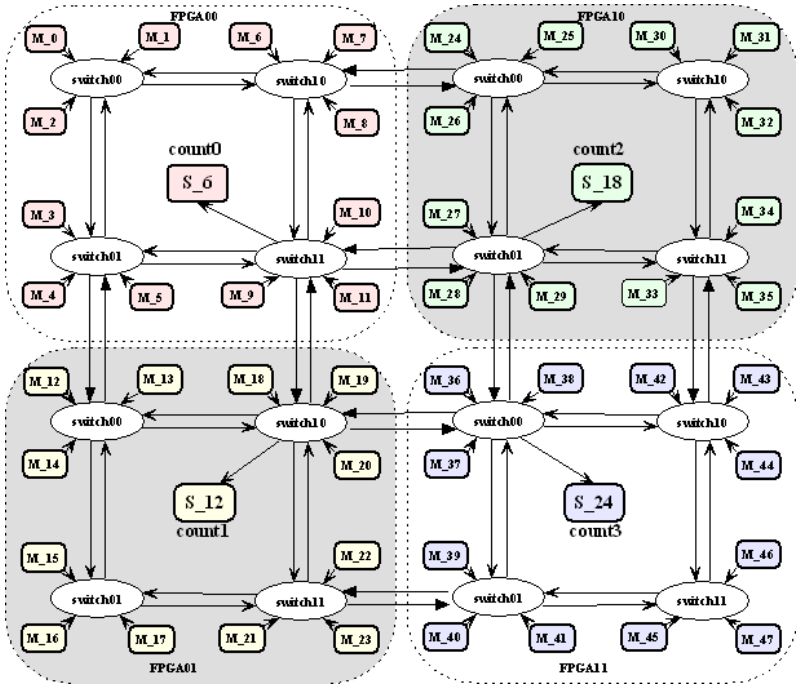


Figure 7.11 **Block-by-block synchronization.**

Table 7.5 Synchronization Time of MicroBlaze36 using different methods

Synchronization method	Execution cycle
Global variable	2917
Block-by-block	856

The block-by-block method takes full advantage of the network and can dramatically reduce the total synchronization time. This again tells the programmers should consider the architecture of multiprocessor. So the block-by-block method is more complex than the global one.

7.5 Automatic Exploration of Pipelined Data Parallel Applications

The existing data parallelizing algorithm for multiprocessor are insufficient for this novel MPSoC with NoC, in which the bandwidth is architecture restricted and communication latency must be considered. We describe in this section an automatic data parallel and pipeline exploration flow based on Fork-Join parallelism model, which is well suited for signal processing application on MPSoC. This large scale exploration is enabled by direct FPGA platform emulation, which is till more fast and data precise than traditional high level of abstraction SystemC like simulation. We selected the block cipher TDES (Triple Data

Encryption Standard) cryptographic algorithm on the 48 PE single chip distributed memory multiprocessor with NoC as an application example of the flow. Our experimental results show significant productivity gains and performance improvement of our exploration flow.

7.5.1 Related work on data parallelization

Multiprocessor mapping and scheduling algorithms have been extensively studied over the past few decades and have been tackled from different perspectives. Bokhari [40, 44] have addressed partitioning problems in parallel, pipeline and distributed computing. The problem of optimally assigning the modules of a parallel program over the processors of a multiple-computer system is addressed. A sum-bottleneck path algorithm is developed that permits the efficient solution of many variants of this problem under some constraints on the structure of the partitions. The problem of optimally partitioning the modules of chain- or tree-like tasks over chain-structured or host-satellite multiple computer systems is treated. Prior research has resulted in a succession of faster exact and approximate algorithms for these problems. Several polynomial exact and approximate algorithms for this class are proposed. King and al [41-42] have addressed the modeling and design of pipelined data parallel algorithms. A decision-free timed Petri net tool have been used for this purpose. Several other mapping proposals [43-46] have been made with variations on constraints and hypothesis. A multi-objective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design have been proposed [47]. Sesame is a software framework that aims at developing a modeling and simulation environment for the efficient design space exploration of heterogeneous embedded systems. Since Sesame recognizes separate application and architecture models within a single system simulation, it needs an explicit mapping step to relate these models for co-simulation. The design tradeoffs during the mapping stage, namely, the processing time, power consumption, and architecture cost, are captured by a multi-objective nonlinear mixed integer program. Multi-objective evolutionary algorithms (MOEAs) are introduced on solving large instances of the mapping problem. In [49] a top-down system level design flow has been proposed which allows code and data structure partitioning for MPSOC. The evaluation has been conducted on limited size architecture. Recently [51] a mapping framework based on packing for design space exploration of heterogeneous MPSoCs have been proposed. The framework is based on static,

analytical, bottom-up temporal and spatial mapping of applications based on packing. The problem is formulated as mixed integer linear program (MILP) problem. The idea of the framework is to explore and reduce the design space to find promising candidates that can be simulated more detailed. An energy/delay exploration of a distributed shared memory architecture for NoC based MPSoC is proposed in [52]. The work focuses on data allocation on the distributed shared memory space, and the exploitation of the HwMMU (hardware memory management unit) primitives which dynamically managed the shared data. Power model is estimated by gate-level simulation. Exploration is based on cycle-accurate level simulation and the scale of MPSoC is limited at 8 PEs. A customized version of the Ptolemy II framework for MPSoC exploration is presented in [56]. Multiprocessor based on different size of RENATO NoC is evaluated using actor-oriented model of HERMES NoC. Application is mapped onto different configurations of the RENATO platform. Communication latency results from simulation give designer an early estimation of the communication costs. All previous works are either based on theoretical framework or on simulation. Both approaches are not suitable for mid to large scale multiprocessors due to: (1) simulation based approaches suffer from prohibitive simulation time (2) static based approaches makes simplifying assumptions which ignore the communication behavior complexities of actual execution on NOC based mid to large scale multiprocessors.

A two-phase solution for a real-time film grain noise reduction application is proposed based on FPGA platform [56]. The application is mapped to MORPHEUS architecture using NoC inter-chip communication approach. FPGA platform is used as a reference design at the first step. Then a novel heterogeneous reconfigurable computing platform that offers flexibility is used for an efficient mapping. Platform-based software design flow for heterogeneous MPSoC is proposed and validated on FPGA platform [57]. Shapes MPSoC architecture which is a multi-tile architecture based on a Diopsis tile is used as the design and validation platform. The combination of the platform with the software code produces an executable model that emulates the execution of the final system, including hardware and software architecture. MOCDEX [53, 54] is a multi-objective design space exploration for multiprocessor on chip based on FPGA platform emulation. Pareto solutions are found by exploration of processor micro-architecture configuration and the size of FIFO. The hardware configuration exploration is still limited on 4-core multiprocessor. NOCDEX [55] is multi-

objective design space exploration of NoC based on multi-FPGA platform emulation. Different configurations of routers in NoC are explored. The latency of packets both on cycle and real time are reported with other metrics like FPGA Slices and frequency. All these works based on FPGA emulation are still limited by the scale of multiprocessor and they focus on hardware configuration of processor and NoC.

Our proposed automatic mapping exploration approach for pipelined data parallel applications differs from all other approaches as it is based on: (1) multi-FPGA emulation for accurate and fast performance evaluation on a large scale multiprocessor up to 48 cores (2) accurate hardware NOC monitoring for accurate feedback for parallel program tuning. To the best of our knowledge, this is the first exploration automatic exploration of pipelined data parallel applications mappings on network on chip based distributed memory embedded multiprocessor on Multi-FPGA.

7.5.2 Application parallel implementation model and target

A typical signal processing application can be divided into several function blocks and parallelized by mapping function blocks onto different computing elements, and these computing elements can work in pipeline for further parallelization. So signal application is a good candidate for data and function block parallelization.

7.5.2.1 Fork-Join model of parallelization

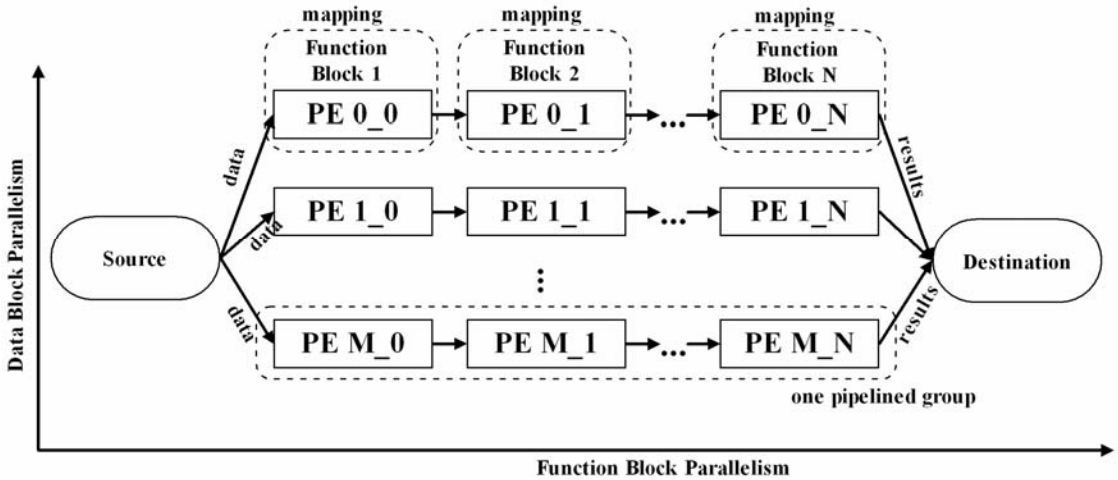


Figure 7.12 Fork-Join Model of data and function block parallelism

Two parallelisms, data and function block parallelism, are studied and combined together to achieve a best performance. The two parallelisms are combined together to work as a Fork-Join model showed in Figure 7.12. Data parallelism means that we distribute different data blocks to different processing groups. These groups process their received data in parallel. In function block parallelism, all the application functions are divided into blocks and mapped onto PEs (Processor Element) sequentially of each data parallel group. Each PE gets input data, calculates its mapped function block and sends the results to the following PE processor and finally to the destination memory. In this way, the PE processors mapped with function blocks work together as pipelined group.

Assuming there are M*N PEs available in the target embedded MPSoC (Multiprocessor System on Chip), application data can be divided into blocks for M groups of PEs and in each group, the application function can be divided into blocks for N PEs. Not all the PEs in MPSoC must be used for the application.

7.5.2.2 Parallel implementation on MPSoC with NoC

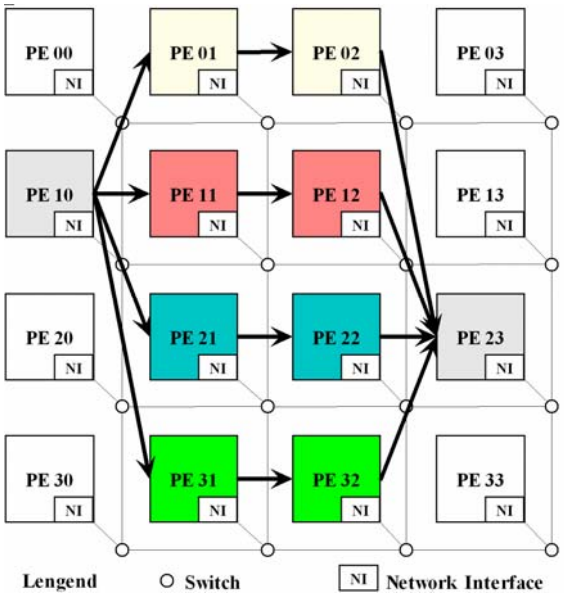


Figure 7.13 MPSoC with NoC and Fork-Join Model Implementation Example

MPSoC with NoC is the trend of future large scale multiprocessor, because of the flexibility of NoC and commercial chips like Tilera TILE64 have widely use mesh topology NoC architecture for the manufacturability reason. To parallelize the application, several PEs in MPSoC is picked up and divided into separate groups for data block parallelization. To

minimize communication latencies, PEs in the same group should be as close as possible. The simplified MPSoC architecture and one implementation of our Fork-Join model of parallelization are illustrated in Figure 7.13. In this example, PE10 works as source and sends separate data blocks to 4 different groups of PEs distinguished by different colours in the figure 2. The application is divided into 2 function blocks and mapped onto the 2 PEs on each group. Finally each group of pipelined PEs sends the results back to PE23 serves as destination. There are totally 4*2=8 PEs used in this implementation example.

7.5.3 Exploration flow

The data parallel and pipeline exploration flow is described in figure 3. In the exploration, we change the number 'M' of data blocks for data parallelization and the number 'N' of PEs in pipelined group for function block parallelization. For each number of data and function block parallelization, the mapping of pipelined groups is explored in the inner iteration. The flow and all the steps of exploration have been fully implemented in Linux RHEL 5.0 environment through scripts invoking the whole set of EDA tools involved in the generation process. It is obviously easy to port the flow in other operating system environment. The inputs of the flow are: (1) the total number of PEs used for parallelization, fixed as a constant ***MAX*** and (2) the C source codes of original signal processing application. All performance evaluation results will be achieved through actual execution of the applications on a multi-FPGA platform based multiprocessor we fully designed and implemented.

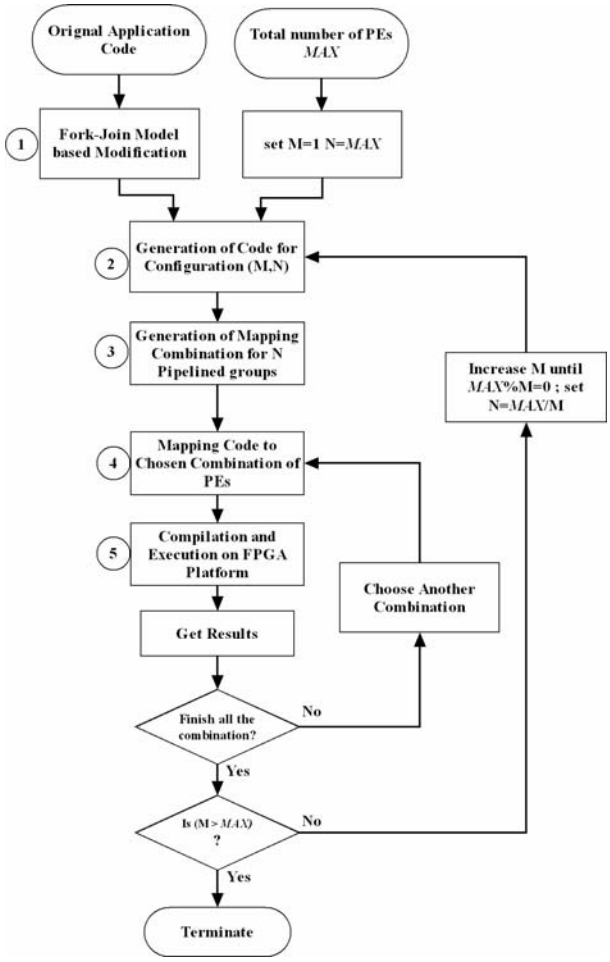


Figure 7.14 Data parallel and pipeline exploration flow

In step 1, the original application must be modified based on Fork-Join model for the next step: automatic generation of C codes in the workflow. The modification depends on the nature of application. If the processing part of application is a repetition of some dedicated function macros, which is the most common case, the repeated part is extracted and written into a C function with a 'for' or 'while' boucle of K. Assuming there are K*N repeated macros in the application, they are divided into N function blocks, each block for one PE in the pipelined group. Each PE get the data from the proceeding PE or from the source; repeat K times of macros; and send the processed data to the next PE or to the destination. Here we use the simplest way, the function is written like:

```
#define K K_Value
Void Function () {
        Get data from predecessor PE or resource;
```

```
        For(int i=0; i<K; i++)
            Repeated Macros;
        Send data to successor PE or destination;
    }
```

In step 2, codes are automatically generated. For each function block parallelization of 'N', we replace the 'K_Value' with 'N'. If the beginning or ending part of application can not be repeated, these parts are written into separate C functions and are mapped to dedicate PEs, which serve as source and destination in the Fork-Join model.

In the case where the application has no repeated macros, the application is divided to a fixed number of function blocks. The number of block function is fixed a constant but the exploration of data parallelization and mapping of pipelined groups remain. So the principle of our exploration methodology does not change.

In step 3, the mapping combination of 'N' pipelined groups onto MPSoC is generated. To minimize communication latencies, PEs in the same pipelined group should be as close as possible. Applying this principle on mesh topology NoC based MPSoC, we put the PEs connected on the same switch or on the same line of mesh together into one pipelined group. And the direction of this sequential pipeline must be from source to destination. This principle of same line is illustrated in Figure 7.13

In step 4, the codes are mapped to the chosen PEs for compilation. Each PE has a unique identity in our multiprocessor, which is generated by EVE compilation tool zCui [38, 39, 40]. A mapping configuration file is generated designating the corresponding between dedicate compiled code file and PE's identity. This mapping configuration file is used to download executables file into PEs' local instruction memories during the FPGA download step.

In step 5, the mapped codes for PEs are compiled using Xilinx EDK tools [37]. The synthesized bit files for FPGA with compiled executable file are downloaded with mapping configuration file and executed on the FPGA platform to get results. The details of synthesis and execution flow can be found in our previous work [39, 40]. The fast emulation on FPGA platform gives us the possibility to explore all the possible mapping and choose the best mapping results from them.

After all the combination of mapping is explored, the number of data parallelisation 'M' is increased to another value, which makes N=***MAX***/M is an integer. New codes are generated,

compiled and executed. The exploration stops when the number of data parallelization 'M' is greater than total number of PEs ***MAX***.

Our proposed exploration flow is general and can be applied onto different FPGA emulation platform and MPSoC design. We use Eve Zebu-UF4 multi-FPGA platform and our 48-core multiprocessor, which are presented in the next section as an example.

7.5.4 TDES algorithm

TDES algorithm was introduced as a security enhancement of the aging DES, a complete description of the TDES and DES algorithms can be found in [19-21].

7.5.4.1 The algorithm

The TDES is a symmetric block cipher that has a block size of 64 bit and can accept several key size (112, 168 bits) which are eventually extended into a 168 bit size key, the algorithm is achieved by pipelining three DES algorithms while providing 3 different 56 bits keys (also called key bundle) one key per DES stage. The DES is a block cipher with 64bit block size and 54 bit key.

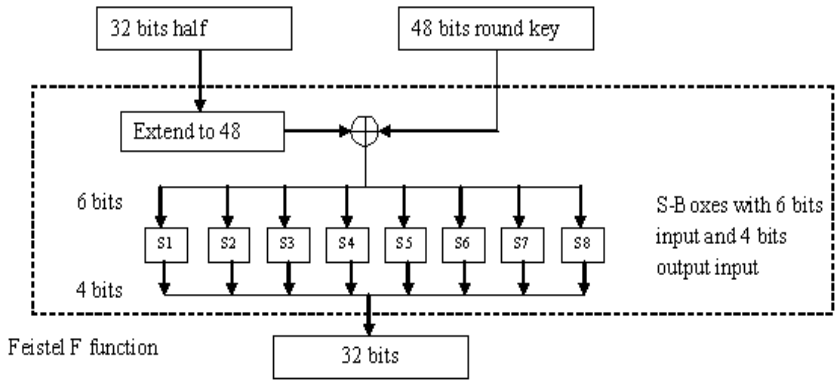


Figure 7.15 Feistel function F (SBoxes)

The TDES starts by dividing the data block into two 32 bits blocks which are passed into a Forward Permutation (FP) then criss-crossed in what is known as Feistel scheme (or Feistel Network) while being passed into a cipher Feistel function F, as illustrated in Figure 7.16, this operation is repeated for 48 rounds followed by one IP (Inverse Permutation). The F function expands the 32 bit half block input into 48 bits that is mixed with a round key that is 48 bit

wide, the result is divided into 8 blocks 6 bits each which in turn are passed into 8 S-Box (Substitution Box) that returns only 4 bits each making an output of 32 bits. round keys are calculated for each round by a expanding the 56 bit key through a specific key scheduling process, then dividing the result into 48 bit keys. Figure 7.15 shows the Feistel F function.

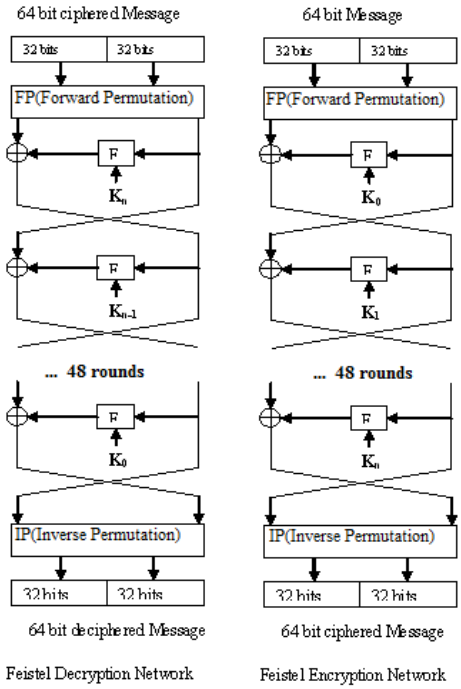


Figure 7.16 TDES encryption and Decryption schemes (Feistel Network)

The TDES algorithm clearly process data on a pipelined mode in both the encryption and the decryption mode.

7.5.4.2 Operation mode

Block cipher algorithms have different operation modes; the simplest is called ECB (Electronic Code Book) in this mode the block cipher is used directly as illustrated in Figure 7.17.

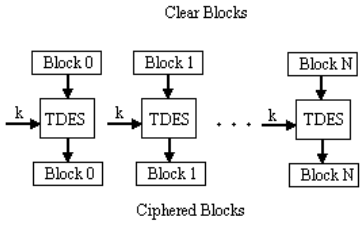


Figure 7.17 ECB operation mode for the TDES block cipher

The problem with ECB is that encrypting the same block with the same key gives an identical output, as counter measure to reveling any information, blocks are chained together using different modes like CFB (Cipher Feed Back), OFB (Output Feed Back) and CBC (Cipher Block Chaining) illustrated in Figure 7.18.

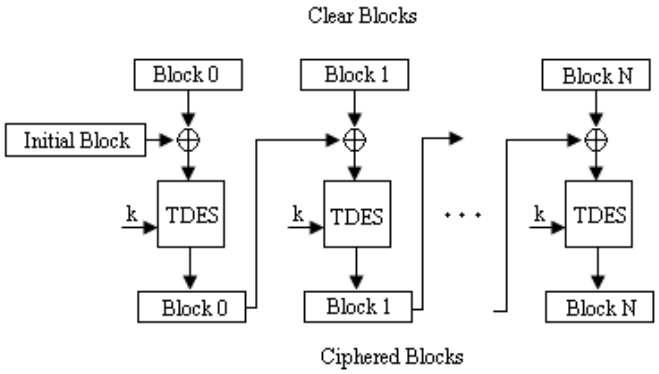


Figure 7.18 CBC operation mode for the TDES block cipher

The TDES represents a data parallel and pipelined signal processing application. So far TDES have been implemented as a hardware IP core, and shared memory parallel software implementation [12] but to the best of our knowledge, no parallel software implementation on distributed memory embedded multiprocessor has been reported.

7.5.5 Parallel Implementation of algorithm

We base our work on the C implementation from NIST [21] (National Institute of Standards and Technology), the sequential TDES encryption C code consists from a Forward Permutation (FP), a 48 calls to an F macro that executes both F boxes and the Feistel network, and finally there is an Inverse Permutation (IP), the C Code is a LUT (Look Up Table) based implementation with combined SPBox tables meaning all arithmetic operations are pre-calculated and stored in tables.

To fully use our 48-PE multiprocessor, two parallelisms, data block and function block parallelism, are studied and combined together to achieve a best performance. An example is given in Figure 7.19:

- 24 MicroBlaze PEs are chosen for implementation;

- all data are divided into 4 blocks and the 24 MicroBlaze PEs are divided into 4 groups correspondingly ;
- to encrypt each data block, each pipelined group has 24/4=6 MicroBlaze PEs;
- in each pipelined group, each MicroBlaze PE will calculate 48/6=8 F macro calls.

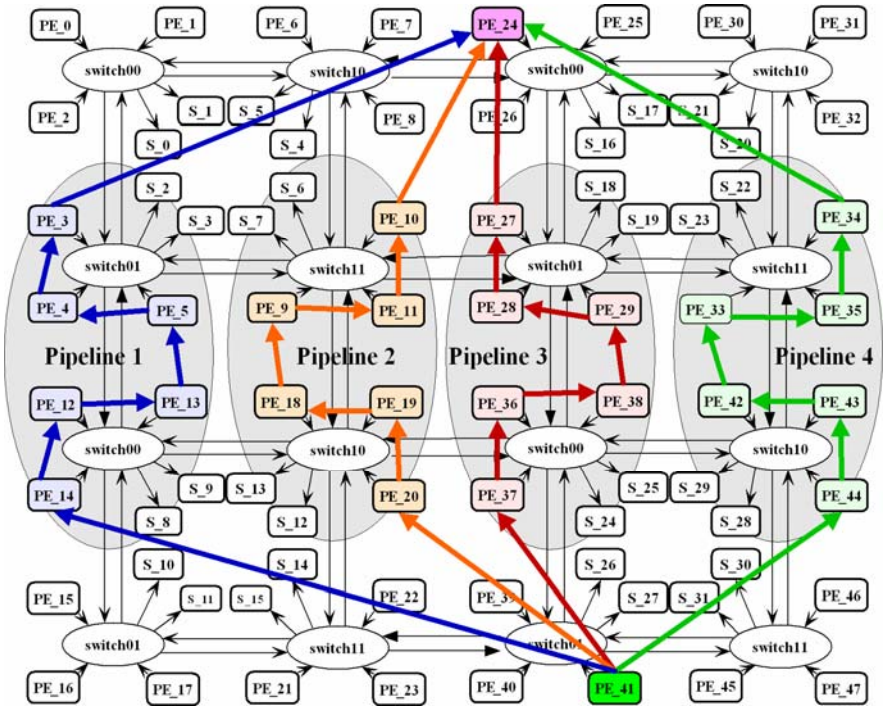


Figure 7.19 Fork-Join Model mapped to 48-PE multiprocessor example

The two parallelisms are combined together to work as a Fork-Join model showed in figure 11. In order to measure the exact execution time and verify the result's validity, two MicroBlaze processors are reserved as the source and destination; in a real case, the source and destination are the memory blocks, where original and worked data are stocked. The source MicroBlaze processor generates the original data, does the FP function and sends one-by-one to the first MicroBlaze processors of different pipelined groups. Each pipelined group of MicroBlaze processor calculates their received data in parallel and finally sends the results to the destination. The destination MicroBlaze processor gets the data from the last MicroBlaze processors of pipelined groups, does the IP function and stocks them to the memory. As described in the architecture section, our target architecture is a 48-PE multiprocessor organized as a 4x4 mesh of clusters fully interconnected through a network-on-chip (NOC). Each cluster includes 3 local MicroBlaze processors and 2 local memory blocks connected to a NOC switch. One local memory block is used for data communication

between MicroBlaze processors and the other one is used for synchronization service between MicroBlaze processors in the same pipelined group. Separation of data and synchronization memory location will avoid unnecessary communication conflicts and improve system performance. MicroBlaze processors in the same cluster communicate using local memory blocks to minimize the communication time; in each memory block, addresses have been reserved for inter-cluster communication.

7.5.5.1 Single task implemented on different number of pipelined processors

The combination of data and task parallelisms on our 48-PE multi-processor is a very large exploration space. Here we give one example result. Only 24 cores in maximum among 48 PEs are used for data and task mapping. As mentioned before, MicroBlaze 42 is used as source and MicroBlaze 24 as destination. At the first step, only one pipelined group is used to map the TDES application. Different size of data is sent by the source: from 10 packets up to 400,000 packets as in Figure 7.20. The number of MicroBlaze processor in the pipelined group augments from 1 PE to 24 PEs in maximum. As the size of packets increasing, the average cycle to treat one packet of all the pipelined groups converges to a constant, which is in the fact the calculation time of one MicroBlaze processor in the pipeline (48/N * calculation time of F). For the next step of experiment, in total 96000 packets traffic is used for data and task parallelization, at most 24 pipelined group will get 4000 packets for each group. The observation of stable average cycle for one packet will ensure that the variety of packet number will not impact the result precision in the next step of experiment.

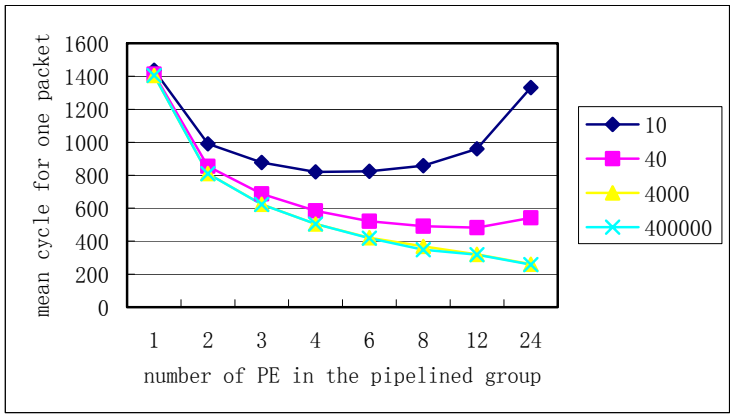


Figure 7.20 Results of one pipelined group with different size of data

Another important observation is that small size of packets treated by large number pipelined group will use more time than the one treated by small number pipelined group, for an extreme example 10 packets treated by 24-PE pipeline uses almost the same time as 10 packets treated by 1-PE pipeline. So task parallelism is suitable for large scale data application.

7.5.5.2 Data parallelization with 24 processors

At the second step, task parallelism and data parallelism are combined to find out a good trade-off between the 2 parallelisms. In this example, at most 24 PEs are used, different combination of data and task parallelism is listed in Table 7.6:

Table 7.6 combination of data and task parallelism (24 cores case)

Number of pipelined group	Number of PE in 1 pipelined group	Number of F micro mapped to 1 PE
24	1	48
12	2	24
8	3	16
6	4	12
4	6	8
3	8	6
2	12	4
1	24	2

Figure 7.21 shows the comparison between single pipelined group (only task parallelism) and multiple pipelined groups (task parallelism combined with data parallelism) both treating 96000 packets. Signal pipelined group means that only one pipeline with different number of PE is used to encrypt the 96,000 packets. Multiple pipelined groups fully use all the 24 PEs. Results show multiple pipelined group mapping is much quicker than single pipelined group mapping, which is obvious. And we can see that a large number PE (24 PEs) pipelined group is not a good choice. The results of the other 7 multiple pipelined groups are zoomed in figure 14 to find out the best trade-off between data and task parallelism.

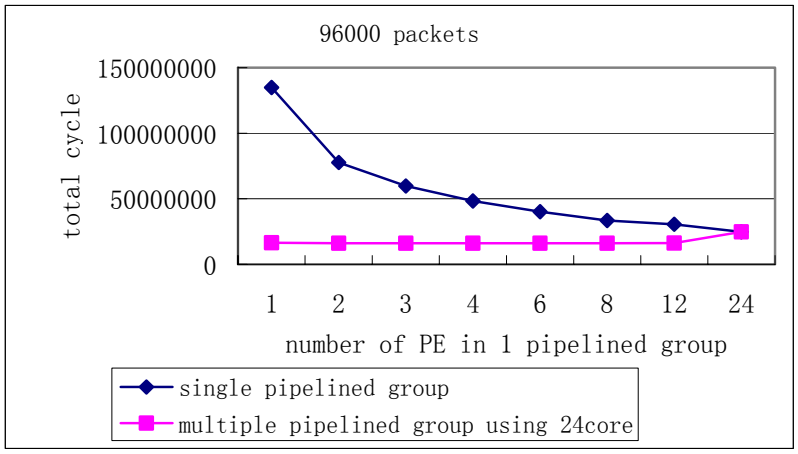


Figure 7.21 Single pipelined group vs. multiple pipelined groups

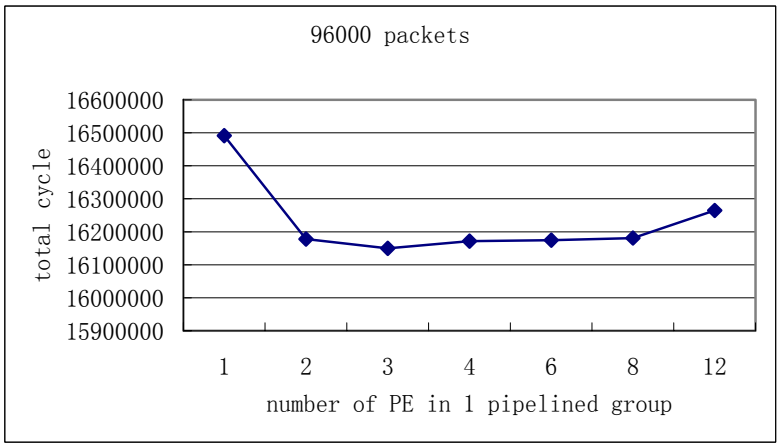


Figure 7.22 Tradeoff between task and data parallelism (24 core limited case)

In Figure 7.22, it is clear that the combination of 8 pipelined groups, each with 3 MicroBlaze processors in group, is the mapping of TDES with minimum execution time. So it's the best trade-off between data and task parallelism. Our principle of PE grouping is to make the most nearby MicroBlaze processor together: meaning that use inner-cluster MicroBlaze processor as most as possible, inter cluster MicroBlaze processor as close as possible. This principle can explain the best trade-off combination of 8 pipelined groups each with 3 MicroBlaze processors. And this result shows the great impact of system architecture on parallel application performance.

7.5.6 NoC Monitoring and Traffic monitoring results example

Network on chip traffic is monitored through hardware performance monitoring unit. The overall concept is to probe the Network Interface Unit (NIU) from initiator and target.

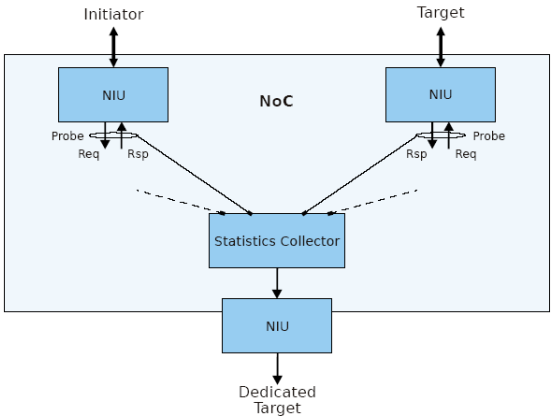


Figure 7.23 Performance monitoring concept

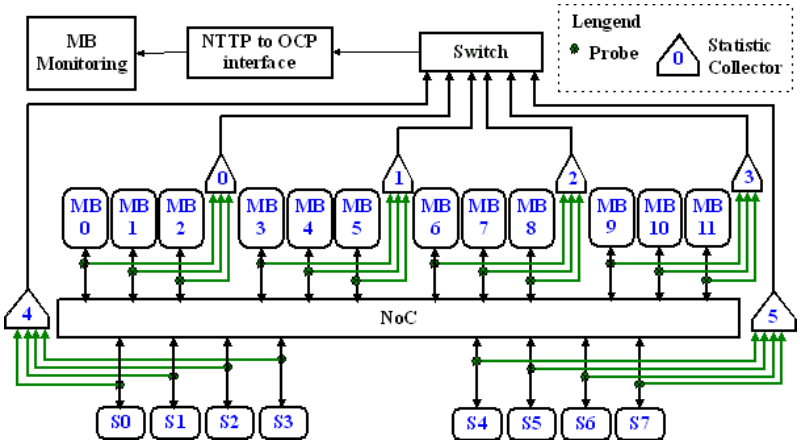


Figure 7.24 performance monitoring network on each FPGA

Statistics collector are added for each group of 3 processors and 4 on-chip SRAM and collected through a dedicated performance monitoring network on chip in figure 15. The monitoring unit and performance monitoring network on chip uses dedicated hardware resources and thus do not interfere of affect the measure. There is a trade-off between emulation frequencies and hardware monitoring area which is significant. These collectors can record the packet latency and traffic throughput of each processor and SRAM.

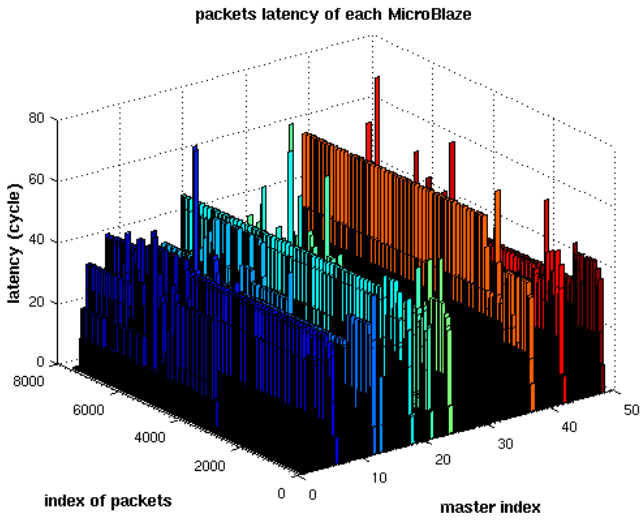


Figure 7.25 NOC monitoring results example

Hardware performance monitoring network, which is also integrated in the 48 cores multiprocessor, records the latency of each packet sent by the 24 MicroBlaze processors. As we have separated the data and synchronization communication and most nearby PE grouping principle, there are not many routing conflicts on the NoC and the packets latency does not change too much especially for the PEs in the middle of pipelined group. Most of the conflicts are found at the connection from source or destination, which is the bottleneck of system communication. With hardware performance monitoring network, we can analyze the system communication situation and find out the limits of system to easily improve performance.

Hardware performance monitoring is a good feedback tools for software programmers, as the real-time traffic distribution is now available to analyze the data throughput on the NoC and find out the traffic jam. This will help programmer to change the mapping of application onto different PEs to avoid large traffic conflicts and ameliorate the system performance. And the statistic data from the hardware monitor can be used as a fitness of our future work flow for multi-objective exploration.

7.5.7 Conclusion

High performance embedded applications based on image processing and single processing will increasingly be moved to embedded multiprocessors. Novel NoC communication architecture enforces the impacts of bandwidth and latency restricts, which is considered in existing parallel programming algorithms. Parallel programming for application on MPSoC is

getting more complex with large scale of parallel resource on chip. Traditional simulation based exploration is no longer fast enough for exploration on large scale MPSoC.

We have proposed an automatic data parallel and pipeline exploration flow based on Fork-Join parallelism model for signal processing applications on embedded multiprocessor with NoC. Our work flow based on direct multi-FPGA platform emulation is much faster for exploration and more precise for task granularity analysis. Cryptographic application TDES is implemented onto our 48 core multiprocessor and exploration flow is used to find the best parallelization solution. Exploration results show that the parallelization on embedded multiprocessor is architecture depended and our proposed exploration flow guarantees not only system performance improvement but also design productivity.

A hardware based network on chip monitoring drives the task placement process to reduce communication bottlenecks. This hardware monitoring network will be automatically integrated in to our new exploration workflow. Future work also will add automatic parallelization as well as optimization algorithm into the proposed exploration flow.

7.6 Heterogeneous design flow with HLS

Embedded system design is increasingly based on single chip multiprocessors because of the high performance and flexibility requirements. Embedded multiprocessors on FPGA provide the additional flexibility by allowing customization through addition of hardware accelerators on FPGA when parallel software implementation does not provide the expected performance. And the overall multiprocessor architecture is still kept for additional applications. This provides a transition to software only parallel implementation while avoiding pure hardware implementation. We selected the block cipher TDES (Triple Data Encryption Standard) cryptographic algorithm on a 48 PE single chip distributed memory multiprocessor with coprocessors as an application example of the flow.

7.6.1 Heterogeneous design flow with HLS

The heterogeneous design flow used for this study is described in figure 1. The inputs of the flow are the TDES application provided as an ANSI C code and a small scale multiprocessor (SSM) IP which will serve as the basic component for the parallelization process. All

performance results will be achieved through actual execution on a multi-FPGA platform. The process starts from mapping the TDES application onto the small scale multiprocessor and automatically increases the size of the multiprocessor to reach the maximum multi-FPGA emulation.

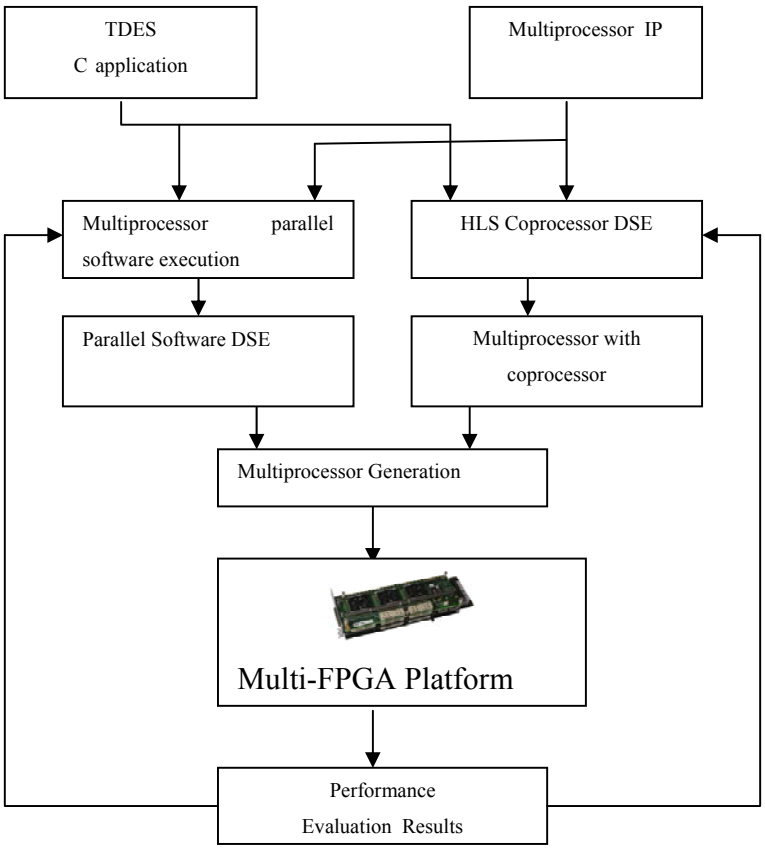


Figure 7.26 Automatic heterogeneous Design Flow with HLS

The execution on the multiprocessor will be based on a fork-join multiple-data multi-pipeline exploiting both the data parallel and the pipeline feature of the TDES algorithm. Having achieved maximum design space exploration through parallel programming the second step is to explore coprocessor based TDES parallel execution by incrementally adding TDES C-based synthesis generated coprocessor. Final step will compare both paths to select the most appropriate implementation.

Our proposed methodology for data parallel and pipeline signal processing applications combines: (1) multi-FPGA emulation for accurate and fast performance evaluation (2) automatic multiprocessor generation up to 48 processors, synthesis and execution (3) accurate

hardware NOC monitoring for accurate feedback for parallel program tuning (4) automatic synthesis and inclusion of hardware accelerators through HLS (High Level Synthesis).

7.6.2 High Level Synthesis: C-Based

Hardware accelerators are blocks of logic that are either automatically generated or manually designed to offload specific tasks from the system processor. Many math operations are performed more quickly and efficiently when implemented in hardware versus software. Adding powerful capability to FPGAs, hardware accelerators can be implemented as complex, multi-cycle coprocessors with pipelined access to any memory or peripheral in the system. They can utilize FPGA resources (such as on-chip memory and hard-macro multipliers) to implement local memory buffers and multiply-accumulate circuits. Using as many master ports as necessary, they can initiate their own read and write operations and access any I/O pin in the system. Hardware accelerators are a great way to boost performance of software code and take full advantage of the high-performance architecture of FPGAs. The design and software simulation of a hardware accelerator used a CAD tool called ImpulseC. This software allows the creation of applications intended for FPGA-based programmable hardware platforms similar to [11] and design space exploration at the HLS level [12-13].

The ImpulseC compiler translates and optimizes ImpulseC programs into appropriate lower-level representations, including Register Transfer Level (RTL) VHDL descriptions. Impulse CoDeveloper™ is an ANSI C synthesizer [37] based on the Impulse C™ language with function libraries supporting embedded processors and abstract software/hardware communication methods including streams, signals and shared memories. This allows software programmers to make use of available hardware resources for co-processing without writing low-level hardware descriptions. Software programmers can create a complete embedded system that takes advantage of the high-level features provided by an operating system while allowing the C programming of custom hardware accelerators. The ImpulseC tools automate the creation of required hardware-to-software interfaces, using available on-chip bus interconnects.

- **Concurrency model**: the main concurrency feature is pipelining. As pipelining is only available in inner loops, loop unrolling becomes the solution to obtain large pipelines. The

parallelism is automatically extracted. Explicit multi-process is also possible to manually describe the parallelism.

- **Types:** ANSI C types operators are available like int and float as well as hardware types like int2, int4, int8. The float to fixed point translation is also available.
- **Timing specification: t**he only way to control the pipeline timings is through a constraint on the size of each stage of the pipeline. The number of stages of the pipeline and thus the throughput/latency are tightly controlled.
- **Memory**: all arrays are stored either in RAM or in a set of registers according to a compilation option.

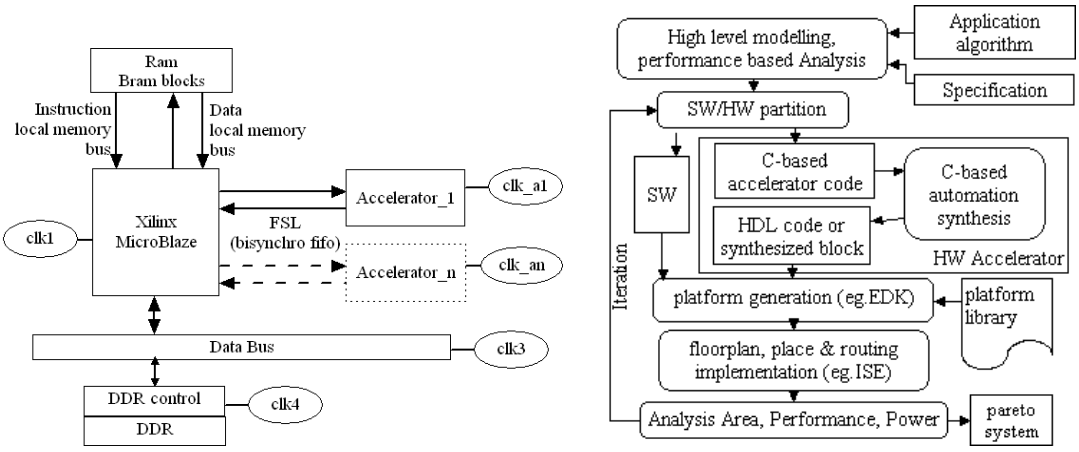


Figure 7.27 (1) Block Diagram of Accelerator Connection Forms (2) C-based HW Accelerated System Design Workflow

7.6.3 EDA Tools Combination

Design automation tools of 4 commercial companies are combined together to generate our multi-FPGA MPSoC and the parallelized execution files for emulation as described in Figure 7.28.

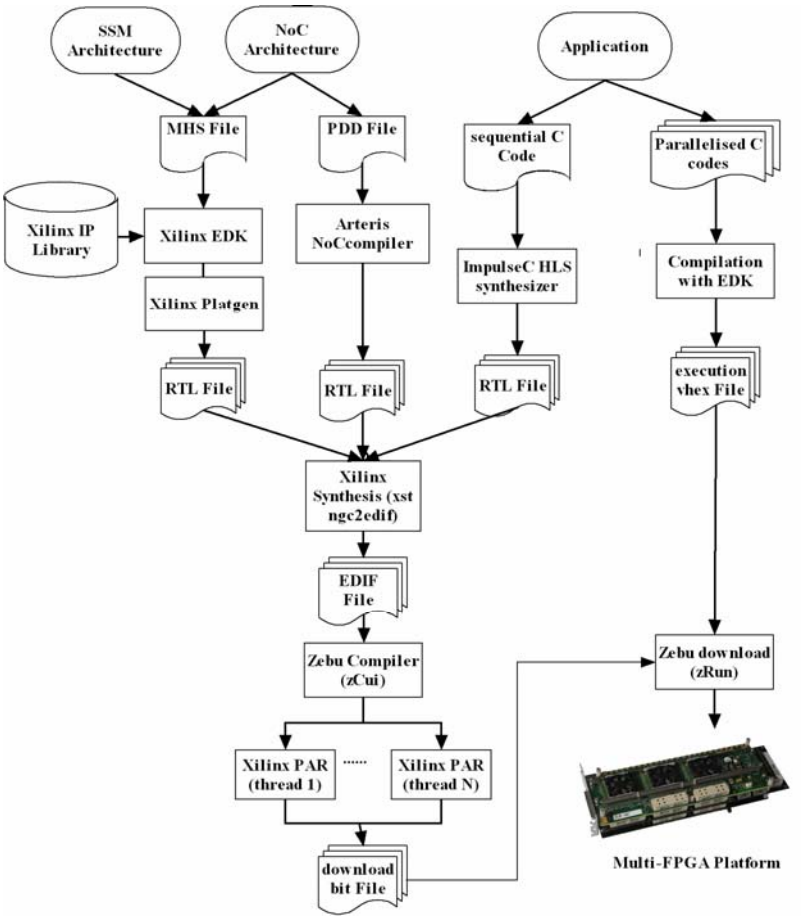


Figure 7.28 Workflow of Multi-FPGA MPSoC with HLS

The Xilinx EDK tool is used to generate our SSM (Small Scale Multiprocessor) multiprocessor, which is described in section V. Once the RTL files of SSM are generated, they are reused for the multi-FPGA large scale multiprocessor synthesis, which can largely reduce system design time. Different NoC files are synthesized for each SSM on different FPGA chips of Zebu platform by changing the generic route-table according to the XY routing algorithm and the SRAM addresses on each FPGA. These NoC RTL files are generated by Arteris NoCcompiler tool, which allows the export of NoC using the Arteris Danube Library. Sequential C code of application can be synthesized to RTL codes by ImpulseC High Level Synthesis (HLS) tools to generate coprocessor to SSM IPs. Eve Zebu compiler takes the EDIF files converted by Xilinx synthesis tools for the implementation. Different SSM IPs are analyzed and distributed onto FPGAs. User constraints can be used to accelerate this incremental process. Finally Xilinx place and rout tools are used to generate the download bit files of FPGA. This phase can be parallelized to reduce the turnaround time.

Application is programmed into parallelized C codes, compiled with Xilinx EDK tools to generate execution file of each processor. Finally system bit file is downloaded to multi-FPGA platform and application is executed.

7.6.4 Data parallelism with coprocessor

Data parallelism with coprocessor is another method to realize TDES application parallelization onto multiprocessor. Coprocessor is used to execute complex math operation, which can greatly improve system performance. In our case, each MicroBlaze processor of the 48-PE multiprocessor has a coprocessor to do the whole TDES functions; the MicroBlaze processor is only in charge of communication: they get the original data from the source, send them to coprocessor, wait until coprocessor sends back the results and finally send the results back the destination.

We use ImpulseC tools to generate our TDES coprocessor directly from our C code to VHDL source, which greatly improves our design productivity.

7.6.4.1 TDES Coprocessors generation using ImpulseC

The TDES coprocessor is designed for the Xilinx MicroBlaze processor using an FSL interface. The Triple DES IP was synthesized for a Virtex-4 platform by Xilinx ISE. Our 5-stage pipeline implementation uses 2877 slices and 12 RAM blocks with a maximum frequency of 169.75 MHz. The occupation for the same IP using LUTs instead of RAM blocks is 4183 slices. The maximum frequency for this version is 162.20 MHz. 12 instances of our IP were successfully implemented on an Alpha Data XRC-4 platform (Virtex-4 FX-140) using 12 MicroBlaze processors within a Network-on-Chip.

The chosen architecture for our Triple-DES hardware accelerator is the following 5-stage pipeline:



Figure 7.29 5 Stage pipeline TDES

This IP was synthesized for a Xilinx Virtex-4 LX-200 FPGA. RAM blocks can be saved by using LUTs, if necessary. The chart below gives us an idea of the surface and performance of our IP. Xilinx's implementation uses a fully pipelined architecture, which allows a very high

throughput. But it is impossible to reach such a throughput on a NoC. Helion's architecture uses a simple loop, which saves a lot of slices. Our IP was generated by ImpulseC whereas Helion's one was coded directly in VHDL/Verilog. This is the main reason why our IP is not as efficient.

Table 7.7 HLS based TDES IP vs optimized IPs

	Helicon	Xilinx	HLS (RAM)	HLS (LUT)
Slices	467	16181	2877	4183
Max frequency (MHz)	196	207	170	162
Throughput at 100 MHz	255.6 Mbps	6.43 Gbps	305 Mbps	305 Mbps

The execution time was measured for several input data sizes

7.6.4.2 Parallel Software vs Coprocessors

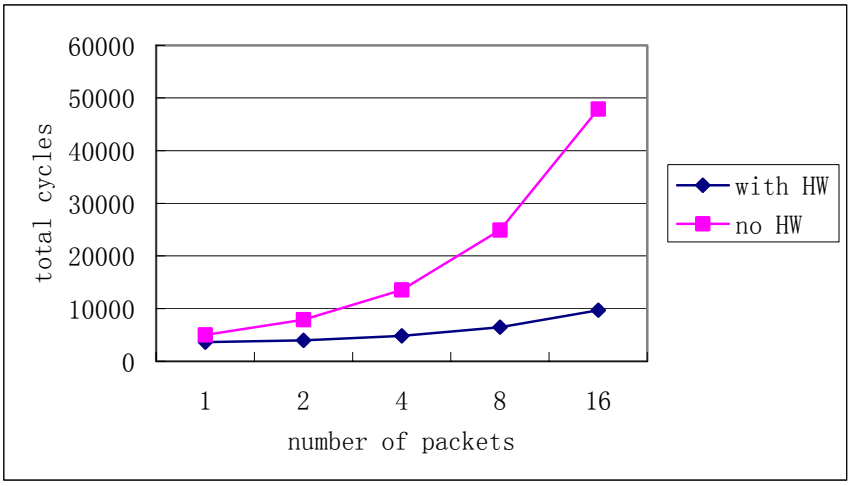


Figure 7.30 Parallel Software vs Coprocessors on a 48 PE Multiprocessor

In this case, all the 48 MicroBlaze processors do the TDES application in parallel. When they finish all the packets, they will send a finish flag to the synchronization memory. MicroBlaze 0 will verify all the PEs finish their jobs and records the execution time. Results of TDES application using coprocessor on the 48-PE multiprocessor are showed in figure 20. The results of software parallel are used to illustrate the acceleration with hardware coprocessor. It can improve system performance by 5 times.

7.7 OCP-IP Micro-benchmarks on LSM processors

7.7.1 OCP-IP Micro-benchmarks

The OCP has released a comprehensive set of synthetic workloads as micro-benchmarks for the evaluation of network on chip. Although micro-benchmarks cannot represent a real application well they are complementary to application benchmark. OCP defines two classes of communication services: best effort and guaranteed services. The best effort is connection-less, delivering packets in a best-effort fashion. It has no establishment phase, and sources send packets without the awareness of states in destinations. The guaranteed service is connection-oriented providing certain bounds in latency and/or bandwidth. A connection is a unidirectional virtual circuit setting up from a source NI to a destination NI via the network. The network reserves resources such as buffers and link bandwidth for connections. Our MPSOC network being wormhole routing based is then a best effort class of communication services.

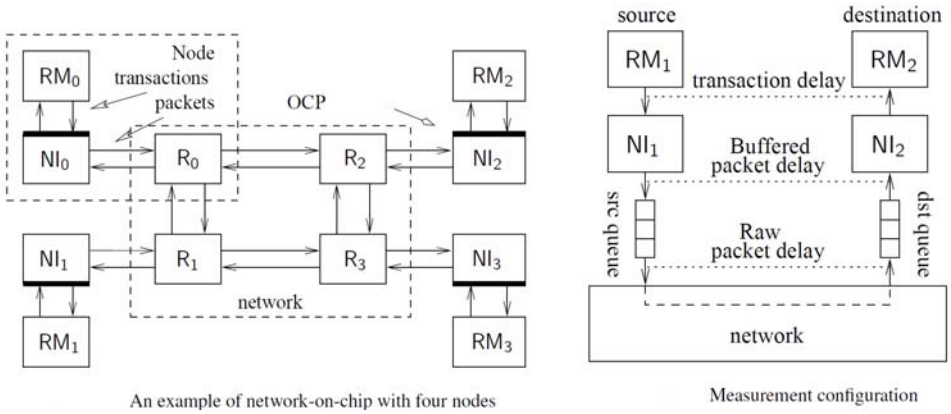


Figure 7.31 **Measurement configuration**

7.7.1.1 Temporal distribution

OCP-IP micro-benchmarks tackle temporal distribution of traffic based on the b-model [20]. This model through a bias parameter b with $0 < b \leq 0.5$ generates more or less burstiness in the traffic. A bias parameter b = 0.3 means that within a given time interval 30% of the data are generated in one half of the time interval and the remaining 70% in the other half and this continues recursively until reaching the time resolution. The case b= 0.5 means that there is

no burstiness and the emission probability is constant. The burstiness of the traffic increases as b converges to 0. OCP-IP specifies 4 types of burst traffic.

Table 7.8 Temporal Distribution

Bursty Traffic	b
type 1	0.5
type 2	0.4
type 3	0.3
type 4	0.2

7.7.1.2 Spatial distribution

OCP-IP specifies 6 spatial traffic patterns: (1) uniform (2) locality (3) bit rotation (4) N complement (5) Hot spot (6) Fork join pipeline. Each of these spatial traffic patterns are representative of a distinct spatial pattern. For example, Fork join pipeline is a pattern where a fork feeds c nodes that are the starting point of c parallel pipelines while uniform is the uniform distribution. We focused on 2 spatial patterns, locality and hot spot. Locality spatial pattern describes a traffic with spatial locality that is the probability to send a packet to a destination node is higher when the destination node is spatially closer. More precisely if we consider the distance d as the source-destination distance then $P(d) = 1/(A(D)2^d)$ where D is the maximum distance in the network and $A(D) = \Sigma_{d= 1..D} (1/2^d)$ is a normalizing factor. Within a set of nodes with the same distance, each node is selected with uniform probability.

Table 7.9 Selected Micro-benchmark

Spatial Distribution	Description
Locality	$P(d) = 1/(A(D)2^d)$
Hot Spot	N/M of the nodes hot spots $M \in \{2, 4, 8, ..., N\}$ $\rho \in \{0.5, 0.7\}$ of traffic sent to these hot spots remaining sent uniformly

The hot spot model selects N/M^2 of the nodes as hot spots $M \in \{2, 4, 8,..., N\}$. A certain fraction $\rho \in \{0.5, 0.7\}$ of traffic is sent to these hot spots while the remaining is sent uniformly to all other nodes.

7.7.1.3 Example Traffics

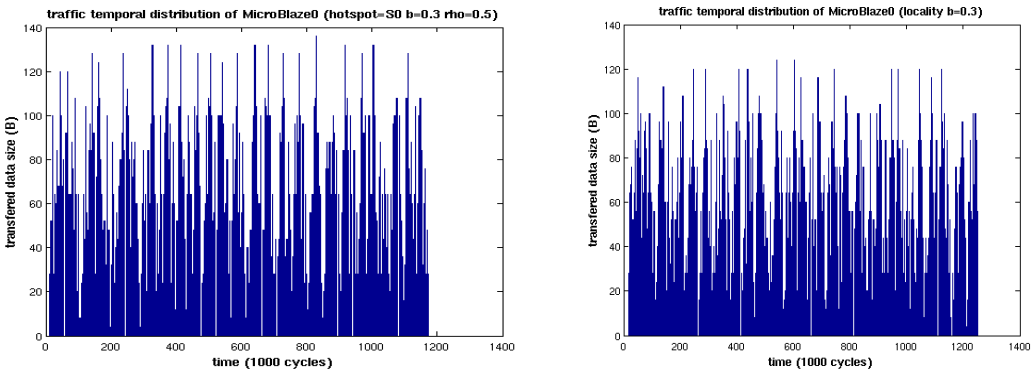


Figure 7.32 (a) Hot spot traffic b = 0.3 = 0.5 (b) locality b = 0.3

The above figures provide examples of hot spot and locality traffic.

7.7.2 Performance Evaluation

We conducted performance evaluation of the 2 OCPIP micro-benchmarks described in section 4, which are hot spot and locality, on two large scale multiprocessors.

For hot spot we considered 2 hot spot cases: (1) S0 being on-chip SRAM S_0 located on the top left corner of the architecture and (2) S24 being on-chip S_24 located on bottom right corner at another position.

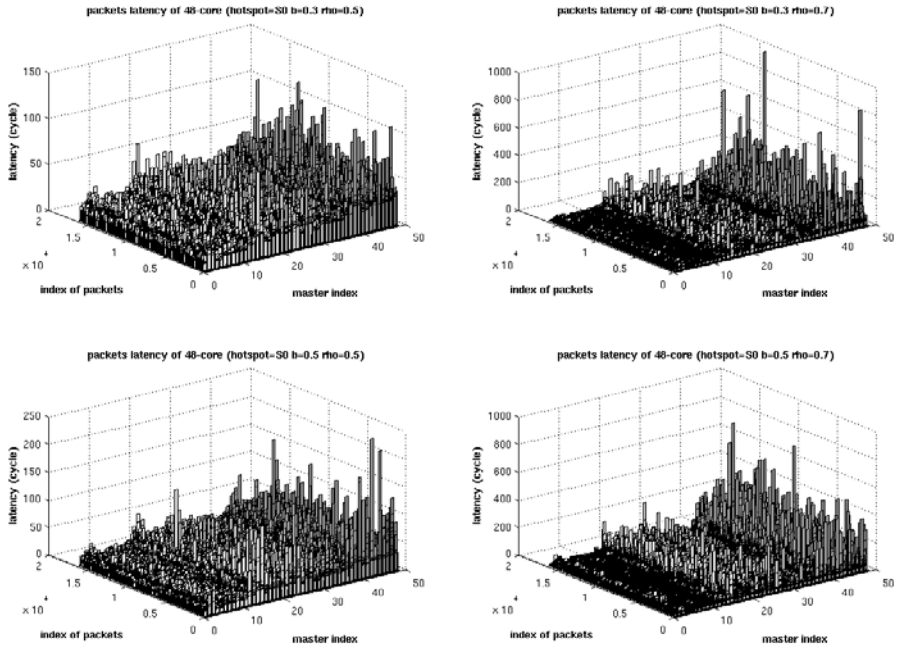


Figure 7.33 Hot spot benchmark packets latency with target S0: (1) b = 0.3 ρ = 0.5 (2) b = 0.3 ρ = 0.7 (3) b = 0.5 ρ = 0.5 (4) b = 0.5 ρ = 0.7

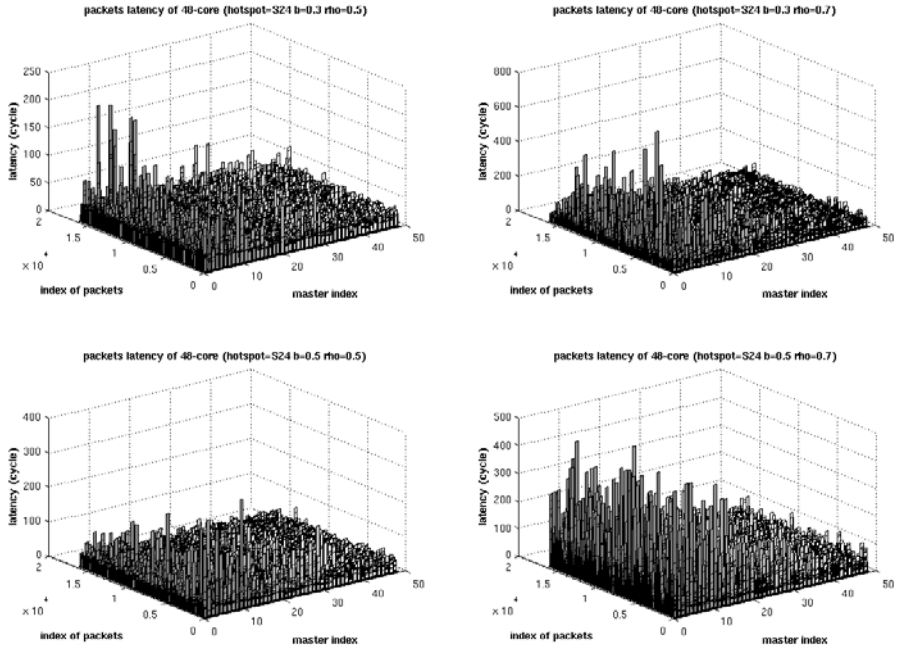


Figure 7.34 Hot spot benchmark packets latency with target S24: (1) b = 0.3 ρ = 0.5 (2) b = 0.3 ρ = 0.7 (3) b = 0.5 ρ = 0.5 (4) b = 0.5 ρ = 0.7

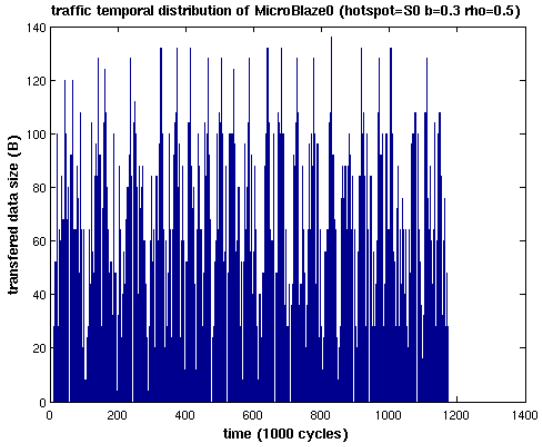


Figure 7.35 Hot spot benchmark data transferred at MB0 b = 0.3 ρ = 0.5

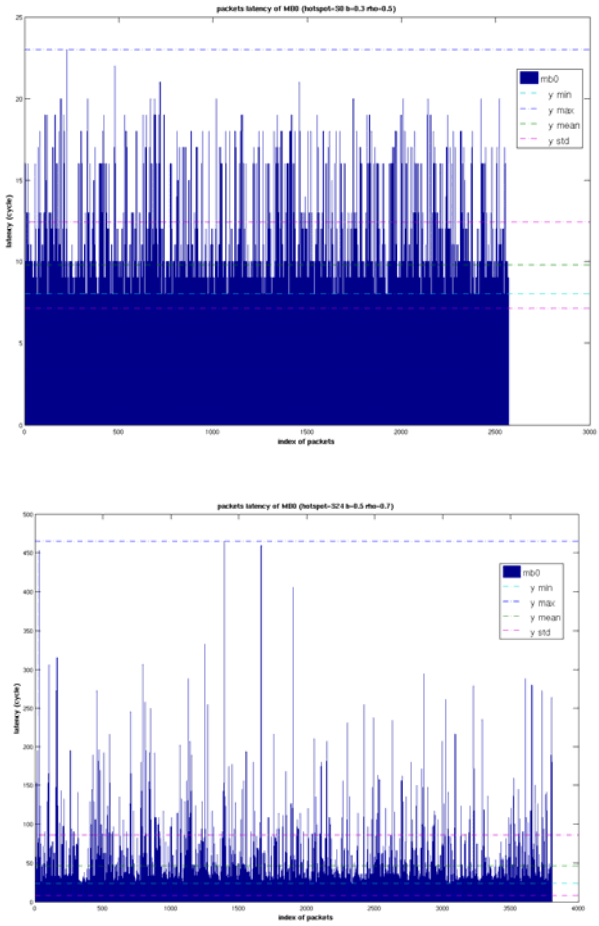


Figure 7.36 Hot spot benchmark – MB0 (1) S0 b = 0.3 ρ = 0.5 (2) S24 b = 0.3 ρ = 0.7

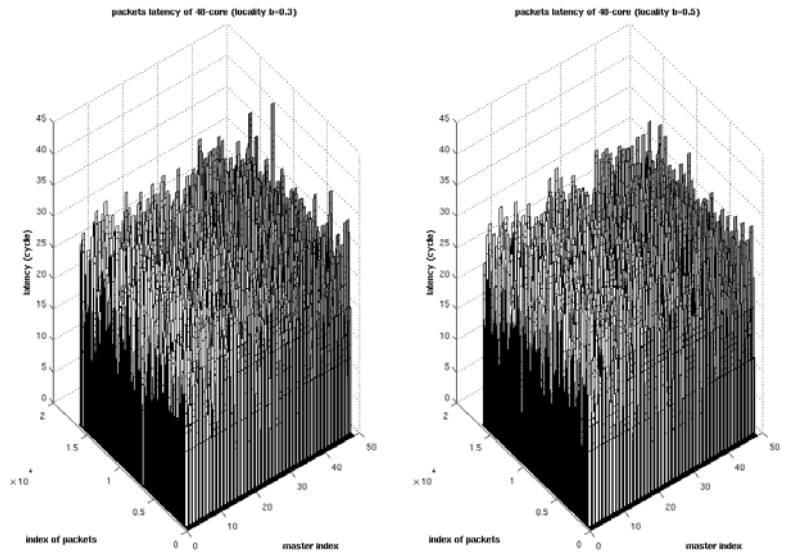


Figure 7.37 Locality benchmark packets latency : (1) b = 0.3 (2) b = 0.5

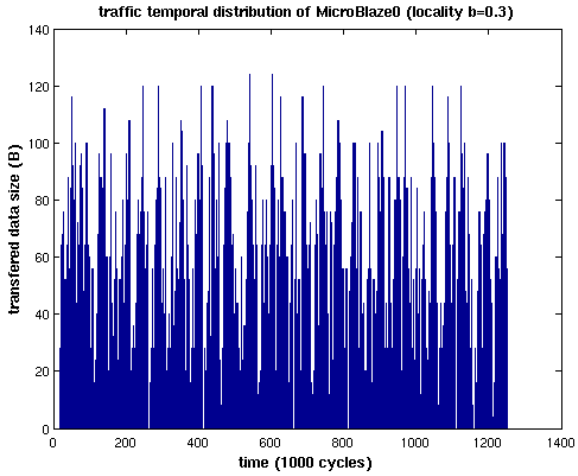


Figure 7.38 Locality benchmark data transferred at MB0 b = 0.3

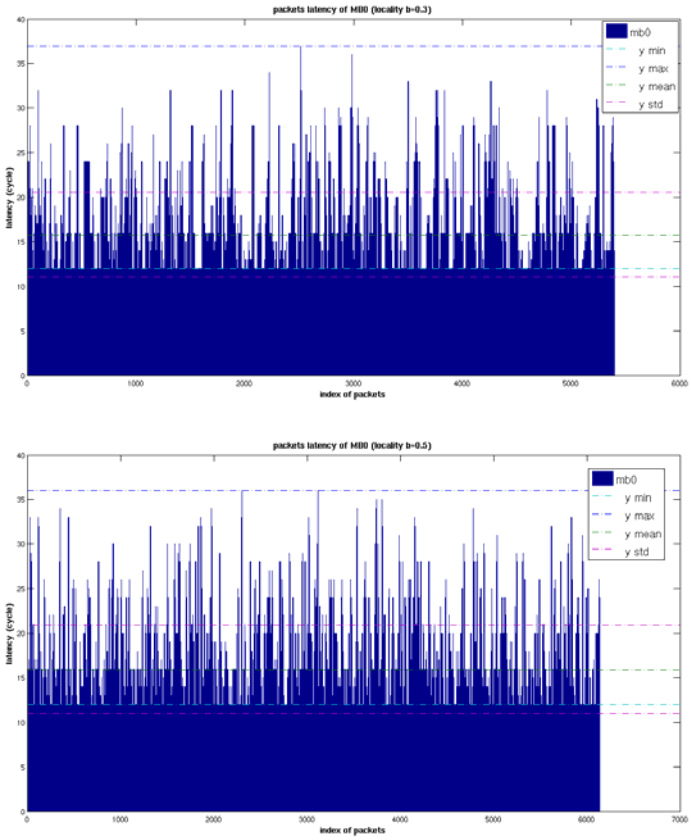


Figure 7.39 Locality benchmark packets latency MB0 : (1) b = 0.3 (2) b = 0.5

Figure 7.33 to Figure 7.39 describe the achieved results. Figure 7.33 describes the hot spot benchmark packets latency with target S0 under the following values: (1) b = 0.3 ρ = 0.5 (2) b = 0.3 ρ = 0.7 (3) b = 0.5 ρ = 0.5 (4) b = 0.5 ρ = 0.7. In Figure 7.34 the hot spot benchmark packets latency target S24. It is clear from both figures that the position of the hot spot coupled with the burst factor affect considerably the results. Figure 7.35 and Figure 7.36 provide the results for master MB0. The locality benchmark is exposed in Figure 7.37 and shows as expected less variance among latencies. The consequence for parallel programming is tremendous as parallel program execution is dominated by critical path. Figure 7.38 and Figure 7.39 provide details for master MB0

7.8 Conclusion

Next generation multiprocessor on chip will be based on hundreds of processors. Multiprocessor design is very complex and in order to reach efficient working silicon in

reasonable time large scale prototyping is needed. We propose a small scale multiprocessor design as a building block for large scale multiprocessor. This SSM IP can be quickly extended in order to build larger scale multiprocessor. We validated our approach on a 48 processors system by automatically extending a 12 processors small scale multiprocessor IP. Future work will deepen the compiler and applications part and find cross-covering niches of optimality.

Design productivity for large scale multiprocessor on chip is a major challenge. Large scale multiprocessors system on chip design can benefit from: (1) larger IP design and reuse such as small scale multiprocessor IP (2) multi-FPGA emulation platforms for quick and modular duplication combined with fully integrated EDA tools and (3) quickly redeployable benchmarks for efficient design space exploration. Indeed, considering the number of processors as a variable in the design space exploration process requires having application suites and benchmarks which can scale within the range of considered number of processors.

High performance embedded applications based on image processing and single processing will increasingly be moved to embedded multiprocessors. We have proposed an automatic design flow for data parallel and pipelined signal processing applications on embedded multiprocessor with NoC for cryptographic application TDES. Our flow explore through execution on multi-FPGA emulation for parallel software implementation with task placement exploration and task granularity analysis. A hardware based network on chip monitoring drives the task placement process to reduce communication bottlenecks. In the second phase, high level synthesis generated hardware accelerators are added to explore the tradeoff in area-performance while still privileging multiprocessor basis for the implementation. Future work will add reconfigurability management of dedicated area for hardware accelerators as well as automatic parallelization.

In the absence of such applications and benchmarks for large scale multiprocessor synthetic workloads and network on chip micro-benchmarks can play such a role. OCP-IP has proposed such a suite of micro-benchmarks. To the best of our knowledge this is the first work which evaluates through actual execution OCP-IP benchmarks on large scale multiprocessors with network on chip.

Reference

[1] ITRS http://www.itrs.net

[2] J.L. Hennessy and D.A. Patterson Computer Architecture, 4th Edition A Quantitative Approach , Morgan Kauffmann, 2006.

[3] D.Culler, J.P.Singh, A.Gupta, Parallel Computer Architecture : A Hardware/Software Approach , Morgan Kaufmann Pub. , 1999.

[4] A.A. Jerraya and Wayne Wolf , “Multiprocessor Systems-on-Chip”, Morgan Kaufman Pub, 2004

[5] Benini, L. ; De Micheli, G., “Networks on Chips: Technology and Tools”, Morgan Kaufmann, 2006.

[6] T.Miyamori, Venezia: a Scalable Multicore Subsystem for Multimedia Applications, 8th International Forum on Application-Specific Multi-Processor SoC 23 - 27 June 2008, Aachen, Germany http://www.mpsoc-forum.org/

[7] T.Isshiki, *MAPS-TCT: MPSoC Application Parallelization and Architecture Exploration Framework*, 8th International Forum on Application-Specific Multi-Processor SoC 23 - 27 June 2008, Aachen, Germany http://www.mpsoc-forum.org/

[8] L;Gao, K.Karuri, S.Kraemer, R.Leupers, G.Ascheid and H.Meyr, Multiprocessor Performance Estimation Using Hybrid Simulation", in Proc. of DAC 2008, pp.325-330, June 8-13, 2008, CA, USA.

[9] R.Ben Mouhoub and O.Hammami MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution, Volume 2006 (2006), Article ID 54074, 14 pages

[10] M.O.Cheema, O.Hammami**,** Application-specific SIMD synthesis for reconfigurable architectures, *Microprocessors and Microsystems*, *Volume 30, Issue 6*, *4 September 2006*, *Pages 398-412*

[11] M.O.Cheema, L.Lacassagne, and O.Hammami , System-Platforms-Based SystemC TLM Design of Image Processing Chains for Embedded Applications, Volume 2007 (2007), Article ID 71043, 14 pages

[12] O.Hammami, Z.Wang, V.Fresse, and D.Houzet, A Case Study: Quantitative Evaluation of C-Based High-Level Synthesis Systems, Volume 2008 (2008), Article ID 685128, 13 pages

[13] Z.Wang and O.Hammami, C-Based Hardware-Accelerator Coprocessing for SOC An Quantitative Area-Performance Evaluation, In Proc of the 15th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2008, Malta, 31st Aug. - Sept3rd. 2008.

[14] Y.Ahn, K.Han, G.Lee, H.Song, J.Yoo and K.Choi, "SoCDAL: System-on-chip Design AcceLerator", ACM TDAES, Vol. 13, No.1, pp. 17:1-17:38, Jan. 2008.

[15] H. Nikolov, M. Thompson, T.Stefanov, A. Pimentel, S. Polstra, R.Bose, C. Zissulescu, E.Deprettere, "Daedalus: Toward Composable Multimedia MP-Soc Design", in Proc. of DAC 2008, pp.574-579, June 8-13, 2008, CA, USA.

[16] C. Haubelt, T. Schlichetr, M. Meredith, "SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models", in Proc. of DAC 2008, pp.580-585, June 8-13, 2008, CA, USA.

[17] G.Chen, F.Lui, S.W.SOn, and M.Kandemir, "Application Mapping for Chip Multiprocessors", in Proc. of DAC 2008, pp.620-625, June 8-13, 2008, CA, USA.

[18] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002

[19] FIPS 46-3: The official document describing the DES standard

[20] FIPS 197: The official document describing the AES standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[21] “Recommendation for Block Cipher Modes of Operation-Methods and Techniques”, NIST Special Publication 800-38A 2001 Edition

[22] Patrick Crowley “The future in your pocket” March 2008 SIGCOMM Computer Communication Review , Volume 38 Issue 2

[23] Rizk, M.R.M.; Morsy, M “Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA” International Design and Test Workshop, 2007 2nd 16-18 Dec. 2007 Page(s):207 – 217

[24] P. Kitsos, N. Sklavos, M.D. Galanis, O. Koufopavlou, VLSI Implementations of the Triple-DES Block Cipher, Proceedings of the 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS'03), United Arab Emirates, December 2003

[25] Helion Technology, – High Performance DES and Triple-DES cores for Xilinx FPGA, http://www.heliontech.com , 2003.
[26] Xilinx Inc., Pasham V, Trimberger S, High-Speed DES and Triple-DES Encryptor-Decryptor, http://www.xilinx.com, August 2001
[27] Yao Yue, Chuang Lin, Zhangxi Tan "NPCryptBench: a cryptographic benchmark suite for network processors" March 2006 MEDEA '05: Proceedings of the 2005 workshop on MEmory performance: DEaling with Applications , systems and architecture
[28] Divya Arora, Anand Raghunathan, Srivaths Ravi, Murugan Sankaradass, Niraj K. Jha, Srimat T. Chakradhar "Software architecture exploration for high-performance security processing on a multiprocessor mobile SoC" July 2006 DAC '06: Proceedings of the 43rd annual conference on Design automation ACM.
[29] Jung-Ho Lee, Sung-Rok Yoon, Kwang-Eui Pyun, Sin-Chong Park "A multi-processor NoC platform applied on the 802.11i TKIP cryptosystem" January 2008 ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation IEEE Computer Society Press.
[30] Klimm, A.; Braun, L.; Becker, J.; "An adaptive and scalable multiprocessor system For Xilinx FPGAs using minimal sized processor cores" IPDPS2008, April 2008
[31] OCP-IP Open Core Protocol Specification 2.2.pdf http://www.ocpip.org/, 2008
[32] Arteris http://www.arteris.com/
[33] NoC Solution 1.12, NoC NTTP technical reference, o3446v8, April 2008
[34] Arteris Danube 1.12, Packet Transport Units technical reference, o4277v11, April 2008
[35] Xilinx Virtex-4 http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/virtex4/index.htm
[36] Xilinx EDK 9.2 http://www.xilinx.com/ise/embedded/edk_docs.htm
[37] http://www.impulsec.com/
[38] A.Kumar, S.Fernando, Y.Ha, B. Mesman and H.Corporall, "Multiprocessor Systems Synthesis for Multiple Use-Cases of Multiple Applications on FPGA", ACM TDAES, Vol. 13, No.3, p. 40:1-40:27, July 2008.
[39] Eve ZeBu UF-4 http://www.eve-team.com
[40] S.H.Bokhari,; *Partitioning problems in parallel, pipeline, and distributed computing*, Computers, IEEE Transactions on, Volume 37, Issue 1, Jan. 1988 Page(s):48 – 57
[41] King, C.-T.; Chou, W.-H.; Ni, L.M.; *Pipelined data parallel algorithms-I: concept and modeling*, Parallel and Distributed Systems, IEEE Transactions on, Volume 1, Issue 4, Oct. 1990 Page(s):470 – 485
[42] King, C.-T.; Chou, W.-H.; Ni, L.M.; *Pipelined data parallel algorithms-II: design*, Parallel and Distributed Systems, IEEE Transactions on, Volume 1, Issue 4, Oct. 1990 Page(s):486 - 499
[43] P. Hansen, K.-W. Lih, *Improved algorithms for partitioning problems in parallel, pipeline, and distributed computing*, IEEE Transactions on Computers 41 (6) (1992) 769–771.
[44] M.A. Iqbal, S.H. Bokhari, *Efficient algorithms for a class of partitioning problems*, IEEE Transactions on Parallel and Distributed Systems 6 (2) (1995) 170–175.
[45] Yu-Kwong Kwok, I.Ahmad *Static scheduling algorithms for allocating directed task graphs to multiprocessors* December 1999 Computing Surveys (CSUR) , Volume 31 Issue 4
[46] M.A. Senar, A.Ripoll, A.Cortés, E.Luque, *Clustering and reassignment-based mapping strategy for message-passing architectures*, Journal of Systems Architecture, Volume 48, Issues 8-10, March 2003, pp. 267-283
[47] Erbas, C.; Cerav-Erbas, S.; Pimentel, A.D.; *Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design*, Evolutionary Computation, IEEE Transactions on, Volume 10, Issue 3, June 2006 pp. 358 – 374
[48] Kianzad, V.; Bhattacharyya, S.S.; *Efficient techniques for clustering and scheduling onto embedded multiprocessors*, Parallel and Distributed Systems, IEEE Transactions on, Volume 17, Issue 7, July 2006 pp. 667 – 680
[49] P.Chandraiah, R.Domer, *Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Recoding*, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, Volume 27, Issue 6, June 2008 pp.1078 – 1090
[50] B. Ristau, T. Limberg and G. Fettweis, *A Mapping Framework Based on Packing for Design Space Exploration of Heterogeneous MPSoCs*, Journal of Signal Processing (2009) 57:45-56
[51] G.Chen, F.Lui, S.W.SOn, and M.Kandemir, *Application Mapping for Chip Multiprocessors*, in Proc. of DAC 2008, pp.620-625, June 8-13, 2008, CA, USA.
[52] M. Monchiero, G. Palermo, C. Silvano, O. Villa, *Exploration of distributed shared memory architectures for NoC-based multiprocessors*, *Journal of Systems Architecture*, *Volume 53, Issue 10*, *October 2007*, *Pages 719-732*

[53] R. Ben Mouhoub and O. Hammami, *"MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution,"* EURASIP Journal on Embedded Systems, vol. 2006, Article ID 54074, 2006.

[54] R.B.Mouhoub and O Hammami,.;*Multiprocessor on chip: Beating the Simulation Wall Through Multiobjective Design Space Exploration with Direct Execution* , IEEE PDPS 2006, 25-29 April 2006 Page(s):8 pp.

[55] X.Li and O.Hammami, *NOCDEX: Network on Chip Design Space Exploration Through Direct Execution and Options Selection Through Principal Component Analysis*, Industrial Embedded Systems, 2006. IES '06. International Symposium on 18-20 Oct. 2006 Page(s):1 – 4

[56] Sean Whitty and Henning Sahlbach and Wolfram Putzke-Röming and Rolf Ernst. "Mapping of a Film Grain Removal Algorithm to a Heterogeneous Reconfigurable Architecture." In Proc. of Design, Automation and Test in Europe (DATE), Nice, France, April 2009.

[57] Katalin Popovici, Xavier Guerin, Frédéric Rousseau, Pier Stanislao Paolucci, Ahmed Amine Jerraya, "Platform-based software design flow for heterogeneous MPSoC", ACM Transactions on Embedded Computing Systems, Volume 7 , Issue 4 (July 2008)

8. Very Large Scale Multiprocessor Design Automation

The previous chapters have demonstrated that small to medium scale multiprocessor design on single or multiple chips with fast productivity is a reality. This addressed our original concern on tackling the design productivity gap for MPSOC design of small to medium scale. The next challenge for the research community is to address large scale multiprocessors, which are beyond 512 PE or even more [1].

To the best of our knowledge there is no reported work in the world in 2010 besides the RAMP project, of prototyping multiprocessor larger than 512 cores on multi-FPGA platforms or on single ASIC chip.

Very large scale multiprocessor (VLSM) is very difficult to be implemented on to ASIC or FPGA circuit because of its complexity and the lack of CAD supported design framework. The design productivity challenge is at its best in terms of hardware generation and in performance evaluation in face of the simulation wall. To our best knowledge, there is no report on either high level modeling (e.g. SystemC) or high level performance evaluation of VLSM. As the simulation wall blocks the VLSM design path, FPGA platform based emulation is selected as the only solution for billion cycle applications. Based on the experience achieved and described in the previous chapters, the framework should support both cycle-accurate emulation of detailed parameterized machine models and rapid functional-only emulations. Details in the underlying FPGA emulation should be hidden from architects and software designers. Automatic design work flow is mandatory at this scale to accelerate the design and debug process.

We achieved the design, implementation and validation of a 672 cores cluster-mesh NoC based multiprocessor architecture onto a very large scale emulator: the Zebu XXL multi-

FPGA platform. Compared to RAMP Blue 1008-core multiprocessor, our system is more flexible in communication architecture and it is fully implemented with NoC technology.

8.1 State of the art

The state of the art at this level of complexity is restricted in 2010 to the University of California Berkeley project, RAMP (Research Accelerator for Multiple Processors) [2].

The project aims to ramp up the hardware and software multiprocessor research. From 2005, three prototype systems have been developed and implemented, named as RAMP Red, RAMP Blue and RAMP White. New systems have been developed such as RAMP gold and RAMP purple from 2007. The table below summarizes the major features of these systems.

Table 8.1 RAMP systems summary

System	Processor	OS	Communication	Number of FPGA	Max number	Frequency (MHz)
RAMP Red (2005)	PowerPC	Linux	Transactionnal Memory	4	8	100
RAMP Blue (2005)	MicroBlaze V4.0	ucLinux	Message Passing	64	1008	90-100
RAMP White (2005)	PowerPC LEON	Linux	Coherent Shared Memory	1	2	50 ?
RAMP Purple (2007)	MicroBlaze V6.0	Xilinx MicroKernel	Coherent Shared Memory	8 ?	64 ?	90-100
RAMP Gold (2007)	SPARC V8	No report	Coherent Shared Memory	No report	64	50

Seen from the above table, the RAMP project has evolved through different versions in the communication paradigm and in the number of processors. Only the RAMP blue has hundreds of processors based on message passing communication. The detail of this system is described in the next section.

The RAMP framework is based on units which communicate and synchronize with each other over channels. Units can be developed as RTL models implemented onto FPGA or as

software codes executed on attached computer for co-simulation or on processor cores embedded within FPGA platform for HW/SW co-synthesis.

The RTL code of unit must be provided by model designers. The instances of each unit and their interconnection by channel are described using RDL (RAMP description language). Then RAMP framework tools automatically generate RTL code of the channels and the interfaces to the channels for a unit. The configuration tools in RAMP framework provide the option to change the channel's parameters dynamically at system boot time. This configuration tool also supports control of the inter-unit channels for monitoring and debugging facilities. Messages can be inserted and read out from the channels by using a automatically inserted debugging network. More detailed workflow is not found in published reports.

8.1.1 BEE multi-FPGA platform

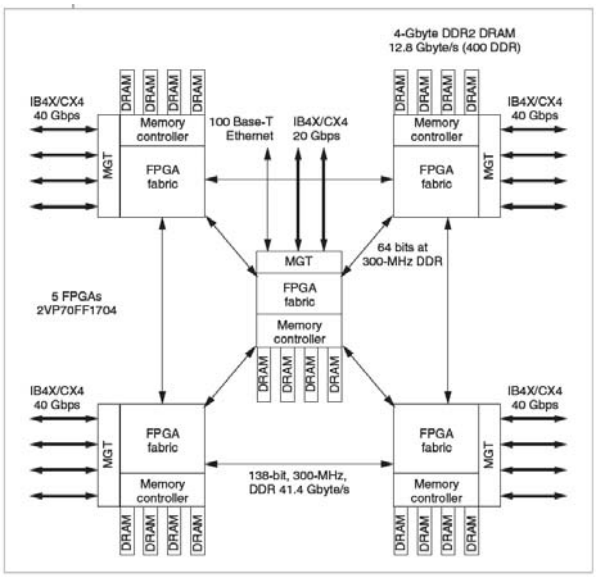


Figure 8.1 general architecture of BEE2 module

RAMP systems have been implemented at first on BEE2 (Berkeley Emulation Engine) multi-FPGA platform [6]. The main board contains 5 Xilinx Virtex-II Pro 70 FPGAs, among which 4 FPGAs are used for user implementation and are connected in a ring topology while the 5th FPGA serves as the control FPGA, which is connected in a star topology with each user FPGA as presented in Figure 8.1. On the board, each FPGA is mapped to 4 DDR2 banks of 1GB each with 3.4 GBps peak throughput per channel. Each user FPGA has four sets of

multi-gigabit transceiver (MGT) for off-board communication. These serial channels are connected to 10GBase-CX4 Ethernet interface for Ethernet or simple P2P connection over cables.

New version project RAMP-2 will base on BEE3 platform. The BEE3 board has 4 Xilinx Virtex 5 LX 110T FPGAs, along with 64 GB DDR2 DRAM and eight Ethernet interfaces for inter-board communication. Over 64 MicroBlaze processors can be implemented on this board.

The major challenge using BEE platform is porting designs that span multiple FPGAs [3]. The interconnection between IPs is often constrained by the inter-FPGA network and especially inter-board interconnection as Ethernet. The initial bus connection of RAMP Red system has be replaced by a star-like interconnection network constrained by the BEE2 board layout.

It is important to note that in our project based on EVE multi-FPGA platform, there is no interconnection constraints. The emulator allows any type of communication architecture to be implemented.

8.1.2 RAMP Blue

The RAMP Blue system [4] consists of 768-1008 MicroBlaze cores implemented on BEE2 platform which has at most 21 boards. It is the first multiprocessor on chip surpassing the milestone of thousand cores. Inter-processor communication is based on MPI standard or for global address space language such as Unified Parallel C (UPC). We describe in details this platform.

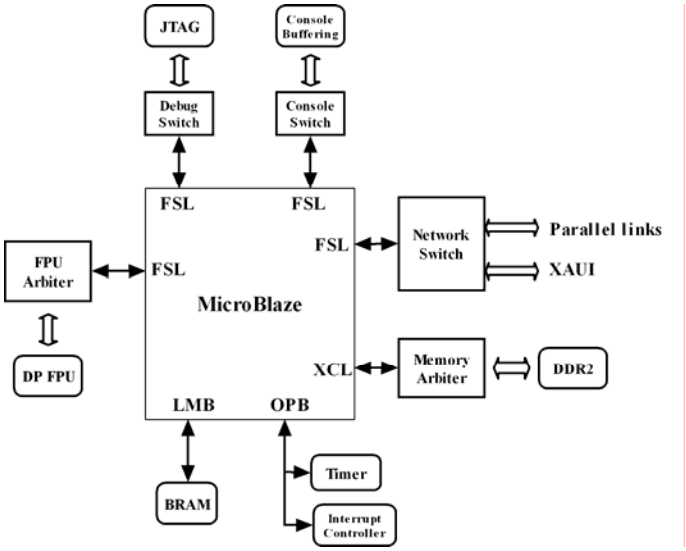
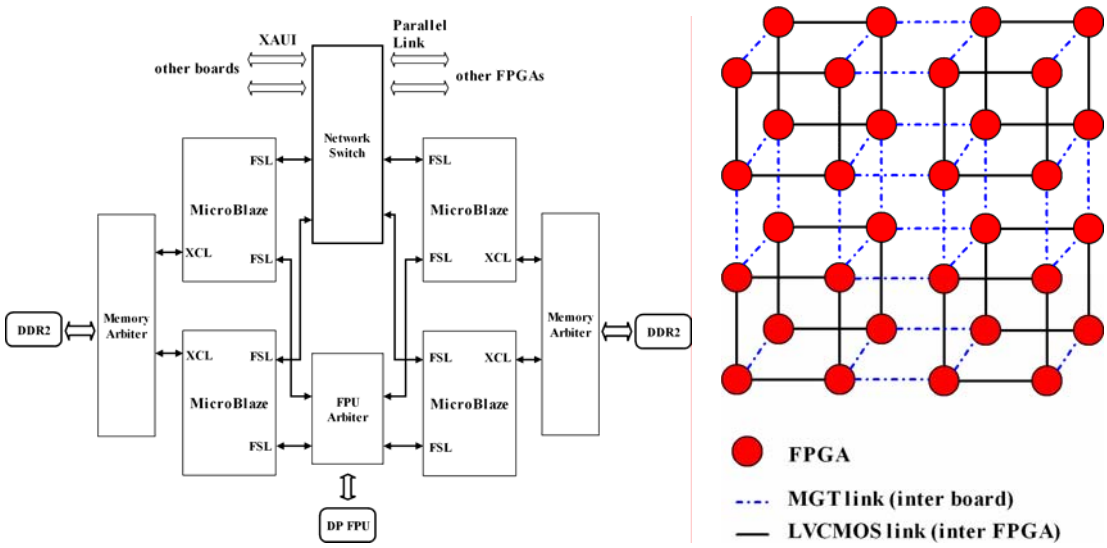


Figure 8.2 Architecture of MicroBlaze processor for RAMP Bleu system interconnection

The Xilinx MicroBlaze processor has been selected in RAMP Blue for its better resource utilization and its FIFO-like FSL interface which provides unidirectional point-to-point connections. At most 12 MicroBlaze processors can be implemented on Virtex-II Pro 70 FPGA, with the following processor options : 90 MHz core clock, no optional functional units, all optional exceptions, 2 KB I-cache, 8 KB D-Cache, LMB block RAM and OPB peripherals. The full 12-core design consumes 32,991 slices (99%), 61,891 LUTs (93%), 37,198 flip-flops (56%) and 181 block RAMs (55%) [4]. The FSL (Fast Simplex Link) is used for most of the interconnection for inter and intra FPGA communication between processors. Packets are transmitted between the FSL interfaces of different processors through statically routed network.



(a) (b)

Figure 8.3 Intra and inter board communication and RAMP Blue 3D mesh architecture

RAMP Blue communication architecture is based on 3D cluster mesh topology. Inter board connection is 3D mesh as shown in figure 8.4, while inter FPGA network at the same board is a ring. On each FPGA, processors are connected through a crossbar switch. The intra and inter BEE2 communication is based on buffer and crossbar switch units. Packets are routed by the buffer units to MicroBlaze or across another link. For compatibility, the packet format is Ethernet II encapsulated with route header. The 4 off-board serial channels are used for inter board connection while two parallel links are used for inter FPGA connection on the same board. Virtual cut-through routing is used and it will be blocked if the next buffer is not available. Board-to-board connections use 4 sets of bonded MGT (multi-gigabit transceiver) forming four independent 10 Gbps high-speed serial I/O channels, so the latency of these links is from tens to hundreds of cycles while the intra-board LVCMOS Parallel links latency is two or three cycles. The delivery is guaranteed end-to-end in software. The gateware does not perform check-summing or retransmission. The prototyping platform system is shown in the following figure. Each BEE2 board is in preliminary 2U chassis and assembled in a standard 19'' rack with external supplies. Physical connection among the boards is through 10 Gbps cables. System configuration, debugging and monitoring are through a 100 Mbps Ethernet switch with connection to the control FPGA of each board.

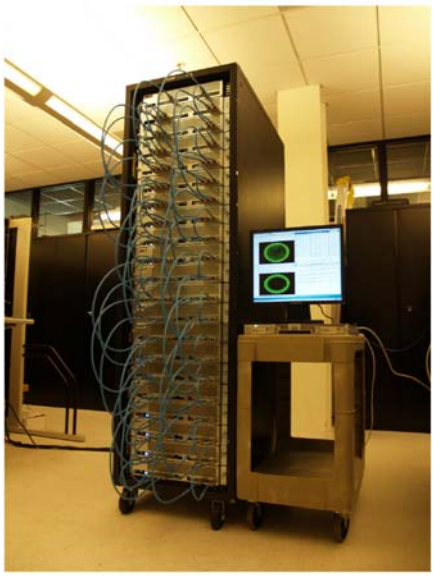


Figure 8.4 1008 cores RAMP Blue system on 21 BEE2 boards with management server and monitor [5]

The NAS parallel benchmarks are run on RAMP Blue system with class-S datasets due to memory limitations [5] The execution time is dominated by the communication costs, a large fraction of which can be eliminated by implementing a DMA-based network interface. Five of the NPB benchmarks are executed on RAMP Blue system: Embarrassingly Parallel (EP), Multigrid (MG), Conjugate gradient (CG), 3D FFT PDE (FT) and Integer sort (IS). The performance results are shown in the following table.

Table 8.2 NAS Parallel Benchmarks performance results on RAMP Blue system [5]

Benchmark	**Size**	**Iteration**	Processors	**Time (s)**	**Mops/s**
EP	25	256	256	27.75	1.21
MG	32x32x32	4	256	13.22	0.57
CG	1400	15	256	190.19	0.35
FT	64x64x64	6	32	12.77	13.88
IS	65536	1	256	4.39	0.01

Although its performance results is not good as the latency of communication is high, RAMP Blue is the first step for developing a robust library of RAMP infrastructure for building more complicated parallel systems.

8.2 General VLSM framework

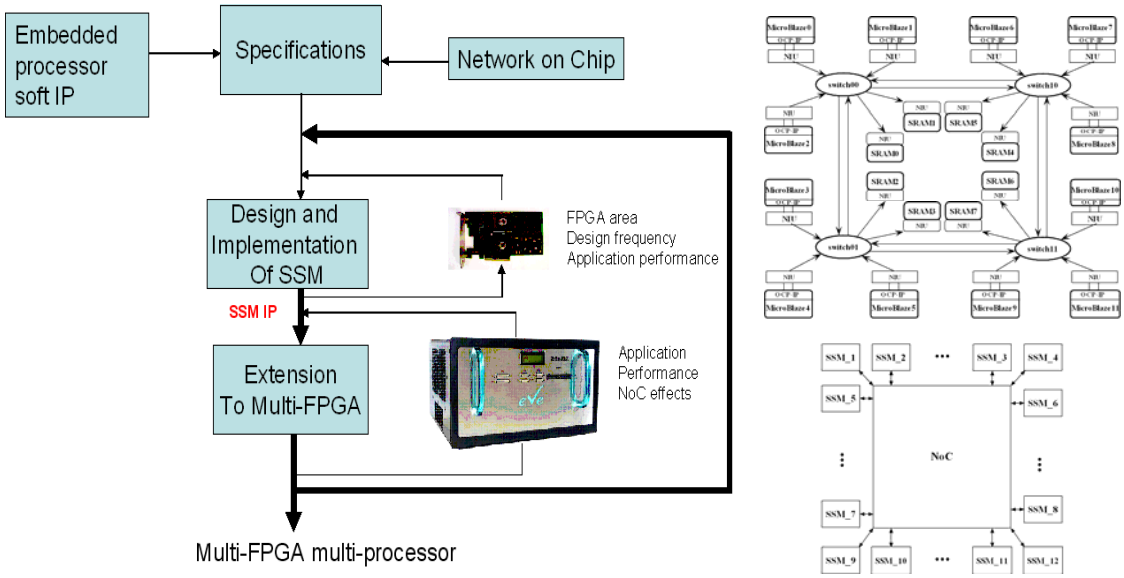


Figure 8.5 General methodology for VLSM multiprocessor design

Our general VLSM design methodology is based on the idea of SSM IP reuse and Network on Chip interconnection technology. Embedded processor soft IPs and NoC are used for the design and implementation of SSM IP. At this step, we focus on the improvement of system area, frequency and application performance. The exploration of SSM IP design can be based on single FPGA platform. Different SSM IPs can be used to realize a heterogeneous VLSM multiprocessor. After all the SSM IPs are designed, they are used and duplicated onto multiple FPGA chip to form the large scale multiprocessor. The interconnection between SSM IPs is based on another top level of Network on Chip. This top level NoC can be realized by the interconnection between SSM IP's switches as in [7], or by a separate topology of NoC to which all the SSM IP are connected. The final multiprocessor is implemented onto multi-FPGA platform without any modification, by using the EVE Zebu compilation and implementation technology.

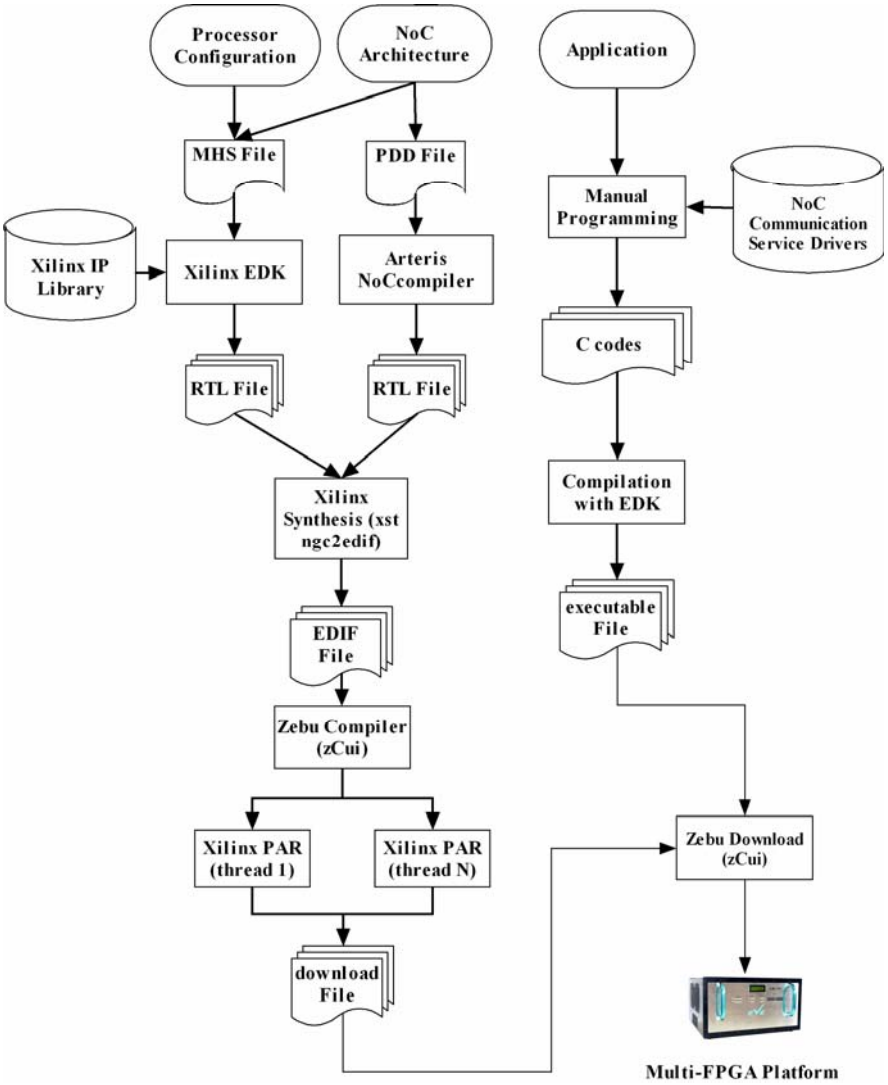


Figure 8.6 target Automatic workflow for VLSM multiprocessor

The automatic workflow of our VLSM multiprocessor design is presented in the above figure. Design automation tools of 3 commercial companies are combined together to generate our multi-FPGA MPSoC. The Xilinx EDK tool is used to generate our SSM multiprocessor using Xilinx IPs. Once the RTL files of SSM are generated, they are reused for the multi-FPGA large scale multiprocessor synthesis, which can largely reduce system design time. Different NoC files are synthesized for each SSM on different FPGA chips of Zebu platform. These NoC RTL files are generated by Arteris NoCcompiler tool, which allows the export of NoC using the Arteris Danube Library. Eve Zebu compiler takes the EDIF files converted by Xilinx synthesis tools for the implementation. Different SSM IPs are analyzed

and distributed onto FPGAs. Xilinx place and route tools are used to generate the download bit files of FPGA. This phase can be parallelized to reduce the turnaround time. The application is parallelized onto our multiprocessor using the NoC communication service drivers. All the C codes are compiled using the gcc compliant compiler integrated in EDK design environment. Finally the compiled execution files and download bits files are downloaded to EVE Zebu XXL multi-FPGA platform using Zebu download tools. The execution time is recorded after the execution.

8.3 BB-672 VLSM

There are 56 Xilinx Virtex 4 LX200 FPGAs available on the Zebu XXL platform. Duplicating our SSM IP onto these 56 FPGAs, our VLSM with 672-core is generated automatically [8,9]. We name our VLSM as BB-672 as a consequence. The general global architecture is presented in the figure. Each square represents one SSM IP which is implemented on one FPGA. The global architecture stays as cluster mesh topology. Each of our SSM IP consumes 53,438 slices (59%), 88,432 LUTs (49%), 50,999 flip-flops (31%), 91 DSP48s (94%) and 289 block RAMs (86%) of Virtex-4 LX200 FPGA.

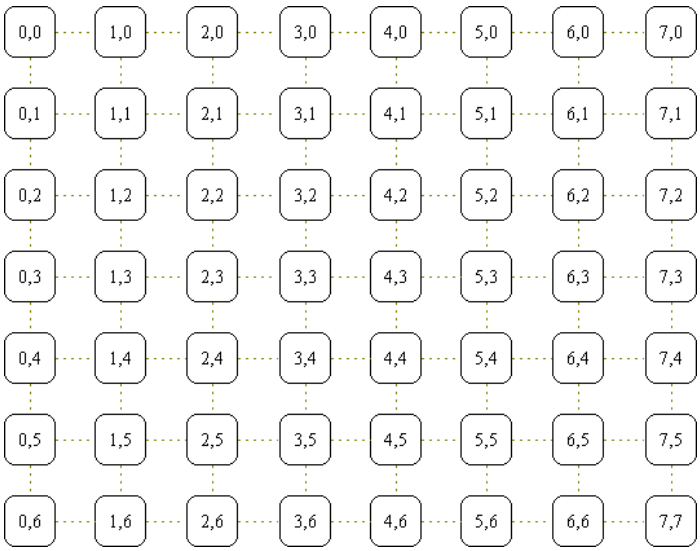


Figure 8.7 Cluster-mesh architecture of 672-core VLSM

8.3.1 Zebu XXL multi-FPGA platform

Our very large scale multiprocessor is implemented on ZeBu-XXL platform. It is a high capacity system emulator with the easy setup and debugging associated with emulation, and

the performance of rapid prototyping. Configurable to handle designs from 12.5M to 100M ASIC gates in a compact, rack-mountable unit, and expandable to accommodate up to 200M ASIC gates via 2 interconnected units. It is ideal for the system-integration phase of the design cycle where multiple logic blocks, multiple chips, and embedded software all must be verified together. Hardware design and software development teams can share the same system and design representation, and can easily collaborate when debugging complex hardware/software interactions.



Figure 8.8 Eve Zebu XXL multi-FPGA platform

ZeBu-XXL is a 19" rack system and includes a PCI (or PCIe) interconnection board (with two cables) to connect to the host PC running under Linux. ZeBu-XXL includes the following modules to implement the DUT:

• 4 memory modules with 1 GBytes of DDR2 DRAM each

• 8 modules slots used to customize the ZeBu-XXL configuration, each able to host one of the following modules:

o Single-Slot FPGA module with 8 Virtex-4 LX200

o Double-Slot FPGA module with 8 Virtex-4 LX200 and 2 Gbit of RLDAM

o Single-Slot DirectICE Module with 2 Virtex-4 LX100 and 128 Mbit of SSRAM

ZeBu-XXL is an open system in that it interfaces with the best-in-class EDA tools, such as listed in the table:

FPGA synthesis (RTL emulation)	Synplify Pro™ Precision Synthesis™ Xilinx XST
ASIC synthesis (gate level emulation)	Design Compiler™ (through a GTECH or Xilinx Library)
HDL simulators	VCS™ ModelSim™ NC-Sim™
C++ modeling	SystemC

The Zebu compilation tools make sure the initial user system is correctly partitioned and placed onto the multiple FPGA circuits.

8.3.2 OCP-IP benchmarks

As presented in the last chapter, OCP-IP specifies 6 spatial traffic patterns [6]: (1) Uniform (2) Locality (3) Bit Rotation (4) N Complement (5) Hot Spot (6) Fork Join Pipeline. Each of these spatial traffic patterns are representative of a distinct spatial pattern. For example, Fork join pipeline is a pattern where a fork feeds c nodes that are the starting point of c parallel pipelines while uniform is the uniform distribution. In our 672-core case, all the spatial benchmarks are executed except the Fork and Join Pipeline.

Table 8.3 OCP-IP specific spatial distribution

Spatial Distribution	Description
uniform	P=1/(N-1), assuming N nodes in network.
locality	P(d) = $1/(A(D)2^d)$, where D is the max distance and A(D) = $\sum_{d=1}^{D}(1/2^d)$
Bit rotation	Right rotating the bit string representation of the source node address by one .
N complement	$ns + nd = N$, where *ns* is the source node's address and *nd* is its destination address
Hot spot	N/M of the nodes selected as hot spots M ∈ {2, 4, 8, ..., N}; ρ . ∈{0.5, 0.7} of traffic sent to these hot spots remaining sent uniformly

In Uniform case, the probability for one node to send another node a packet is 1/(N-1), assuming there are N nodes in the network. A node does not send data to itself.

Locality spatial pattern describes traffic with spatial locality that is the probability to send a packet to a destination node is higher when the destination node is spatially closer. More precisely if we consider the distance d as the source-destination distance then P(d) = $1/(A(D)2^d)$ where D is the maximum distance in the network and A(D) = $\sum_{d=1}^{D}(1/2^d)$ is a normalizing factor. Within a set of nodes with the same distance, each node is selected with uniform probability.

Both Bit Rotation and N Complement define a destination node address by a function of the source node address. Bit Rotation pattern means that the destination address is obtained by

right rotating the bit string representation of the source node address by one. While in N Complement scenario, if the source node address is *ns*, its destination node address is *nd*, then *ns* + *nd* = *N*, supposing there are N nodes in the network and they are numbered as naturals from 0 utile N-1.

The hot spot model selects N/M^2 of the nodes as hot spots $M \in \{2, 4, 8, ..., N\}$. A certain fraction $\rho \in \{0.5, 0.7\}$ of traffic is sent to these hot spots while the remaining is sent uniformly to all other nodes.

8.3.3 Performance evaluation

We have designed a 672-core multiprocessors extend version of the 48-processors multiprocessor. This multiprocessor has been implemented on a large scale emulation platform (Eve ZeBu XXL) composed of 56 FPGA Virtex-4 LX200.

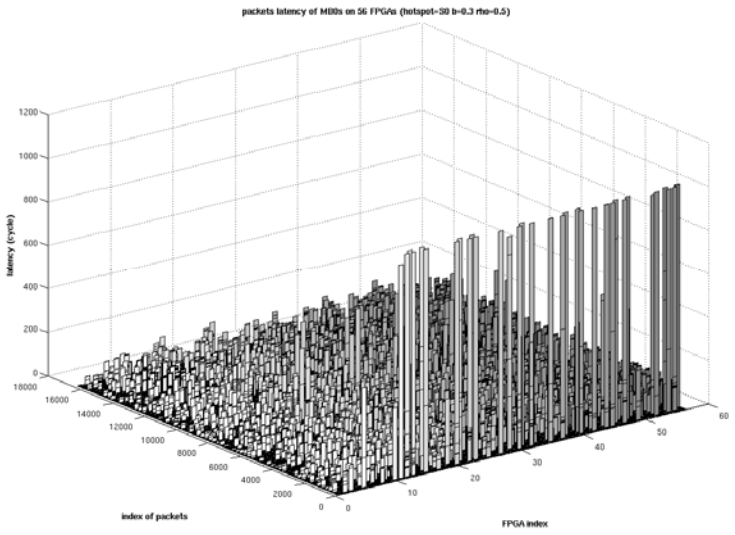


Figure 8.9 Hot spot benchmark packets latency S0: b = 0.3 ρ = 0.5 (672 core)

At first, we ran the hot spot benchmark targeting S0 as the hot spot with b=0.3 and ρ = 0.5. Monitoring networks have been applied to all masters using the same overall monitoring configuration. Due to the large amount of collected data from this large number of processors we describe the traffic latency of only 56 masters (1 per FPGA chip). A part from the starting phase where some masters experience large latencies , it appears that latencies smooth out on

the large scale multiprocessor and this contrary to the 48-core version where under the same benchmarking conditions the hot spot is clearly pronounced as described in figure 8.10. So we plan to evaluate the full set of OCP-IP benchmarks on this large scale multiprocessor platform.

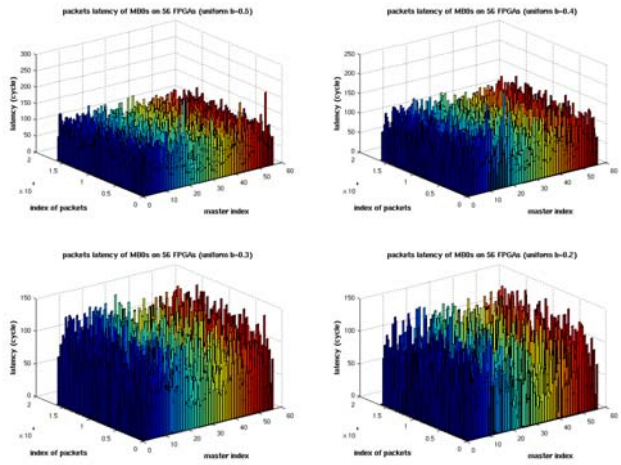


Figure 8.10 Uniform benchmark packet latency with b = {0.5, 0.4, 0.3, 0.2}

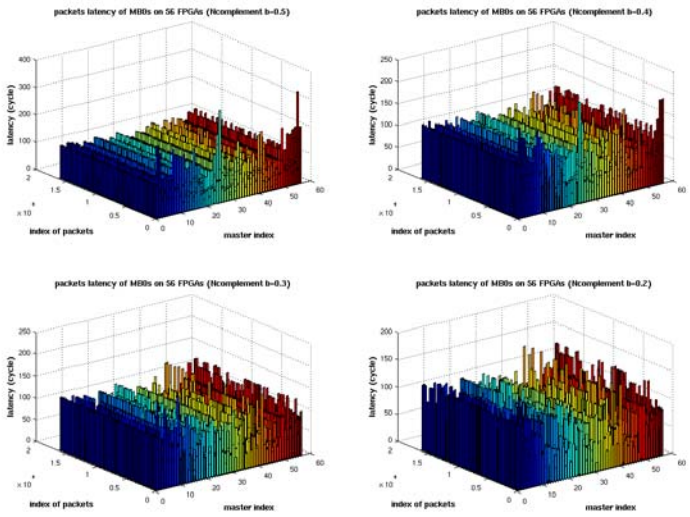


Figure 8.11 N complement benchmark with b = {0.5, 0.4, 0.3, 0.2}

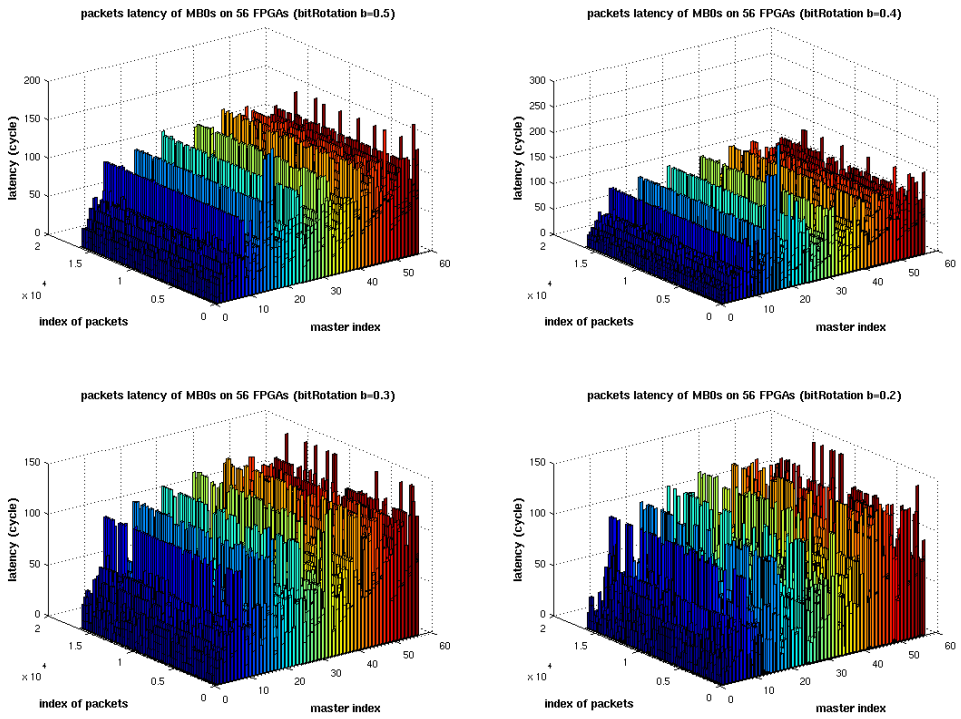


Figure 8.12 Bit rotation benchmarks with b = {0.5, 0.4, 0.3, 0.2}

It's interesting to analyze the Uniform, Bit Rotation and N Complement benchmarks together. As we can see from the above figures, changing the parameter b from 0.5 to 0.2, the packet latency does not fluctuate very much, especially in the Bit Rotation and N complement cases. While in Uniform case, for the same master, the packet latency changes from one to each other, as the packets go to different nodes. But still the traffic bursty does not change the global fluctuation. The impact of burst is the maximum packet latency decreases as the bursty increase, as there are less conflict when the traffic becomes more bursty.

Comparing the Bit Rotation and N Complement results, we can see that how nodes are numbered determines the traffic spatial distribution. The node should be regularly numbered according to the topology. The pair of source and destination nodes should be close to decrease packet latency.

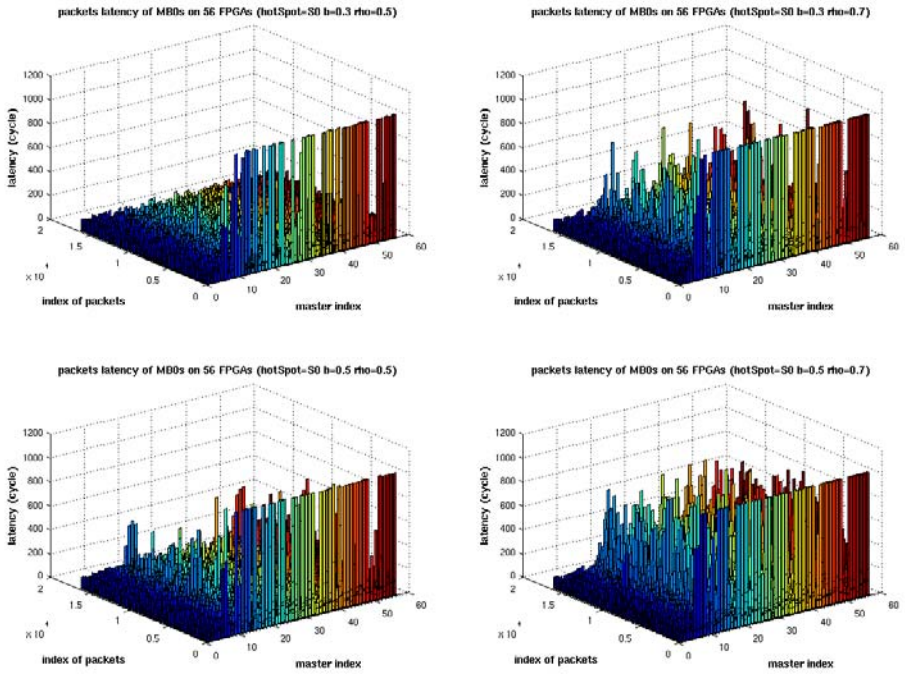


Figure 8.13 Hot spot benchmark packets latency with b = {0.5, 0.3}, ρ = {0.5, 0.7}, and S0 as hot spot

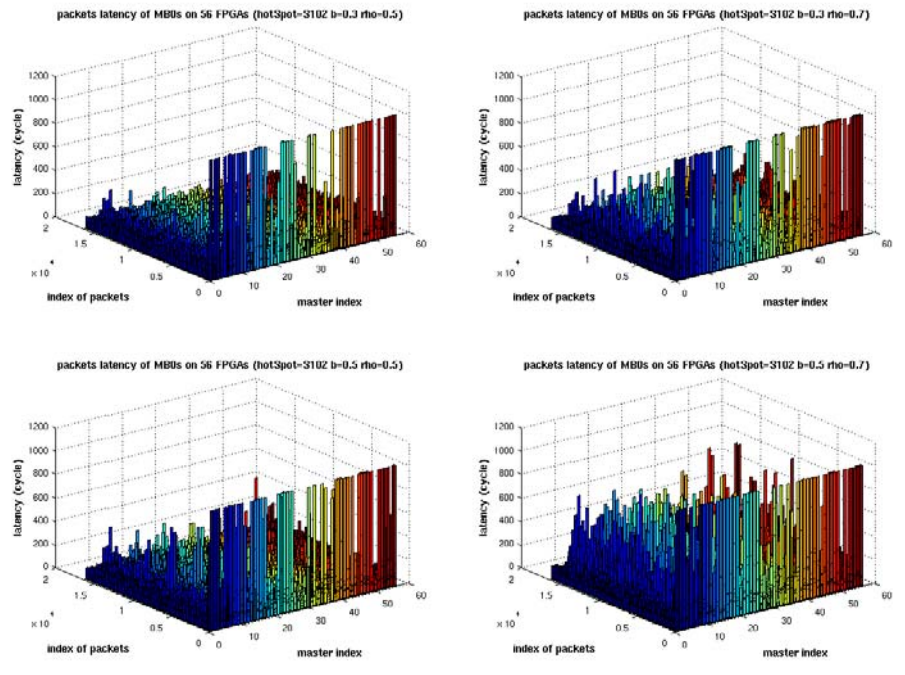


Figure 8.14 Hot spot benchmark packets latency with b = {0.5, 0.3}, ρ = {0.5, 0.7}, and S102 as hot spot

Hot Spot benchmarks are rerun on our 672-core processor. At first, S0 SRAM is still selected as hot spot, while the parameters b and ρ are changed. Parameter b changes from 0.5 to 0.3, changing the traffic from no burstiness to bursty and the fraction of traffic to hot spot changes from 0.5 to 0.7. From figure 8.14, comparing the 2 figures on the top to the other 2 on the bottom, we can see that if there are no bursts in the traffic, the packet latency will increase, as there will be more traffic going into the hotspot. Comparing the 2 figures from left to right, if there are more traffic goes to the hot spot, the packet latency will increase too. Change the hot spot from S0 to S102, we get the same observation from Figure 8.15.

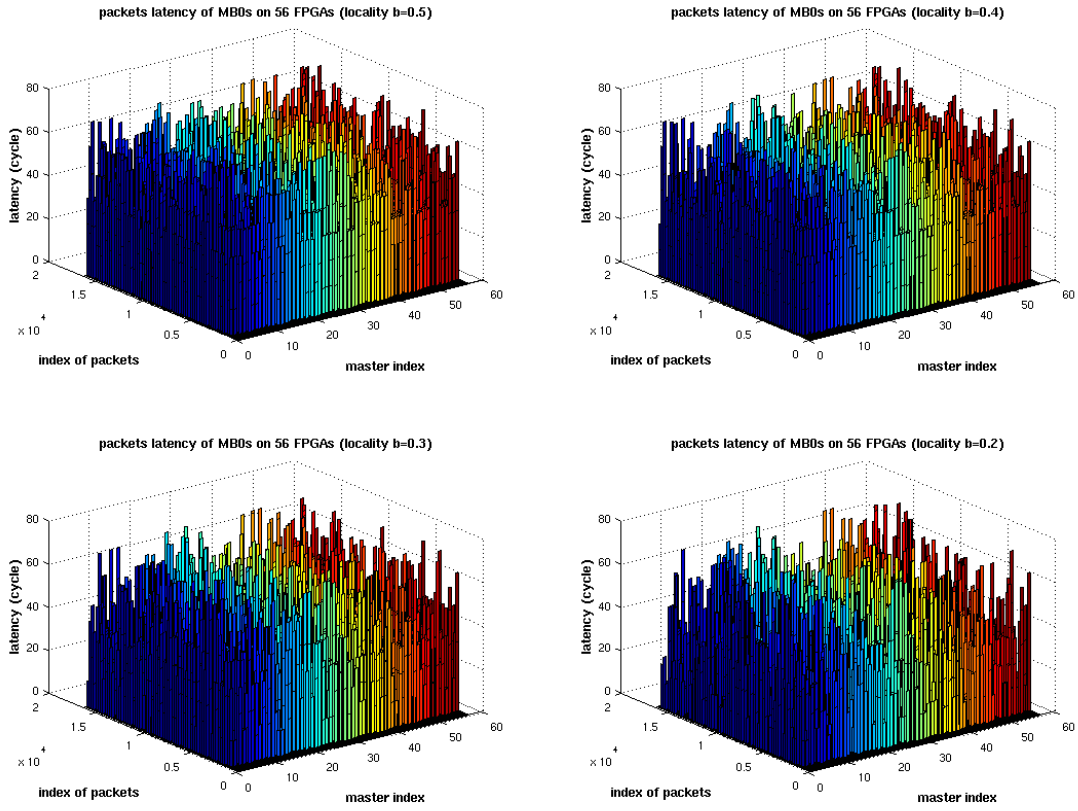


Figure 8.15 Locality benchmark packets latency with b = {0.5, 0.4, 0.3, 0.2}

When comparing Locality benchmarks results with other benchmark, the average packet latency is as it can intuitively been expected the smallest, which is the nature of this scenario. Changing the parameter b from 0.5 to 0.2, the packet latency does not change too much from Figure 1.13. So the locality pattern traffic is not sensitive to burstiness, which suggests us that

bursty traffic should have local memory access to have better communication latency. In a VLSM clearly the impact of parallel programming on communications needs to be very carefully analyzed in order to feedback to the parallel programmer and help tune his application [13-14].

8.4 Comparison between RAMP Blue and our VLSM

According to our best acknowledge, the RAMP blue and our VLSM are the only two multiprocessors which have surpassed the milestone of 512 cores on multi-FPGA platforms. The basic idea of multi-FPGA platform emulation is the same, while the approaching framework is quite different. Although Berkeley decided to go for its own board designs we selected industry-class emulator in order to focus on the methodologies and design flows.

Table 8.4 comparison between RAMP Blue and BB-672

	RAMP Blue	BB-672
Max number of processor	1008	672
FPGA number and type	84 Virtex-II Pro 70	56 Virtex-4 LX200
Emulation frequency	90 MHz	10 Mhz
Processor	MB V4.0 no optional units	MB V6.0 full optional units
OS & compiler	ucLinux & GCC	Standalone & GCC
FPU	64 bits FPU / 12 cores	32 bits FPU / core
DDR memory	250 MB / processor	Not implemented
Monitoring & Debug	Control network Debug interface	Monitoring network Dynamic probes
Network topology	Cluster 3D mesh	Cluster mesh
Interconnection	Crossbar switch + Ethernet	Network on Chip
Communication	Message passing	Distributed shared memory
Language	MPI or UPC	ENSTA proper C driver
Benchmarks	Netpert (for network) NPB class-S (on 256 cores)	OCP-IP micro-benchmarks (on 672 cores)

Design flow	Not reported	Full automatic

Both systems have surpassed the milestone of 512 cores: RAMP Blue has 1008 MicroBlaze processor implemented on 84 Xilinx Virtex-II Pro 70 FPGAs; while our BB-672 has 672 cores on 56 Xilinx Virtex-4 LX200 FPGAs. As there are more hardware resources on Virtex-4 LX200 than Virtex-II Pro70, We can place at most 24 with all full optional units on one FPGA and 12 cores with full optional units and hardware multiplier are placed with our BB-672 project; while there are 12 processor with no optional units in RAMP Blue case. Each processor has one 32 bits FPU (floating point unit) in our case; while all the 12 processors share one 64 bits FPU in RAMP Blue case. In RAMP Blue system, each processor can access up to 250 MB DDR2 memory; while in our system DDR memory has not been implemented until now, which will be used in the next version. Console and control network is used for code, data and user input from NFS or TELNET service as each MicroBlaze run uClinux operating system in RAMP Blue; While in our case, OS is not implemented yet as the high level parallelization service is not used for our program execution, so the execution codes are initiated to FPGA download bit files for standalone execution. JTAG port of each FPGA is used by Xilinx Microprocessor Debugger for software debug and by Xilinx ChipScope for signal tracing; in our BB-672 system ,a dedicate monitoring network is used for communication surveillance and analysis, and dynamic probes can be added into system for signal tracing by the tools of EVE company. Using this monitoring network, we can get the throughput and latency of traffic as shown in the last section, although this network takes almost 10% of FPGA hardware resource. During the debug process, we can trace the signals we want to check out with the dynamic probes, which will slow down the emulation speed.

As presented in section 1, the RAMP Blue network topology is cluster 3D mesh. On each FPGA, 12 processors are connected to one switch as a cluster. The 4 FPGA modules on the same board are connected by the switches as a ring by parallel links. The inter board interconnection is based on MGT XAUI serial links. The FPGA modules are globally connected as a 3D mesh topology. So the interconnection is based both on switch and Ethernet compatible network to realize the message passing mechanism. As complained in [3], the interconnection between IPs is constrained by the inter-FPGA network. Considering also the inter-board interconnection by serial links, the BEE2 multi-FPGA platform is not

very flexible. Designers have to modify their original designs to adapt to the platform interconnection for implementation. In this way, the BEE2 platform is not a full emulation platform for ASIC design and is not representative of current MPSOC design efforts

In contract, the automatic partition and placement technology of EVE Company can make sure that the user's original large size design is implemented onto Zebu multi-FPGA platform without any modification of system architecture. From user's view, it looks like his original design is placed and routed onto a really large FPGA. So there is not any constraint for user's architecture. To achieve this objective, the frequency of system is less than 20MHz until now. Our BB-672 system runs at 10 MHz while RAMP Blue is at 90 MHz. Considering the flexibility of system, which should be one advantage of FPGA platform, the frequency difference of 80 MHz is adaptable for hardware emulation purpose. And our emulation of large OCP-IP benchmarks is still hundreds time quicker than VHDL simulation.

Our BB-672 system is built on the reuse of SSM IP. As presented in chapter 7 and last section, the system topology is still cluster mesh. The interconnection is fully based on Network on Chip technology. The inter-processor communication is based on distributed shared memory. Because of the flexibility of Zebu multi-FPGA platform, it is easy to change our VLSM architecture to other regular topologies including the 3D mesh. We just have to modify the SSM IP output interconnection, and then the whole system can be regenerated and implemented onto Zebu XXL platform automatically. While there is no automatic workflow reported for RAMP project.

Proper drivers for OCP-IP communication protocol are developed and used for our BB-672 system programming. The OCP-IP mirco-benchmarks for NoC are used for the performance measurement as presented in section 2. Processor executes the benchmarks to generate different traffic which pass through the communication architecture based on NoC technology. Common benchmarks can be executed to comparing the two system performance. In fact performance is not the major objective of VLSM framework. In this thesis, we have proposed a fully automatic workflow for system generation and implementation, which can greatly accelerate the large scale multiprocessor design.

8.5 VLSM and Benchmarking: where are the benchmarks ?

VLSM multiprocessor on chip is technically realizable because of the advances in silicon technology. The inter-core communication is considerable faster on a MPSOC than in a multi-node system. New benchmarks are needed which should thoroughly characterize the new communication patterns.

MultiBench 1.0 [12] is proposed by EEMBC (Embedded Microprocessor Benchmark Consortium) as a suite of industrial embedded benchmarks to analyze, test, and improve multicore architectures and platforms. It measures the impact of parallelization and scalability across both data processing and computationally-intensive tasks. This is first generation targets the evaluation and development of scalable SMP (symmetrical multicore processor) architectures with shared memory.

The PARSEC [13] (Princeton Application Repository for Shared-Memory Computers) is a benchmark suite composed of multithreaded programs, which also targets the shared-memory based chip-multiprocessors design. It supports two parallelization models: OpenMP and pthreads. In the second version, the TBB (Intel Treading Building Blocks) is supported.

Both benchmarks presented here target multiprocessor with large size of shared memory, which is not available in the embedded system design. Taking the ‘dedup’ program of PARSEC benchmarks for example, the input set of small problem size has reached 10 MB. While in our multiprocessor system, each processor has only 32 KB local memory. The future benchmarks for embedded multiprocessor design should take the memory limitation as one important criteria. The new version of PARSEC [19] is still not adapted to embedded multiprocessors.

The interface between processor and NoC of our multiprocessor is based on OCP-IP protocol. The cache coherency is not supported until the middle of 2009 as the OCP-IP version 3.0 was released. In this new release, coherency extension is available to support cache coherency of multiprocessor with OCP-IP interface. As our BB-672 system is based on OCP-IP version 2.2, cache coherency cannot be implemented, and that is the reason why we do not use DD2 extern memory as application parallelization on the multiprocessor without hardware cache coherency is considering low. Cache coherency based on OCP-IP version 3.0 has been developed thins its release. A simple directory based cache coherency has done.

Full hardware cache coherency unit will be integrated in the next version of our multiprocessor. A new parallelization flow with automatic parallelization should be supplied with this new version of multiprocessor. The Parallelization is based on MPI. A new NoC synthesis method will be proposed from MPI parallelization to RTL implementation.

An extension of this work will be to realize a multiobjective design space exploration [12, 16] for VLSM. With 2.3 billion transistors and 8 cores the most recent intel microprocessor [20] represent state of the art in 2010 of multicore. We expect that with the dawn of terascale computing VLSM Design Space Exploration is just at the beginning [21].

8.6 Conclusion : EDA vs Computer Architecture

Very large scale multi-processor with 100+ cores is very complex to design and evaluate. Facing the simulation wall, the multi-FPGA platform is the only solution to realize such large system. Using our automatic generation workflow a 672-core multiprocessor can be generated and implemented onto FPGA platform in less than one day. OCP-IP micro-benchmarks are evaluated on our NoC based 672-core. Packet latency results from monitoring network are analyzed and they are useful for architecture and software designers to improve system performance. Comparing to RAMP Blue multiprocessor, our VLSM is more flexible on communication architecture as different topologies can be implemented using our automatic workflow within one day. This new design framework based on multi-FPGA emulation has been validated by our project to help and accelerate large scale multiprocessor hardware and software design. The fundamental issue in our approach is that we have put the emphasis on system level design methodologies and tools rather than on a computer architecture only approach.

We believe that time has come for EDA to move up the scale and integrate 50 years of parallel computer architecture, automatic parallelization compiler and operating system research results to efficiently tackle in a unified and integrated framework the design productivity gap.

Reference

[1] John H. Kelm, Daniel R. Johnson, Matthew R. Johnson, Neal C. Crago, William Tuohy, Aqeel Mahesri, Steven S. Lumetta, Matthew I. Frank, Sanjay J. Patel, **Rigel: An Architecture and Scalable Programming**

Interface for a 1000-core Accelerator, International Symposium on Computer Architecture (ISCA'09), June 2009.

[2] Wawrzynek, J.; Patterson, D.; Oskin, M.; Shin-Lien Lu; Kozyrakis, C.; Hoe, J.C.; Chiou, D.; Asanovic, K.; , "**RAMP: Research Accelerator for Multiple Processors**," Micro, IEEE , vol.27, no.2, pp.46-57, March-April 2007

[3] Njuguna Njoroge, Jared Casper, Sewook Wee, Yuriy Teslyar, Daxia Ge, Christos Kozyrakis, and Kunle Olukotun, **ATLAS: A Chip-Multiprocessor with Transactional Memory Support**, Proceedings of the Conference on Design Automation and Test in Europe (DATE), Nice France, April 2007

[4] Alex Krasnov, Andrew Schultz, John Wawrzynek, Greg Gibeling, and Pierre-Yves Droz, **RAMP Blue: A Message-Passing Manycore System In FPGAs**, Proceedings of International Conference on Field Programmable Logic and Applications, Amsterdam, The Netherlands, August 2007

[5] Andrew Schultz, **RAMP Blue: Design and Implementation of a Message Passing Multi-processor System on the BEE2**, master thesis, University of California at Berkeley 2007

[6] Asanovic K, Patterson DA, Tan Z, Waterman A, Avizienis R, Lee Y. **RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors**. In: The 4th Workshop on Architectural Research Prototyping. Austin, Texas; 2009

[7] Chang, C.; Wawrzynek, J.; Brodersen, R.W.; , "**BEE2: a high-end reconfigurable computing system**," Design & Test of Computers, IEEE , vol.22, no.2, pp. 114- 125, March-April 2005

[8] **OCP-IP Network-on-chip Benchmarking Specification Part2: Micro-benchmark Specification v1.0**, May 23rd; 2008. http://www.ocpip.org

[9] Xinyu Li, Omar Hammami, **BB-762: Design and Implementation of 762 Processor Multiprocessor and OCP-IP Benchmarking**, DATE 2009, university booth.

[10] O. Hammami, X.Li, L.Larzul and L.Burgun, **Automatic Design Methodologies for Large Scale MPSOC and Prototyping on Multi-FPGA Platforms**, International SoC Design Conference (ISOCC) 2009, invited talk, Nov. 22-24 2009, South Korea.

[11] Xinyu Li and Omar Hammami, **Multi-FPGA emulation of a 48-cores multiprocessor with NOC**, Design and Test Workshop, 2008. IDT 2008. 3rd International, 20-22 Dec. 2008

[12] Xinyu Li and Omar Hammami, "**An Automatic Design Flow for Data Parallel and Pipelined Signal Processing Applications on Embedded Multiprocessor with NoC: Application to Cryptography**," International Journal of Reconfigurable Computing, vol. 2009, Article ID 631490, 14 pages, 2009.

[13] Riad Ben Mouhoub; Omar Hammami; **NOC Monitoring Feedback for Parallel Programmers Circuits and Systems**, 2006 IEEE North-East Workshop on .

[14] Riad Ben Mouhoub; Omar Hammami;**NOC Monitoring Hardware Support for fast NOC Design Space Exploration and Potential NOC Partial Dynamic Reconfiguration**", IES 2006.

[15] Mouhoub, R.B.; Hammami, O.; **Multiprocessor on chip: beating the simulation wall through multiobjective design space exploration with direct execution** , Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International

[16] R;Benmouhoub and O.Hammami, "**MOCDEX: Multiprocessor on Chip Multiobjective Design Space Exploration with Direct Execution**", EURASIP Journal of Embedded Systems, 2006.

[17] **EEMBC, MultiBench™ 1.0 Multicore Benchmark Software** , http://www.eembc.org/benchmark/multi_sl.php

[18] Christian Bienia and Sanjeev Kumar and Jaswinder Pal Singh and Kai Li, **The PARSEC Benchmark Suite: Characterization and Architectural Implications**, Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, October 2008

[19] C. Bienia and K. Li. PARSEC 2.0: A **New Benchmark Suite for Chip-Multiprocessors** In Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation, June 2009.

[20] Rusu, S. Tam, S. Muljono, H. Stinson, J. Ayers, D. Chang, J. Varada, R. Ratta, M. Kottapalli, S. Vora, S. **A 45 nm 8-Core Enterprise Xeon¯ Processor**, Solid-State Circuits, IEEE Journal of Volume: 45 Issue:1 page(s): 7 – 14, Jan 2010.

[21] [21] Rattner, J.; **The dawn of terascale computing**, Solid-State Circuits Magazine, IEEE Volume: 1 , Issue: 1 2009 , Page(s): 83 – 89

9. Conclusion and Perspective

The final part concludes the thesis by summarizing all the proposed methodologies for MPSoC design and major results. An overview of future research directions are given for future large scale MPSoC design. And the major contributions are listed.

9.1 Summary of the Thesis

In this thesis we present a new methodology for large scale MPSOC design to resolve the design challenges of complexity and productivity. To achieve this object, 3 strategies are proposed and used:

1. Combination of different system design levels: from TLM level to FPGA emulation
2. Reuse of IPs and components: extend SSM IP to 48-core and 672-core MPSoC
3. Utilization of new technologies: Arteris NoC, M2000 eFPGA, EVE multi-FPGA platform.

A survey of until recent work on NoC and MPSoC design shows that there is no mature design workflow for the future large scale MPSoC. The interconnection architecture of MPSoC greatly impacts the system performance. Three different interconnection methods have been used: bus, crossbar and network on chip (NoC). And NoC is proposed as the only solution for future large scale MPSoC design. Arteris NoC design tools are used as industrial support for our design.

Real time constrains must be considered during MPSoC design. Homogeneous and heterogeneous multiprocessors are two important and distinct branches of MPSoC design. Analysis and comparison of different MPSoC design methodologies help to better understand different design approach and overcame their shortcomings. The mapping of core graph to NoC topologies is well known NP-Hard, only heuristic algorithms can be used to approach

the optimal solution according to different objective functions. Different approximation algorithms are proposed to reduce the execution time of ILP problem.

Design space exploration of network-on-chip can be conducted at multiple levels of abstraction from transaction level modeling down to emulation. Although, each level brings its own benefits multiple constraints may push for a given level of abstraction. At first a fully automatic design flow for network on chip at TLM level is proposed. Combining this flow with our following emulation work will allow fully integrated solutions.

Reconfigurable network on chips require efficient reconfigurable hardware support in ASIC environment. The emerging eFPGA IPs allow the integration of reconfigurable area in ASIC devices. The organization and dimensioning of this area is an important issue to be tackled in order to maximize the efficiency of network on chip mapping. Our linear programming methodology provides a solution to this problem.

Next generation MPSoC will be based on hundreds of processors. MPSoC design is very complex and in order to reach efficient working silicon in reasonable time we propose a small scale multiprocessor design as a building block (soft IP) for large scale multiprocessor. We proposed a Cluster-Mesh NoC based small scale multiprocessor IP which have been fully prototyped on a large scale FPGA chip. Building large scale multiprocessors from the proposed SSM IP can be fast as the main design effort resides in the connection and adaptation of NOC addressing.

We validated our approach on a 48 processors system by automatically extending our 12 processors SSM IP and finally extend to a 672-core MPSoC on EVE Zebu-XXL multi-FPGA platform. Different industrial design tools are combined into our automatic design flow.

We have proposed an automatic design flow for data parallel and pipelined signal processing applications on embedded multiprocessor with NoC using cryptographic application TDES as an example. Our flow explores through execution on multi-FPGA emulation for parallel software implementation with task placement exploration and task granularity analysis. A hardware based NoC monitoring drives the task placement process to reduce communication bottlenecks. In the second phase, high level synthesis generated hardware accelerators are added to explore the tradeoff in area-performance while still privileging multiprocessor basis for the implementation.

OCP-IP has proposed a suite of micro-benchmarks for network on chip benchmarking. We evaluate through actual execution of OCP-IP benchmarks on large scale multiprocessors with network on chip. And traffic information and statistics are obtained from our hardware monitoring network.

9.2 Future Research

Large scale MPSoC design is a new and open research area from software programming to hardware design. In the thesis, approaches are proposed and can be extended in several different research directions.

Energy consumption is very important for nowadays electronic system not only for commercial reason but also for earth protection. The energy consumption model library can be integrated into our workflow for power-area- performance multi-criteria system on chip design. Combining this flow with our previous work at RTL level will allow a fully integrated solution.

Large scale MPSoC exploration is very complex and takes a long time. Take advantage of the combination of different levels from TLM to RTL will prune exploration space at fast evaluation level to accelerate exploration. Neural network theory estimation and the PR technology of FPGA can also achieve this goal.

Add reconfigurability management of dedicated reconfigurable eFPGA area for hardware accelerators as well as automatic parallelization is the future work too.

9.3 Contribution

- The main contribution is our automatic design flow for large scale MPSoC design based on the reuse of SSM IP. Based on this, an automatic design flow for data parallel and pipelined signal processing applications on embedded multiprocessor with NoC for cryptographic application TDES. High level synthesis is added to generated hardware accelerators, which are added to explore the tradeoff in area-performance while still privileging multiprocessor basis for the implementation.

- A lot of work has been done to summarize the state-of-art NoC and MPSoC design flow. Analysis and comparison of different MPSoC design methodologies help to better understand different design approach and overcame their shortcomings. The same case for eFPGA and reconfigurable NoC design.
- We propose a fully automatic multi-objective design workflow for network on chip at TLM level. The timing and area criteria from RTL level are explored but not limited using the TLM NoC models of NoCexplorer.
- A linear programming methodology is provided as a solution to the problem: organization and dimensioning of eFPGA reconfigurable area to maximize the efficiency of network on chip mapping.
- We propose to reuse and extend SSM IP multiprocessor to large scale MPSoC as a solution for design complexity and productivity.
- We first evaluate through actual execution OCP-IP benchmarks on large scale multiprocessors with network on chip using multi-FPGA emulation platform.

All the work done in this thesis is the basis of “MPSOC explorer”, an ongoing industrial project for large scale MPSoC design exploration supported by European Union and French government.



Thèse préparée dans Laboratoire Electronique et Informatique d'ENSTA