Design and Multi-Technology Multi-objective Comparative Analysis of Families of MPSOC

Zhoukun Wang

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'Institut polytechnique de Grenoble

Spécialité : *« Micro et Nano-Electronique »*

préparée au laboratoire _GIPSA-lab et ENSTA ParisTech UEI-lab

dans le cadre de **l'Ecole Doctorale** *« Electronique, Electrotechnique, Automatique & Traitement du Signal »*

présentée et soutenue publiquement

par

_____Zhoukun WANG_____

le ____12.Novembre.2009__

Conception et Analyse Comparative Multi-objectif Multi-Technologies de Famille de Multiprocesseurs sur Puce

DIRECTEUR DE THESE : Dominique HOUZET
CO-DIRECTEUR DE THESE : Omar HAMMAMI

JURY

M.	Kees	GOOSSENS	, Rapporteur
M.	Lionel	TORRES	, Rapporteur
M.	Dominique	HOUZET	, Directeur de thèse
M.	Omar	HAMMAMI	, Co-Directeur de thèse
M.	Marcello	COPPOLA	, Examinateur
M.	Ian	O'CONNOR	, Examinateur

Acknowledgement

Je tiens d'abord à exprimer ma profonde reconnaissance à mon co-directeur de thèse, le Professeur Omar Hammami, professeur de l'ENSTA ParisTech, pour toute la confiance qu'il m'a témoignée et toute sa disponibilité à diriger cette thèse. Je le remercie également pour les longues discussions techniques, pour les précieux conseils qu'il m'a donnés, et pour tout ce qu'il m'a appris pendant cette thèse et lors de la rédaction de ce manuscrit. Je le remercie enfin de m'avoir permis d'intégrer son équipe.

Je remercie vivement mon directeur de thèse, le Professeur Dominique Houzet, professeur de l'INPG, pour tous ses conseils et discussions techniques.

Je remercie Monsieur Kees Goossens, professeur de the Delft University of Technology, et Monsieur Lionel Torres, professeur de l'Université Monpellier II, qui ont accepté d'être les rapporteurs de ma thèse de doctorat.

Je remercie Monsieur Ian O'connor, professeur à l'école centrale de Lyon, et Monsieur Marcello Coppola, qui ont bien voulu en être les examinateurs.

Je remercie, bien évidement, Xinyu LI, Guangye Tian, Mazen Khaddour, Muhammad Imran Taj et tous mes amis, collègues et doctorants de l'ENSTA.

Enfin, et surtout, je remercie ma famille pour son soutien, sa compréhension et ses encouragement. Je présente ici mes reconnaissances à ma chère épouse Jiangdong! Celle qui me comprend et me soutient dans les moments les plus difficiles, surtout pendant les derniers mois de la rédaction de ce manuscrit.

Résumé

Les systèmes multiprocesseurs sur puce (MPSoC) ont fortement émergé durant la dernière décennie en matière de communication, multimédia, réseaux et d'autres domaines embarqués. Le MPSOC est devenu un nouveau paradigme à haute performance pour la conception d'applications embarquées. Cette thèse aborde la conception et l'implémentation physique d'un réseau sur puce (NoC) sur multiprocesseur System on Chip. Nous avons étudié plusieurs aspects de la conception à des stades différents: la synthèse de haut niveau, la conception architecturale, la mise en œuvre sur FPGA, ASIC et l'évaluation de mise en œuvre physique. Nous avons analysé et présenté les impacts de ces aspects sur la performance finale des MPSOC, la consommation de puissance et le coût en surface.

Nous avons implémenté une famille de MPSoC par prototypage sur FPGA. Un exemple de 16 processeurs (PE) basé sur un système embarqué à base de NoC sera introduit. Trois NoC fournissent des fonctionnalités différentes pour les 16 tuiles PE. Le Data-NOC relie les tuiles PE et les mémoires DDR2 avec une grande bande passante ; le Synchronisation -NOC offre deux modes de synchronisation pour les tuiles de PE. Les utilisateurs peuvent vérifier et configurer les adresses des blocks IP connecté au Data-NOC grâce à notre Service-NOC. Nous avons également montré l'utilisation de notre système de suivi des performances pour le déboguage et le réglage du logiciel embarqué. Avec les FIFO bi-synchrone, notre architecture GALS résout avec succès le problème de distribution du signal d'horloge et permet que chaque domaine d'horloge

puisse fonctionner à sa fréquence propre. D'autre part nous avons mis en œuvre avec succès les algorithmes AES et TDES de chiffrement sur cette plate-forme et les résultats montrent une accélération linéaire en temps de calcul.

La partie réseau de notre architecture a été mis en œuvre sur la technologie ASIC et a été explorée avec des contraintes temporelles différentes et des bibliothèques de différentes catégories de technologies 65nm et 45nm de STMicroelectronics. Les résultats expérimentaux sur ASIC et FPGA sont comparés, et nous avons discuté l'impact du changement de technologie sur la programmation parallèle.

Une accélération matérielle de systèmes embarqués à base de synthèse de haut niveau basée sur le langage C a été proposée pour s'attaquer à l'augmentation du temps de conception et à la pression du marché ainsi qu'à la complexité croissante des systèmes sur puce (SoC). Dues à la sélection des outils et à différents jeux d'options de synthèse et de placement/routage, de nombreuses solutions de bas niveau en termes de superficie et de fréquence peuvent être produites et doivent être prises en compte au plus haut niveau d'abstraction. Nous effectuons ici une évaluation quantitative des coûts/performances de la synthèse de haut niveau basée sur le C d'accélérateurs matériel de co-traitement. Plusieurs résultats expérimentaux sont présentés pour montrer l'impact de divers outils de synthèse (SystemC Agility, Handel-C, ImpulseC) et l'impact des sélections d'options dans le cadre du contexte général des SOC.

Mots clés: Network-on-Chip, système multiprocesseurs sur puce, synthèse de haut niveau, FPGA, ASIC

Titre français : Conception et Analyse Comparative Multi-objectif Multi-Technologies de Famille de Multiprocesseurs sur Puce

Abstract

Multiprocessor System on Chip (MPSOC) has strongly emerged in the past decade in communication, multimedia, networking and other embedded domains. MPSOC became a new paradigm of high performance embedded application design. This thesis addresses the design and the physical implementation of a Network on Chip (NoC) based Multiprocessor System on Chip. We studied several aspects at different design stages: high level synthesis, architecture design, FPGA implementation, application evaluation and ASIC physical implementation. We try to analysis and find the impacts of these aspects for the MPSOC's final performance, power consumption and area cost.

We implemented a family of MPSoC on FPGA prototyping. An example of 16 processors based on NoC embedded system will be introduced. Three NoCs provide different functionalities for sixteen PE tiles. The cascading Data-NoC connects PE Tiles and DDR2 memories with a high bandwidth; synchronization-NoC offers two synchronization modes for PE tiles. And users can check and configure IPs of Data-NoC through our service NoC. We also demonstrated the use of our performance monitoring system for software debugging and tuning. With the bi-synchronous FIFO method, our GALS architecture successfully solves the long clock signal distribution problem and allows that each clock domain can run at its own clock frequency. On the other hand we successfully implemented AES and TDES block cipher cryptographic algorithms on this platform and results show linear speedup in computation time.

The network part of our architecture has been implemented on ASIC technology and has been explored with different timing constraints and different library categories of STmicroelectronics' 65nm/45nm technologies. The experimental

results of ASIC and FPGA are compared, and we conducted the discuss of technology change impact on parallel programming.

C-based hardware-accelerated embedded system has been proposed to tackle the increasing time-to-market pressure and the growing complexity of system on chip (SoC). Due to tools selection and different set of synthesis, place and route options, numerous low level solutions in term of area and frequency can be produced and must be considered in high abstraction level. we conduct a quantitative area-performance evaluation of C-based high level synthesis of hardware-accelerator co-processing. Several experimental results are presented to show the impact of various C-based synthesis tools (SystemC Agility, Handel-C, ImpulseC) and the impact of option selections in the context of complete SOC environment.

Key words: Network-on-Chip, multiprocessor system on chip, high level synthesis, FPGA, ASIC

English Title: Design and Multi-Technology Multi-objective Comparative Analysis of Families of MPSOC

Table of Contents

Table of Figure

List of Table

Version française

Conception et Analyse Comparative Multi-objectif Multi-Technologies de Famille de Multiprocesseurs sur Puce

Gordon E. Moore affirme que le nombre de transistors sur une puce double environ tous les deux ans, maintenant communément appelée loi de Moore. La dimension de l'élément de transistor devrait continuer à diminuer. Une réduction de 70% en dimensions linéaires des transistors en passant à un nouveau procédé de fabrication (par exemple de 65 nm à 45 nm) permet un facteur 2 d'augmentation de la densité de puces.

Comme le montre la figure 1, l'International Technology Roadmap for Semiconductors (ITRS) de 2008 prévoit que des puces à plusieurs milliards de transistors apparaîtront en production d'ici la fin de cette décennie.

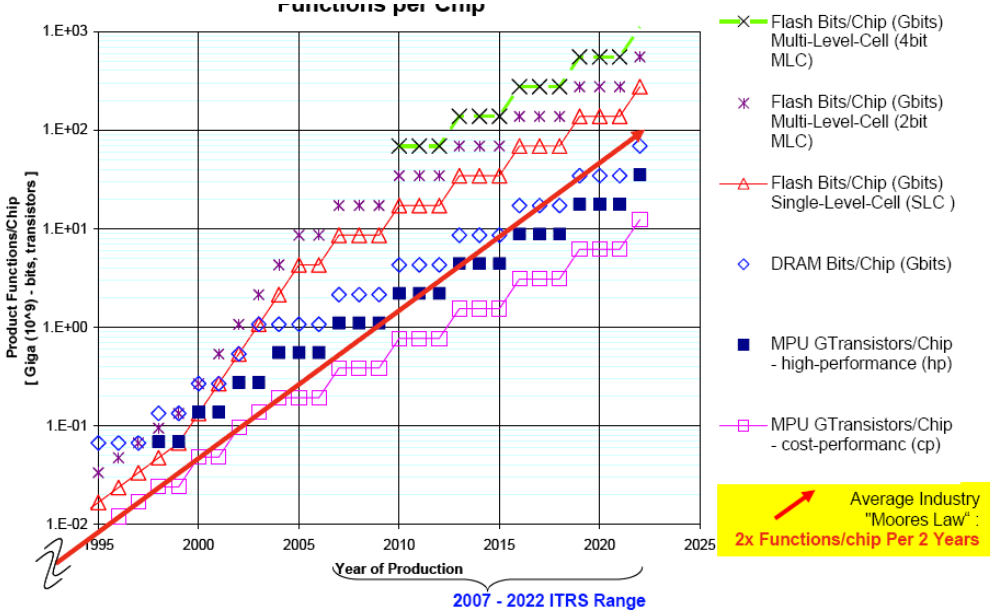


Figure 1 : évolution des technologies (2008 ITRS)

Actuellement les technologies submicronique (DSM) sont le procédé utilisé dans la plupart des VLSI et ULSI. La taille de gravure peut augmenter la dimension effective des ASIC, et introduire de nouvelles applications, mais aussi aggraver les problèmes actuels de conception VLSI / ULSI, et d'ailleurs en introduire de nouveaux. Récemment, l'équipe de recherche Intel Tera-Scale Computing a mentionné dans un livre blanc que «les questions d'énergie thermique, telles que la dissipation de la chaleur provenant des transistors de plus en plus denses, ont commencé à limiter la vitesse à laquelle la fréquence du processeur peut également être augmenté. Bien que les hausses de fréquence ont été un aliment de base de conception pour les 20 dernières années, les 20 prochaines années, il faudra une nouvelle approche. Fondamentalement, l'industrie a besoin de développer l'amélioration des micro-architectures à un rythme plus rapide et en coordination avec chaque nouveau processus de fabrication de silicone, de 45 nm à 32 nm et au-delà ».

La performance est toujours l'une des caractéristiques les plus importantes pour la mesure de la conception VLSI. Nous ne pouvons plus simplement augmenter la fréquence d'horloge à la même vitesse que nous avons fait dans le passé afin d'augmenter les performances. L'énergie et les exigences thermiques commencent à surpasser les avantages que les fréquences d'horloge plus rapides offrent. Toutefois, en raison de la trajectoire de la loi de Moore qui se poursuivra dans la prochaine décennie, nous prévoyons de continuer à doubler les transistors tous les 18-24 mois pour les années à venir. L'exécution parallèle dans des conceptions multi-coeurs nous permettra ensuite de tirer parti de ces densités de transistors accrues pour fournir des performances accrues.

Les système multiprocesseur sur puce (MPSoC) ont fortement émergé durant la dernière décennie en matière de communication, multimédia, réseaux et d'autres domaines embarqués. Le MPSOC est devenu un nouveau paradigme de haute performance pour la conception d'applications embarquées.

Un MPSOC est une agrégation d'un système sur puce et d'un multiprocesseurs traditionnel. Mais nous ne pouvons pas appliquer directement le modèle scientifique de super-calculateur pour les systèmes sur puce. Ce n'est pas tout simplement des

processeurs multiples et sous-systèmes de matériels périphériques réduits à une seule puce de silicium. Les MPSOCs ont été conçus pour satisfaire les exigences des applications embarquées.

Les trois plus importantes caractéristiques communes de ces applications embarquées sont la haute performance, le temps réel et la faible puissance.

Les performances plus élevées sont toujours demandées par les consommateurs, et cela encourage la recherche et le développement de plates-formes à haute performance, pouvant satisfaire les nouvelles exigences et les nouvelles normes. Des performances supérieures impliquent des moyens et des algorithmes de plus en plus complexes qui ne peuvent être réalisés par du matériel simple ou des SoC monoprocesseur. Les MPSOCs ont été conçus à cet effet.

L'nformatique temps réel est bien plus qu'un simple moyen de calcul de haute performance. Dans le calcul interactif traditionnel, nous nous soucions de la vitesse, mais pas des délais. La plupart des systèmes embarqués se soucient des performances moyennse, mais aussi que les tâches soient accomplies dans un délai donné. Les architectures MPSoC doivent être prévisibles. Les applications qui peuvent fonctionner avec des performances prévisibles forment l'architecture MPSoC.

La consommation électrique est une autre contrainte importante pour la conception de MPSoC. Ces contraintes sont beaucoup plus strictes dans les systèmes MPSoC que pour les supercalculateurs traditionnels ou les systèmes informatiques de bureau. Réduire la consommation électrique peut prolonger la vie de la batterie d'un MPSoC, et l'énergie limitée fournie par la batterie nécessite une réduction de la consommation d'énergie des MPSoC des que possible lors de la conception. Même pour les MPSOC sans batterie, la faible puissance est également requise du fait de la chaleur des puces et pour des raisons de coût.

Les contraintes de puissance et d'énergie des MPSoC doivent être prises en compte à chaque niveau d'abstraction.

Dans les dernières années, beaucoup de papiers ont été publiés sur les MPSoC. Beaucoup de ces recherches se soucient de la simulation, et s'arrêtent à la simulation de

haut niveau d'abstraction. À un haut niveau d'abstraction, la simulation peut rapidement réaliser et vérifier la fonctionnalité du système et de l'algorithme. Lors de la simulation au niveau transfert de registres on peut vérifier la correction du code RTL et évaluer la performance du système avec au niveau cycles. La simulation peut également aider le partitionnement logiciel / matériel. Pourquoi dans cette thèse nous nous soucions de prototypage et d'émulation réelle et non seulement de simulation ?

L'émulation avec prototypage et la réalisation effective peuvent fournir des mesures de performances réelles. Les performances d'un système peuvent être mesurées en fournissant le nombre de cycles, mais la simulation ne peut pas fournir les mesures avec le timing précis. La fréquence définie dans la simulation n'a pas de sens, nous ne pouvons pas obtenir la fréquence maximale du système par la simulation. Un MPSoC est un système temps-réel de haute performance. L'émulation et la mise en œuvre sont nécessaires pour mesurer la période minimale d'horloge avec prototypage réel.

L'émulation par prototypage et la réalisation effective peuvent mesurer la surface d'utilisation de silicium. Les résultats de mise en œuvre peuvent nous donner ment l'utilisation de cellules pour les cibles ASIC, et le pourcentage d'utilisation des FPGA. D'autres questions importantes peuvent être atteintes avec la mise en œuvre, telles que des contraintes de nombres de broches, le floorplaning, la consommation d'énergie.

Conception et Implémentation de MPSOC

Avec l'avènement des nouvelles technologies de conception de circuits intégrés, le MPSoC est apparu comme une solution prometteuse à la complexité grandissante et la fonctionnalité croissante des systèmes embarqués sur puce. Le grand nombre de processeurs et modules requièrent une dorsale de communication qui assure la flexibilité, l'évolutivité et la qualité de service (QoS) garantie. Les moyens classiques de l'interconnexion, tels que les bus et les crossbars, ne peuvent satisfaire à ces exigences en raison des contraintes de surface, de temps et de consommation d'énergie. On-Chip Network (OCN), ou réseau sur puce (NOC), a été proposé comme une approche systématique pour traiter le défi de la conception centrée sur la communication.

Contrairement aux méthodes traditionnelles de raccordement, la structure modulaire de l'architecture multiprocesseur à base de NoC rend hautement évolutive et améliore la fiabilité et la fréquence de fonctionnement des modules sur puce. En outre, l'approche NoC offre des possibilités incomparables pour mettre en œuvre de manière Globalement Asynchrone, Localement Synchrone (GALS) la conception, qui font de la distribution d'horloge et du temps-réel des problèmes plus gérable. Pour réduire la pression du temps de mise sur le marché et face à la complexité croissante des systèmes sur puce, la nécessité de prototypage rapide est en pleine croissance. Prenant avantage de la technologie submicronique profonde, les FPGA modernes offrent un prototypage rapide et à faible coût avec des moyens logiques de grandes et hautes performances.

Nous décrivons ici notre système multiprocesseur comprenant des tuiles de seize processeurs, une mémoire partagée synchronisée et quatre banques de mémoire DDR2. Toutes ces tuiles de processeurs sont reliées à des esclaves DDR2 grâce au Data-NOC et sont synchronisés à l'aide du Synchronisation-NoC. Un réseau de service est également intégré dans le système pour fournir des fonctionnalités comme la gestion d'erreur, vérifier l'état des blocs IP et les services de reconfiguration à l'exécution. Les NoC sont implémentés en utilisant la bibliothèque Arteris Danube, qui est basée sur le routage et la commutation par paquets wormhole. L'observabilité au moment de l'exécution est une nécessité pour le déboguage et le contrôle de systèmes embarqués, ce qui remet en cause la conception de SoC basée sur les NoC. La surveillance des performances à l'exécution mise en œuvre dans notre système fournit des informations de comportement du NoC, comme le débit, la latence, et un soutient au déboguage et réglage des applications et logiciels. Le MPSoC est une conception GALS réalisée avec l'aide de FIFO bi-synchrones. L'adaptateur NoC contenant une paire de FIFO bi-synchrone a été inséré entre le processeur et le NoC, pour lutter contre le problème de communication entre deux domaines d'horloge différents (domaine d'horloge du processeur et le domained' horloge du NoC).

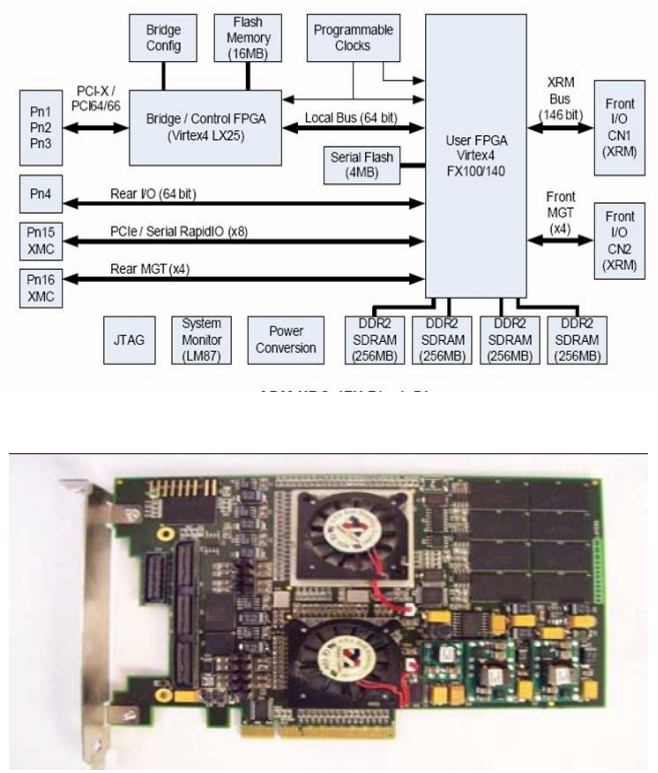


Figure 2 plate-forme Alpha-Data FPGA ADPe-XRC-4

Le système multiprocesseur est mis en œuvre sur plate-forme Alpha-Data à base de carte FPGA ADPE-XRC-4. Comme le montre la figure 2, l'ADPE-XRC-4 est une performance reconfigurable à base de carte PCI Express sur la base des FPGA Xilinx Virtex-4 FX140. Le FPGA est connecté à quatre banques de mémoire DDR2, tandis que chaque banque DDR2 256Mbytes se compose de deux MT47H64M16 de Micron. Le MPSoC est synthétisé, mis en oeuvre par les outils Xilinx: EDK 9.2 et ISE 9.2, et les résultats d'utilisation des ressources des FPGA sont présentés au tableau 1. Le système multiprocesseurs utilise environ 90% des tranches de FPGA et 65% de blocs de RAM. Chaque MicroBlaze prend 3 blocs DSP48, tandis que 15 processeurs MicroBlaze utilisent 45 blocs DSP48. Les Digital Clock Managers (DCM) sont distribués au sein du FPGA pour générer des fréquences d'horloge différentes pour les différents domaines d'horloge. 16 tuiles PE, le Data-NoC, le Synchronisation-NoC et les DDR2 sont séparés dans 19 domaines d'horloge et nécessitent en tout 19 DCMS. En ce qui concerne

l'utilisation des tranches de FPGA (slices), la figure 3 donne le pourcentage d'utilisation de tranches. 16 tuiles PE prennent environ une moitié des tranches au total. Le Data-NOC et le Synchronisation-NoC occupent environ 29,5% et 9,5% respectivement. Environ 8% des tranches sont utilisées pour les contrôleurs DDR2.

Nombre de DCM_ADVs	19 parmi 20	95%
Nombre de DSP48s	45 parmi 192	23%
Nombre de RAMB16s	357 parmi 552	65%
Nombre de Slices	57344 parmi 63168	90%
Nombre de PPC405s	1 parmi 2	50%

TABLE 1 resources FPGA

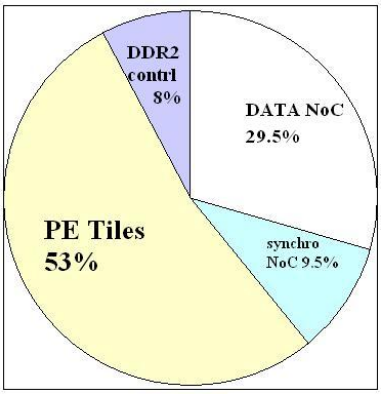


Figure 3 pourcentage d'utilisation de tranches

	Nombre	Surface (slices)	Pourcentage
Master NIUs	16	3504	16%
Slave NIUs	4	1740	8%
Statistic collectors	5	9976	45%
Switches	16	6896	31%

TABLE 2 resources du data-NOC

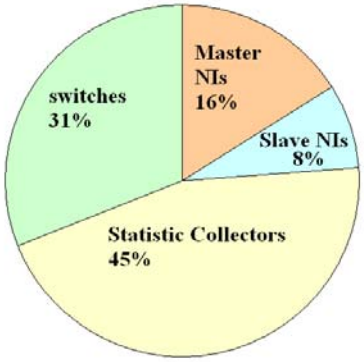


Figure 4 the pourcentage de surface du Data-NOC

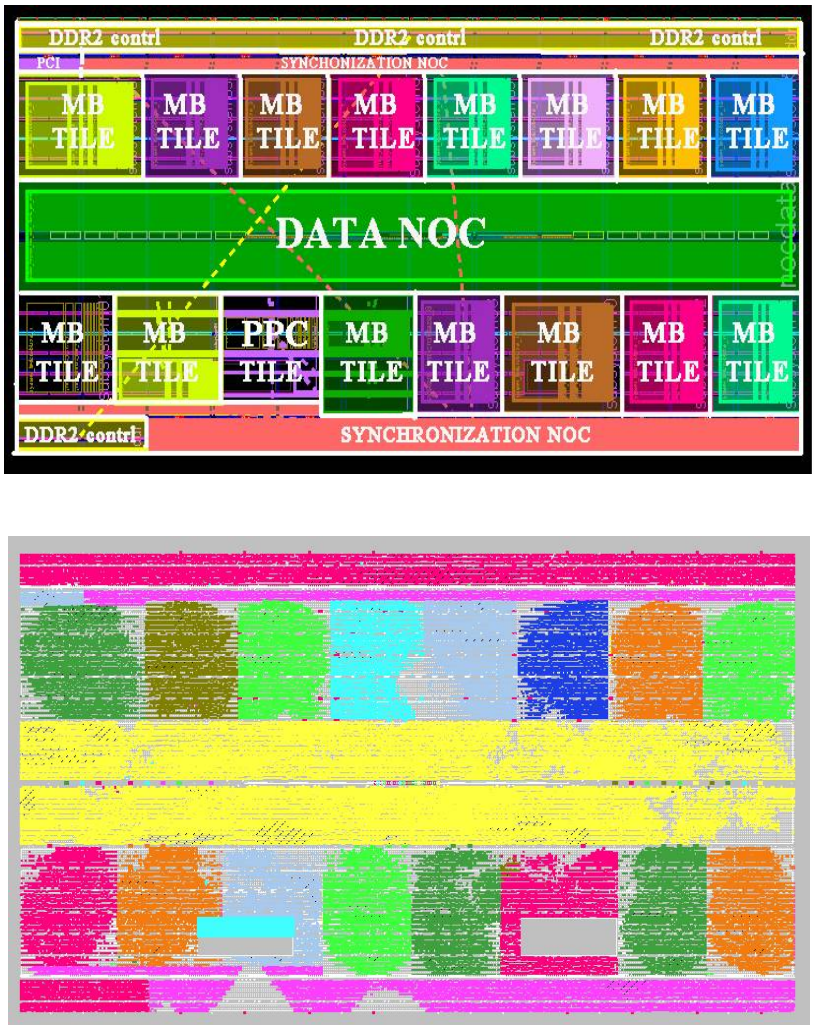


Figure 5 Floorplaning at placement FPGA de la platforme

Nous présentons ici une mise en œuvre d'un FPGA NoC-multiprocesseur de système embarqué. Trois NoC fournissent des fonctionnalités différentes pour les tuiles de seize PE. Le Data-NOC relie les Tuiles PE et les mémoires DDR2 avec une bande passante élevée, le synchronisation-NOC offre deux modes de synchronisation pour les tuiles de PE.

Evaluation à base d'application de cryptographie

La sécurité des données joue un rôle important dans la conception des systèmes embarqués aujourd'hui, et de nombreuses applications intégrées s'appuient fortement sur le mécanisme de sécurité. Les deux algorithmes de chiffrement couramment utilisés sont: AES (Advanced Encryption Standard) et le TDES (Triple Data Encryption Standard) qui ont été choisis comme application pour évaluer notre plate-forme multiprocesseurs.

Une mise en oeuvre commune qui favorise les performances de ces algorithmes se base sur les LUT où un ou plusieurs des opérations de l'algorithme sont pré-calculées pour toutes les entrées possibles et stockées dans des tables (LUT). Nous utilisons cette solution dans notre document car elle est simple à mettre en oeuvre, rapide et elle s'intègre dans notre architecture (le domaine est déjà fixé par le choix des éléments de traitement).

Le TDES est construit de manière à ce que le cryptage et le chiffrement utilisent la même fonction, mais avec un ordre inverse dans les tours de clés, ce qui conduit à un faible encombrement. L'AES, par contre se base sur un cryptage et décryptage avec des fonctions différentes ce qui signifie que l'AES a un encombrement plus grand en termes de code, où 2 fonctions doivent être mises en œuvre au lieu d'une pour le chiffrement / déchiffrement, et en termes de taille des tables de choix car nous avons besoin de tables différentes pour ces deux opérations, mais l'AES est plus rapide en temps d'exécution et fournit une meilleure sécurité.

Nous avons exploré deux implémentations du TDES sur notre MPSoC à 16 processeurs en utilisant deux méthodes: la première est la division des données entre les processeurs et la seconde est en divisant l'exécution du TDES. Notre architecture mets en oeuvre des éléments de traitement connectés via deux NoC.

Les résultats montrent de meilleures performances pour le parallélisme de données, favorisé par l'architecture, ce résultat peut être inversé dans une architecture pipeline.

MPSoC ASIC Design

Les Multiprocessor Systems-on-Chip (MPSoCs) sont devenus la norme pour les systèmes intégrés. Les modes traditionnels d'interconnexion, tels que les bus et les crossbars, ne peuvent pas satisfaire aux exigences MPSOC de performance, de taille ainsi que d'évolutivité, et de fiabilité. Les On-Chip Network (OCN) ou réseau sur puce (NOC), ont été proposés comme une approche systématique pour traiter le défi de la conception centrée sur la communication. La structure modulaire de l'architecture multiprocesseur NoC rend hautement évolutive et améliore la fiabilité et la fréquence de fonctionnement des modules sur puce. Bien que ces avantages clés des NoC ont été largement discutées, l'application pratique des NoC dans les technologies submicroniques profondes (65 nm et au-dessous) est encore un défi ouvert.

Des recherches récentes ont porté sur les méthodes de synthèse efficaces de NoC et des comparaisons avec des systèmes sur puce à base de bus, ainsi que la conception de chaînes d'outils et d'architectures à base de NoC pour applications spécifiques pour des modes de communication connus. Nous bâtissons sur ces travaux antérieurs pour résoudre le problème de conception de topologie de NoC, prenant en compte l'impact des outils au niveau des technologies 65 nm et 45 nm. Par rapport aux travaux précédents, ici on ajoute plusieurs études et expérimentations sur des questions comme l'impact du changement de technologie sur les performances entre 65 nm à 45 nm ainsi que la migration du FPGA à l'ASIC. Nous utilisons ici un flot de conception complet et

intégré, avec des chaînes d'outils standard industriel pour réaliser des implémentations physiques exactes des NoC.

Nous avons des résultats d'expérience d'exploration totalement opérationnelles sur différentes configurations de NoC sur des technologies submicroniques avec FPGA et ASIC 65-nm/45-nm.

FPGA-ASIC exploration

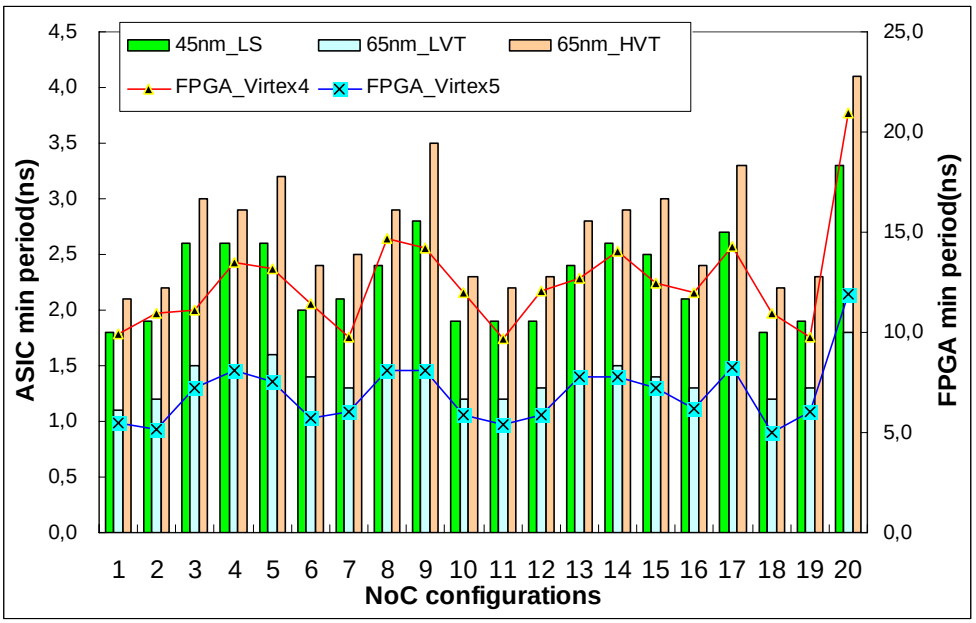


Figure 6 ASIC-FPGA exploration en surface

Nous avons effectué de nombreuses explorations pour différentes configurations de notre NoC. Nous avons considéré les 4 algorithmes d'arbitrage (au hasard, LRU, FIFO, Round-Robin) et différents mécanismes de mémorisation (pipeline) à l'entrée et la sortie des switchs. Nous avons sélectionné 20 configurations représentatives. L'impact de la migration d'une technologie à une autre est présenté sur les Figure 6 et Figure 7 pour la surface et la fréquence respectivement. Nous comparons ici trois technologies ASIC et deux familles de FPGA. Le principal résultat de ces deux figures est que les courbes ont

la même forme pour les superficies et les fréquences. Quand nous considérons une migration, le contexte de la technologie cible permet différents compromis architecturaux et donc des configurations de conception différentes. Ainsi, la nouvelle configuration sélectionnée pour répondre au mieux au contexte de la technologie cible peut être prototypée ou émulée sur le FPGA avec une grande confiance que la surface relative et les performances peuvent être extrapolées à partir des résultats présentés ici. La principale conclusion est que l'exploration peut être réalisée sur FPGA avec une grande confiance avant la mise sur ASIC. C'est la même chose pour une migration à partir de 65-nm vers 45 nm!

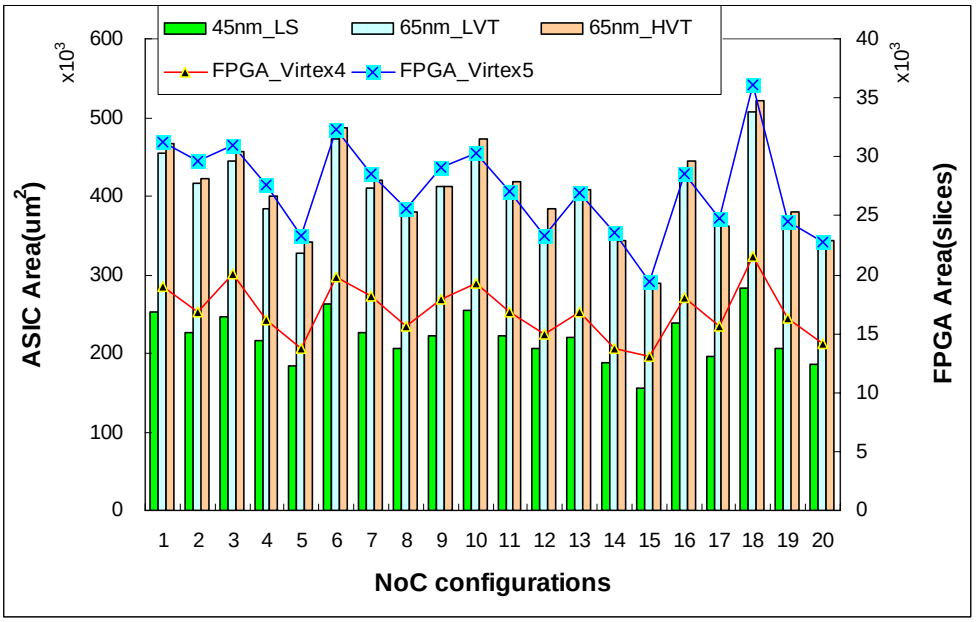


Figure 7 ASIC-FPGA exploration en frequence

Synthèse HLS à base de C

Les systèmes embarqués sont de plus en plus complexes à concevoir, valider et mettre en œuvre. Le système de composition de processeurs embarqués et d'accélérateurs matériels conduit à l'HW / SW co-méthodologis de conception. La

coopération traditionnelle des méthodologies de conception exige que le matériel et le logiciel soient spécifiés et conçus de manière indépendante. La validation de matériel et de logiciel sont ainsi fait séparément, sans validation des interfaces. Le partitionnement est donc décidé à l'avance et des modifications de la partition peut nécessiter un vaste remaniement ailleurs dans le système. Cette décision est respectée autant que possible. Elle peut conduire à des sous-optimaux et les techniques de partitionnement s'appuient souvent sur l'expérience du concepteur. Ce manque d'une représentation unifiée du matériel et du logiciel peut conduire à des difficultés dans la vérification de tout le système, et donc à des incompatibilités entre les matériels / logiciels frontière. Utiliser un langage unique pour les deux simplifie la tâche de migration et assure une vérification de l'ensemble du système. L'utilité d'un langage unifié de conception, en plus de la synthèse, sont la validation et l'exploration de l'algorithme (y compris un cloisonnement efficace). C langage est beaucoup plus contraignant pour ces tâches, et un en particulier SystemC est désormais largement utilisé pour la conception au niveau système, comme le sont de nombreuses variantes ad-hoc. C permet la synthèse rapide et les techniques d'émulation rapide du matériel et des logiciels utilisés pour l'exploration de l'architecture. La synthèse issue de ces méthodologies de conception HW / SW a pour objectif de réduire le défi de la productivité de conception. Toutefois, bien que relever le niveau d'abstraction de la conception de systèmes va contribuer à réduire la complexité de conception fiable et prévisible, la mise en œuvre sur silicium reste obligatoire. La question est donc sur l'impact de la synthèse C dans un cadre de modélisation de systèmes embarqués.

Nous évaluons, à travers une étude de cas, la performance et l'efficacité de plusieurs langages de description de haut niveau (SystemC, Handel-C) à base de FPGA pour les systèmes embarqués.

Conclusion

Dû à des applications embarquées de plus en plus complexes, la conception et la mise en œuvre des MPSOC (Multi-Processor System on Chip) sont l'objet de nombreuses recherches académiques et industrielles. Les principaux axes de recherche comprennent les flots de conception MPSOC, l'architecture MPSOC, la modélisation de MPSOC, les techniques d'évaluation de performances et d'implémentation de MPSOC. Il reste que malgré de nombreuses recherches, il existe très peu d'implémentations existantes. En outre, la plupart des implémentations existantes offrent en général très peu de processeurs. Intuitivement, il semble évident que les problèmes et solutions dans les petites MPSOC seront différents des MPSOC de plus grande taille. Il existe donc un besoin pour concevoir et construire des plates-formes MPSOC avec un nombre important de processeurs en tant qu'outil expérimental pour l'évaluation des flots de conception actuels de MPSOC. L'objectif principal de cette thèse est cette mise en œuvre afin de mieux extraire les propriétés physiques de la conception qui pourraient être utilisés dans la conception au niveau supérieur et en même temps fournir une précieuse plateforme expérimentale. En effet, avec l'avènement de la modélisation de haut niveau d'abstraction et des techniques de conception telles que SystemC TLM il semble qu'il y ait un besoin d'enrichir la précision de ces modèles avec implémentation physique. D'ailleurs, le suivi précis du trafic des applications complexes de communication est nécessaire pour comprendre comment régler les futures applications parallèles.

Nous avons abordé plusieurs questions dans cette thèse:

1. La synthèse de haut niveau et l'utilisation potentielle pour la conception de MPSOC,
2. La mise en œuvre sur FPGA de MPSOC homogène et hétérogène à base de mémoire partagée,
3. Le matériel de surveillance de NoC.

En ce qui concerne la synthèse de haut niveau, des accélérateurs ou des co-processeurs basés sur de la synthèse C ont été proposés pour le prototypage rapide

d'applications. Nous avons mené une étude de cas sur l'évaluation de la synthèse de haut niveau basée sur le C. L'objectif était d'évaluer le potentiel d'utilisation plus élevé avec contraintes de surface et avec partitionnement multi-objectifs et comment les décisions au niveau système pouvaient être touchées. En effet, bien que la complexité croissante des systèmes pousse pour la modélisation de haut niveau d'abstraction, il est toujours impératif de prendre en compte des informations de mise en œuvre précises pour améliorer les performances. Cela remet en question la capacité des outils de synthèse C pour relever ce défi. Des études de cas montrent des variations significatives entre les résultats des différents outils de synthèse de haut niveau et en particulier avec exploration des options de synthèse physique, ce qui questionne l'utilisation du C pour la modélisation et la conception multi-objectifs de systèmes. Une approche de conception de systèmes multi-objectifs dans un environnement avec compromis de coût/performance devrait être fondée sur des données aussi précises que possible, sans quoi des décisions de conception inappropriées pourrait être faites. Nous soutenons que les questions d'implémentation (surface, fréquence, et floorplan) pour les systèmes complexes à grande échelle devraient être prises en compte lors de la modélisation de haut niveau basée sur le C puisque actuellement les outils ne garantissent pas que les propriétés de la sémantique de la concurrence de haut niveau sont préservées. En effet, la concurrence extraite à un haut niveau est perdues partiellement dans les transformations de code et de représentation ainsi qu'avec les contraintes de ressources. Cette conclusion affecte l'utilisation potentielle de la synthèse de haut niveau pour la conception de MPSOC avec contrainte de surface. Toutefois, pour la conception de coprocesseur dans les MPSOC, la conception HLS est appropriée. Nous devons nous concentrer sur les plateformes basées MPSOC et les problèmes de conception tels que la personnalisation peut être atteinte avec une synthèse de haut niveau de coprocesseur, tout en gardant la structure principale du MPSOC.

Nous avons implémenté une famille de MPSOC à base de NoC avec un plus ou moins grand nombre de processeurs (2 - 24 PE), homogènes et hétérogènes (Microblaze, PPC) sur un seul FPGA. Trois NoC offrent des fonctionnalités différentes pour 16 tuiles PE sur une puce unique et il est possible d'augmenter le nombre de processeurs en

réduisant le nombre de NoC pour conserver la même surface. De toute évidence, avec des contraintes de ressources en surface, il existe un compromis entre une architecture avec moins de processeurs, mais avec un support de communication et de contrôle des performances fort et une architecture avec 50% de processeurs en plus et moins de support de communication. Cette thèse de doctorat a clairement affirmé l'impact du placement/routage de NoC sur la conception MPSOC. Le Data-NOC relie les tuiles de PE et les mémoires DDR2 avec une bande passante élevée, le Synchronisation-NOC offre deux modes de synchronisation pour les tuiles de PE. Les utilisateurs peuvent vérifier et configurer les adresses des blocks d'IP connectés au Data-NOC grâce à notre Service-NOC. Nous avons également montré l'utilisation de notre système de suivi des performances pour le déboguage et le réglage le logiciel embarqué. Le Virtex4 FX140 FPGA Xilinx a été sélectionné pour fournir d'importantes ressources logiques avec mise en œuvre rapide et de test simple. L'environnement de conception FPGA peut offrir un grand nombre d'IP pour réduire les efforts de conception et diminuer la pression du temps d'accès au marché. Par exemple, dans notre système un certain nombre d'IP peuvent être issus de la bibliothèque EDK Xilinx, tels que les processeurs MicroBlaze, LMB, FSL, Bram.

Aucun « benchmark » de MPSOC n'existait au moment de cette thèse. Ce n'est que récemment que EEMBC multi-core v.1.0 a été disponible pour notre laboratoire. Nous avons donc mené des études d'évaluation des performances avec notre propre application. Pour évaluer notre MPSoC nous avons mis en œuvre sur notre plateforme avec succès les blocs AES et TDES de chiffrement avec des algorithmes de cryptographie en utilisant deux méthodes. La première consiste à diviser les données entre les processeurs et la seconde se fait en divisant l'exécution de la TDES. Notre architecture met en oeuvre des éléments de traitement connectés via deux NoC. Une telle architecture est essentiellement adaptée aux accès lourds à la mémoire et permet une évolutivité de la construction du système. Les résultats montrent de meilleures performances pour les applications avec données parallèles, favorisées par l'architecture. Ce résultat peut être inversé dans une architecture pipeline. Une autre conclusion montre que les applications avec données parallèles peuvent être appliquées à un plus

grand nombre de processeurs, avec des modes de fonctionnement différents, y compris CBC, parce qu'il a un temps de calcul dominant sur le temps d'accès à la mémoire permettant une accélération linéaire, très efficace. On peut prévoir ainsi que la segmentation physique de la mémoire en banques permet de pousser cette parallélisation et d'atteindre même un nombre plus élevé de processeurs, bien que cela induise une plus grande complexité pour la connexion entre les processeurs. D'autre part nos résultats montrent que notre architecture ne favorise pas la mise en œuvre pipeline en raison de l'absence de lien direct entre les processeur ce qui impacte grandement les performances en raison de la nécessité d'un accès synchronisé à la mémoire pour échanger des données. Nous notons également que réduire le niveau de granularité peut améliorer les performances. Le passage d'une conception de FPGA vers ASIC soulève la question des gains et des avantages qui peuvent être réalisés tant au niveau architectural, mais aussi bien au niveau de la programmation parallèle. Le NoC de nos MPSoC a été mis en œuvre sur la technologie ASIC et a été exploré avec des contraintes temporelles différentes et une bibliothèque de différentes catégories de technologies 65nm/45nm de STMicroelectronics. Les résultats expérimentaux sur ASIC et FPGA sont comparés et nos résultats montrent bien que l'on peut naturellement s'attendre à un gain d'espace alors que la fréquence de travail n'est pas aussi sensiblement augmenté. Ceci suggère que l'amélioration des performances ne peut être atteinte par la seule technologie et de l'avantage en surface doit être exploité en sélectionnant les composants de réseau sur puce avec des fonctionnalités plus agressives.

Cette thèse est une étape préliminaire dans le sens de flots précis de conception physique de MPSoC à la fois vers les technologies FPGA et ASIC. D'autres recherches sont nécessaires en ce qui concerne le DFM, la traduction automatique de FPGA vers ASIC avec des contraintes sensibles au layout, couplée à une exploration de l'espace de conception. Les plates-formes conçues lors de cette thèse vont contribuer à mieux comprendre les techniques de programmation parallèle à travers la surveillance de NoC précis et temps-réel et avec l'appui matériel de NoC multiples. L'analyse comparative des méthodologies pour MPSoC devrait devenir un sujet de recherche majeur car un flot

complet de « Design Space Exploration » peut bénéficier à partir du monitoring de NoC jusqu'à la programmation parallèle de haut niveau et la parallélisation automatique.

Chapter 1 Introduction

1.1 Select MPSoC as the direction of thesis

Gordon E. Moore states that the number of transistors on a chip doubles about every two years, now popularly known as Moore's Law. Transistor feature size is expected to continue to shrink. A reduction of 70% in linear dimensions of transistors by moving to a new fabrication process (for example from 65 nm to 45 nm) allows a 2-fold increase in the chip density.

As shown in Figure 1 the 2008 International Technology Roadmap for Semiconductors (ITRS) [1] projects that multi-billion transistor chips will come to production by the end of this decade.

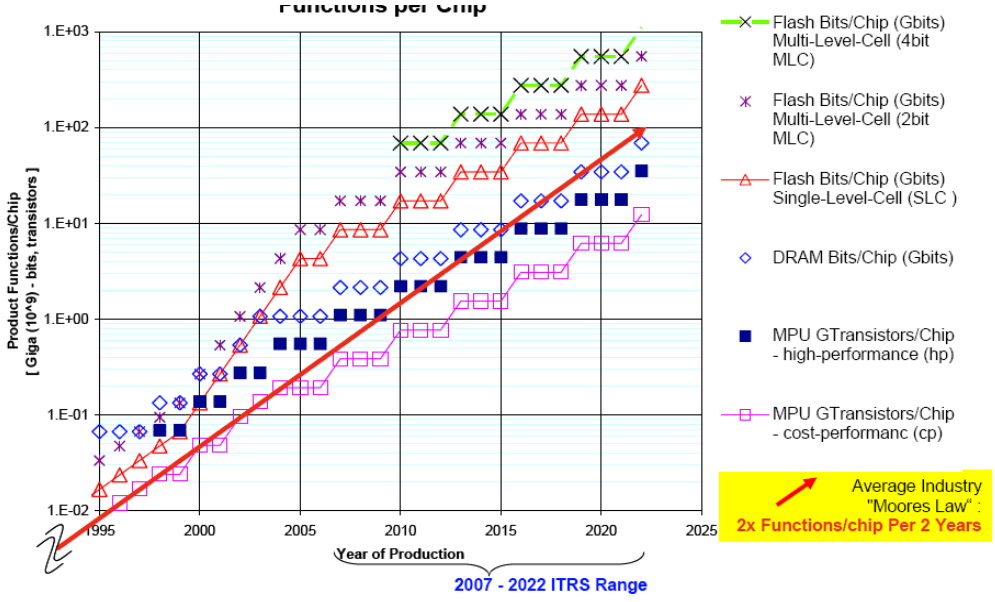


Figure 1 Product Technology Trends (2008 ITRS)

Currently deep submicron (DSM) technologies is proceed in most VLSI and ULSI. Feature sizes can reduce the dimension of actual ASIC application, and introduce new applications, but will also aggravate current problems in VLSI/ULSI design, and moreover, introduce several new ones. Recently the Intel Tera-Scale Computing research team in a white paper has mentioned that "power thermal issues, such as dissipating heat from increasingly densely packed transistors, have begun to limit the rate at which processor frequency can also be increased. Although frequency increases have been a design staple for the last 20 years, the next 20 years will require a new approach. Basically, industry needs to develop improved micro-architectures at a faster rate and in coordination with each new silicon manufacturing process, from 45 nm, to 32 nm, and beyond.

Performance is always one of most important characteristics for the measurement of VLSI design. We can no longer simply increase the clock frequency at the same rate as we have in the past in order to increase performance. Power and thermal requirements are beginning to outstrip the benefits that faster clock frequencies offer. However, because the trajectory of Moore's law will continue well into the next decade, we expect to continue doubling transistors every 18-24 months for the next several years. Parallel execution in Multi core designs will then allow us to take advantage of these greater transistor densities to provide greater performance.

Multiprocessor system on chip (MPSOC) have strongly emerged in the past decade in communication, multimedia, networking and other embedded domains. MPSOC became a new paradigm of high performance embedded application design. As shown in the Figure 2, more and more processing engines are integrated in system on chip, In the future hundreds even thousands processing elements could be used for high performance required system.

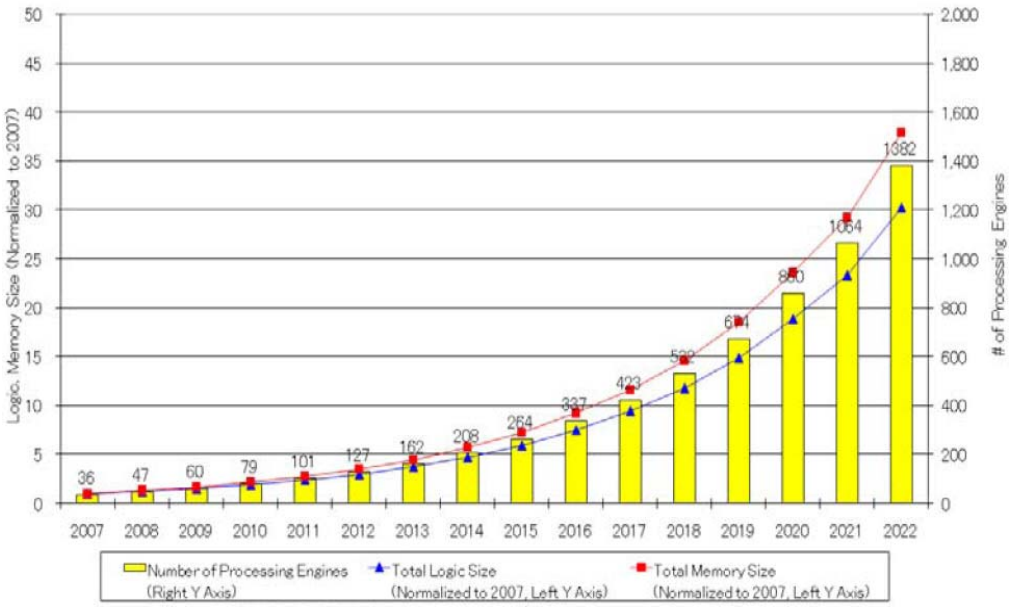


Figure 2 number of processing engines Trends (2008 ITRS)

MPSOC is an aggregation of System on chip and traditional multiprocessors. But we can't directly apply the scientific super computer model to SoCs. It is not simply multiple processors and other hardware peripheral subsystem shrunk to a single silicon chip. MPSOCs have been designed to satisfy the requirements of embedded applications.

Three most important common characteristics of these embedded applications are high performance, real-time and low power.

Higher performances are always required by consumers, and it encourages the research and development of high-performance platforms .that can satisfy new requirements and new standard. Higher performance means more computations and more complex algorithms that can not be realized by simple hardware or single processor SoCs. MPSOCs have been designed at this background.

Real time computing is much more than simple high performance computing. In traditional interactive computing, we care about speed but not about deadlines. Most embedded systems care not just about average performance but also that tasks are done by a given deadline. So MPSoC architectures have to be predictable enough, applications can run with predictable performance form the MPSoC architecture.

Power consumption is another important constraint for MPSoC design. Power constraints are much stricter in MPSoC than in traditional supercomputer system or desktop computer system. Lower power consumption can extend the life of the battery for battery operated MPSoC, and limit of energy provided by battery require MPSoC design reducing energy use as possible. In non-battery operated devices, low power is also required for chip heat and cost reasons. MPSoC's Power and energy constraints must be tacked at every level of abstraction.

1.2 Identify the MPSOC design and implementation issues

In the past several years, lots of paper have been published for MPSoC, many of these researches just care about the simulation, and stopped at the simulation of high level of abstraction. At high level of abstraction simulation can quickly realize and verify the functionality of system and algorithm. At register transfer level simulation can verify the correction of RTL code and evaluate the performance of system with the execution cycles. Simulation can also help partitioning of software/ hardware. Why in this thesis we care about actual prototyping emulation and implementation not just simulation.

Actual prototyping emulation and implementation can provide real performance measures. With simulation performance of system can be measured by providing the numbers of cycles, but simulation cannot provide the measurement with the timing. Even we can set frequencies of system and calculate the timing. The frequency set in simulation have no means, we cannot get the maximum frequency of system by simulation. MPSoC is high-performance real time system, emulation and implementation are necessary to measure minimum period of clocks at actual prototyping.

Actual prototyping emulation and implementation can measure area use and consumption. Implementation results can clearly tell us the cell utilization for ASIC application, and the percentage of FPGA utilization. Other important issues can be reached with implementation, such as pin numbers constraints, floorplaning, power consumption.

In this thesis, we focus on

- Implementation MPSOC on actual prototypes, not just simulation to clearly identify MPSOC design and implementation issues.
- Execute Multi-application on actual platform to evaluate performance of MPSOC and data parallelization
- The impact of technology scaling on NOC and MPSOC performances from 65-nm to 45-nm as well as migration from FPGA to ASIC.
- Evaluate High level synthesis and the potential use for MPSOC platform based design

The rest of thesis are organized as follow the chapter 2 will introduce the states of the art of multiprocessor system on chip and network on chip. The chapter 3 presents our proposed MPSOC architecture and FPGA implementation. The chapter 4 introduce the performance evaluation of the FPGA implemented MPSOC. The primarily ASIC evaluation of MPSOC on 65nm and 45nm technologies will be shown in the chapter 5. The potential use of High level synthesis on MPSoC platform can be found in the chapter 6. The chapter 7 is the conclusion. And finally, at the end of the manuscript, the publications relating to this PhD thesis are listed.

Chapter 2 State of the Art of Multiprocessor system on chip and Network on Chip

2.1 Introduction

As the development of deep submicron technology, a single integrated circuit can contain 2 Billion transistors. Multiprocessor Systems on Chips (MPSoC) are the latest incarnation of very large scale integration (VLSI) technology. An MPSoC is integrated circuit that implements most or all of the functions of a complete electronic system and that uses multiple programmable processors as system components. Nowadays Multiprocessor System on Chips are widely studied in academies and are entering commercial and consumer markets.

2.2 Academic and Commercial MPSoC

Multiprocessors System on Chips are strongly emerging and several products or ongoing R&D projects are tackling the issues related to multiprocessors [22]-[32].

TABLE 1 provides a few examples of commercial multicore implementations. They can be globally divided in 2 categories: (1) general purpose (2) application specific. In the first category we can place the ARM ARM11MPcore [23], the MIPS MIPS32 1004 Core [24] and the Renesas/Hitachi SH-X3 [25]. In the second category we can place Texas Instruments TMS320C6474/TMS320VC5441 DSP [28]-[29],

Freescale QorIQ P4080 [30] and the Toshiba Venezia multicore [31]. Other worth noting are Ambric [26], MAPS-TCT [32] and [22].

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 EX	Bus	8

TABLE 1 industrial MPSoC Implementation

The ARM11 MPcore [23] showed in the Figure 3 is a classical shared memory 4 processors based multiprocessor based on a shared bus architecture with a snoopy cache coherency protocol (MESI).

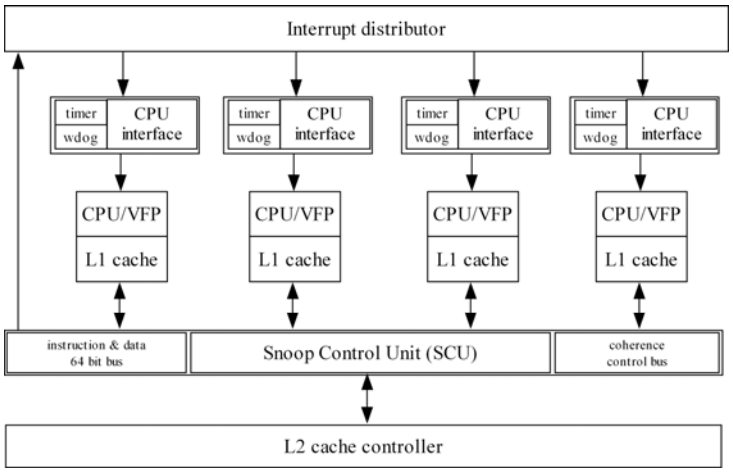


Figure 3 ARM 11 MPcore

The MIPS32 1004 [24] 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 [27]. The Texas Instruments TMS320C6474 [28] 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 [29] is a 4 core multicore with shared bus between 2 cores and HPI for external accesses.

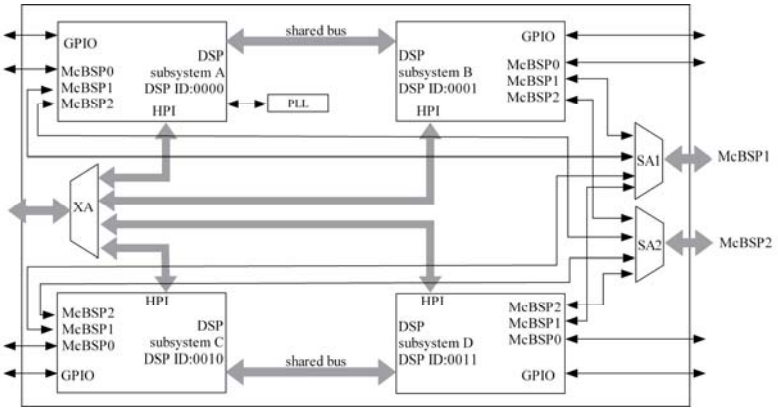


Figure 4 Texas Instruments TMS320VC5441

The Freescale QorIQ™ P4080 [30] 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. CoreNet is presented as a highly concurrent fully cache coherent multi-ported fabric. CoreNet's point-to-point connectivity with flexible protocol architecture allows for pipelined interconnection between CPUs, platform caches, memory controllers. No details are available.

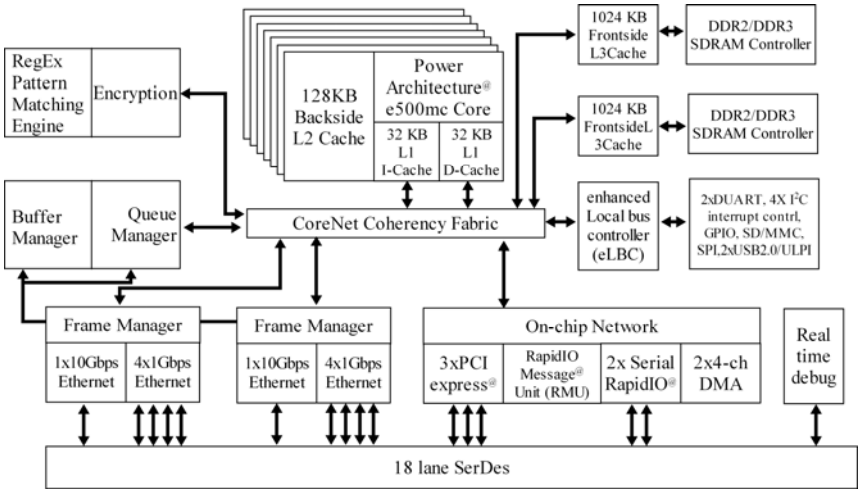


Figure 5 QorIQ™ P4080

Finally Toshiba proposes the Venezia architecture [31]. Our work differs from all the previously described work by a larger number of processors of smaller size emphasizing the choice of coarse grain concurrency over fine grain concurrency exploited by more sophisticated processors (VLIW e.g. MEP). It remains that we are working on architectural support for fine grained parallelism [33] through SIMD coprocessing.

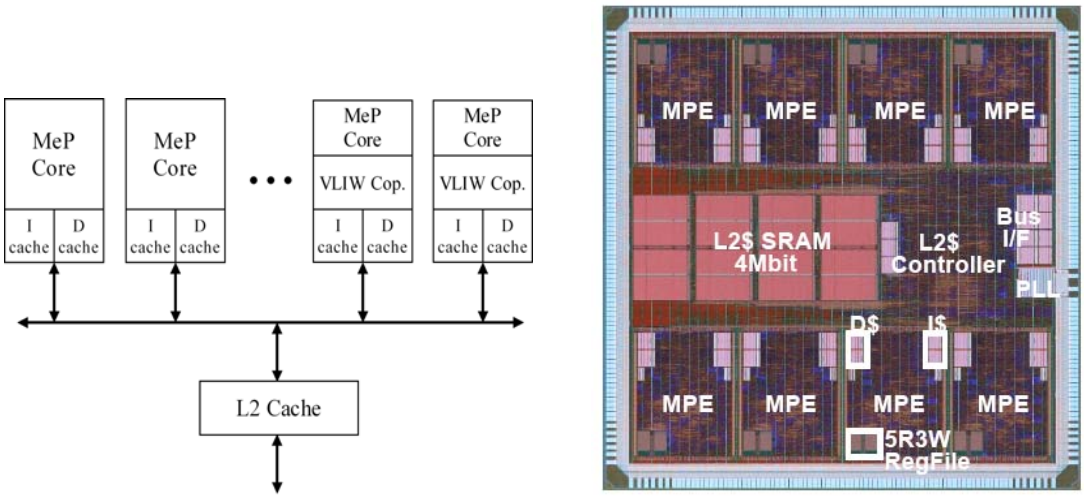


Figure 6 Toshiba Venezia Architecture

2.3 Academic and Commercial NOCs

In this section, we briefly introduce a handful of specific NoC examples, which are studied and published in recent years. This is by no means a complete compilation of existing NoCs, we just address a representative set: SPIN, ÆTHEREAL, NOSTRUM, and MANGO.

the range of the analysis is limited to the describing the design choices of actual implementations and the accompanying work by the research groups behind them. there are many more, rather the purpose of this section is to address a representative set: In Moraes et al. [79], a list in tabular form is provided which effectively characterizes many of the NoCs not covered in the following.

2.3.1 SPIN.

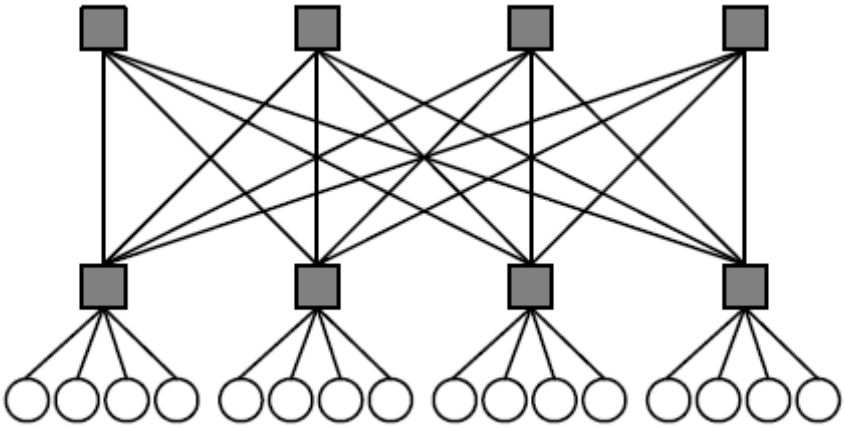


Figure 7 SPIN network topology

The SPIN network (Scalable Programmable Integrated Network) is one of the first published and realized NoC[9] [10]. It is developed by the University Pierre and Marie Curie. It implements a fat-tree topology with two one-way 32-bit datapaths at the link layer, shown in Figure 7. The fat tree is the most cost-efficient topology for VLSI realizations claimed in Leiserson and fat tree provides a simple and effective routing scheme. In SPIN, The routers are packet-based with a flit size of 36 bits.

Wormhole routing is used without limit of packet size. There are three types of flits: first, data and last flit. The first flit contains the address and packet tagging information, while the last flit contains the payload checksum. Adaptive routing algorithm and out-of-order delivery can be used to maximize the network bandwidth. Otherwise, deterministic and in-order delivery is used to avoid the reordering buffers on the output ports. In comparison with tree topology, fat tree duplicate the root of tree to have a large bisection bandwidth, but higher area cost. For larger number of nodes, the fat tree is more efficient in area cost but more difficult in physical layout in comparison with mesh or tori.

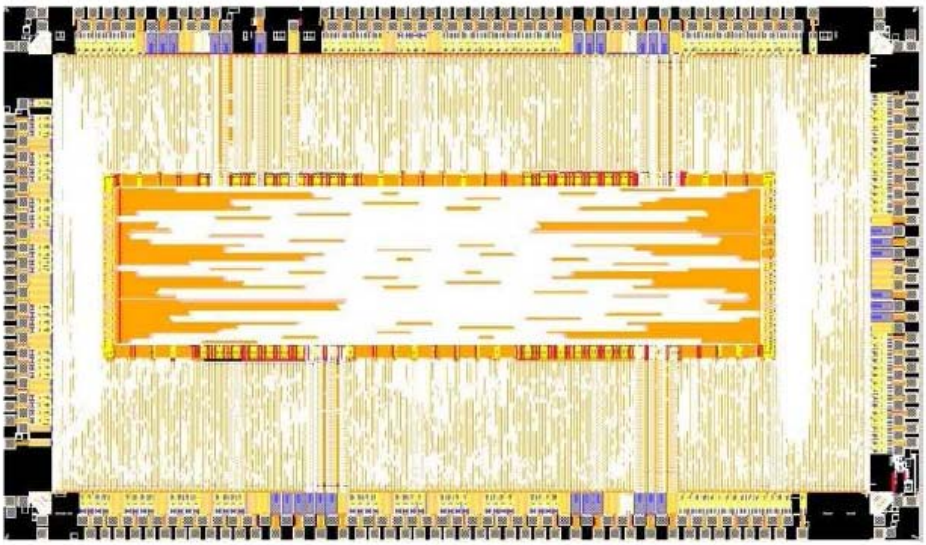


Figure 8 SPIN32 test chip layout

A 32-port SPIN network was implemented with CMOS ST-Microelectronics 0.13 μm technology [10] Each SPIN router has an area of 0.24 mm2 while the total area of 32-port NoC was 4.6 mm2 (0.144 mm2 per port), for an accumulated bandwidth of about 100 Gbits/s. Figure 8 shows the test chip of SPIN which contains traffic generators and analyzers to compute the 32-port SPIN NoC performance.

2.3.2 ÆTHEREAL

The ÆTHEREAL NoC is developed by Philips offering both Guaranteed Service and Best Effort Traffic [11][12] In the ÆTHEREAL the guaranteed services pervade as a requirement for hardware design and also as a foundation for software programming. The router uses a contention-free routing to send independent traffic on the same physical links. Therefore, a time-division multiplexing (TDM) is used to eliminate entirely the contention by scheduling the time and location of all packets globally. Flits wait in the network interfaces until they can be routed while avoiding the contention on the link. Thus propagation speed of individual flits through the NoC is fixed and known in advance. Every TDM slot table has S time slots (rows), and N output ports (columns). There is a logical notion of synchronicity, since all routers in the network are assumed to be in the same fixed-duration slot. Figure 9 illustrates the operation of contention-free routing in ÆTHEREAL. Packet A and B are routed without contention between router R1 and R2 because they use different timing slots. In the same way, packets A and C do not have contention in between routers R2 and R3. The reconfiguration of these tables are carried out by special packets sent over the BE network.

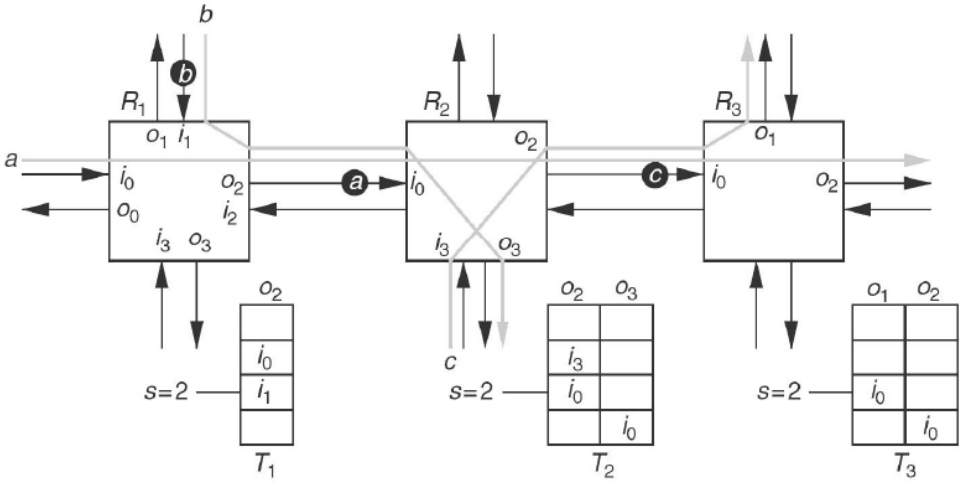


Figure 9 ÆTHEREAL contention-free routing. source[12]

ÆTHEREAL NoC has two visions: distributed vision and centralized vision. On the distributed vision, special BE packets are sent to the router to request the

allocation of a GS traffic. The router decides if the GS packet can be routed or not. On the centralized vision, the slot tables are located on the Network Interface and no longer on the router. Moreover, a centralized slot allocator decide which GS packets can be served by sending Best Effort packets to the Network Interfaces in order to configure the timing slots.

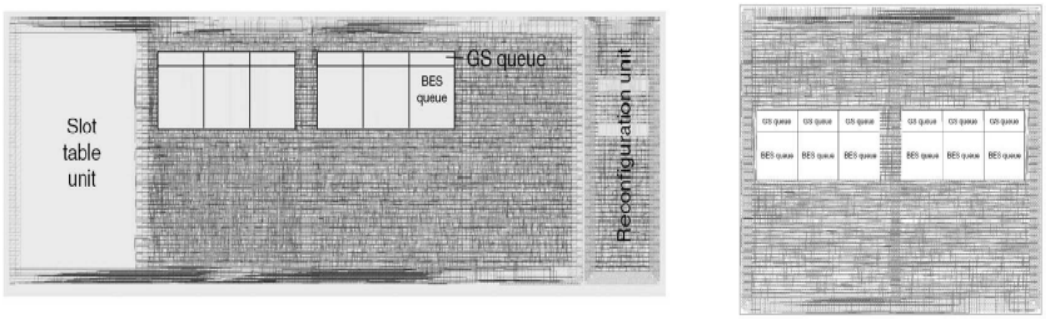


Figure 10 Implementation of GS-BE ÆTHEREAL source [13]

Figure 10 shows the implementation of GS-BE ÆTHEREAL distributed and centralized programming router architecture [12]. A 6-port ÆTHEREAL NoC implementation is detailed on CMOS 130nm technology. A distributed and a centralized programming architecture are described on the paper. Both of them are designed as a hard macro with dedicated hardware FIFOs for the BE and GS queues. Moreover, a dedicated hardware slot table is used on the distributed programming architecture for the congestion-free routing algorithm. These dedicated hardware devices are designed to minimize the router area. The hardware dedicated FIFOs and slot table are depicted on the squares. Moreover, the distributed programming router architecture contains the Reconfiguration unit (on its left side). It is used to dynamically allocate and dislocate the GS traffic. The distributed architecture takes 0.24 mm2 at 500MHz, while the centralized architecture takes 0.13 mm2 at 500MHz. Because there is neither slot table nor reconfiguration unit in the centralized vision, the area of the centralized architecture is smaller than the distributed version.

Due to the same sense of time in all of routers, all the routers have to be fully synchronized. The design of fully synchronous network is not a scalable solution. Thus a waterfall clock distribution and a synchronous Latency Insensitive Design

(SLID) are proposed. The buffers of BE and GS are independents and different timing slots are distributed for different GS traffics. So contentions are eliminated and the throughput is guaranteed. But sometime even there are no other packets in the network, the packets have to wait for their timing slots in the network interface. The average latency for packets is relatively high.

2.3.3 Nostrum

Nostrum is an NoC developed by the LECS (Laboratory of Electronics and Computer Science) at the Royal Institute of Technology in Sweden [17].

The Nostrum NoC architecture follows a regular mesh topology containing switches and network interfaces. Two traffic classes are available, Best Effort (BE) and Guaranteed Bandwidth (GB). In the BE implementation, the packet transmission is handled by datagrams. The switching decisions are made locally in the switches on a dynamic/non-deterministic manner by means of the deflection routing algorithm. Its benefits are robustness against network link congestion and link failure. However, the BE packets may arrive in another order that they were sent; thus, the NI handles the ordering of packets and de-segmentation of messages. The BE packet size is one flit.

The defection routing algorithm guarantees that no packet is stalled in the router, thus no intermediate buffer is required in the network. All packets in the switch are forwarded to an output port; even it is not the requested one. This phenomena requires that the entire network is synchronized (all switches have the same clock frequency and switch at the same clock cycle). The GB traffic is handled using containers [Millbe04]. A container is a network packet that follows a predefined looping path as shown in Figure 11. They can transport the information of GB traffic; but if they do not transport any information, they continue to follow the predefined looping path. Thus, they contain an empty flag to identify if they transport or not data.

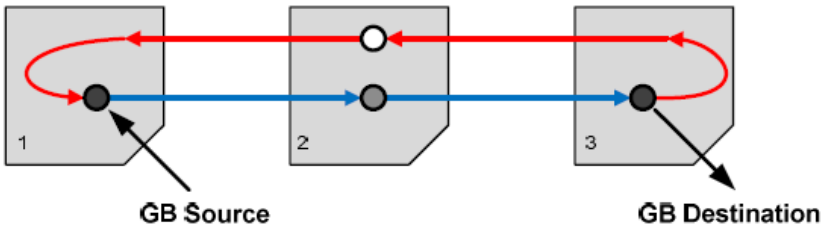


Figure 11 Nostrum looping containers

Figure 11 illustrates an example of a lopping container when a GB source transfers packets to its GB destination. When the empty container arrives to the switch 1 (the GB source), the GB source load the container with the GB traffic and sent it to the east switch (blue line). The container and its load is routed though the network following its predefined looping path. When it reaches the GB destination, the container is unloaded and it is sent back (red line) empty, possibly, with some new information loaded.

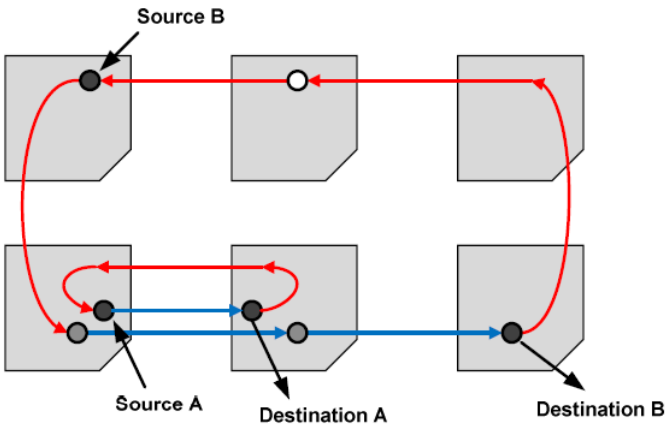


Figure 12 Nostrum bandwidth granularity

When looping containers are temporally multiplexed, the network is able to sent different GB traffics over the same link. Figure 12 shows an example of bandwidth sharing between two independent GB traffics. The containers are launched on the start-up phase of the network when no BE packets are allowed to enter the network. The higher the bandwidth required, the higher the launched containers over the same loop.

2.3.4 MANGO

The MANGO (Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces) architecture, developed at the Technical University of Denmark, is an asynchronous NoC, targeted for coarse-grained GALS-type SoC [15]. MANGO provides connection-less Best Effort (BE) routing as well as connection-oriented Guaranteed Services (GS) . Guaranteed services connections are established by allocating a sequence of Virtual Channels (VCs) through the network. The routers implement VCs as separate physical buffers. A scheduling scheme called ALG (Asynchronous Latency Guarantees), schedules access to the links, allowing latency guarantees to be made.

The router consists of two separate routers: the BE router and the GS router.

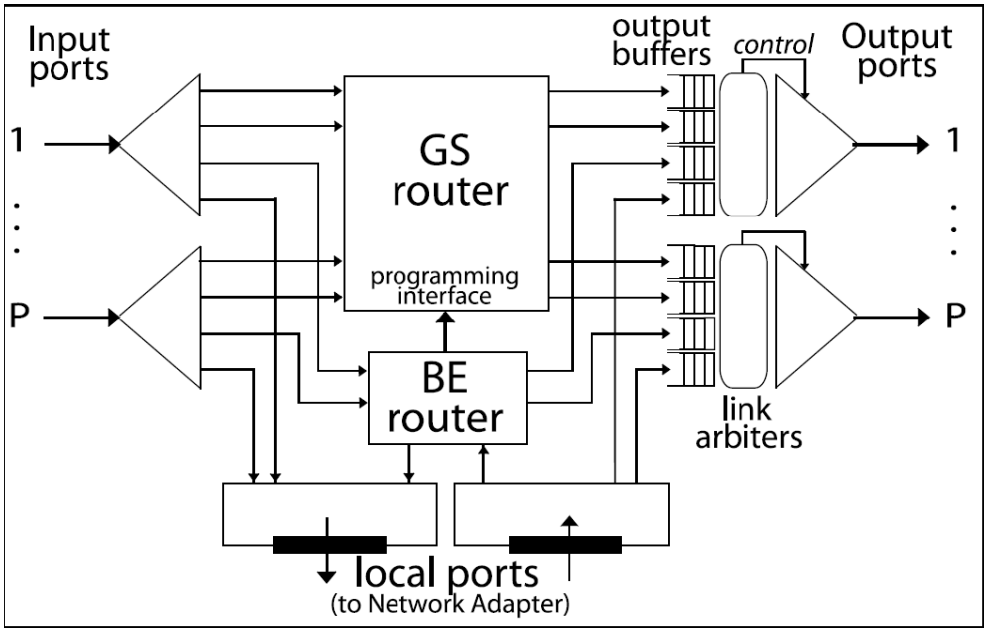


Figure 13 MANGO router

The BE router implements a source routing scheme. The three MSBs of the packet header indicate one of five output ports. After passing the router, the header is rotated three bits, positioning the header bits for the next hop. With a flit size of 33 bits (of which one is the end-of-packet bit) it is thus possible to make only 10 routing hops.

While the routers themselves are implemented using area efficient bundled-data circuits, the links implement delay-insensitive dual-rail data encoding. This makes global timing robust, because no timing assumptions are necessary between routers. However pipelining is necessary in order to keep performance.

2.4 Case study: Arteris Technology

Arteris Company was founded in Paris in 2003 and company focuses their attention on the next-generation of challenges associated with System-on-Chip (SoC) design: on-chip communications, or Network-on-Chip (NoC) [71] [73]. In modern on-chip-system arena, network on chip has been proven to be an effective communication backbone of managing multilevel communication in distributed system. In 2005 Arteris introduced the first commercial implementation of a Network-on-Chip (NoC), delivered in the form of Intellectual Property. Arteris NoC Solution consists of the Danube Intellectual Property Library and a set of EDA tools for configuring and implementing the networking IP cores as synthesizable RTL. Arteris proposes the NoC configuration and design flow as show in the Figure 14.

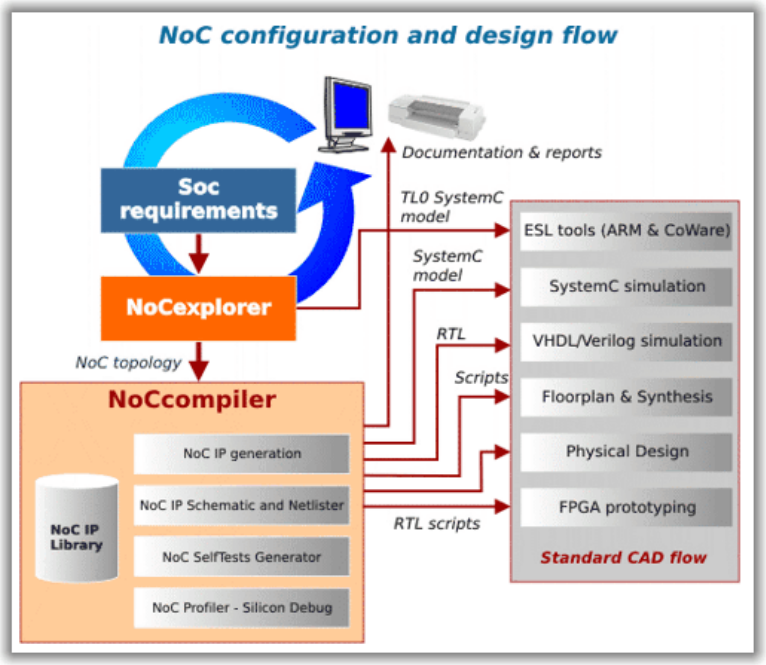


Figure 14 the NoC design flow by Arteris

The Danube Intellectual Property Library contains a suite of configurable building blocks managing all on-chip communications between IP cores in SoC designs. The Packet Transport Units (PTU) from the Arteris Danube library. Can build the packet transport portion of the NoC, which is comprised of a request network and a response network. Each network is sized according to the traffic load expected on request and response paths. At the boundary of network the Network Interface IPs are offered, the third part protocol includes OCP, AXI, AHB or other proprietary interfaces.

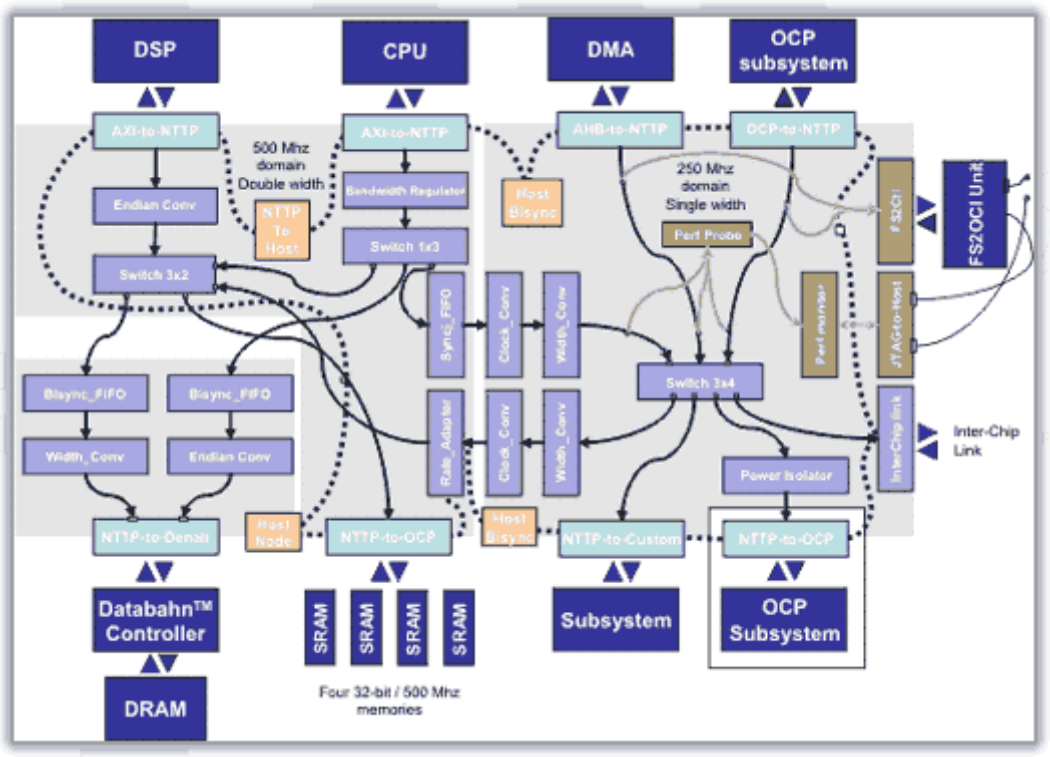


Figure 15 example of Danube IPs

Containing transaction level systemC models of Danube IPs, the architecture tool NoCexplorer can be used to analyze complex traffics of system on chip and evaluate the impact of alternative NoC topologies on overall system performance. NoCexplorer can also generate transaction systemC model of NoC which can be integrated in the complete system in other ESL tools. The NoC systemC model can be simulated with different IP block types such as processors, DSPs, HW accelerators, I/O peripherals, memories, and so on.

In the NoCcompiler environment, selected Arteris NoC IP library units are configured and connected to match the topology obtained in the initial exploration step, in accordance with the specifications of the IP cores connected to the NoC. NoC design views, such as cycle-accurate SystemC models and synthesizable RTL, are then generated for simulation FPGA prototyping, or integration within the SoC using standard design flows.

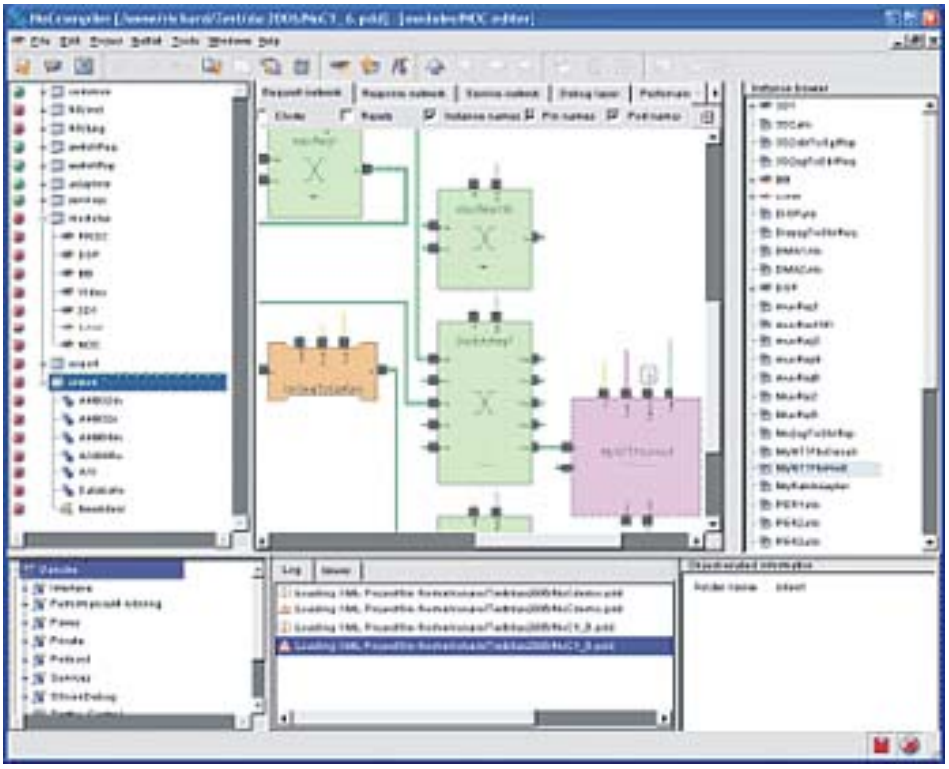


Figure 16 NoCcompiler GUI

2.5 Conclusion

In this chapter, we have presented recent academic and commercial MPSoCs for embedded system. ARM 11 MPcore, MIPS32 1004 Core and the Renesas/Hitachi SH-X3 have been implemented for general purpose. The Texas Instruments TMS320C6474/TMS320VC5441 DSP, Freescale QorIQ P4080 and the Toshiba Venezia multicore have been presented as application specific MPSoC. All these

implementation that we listed are designed specifically for embedded applications, that is also our discuss target, embedded MPSoC. These MPSoC use a fine gain concurrency exploited by more sophisticated processors (VLIW e.g. MEP). The largest number of processor is eight. In the following chapter our MPSoC augmented the number of processors and used smaller size of processor emphasising the choice of coarse grain concurrency.

In the second part of this chapter, a set of representative Network on chip SPIN, ÆTHEREAL, NOSTRUM, and MANGO have been introduced. We limited ourselves in this review of state of the art to the NoCs with physical implementation. An example of an industrial case of NoC IP library, Arteris Danube IP library and the environment NoCcompiler were presented. The NoC of our MPSoC are based on this library. The detail of our MPSoC and NoC will be introduced in Chapter4.

Chapter 3 MPSOC Design and Implementation

3.1 Introduction of MPSOC and NoC Design

With the advent of new technologies in IC design, the MPSoC has emerged as a promising solution for the growing complexity and increasing functionality of the embedded System on Chip. The large number of processors and modules require a communication backbone which provides flexibility, scalability and *quality of service* (QoS) guarantee. The conventional ways of interconnecting, such as buses and crossbars, cannot satisfy these requirements due to timing, power and area limits. *On-Chip Network* (OCN), or *Network on Chip* (NoC), has been proposed as a systematic approach to deal with the communication-centric design challenge. In contrast to traditional connecting ways, modular structure of NoC makes multiprocessor architecture highly scalable and improves reliability and operation frequency of on chip modules. Furthermore the NoC approach offers matchless opportunities for implementing *Globally Asynchronous, Locally synchronous* (GALS) design, which make clock distribution and timing closure problems more manageable. To reduce the pressure of time-to-market and tackle the increasing complexity of SoC, the need of fast prototyping and testing is growing. Taking advantage of deep submicron technology, modern FPGAs provide a fast and low-cost prototyping with large logic resources and high performance.

In this chapter, we describe our multiprocessor system comprising of sixteen processor tiles, one synchronized shared memory and four DDR2 memory banks. All these processor

tiles are connected to DDR2 slaves through a high throughput Data-NoC and are synchronized using a synchronization NoC. A service network is also integrated in the system to provide functionalities like error management, IP status check and reconfiguration services at run time. The NoCs are implemented using Arteris Danube library, which is based upon wormhole routing and packet switching. The run-time observability is a necessity for embedded systems' debugging and monitoring, which challenges the NoC-based SoC design. The run-time performance monitoring implemented in our system provides NoC behavior information, like throughput, latency, to support the application debugging and software tuning. The whole MPSoC is a GALS design realized with the help of bi-synchronization method. The OCP adapter containing one pair of bi-synchronous FIFOs has been inserted between processor and NoC, to tackle the communication issue between two different clock domains (processor clock domain and NoC clock domain).

3.2 MP3NOC: A Family of Architectures

3.2.1 Generic Architecture

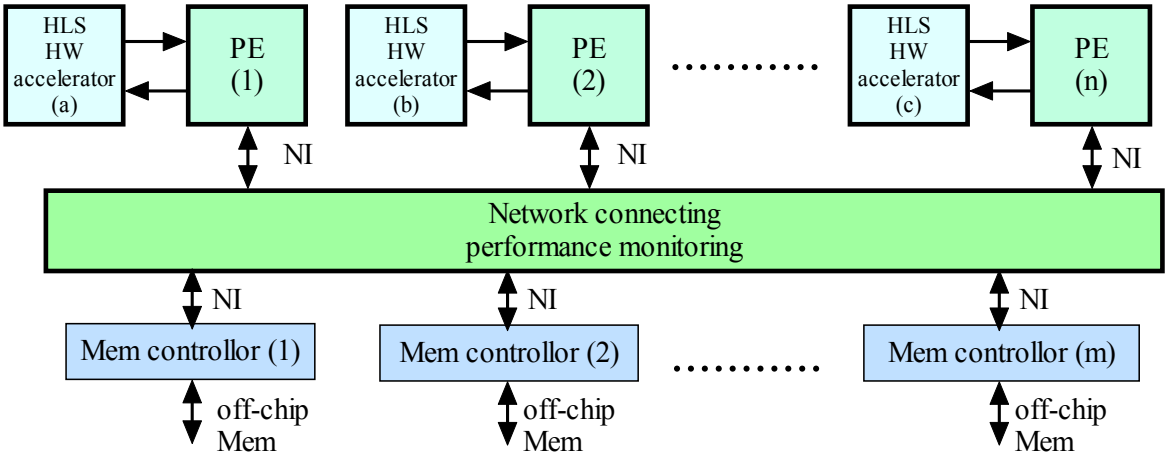


Figure 17 generic multiprocessor architecture block diagram

The block diagram of the generic multiprocessor architecture is shown in Figure 17. N processors are connected to m banks of memories by network connection. In these N processors, some processors can be connected to High level synthesized hardware accelerator to increase the performance of MPSOC. Our proposed MPSOC embedded

system design is a MPSOC platform based design. Design is based on intensive IP design and reuse. From the application which will be implemented and the MPSOC platform, several designs space explorations can be explored, and to find the most suitable architecture. The Figure 18 shows some possible design space explorations

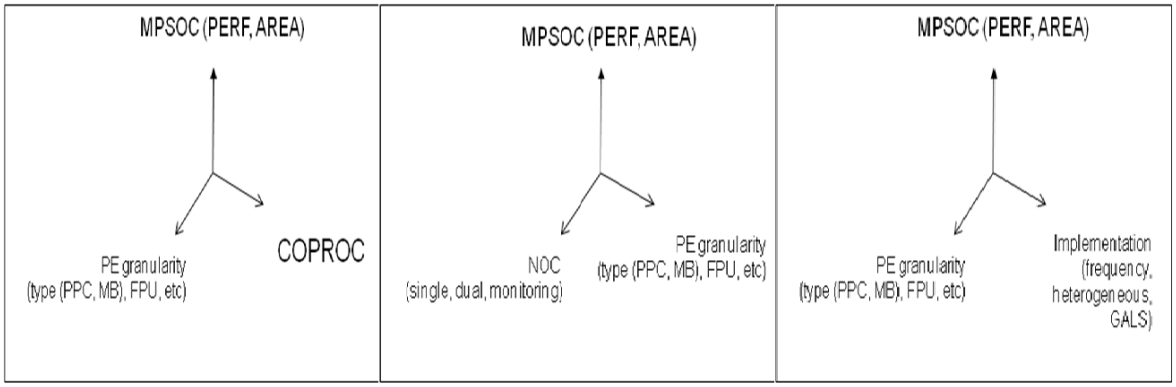


Figure 18 possible design space explorations

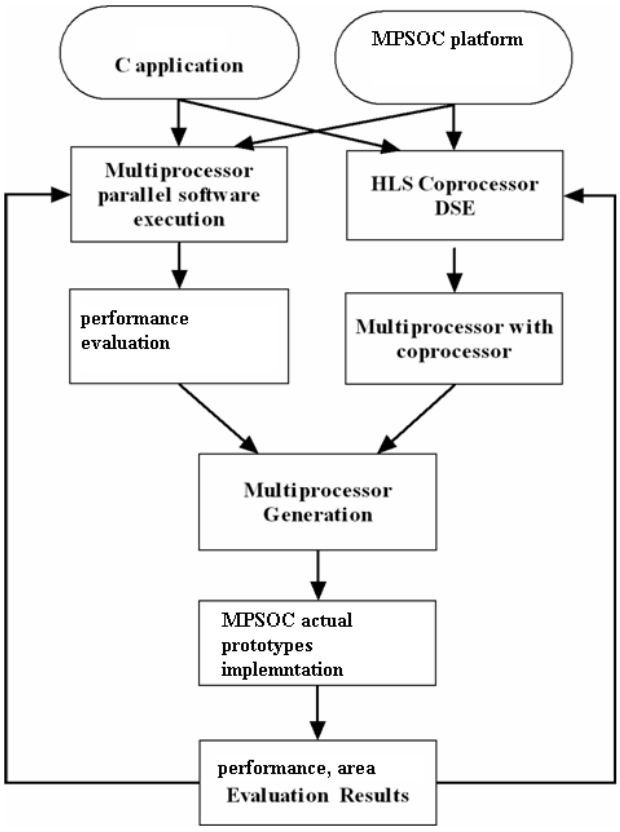


Figure 19 platform based MPSOC design Methodology

The Figure 19 shows the platform based MPSOC design methodology. The application on C firstly is implemented on the MPSOC platform without accelerators. Then we evaluate the performance of this MPSOC, if the performance can satisfy the requirement,

the application is successfully implemented on the MPSOC, if not, from the C code of application, the HW accelerator (or coprocessor) can be quickly generated and integrated on the platform. Then we can do some DSE and get the new MPSOC with HLS HW accelerators, In this chapter, we will introduce the MPSOC platform and FPGA implementation, the High level synthesized HW accelerator generation will be presented in chapter 6.

3.2.2 Architecture overview

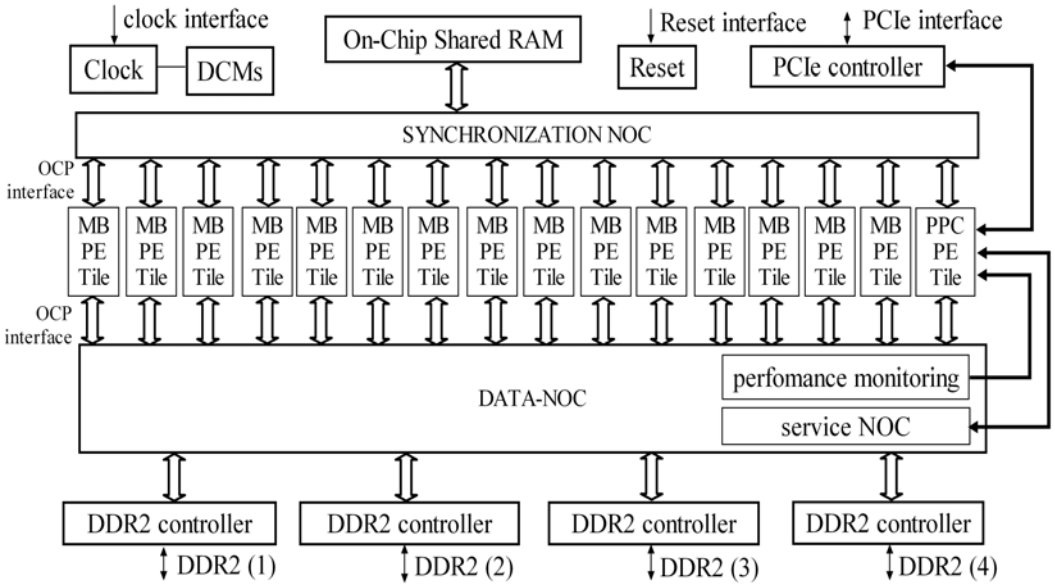


Figure 20 multiprocessor block diagram with N=16 Processors and M=4 DDR2 Banks with 3 NOCs (Data, Synchronization, service)

The multiprocessor system integrates sixteen Processing Elements (PE) Tiles including fifteen MicroBlaze based MB-PE Tiles and one PowerPC based PPC-PE Tile. Each tile is a powerful computing system that can independently run its own program code and its own operating system. PE Tiles are connected to Data-NoC and Synchronization-NoC through OCP data adapters and OCP synchronization adapter.

The OCP adapter can convert processor's FIFO interface into OCP interface and separate PE clock domain from NoC clock domains. One 64KBytes shared on-chip memory is attached to the synchronization NoC, which establishes a synchronization media for the PE tiles. Data NoC is connected to four DDR2 controllers, which in turn connect to

the respective off-chip 256MBytes DDR2 memory banks. Service NoC is integrated in the Data NoC to provide error management, IP status check, reconfiguration services at run time. The performance monitoring probes are attached on all Data NoC inputs and outputs to observe NoC behaviors. Recorded events and results are then fed back to the PPC-PE tile. PPC-PE tile is also connected to a PCI express interface for the off-chip communication purpose.

3.2.3 Processing Element TILE

To increase the compatibility and to facilitate the reuse of the architecture, the OCP-IP standard is used for the connection of PE Tiles and NoCs. Benefiting from the OCP standard, any PE tile with OCP-IP interface can be connected to our system. In our system we have integrated two types of PE tiles: The Xilinx processing soft-core MicroBlaze V7.00 based MB-PE Tile and the IBM processing hard-core PowerPC-405 based PPC-PE Tile. Both processing cores are provided by our FPGA design environment: the Xilinx Embedded Development Kit (EDK).

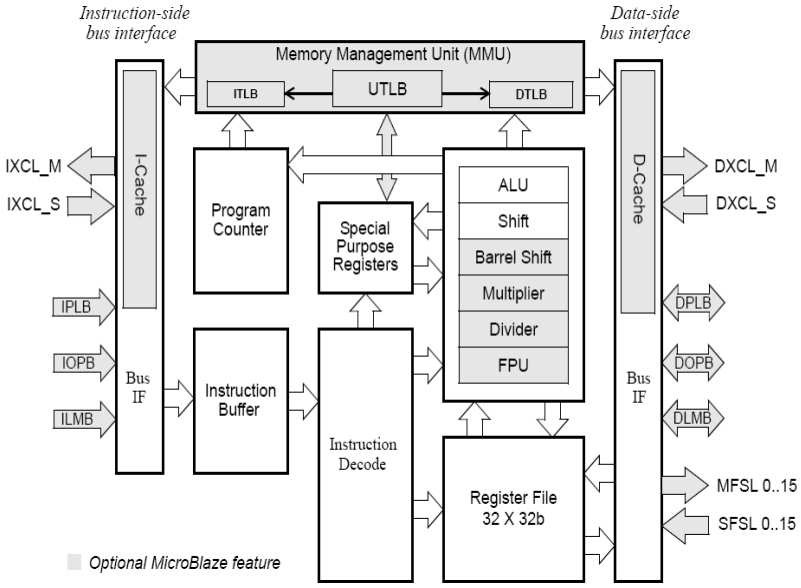


Figure 21 Block diagram of MicroBlaze processor

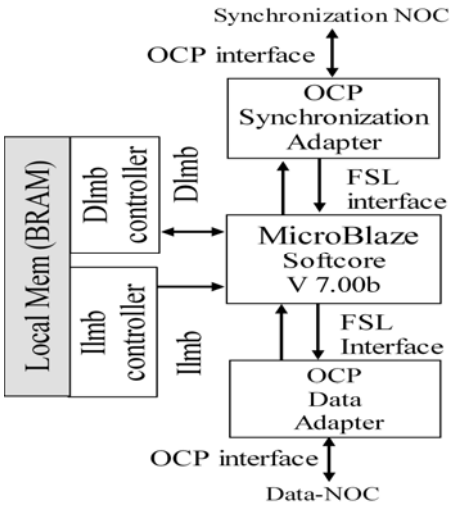


Figure 22 Block diagram of MB PE Tile

MicroBlaze™ v7.00 soft core processor is a 32bit *Reduced Instruction Set Computer* (RISC) optimized for implementation in Xilinx® *Field Programmable Gate Array* (FPGA). MicroBlaze processor is implemented with Harvard memory architecture, and it is highly configurable. As shown in Figure 21, a set of parameters and execution units can be configured at design time to fit design requirement, such as number of pipeline stages, cache size, selectable *Barrel Shifter* (BS), *Floating Point Unit* (FPU), *Hardware Divider* (HWD), *Hardware Multiplier* (HWM) and *Memory Management Unit* (MMU). The performance and the maximum execution frequency vary depending on processor configuration. For its communication purposes, MicroBlaze v7.00 offers a *Processor Local Bus* (PLB) bus interface and up to 16 *Fast Simplex Links* (FSL) interfaces which is a point to point FIFO-based communication channel. In our implementation, MicroBlaze processors are used as simple vision, which contain 5stage pipeline, 32bit integer HWM and enable additional Machine Status Register Instructions, pattern comparator. The MB-PE Tile, shown in Figure 22, contains MicroBlaze, *Instruction side Local Memory Bus* (ILMB), *Data side Local Memory Bus* (DLMB), ILMB BRAM interface controller, DLMB BRAM interface controller and BRAM based 32KByte local on-chip memory. Local memory connects processor through LMB interface controller and LMB memory bus. The FSL interfaces of MicroBlaze are directly connected to OCP Synchronization Adapter and OCP Data Adapter. Processors feed the OCP adapter with data and commands through the

FSL channel. Then the OCP adapter converts these data and commands to OCP compatible signals, which are then consumed by the DATA-NOC and synchronization-NOC. The OCP adapter also can separate PE clock domain from NoC clock domains, benefiting from the bisynchronous FIFOs contained in OCP adapter.

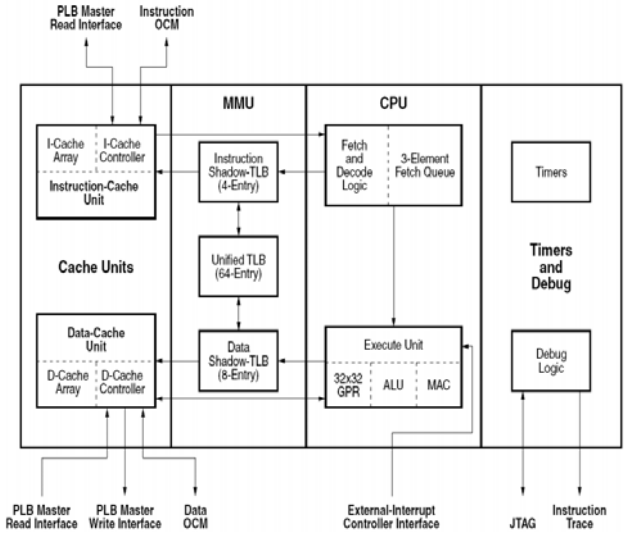


Figure 23 Block diagram of PowerPC processor

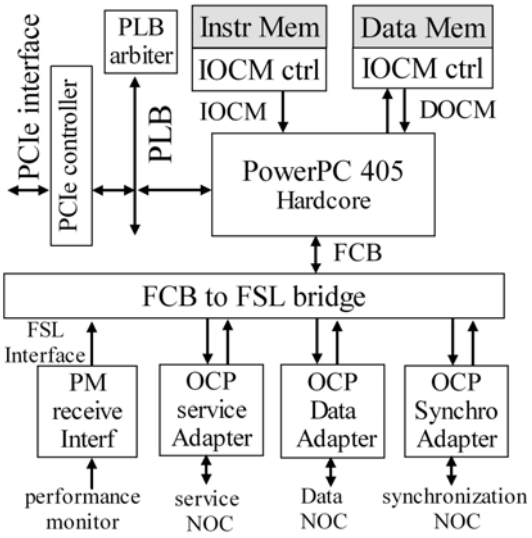


Figure 24 Block diagram of PPC PE Tile

PPC-PE Tile is based on PowerPC 405 processor. As shown in Figure 23 it is a hardcore implemented 32-bit RISC general processor and can run up to 450MHz execution frequency. It contains a 5-stage pipeline, a virtual-memory-management unit, separate instruction cache and data cache units as well as three programmable timers. In our PPC PE

tile, shown as Figure 24, the PowerPC accesses its 32KByte local instruction memory through *Instruction side On-Chip Memory Bus* (IOCM) and accesses his 32KByte local data memory through *Data side On-Chip Memory Bus* (DOCM). The processor joins the FCB-to-FSL Bridge via *Fabric Co-processor Bus* (FCB). We used four of 32 FSL interfaces provided by FCB-to-FSL Bridge. The OCP synchronous adapter, OCP data adapter and OCP service adapter connected to synchronization NOC, DATA-NOC and service NOC respectively. The PowerPC processor can also receive the performance monitoring information from the *Performance Monitoring* (PM) through PM receive interface.

The 16 different tiles can run at largely different clock frequencies. Firstly implementation in hard core or soft core can cause very different frequencies. Furthermore MB-PE tiles with different MicroBlaze configurations can run at different frequencies. The bi-synchronous FIFO pair contained in the OCP adapter makes it feasible to realize the communication between two independent clock domains. The more details and the issue of frequency will be discussed in section IV.

3.2.4 Network on Chip

Our on-chip network connection system is developed with the Packet Transport Units (PTU) from the Arteris Danube library. These PTUs build the packet transport portion of the NOC, which is comprised of a request network and a response network. All the PTUs adopt NoC Transaction and Transport Protocol (NTTP) which is a three-layered approach comprising transaction, transport and physical layers.

The transaction layer is compatible with bus-based transaction protocol implemented in Network interface Units (NIUs). NIU translates third-party protocols to NTTP at the boundary of NoC. We used OCP-to-NTTP NIUs to convert OCP protocol to NTTP protocol.

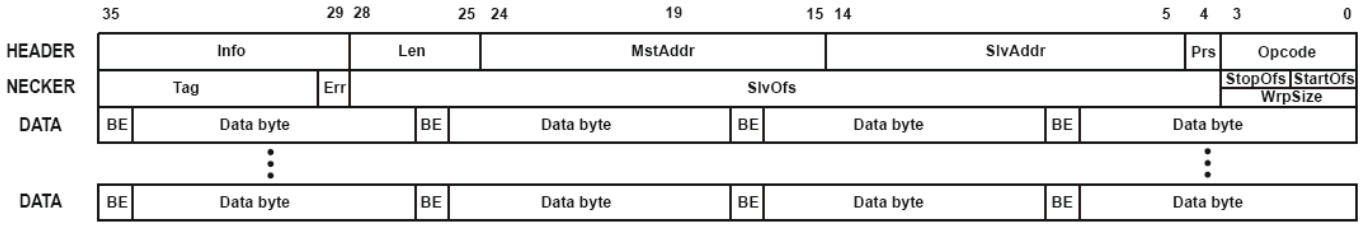


Figure 25 A typical NTTP Request Packet

In the transport layer, NTTP uses the packet-based wormhole scheduling technique. As shown in Figure 25, the packets are comprised of three different cells: a header cell, an optional necker cell and possibly one or more data cells. The header and necker cells contain information relative to routing, payload size, packet type, and the packet target address. The necker cell provides detailed addressing information of the target. The necker cell is not needed in response packet.

NTTP	**Function**
Vld	Data validation
RxRdy	Flow control
Head	Head indication
Tail	Last cell indication
Data	Data word
TailOfs	Packet tail offset
Press	Link pressure

TABLE 2 NTTP signals in physical layer

Packets are split into cells in the transport layer, and then cells are delivered as words in the physical layer. Table 1 shows the NTTP signals using in physical layer. The size and width of link can be defined by the designer according to application case. Within a single clock cycle, the physical layer may carry words comprising of a fraction of a cell, a single cell, or multiple cells. There are five possible link-widths: quarter, half, single, double, and quad.

3.2.4.1 OCP Network Interface Unit:

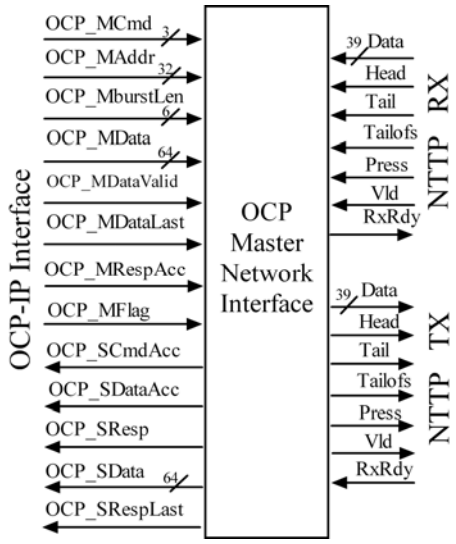


Figure 26. An example of Data OCP master NIU

	Function	
MCmd	master Transfer command	basic
MAddr	master Transfer address	basic
MBurstLen	master Burst length	burst
MData	master Write data	basic
MDataValid	master Write data valid	basic
MDataLast	master Last write data in burst	burst
MRespAcc	master accepts response	basic
MFlag	master flag for pressure level	press
SCmdAcc	Slave accepts transfer	basic
SDataAcc	Slave accepts write data	basic
SResp	slave Transfer response	basic
SData	slave Read data	basic
SRespLast	slave Last response in burst	burst

TABLE 3 the signals for our Data OCP configuration

OCP NIUs interface seamlessly with the Arteris NTTP protocol and provide compatibility with existing protocol OCP 2.2. OCP comprehensively defines an efficient, bus-independent, configurable and highly scalable interface for on-chip subsystem communications and enables IP core creation to be independent of system architecture and application domain. An example of OCP master NIU (OCP-to-NTTP) is shown in Figure 26, it can connect OCP initiator to NTTP interface. OCP master NIU translates OCP requests to NTTP packets and sends them to switch from NTTP TX interface; simultaneously it

receives NTTP packets from NTTP RX and translates to OCP response. OCP slave NIU (NTTP-to-OCP) permits the communication between OCP target and NTTP switching connection system. As described in OCP-IP standard, designers can define different OCP configurations for their IPs. Arteris' OCP NIUs can be configured differently for own utilization at design time and help the NTTP NoC easily communicate with OCP initiators and targets containing different OCP configurations. In our system the OCP configurations in Data-NoC and in synchronization-NoC are different. In Data-NoC the OCP basic signals, burst extensions signals and "MFlag" signal are used, as shown in Table2. The data width of MData and SDatae are 64bits. In synchronization NoC, we just use the basic signals and select 32bits for data width. Burst mode is not supported in Synchronization NoC. Two synchronization modes, locked synchronization and lazy synchronization, are realized thanks to the OCP NIU and the Exclusive Access Manager. The OCP initiator can optionally associate a pressure level to requests in order to indicate service priorities at arbitration points. The pressure-level is passed to the NoC via the "MFlag" input signal, and applies to the "Pressure" field in the packet header cell, as well as the "press" signals in the physical link layer. For address decoding, a Translation Table is integrated in the OCP-to-NTTP unit. This Translation Table receives OCP addresses and split them into two parts. The first part represents the slave address in HTTP header cell. The second is slave offset in NTTP necker cells.

3.2.4.2 Switch:

The Arteris Danube IP switch is the basic element in the connection system. As shown in Figure 27, switch includes several principal elements such as input controller, route table, crossbar, arbiter, output controller. The full crossbar can transfer up to one data word per port and per cycle. The switch also support lock operation and pressure for enhanced arbitration decision making.

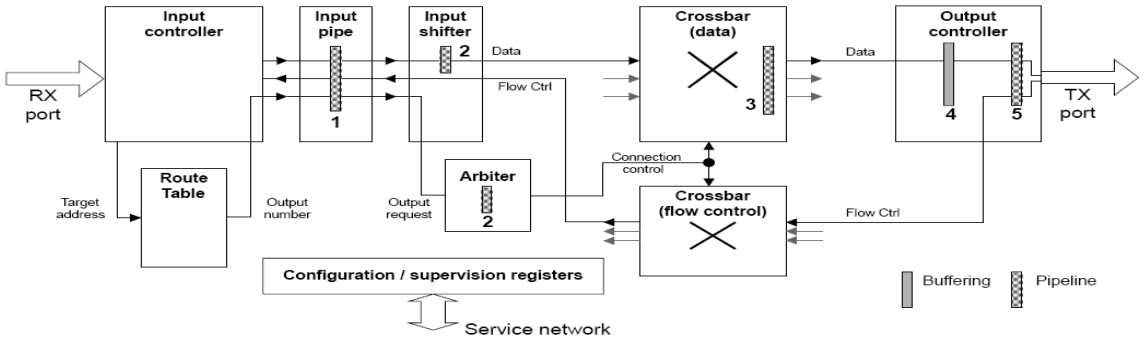


Figure 27. Block diagram of Arteris switch

Four parameters can be customized for specific application requirements.

(1) The ports attributes: up to 16 input ports and output ports of switch can be set.

(2)Arbitration mode: LRU, Round-Robin, Random, FIFO or software programmable.

(3)Route table: it selects the output port that a given packet must take. Types of route table are generic, constant, decode, slice, cluster etc.

(4) Pipelining strategy: several pipeline stages can be added such as input-pipe, output-fwd-pipe, output-bwd-pipe, crossbar-pipe, arbitration-pipe.

3.2.4.3 DATA NOC

As shown in Figure 28, the Data NoC is a cascading multistage interconnection network (MIN), which contains 8 switches for request data routing as well as 8 switches for response. Sixteen OCP-to-NTTP NIUs and four NTTP-to-OCP NIUs are integrated at the boundary of Data NoC. The OCP-to-NTTP NIU converts the OCP master interface to NTTP interface and connects PE tiles to the first stage switches. We place four Master NIUs as a group connected to one switch of the first stage, while the first stage are comprised of four 4IN*4OUT switches for 16 PE tiles. Each switch in first stage distributes four output ports, which are respectively connected to the switches in second stage. The second stage contains four 4IN*1OUT switches. Each output port of switches in second stage is connected to a NTTP-to-OCP NIU, which in turn connects to DDR2 memory controller. Each DDR2 memory controller is attached in 256Mbyte off-chip DDR2 memory. The response part of DATA NOC is a mirror of request part. Through the Data-NOC, sixteen PE tiles connected to OCP Master interfaces can access any DDR2 salve memory.

Between each NIU and switch connecting point, a performance monitoring probe is placed to measure the behaviors of Data NoC. The detail of performance monitoring system will be discussed in 3.4 Performance Monitoring

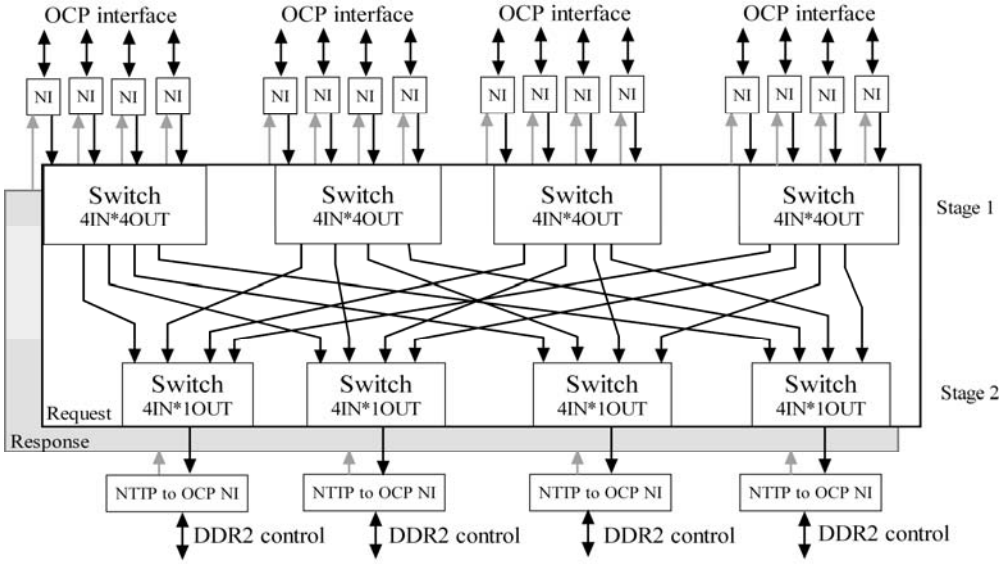


Figure 28. Block diagram of Data NoC

3.2.4.4 Synchronization NOC

MCmd[2 :0]	**Description**
IDLE	Idle
WR	write
RD	Read
RDEX	Read exclusive
WRNP	Write Non post
RDL	Read Linked
WRC	Write Conditional
BCST	Broadcast

TABLE 4 OCP master command MCmd

SResp[1:0]	**Description**
NULL	No response
DVA	Data valid / accept
FAIL	Write conditional fail
ERR	System error

TABLE 5 OCP Slave response SResp

TABLE 4 and TABLE 5 list the Master commands and Slave responses specified in OCP protocol. From these commands and responses OCP protocol can support two synchronization modes among OCP masters. The first one is Locked synchronization, which is an atomic set of transfers. OCP initiator use the ReadExcusive(ReadEX) command and Write or WriteNonpost command to perform a read-modify-write atomic transaction. In our system the NTTP protocol translates such accesses by inserting control packets, Lock and Unlock, on the request flow. As shown in Figure 29, the NIU sends a Lock request packet when it receives the ReadEX command. The Lock request locks the whole path to the NTTP slave. Then a LOAD request packet read the data of NTTP slave. OCP master modifies the data and sends to salve by Write or WritrNonPost command. When the NIU receives the Write command, it write the data to required NTTP salve by a STORE request packet then release the NoC by Unlock request packet. The other competing OCP masters can't access the locked location, until the Unlock packet is sent. Such a mechanism is efficient for handling exclusive accesses to a shared resource, but can result in a significant performance loss when used extensively.

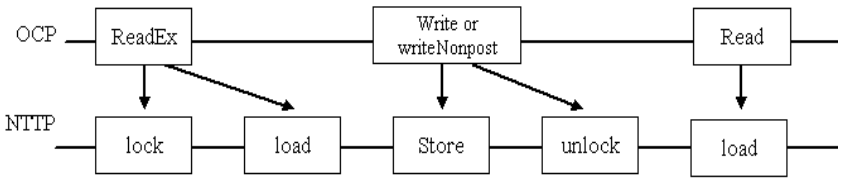


Figure 29. Illustration of Locked synchronization

For allowing other instructions to be executed between the ReadEX and write commands and breaking the atomicity of the exclusive read/write pair, the Lazy synchronization is implemented in our system. This mechanism requires new read and write semantics, commonly known as Load-Linked and Store-Conditional (LL/SC) semantics. LL/SC access must be monitored by access monitor logic, that can be located either in the NoC, or in the memory controller. The ReadLinked command sets a

reservation tag in the monitor logic, with a particular address. The WriteConditional command is locally transformed into a memory write access only if the reservation tag is still set when the command is received. As the tagged address is not locked, the tag can be reset by competing traffic directed to the same location from other Master between ReadLinked and WriteConditional.

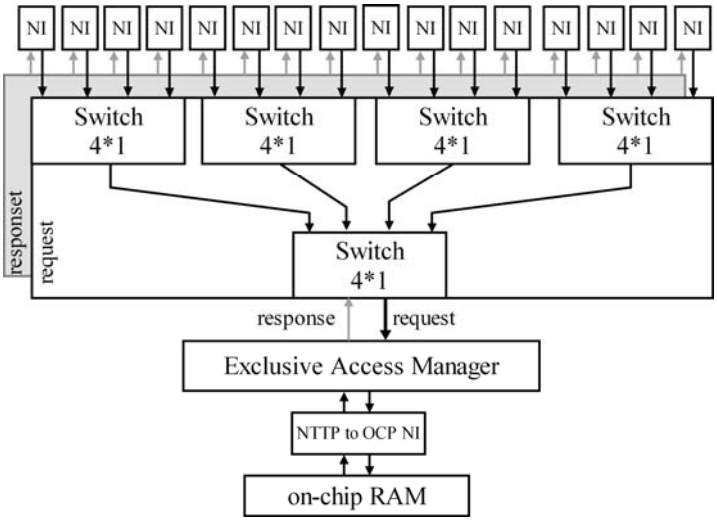


Figure 30. Block diagram of Synchronization NoC

Due to absence of monitor logic in RAM controller, an Exclusive Access Manager from Arteris Danube library is integrated in front of the NI of RAM, as shown in Figure 30. The NTTP protocol controls the lazy synchronization by “exclusive” information bits, which can be found in the information field of the header cell. When the OCP Master-NIU receives ReadLinked command, the “exclusive” bits are encoded in request packets. Next these bits are registered in the Exclusive Access Manager and are updated during transport. When the WriteConditional is received, the Exclusive Access Manager inserts the “exclusive” bits in the response packets for reporting to the initiator. The initiator receives a “Fail” response, that means the other initiators have modified the monitored target. Otherwise it means the monitored target is not modified and WriteConditional is successful. The “Fail” response is different with “Error” response. The “Error” response effectively signals a system interconnect error or a target error.

3.2.4.5 Service NOC

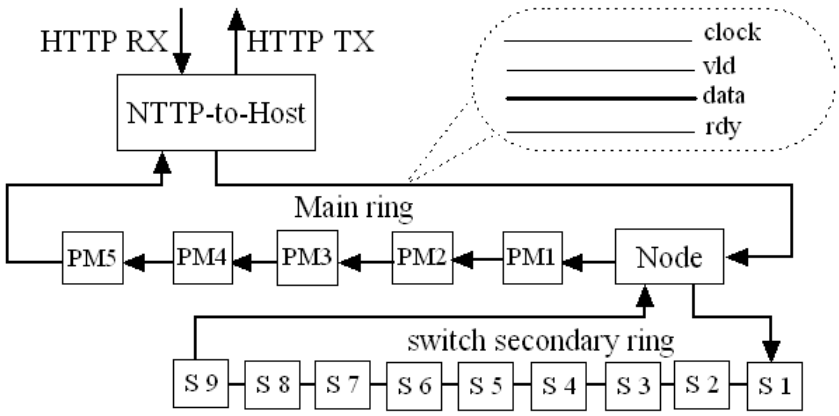


Figure 31. Block diagram of Service NoC

The Service NoC is integrated in the Data NoC to provide reconfiguration services and to report the status of IPs at run time. In the PTUs such as switch and performance monitor, a set of registers can be accessed by the host interface, which is connected to service network. PE tiles can change the values of registers to control the behaviors of PTUs. As shown in Figure 31, the service network includes a main ring and a secondary ring. The main ring comprises a NTTP-to-Host component, a Node and five performance monitors. The NTTP-to-Host translates NTTP data to service host data, which include four signals: clock, Vld, Data, Rdy. Vld indicates valid data; Data carries host packet information; Rdy indicates the receiver is ready to receive data. The Node connects a secondary ring to main ring. The Node component reads and writes data from main ring to its secondary ring only when the data's addresses match one of the base-address; otherwise the data packets continue circulate on the main ring. In our design the switch secondary ring comprises nine switches host slaves. Through this Switch service ring we can check the statuses of switches, configure the route table for each input port of switch and set the arbitration "enable" bit for each output port of switch. The five host interfaces of performance monitors are connected to the main ring. Programmer can control the behavior of performance monitor by updating the registers such as basic status check, monitoring type select, monitoring time select, counter operation select etc.

3.3 Bi-Synchronous FIFO in GALS architecture

The Globally Asynchronous, Locally Synchronous (GALS) approach has been proposed to solve the timing closure problem in deep sub-micron processes by partitioning the SoC into isolated synchronous subsystems that hold own independent frequency. To improve the performance and reduce the power consumption of system, the GALS approach is adopted by using Bi-Synchronization method in our design.

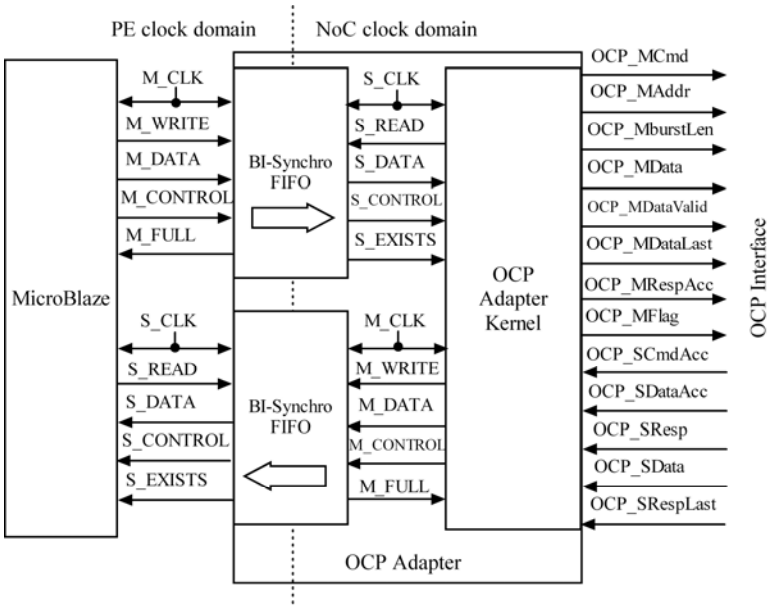


Figure 32. Block diagram of OCP adapter

To tackle the communication issue between two different clock domains and two different protocols, the OCP adapter (as shown in Figure 32) has been designed, which contains one pair of bi-synchronous FIFOs and OCP adapter kernel. The OCP adapter kernel convents FIFO interface to OCP interface. We have three types of OCP Adapter: OCP Data Adapter, OCP synchronization Adapter and OCP service adapter. PE Tiles are connected to Data NoC by OCP Data Adapters, which interface OCP basic signals and OCP burst extension. OCP synchronization Adapter does not support burst mode but it can convert ReadLinked and WriteConditional commands for lazy synchronization. OCP service Adapter is the simplest vision, it supports neither burst mode nor lazy synchronization. All of three types of OCP adapter contain a pair of bi-synchronous FIFOs,

which make PE tiles and NOCs as isolated synchronous islands with independent clock frequencies. Bi-synchronization FIFO is used in Bi-Synchronous communication method to interface two synchronous systems with independent clock frequencies. The signals of Bi-Synchronous FIFO are listed in Table 4. Bi-Synchronous FIFO has two directions interface: sender interface (M_) and receiver interface (S_). Each interface has its own clock signal; M_Clk for the sender and S_Clk for the receiver. PE Tile clock domain can communicate with NoC clock domain via a pair of Bi-synchronous FIFO contained in OCP Adapter.

Name	**Description**
M_Clk	This port provides the input clock to the master interface
M_Data	The data input to the master interface
M_Control	Single bit control signal that is propagated along with the data at every clock edge.
M_Write	Input signal that controls the write enable signal of the master interface of the FIFO
M_Full	Output signal on the master interface of the FIFO indicating that the FIFO is full.
S_Clk	This port provides the input clock to the slave interface
S_Data	The data output bus onto the slave interface
S_Control	Single bit control that is propagated along with the data at every clock edge
S_Read	Input signal on the slave interface that controls the read acknowledge signal of the FIFO.
S_Exists	Output signal on the slave interface indicating that FIFO contains valid data.

TABLE 6 signals for Bi-synchronous FIFO

As mentioned in the part of Processing Element Tile, PPC PE tile and MB PE tiles have different maximum execution frequencies because of different implementation modes: hard-core and soft-core. The maximum execution frequencies of MB PE tiles also largely vary due to different configurations of MicroBlaze. Regarding the NoCs and PE Tiles, different functionalities and performance needs lead different frequencies requirements. The entire system is partitioned into different clock domains depending on the performance requirements and the maximum execution frequencies. In our design, Bi-synchronous FIFO points separate the PE tile clock domains and NoC clock domains as illustrated in Figure 33. Each PE tile can be considered as an isolated frequency island. PPC PE tile containing Hardcore PowerPC405 runs at frequency 300MHz. The Maximum frequencies of MB PE tiles containing soft-cores MicroBlaze vary from 70 MHz to 130MHz depending on the configuration of MicroBlaze and performance requirements. In the NoC side, two

NoCs, Data-NOC and synchronization-NOC have different maximum execution frequencies due to different pipeline strategy, arbitration settings and configuration of input, output numbers. Data-NOC run at 200 MHz, synchronization NoC run at 250 MHz.

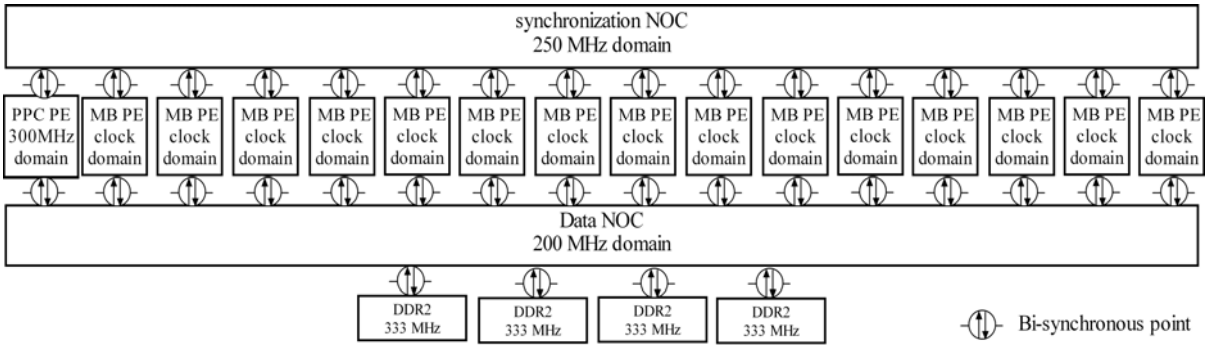


Figure 33 Illustration of clock domain

3.4 Performance Monitoring

Run-time observability is an obligatory condition in embedded system debugging and monitoring. Observability of NoC based SoC is more difficult, the application debugging is impossible without the information of NoC behaviors such as throughput, latency etc. A scalable, reconfigurable, non-intrusive NoC performance monitoring service has been added in our MPSoC. This performance monitoring system can offer run time observability of NoC behavior and export results as frames to a processing target helping application software tuning.

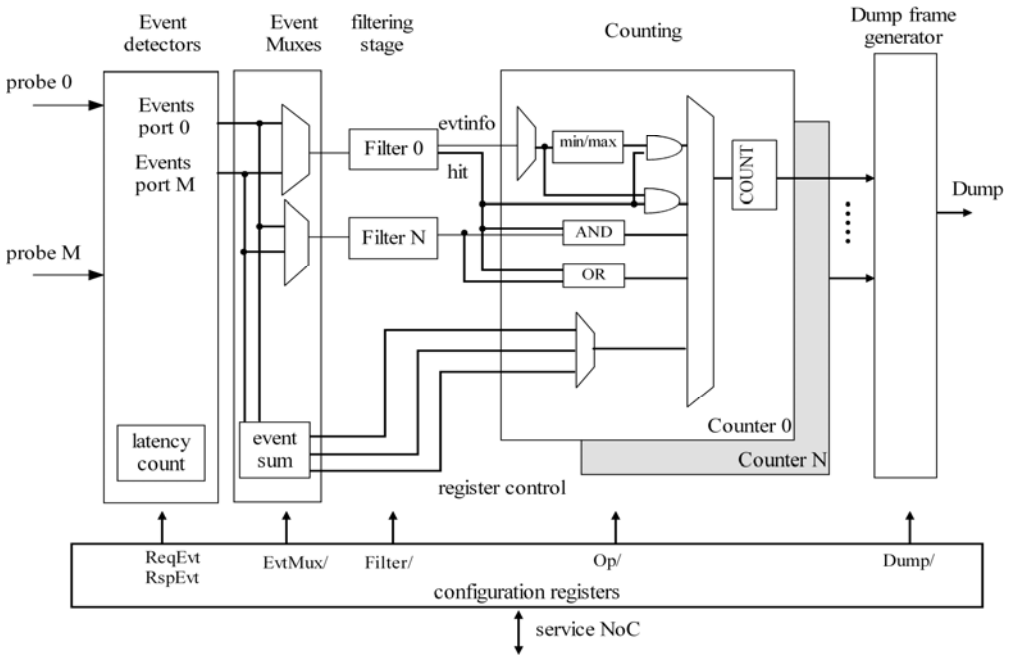
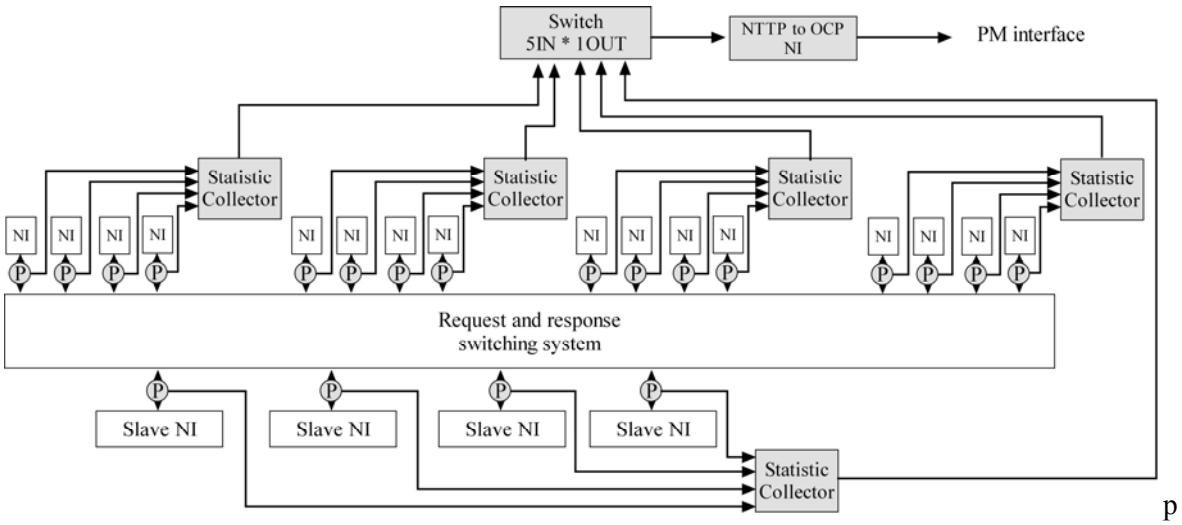


Figure 34. diagram of statistic collector

The basic component in the performance monitoring system is statistic collector from Arteris Danube library, as shown in Figure 34. Statistic collector is based on hardware probes, which can be attached on any NTTP request link or response link in the NoC, or the probes can be directly attached on OCP links. The monitoring probes capture the information passing from the link and detect the monitored events without introducing any flow control in the system. Each statistic collector can provide up to 8 probe input channels, one dump frame output and service host interface. The internal configuration registers control the monitoring behaviors and can be predefined at design time or be modified and control at run time.



means probe point; SC means statistic collector

Figure 35. illustration of performance monitoring system,

As shown in Figure 35, all input and output ports of Data NoC are monitored. Twenty performance monitoring probes are placed at all connection points between NIUs and switches. Each statistic collector provides 4 probes to detect 4 pairs of NTTP request and response links. Four statistic collectors detect and record the 16 connections of PE tiles and one statistic collector for the 4 connections of DDR2 memories. The dump output ports of statistic collector are connected to the PM switch. Statistic Collectors can automatically send the monitoring results to switch when the configured collect time arrive, or programmer can manual set the "send register" to do that. The PM switch exports all the data to PPC PE tile through a NTTP-to-OCP NIU. Using service NoC programmer can change the configuration registers of statistic collector at run time. At the event detector stage, we can configure the detecting event type, such as packets, latency, wait cycle, payload, and idle cycle. The event MUXes select which the input port is monitored by corresponding filter. The filtering stage provides additional selection capabilities by mask and match registers. We can also select the counting mode and the form of output frame.

3.4.1 Performance Monitoring system evaluation

To evaluate the performance monitoring system in our MPSoC platform, we select matrix multiplication as our parallel application, which is widely used in science

computation. Due to its large dimensional data array, it is extremely demanding in computation power and meanwhile it is potential to achieve its best performance in a parallel architecture.

We give a brief description of the algorithm utilized for the matrix multiplication. Suppose that A is an M*K matrix, B is a K*N matrix, and the result C is an M*N matrix. In the phase of task dispatch, A is partitioned into p tasks, each task contains M/p lines of matrix A, where p indicates the number of PE Tiles. After this partitioning is done, each PE tile works with the task from A corresponding to its tile number and the entire B. So at each iteration, it reads in the one block of A, which is then multiplied with all the blocks of B, resulting in a M/P * N partial result matrix. The partial result is first stored in the local memory of PE tile and is stored in the corresponding memory location (according to the row number, the column is complete) before the start of the following iteration.

According to the parallel algorithm described above, which demands a great amount of memory access for loading original matrix blocks and storing partial results, a successful implementation depends largely on memory access scheme. In regarding the NoC architecture, a memory bank can be accessed by one PE tile at a time. If the memory bank is occupied by one PE Tile, the other PE Tile can read or write this memory bank until the path to access this memory bank is free.

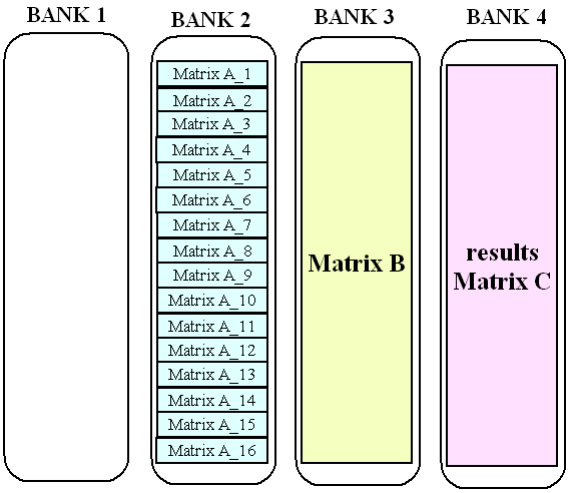


Figure 36 Matrix per bank data distribution scheme

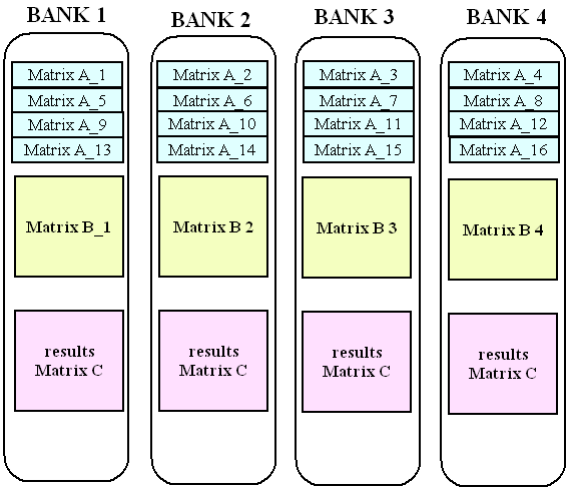


Figure 37 Line interleaved data distribution scheme

Solution1	packets	laten
MB PE0	149760	73,5
MB PE1	149760	73,5
MB PE2	149760	73,6
MB PE3	149760	79,2
MB PE4	149760	79,2
MB PE5	149760	79,2
MB PE6	149760	79,1
MB PE7	149760	80,2
MB PE8	149760	78,0
MB PE9	149760	78,0
MB PE10	149760	77,8
MB PE11	149760	77,9
MB PE12	149760	77,9
MB PE13	149760	77,9
MB PE14	149760	77,8
Bank1	0	
Bank2	28739	10,2
Bank3	213693	118,
Bank4	12661	4,88

TABLE 7 performance monitoring results Solution1 : matrix par bank

Solution2	packets	latency
MB PE0	149760	32,11
MB PE1	149760	32,12
MB PE2	149760	31,66
MB PE3	149760	33,41
MB PE4	149760	33,44
MB PE5	149760	33,55
MB PE6	149760	33,29
MB PE7	149760	33,38
MB PE8	149760	30,90
MB PE9	149760	31,07

MB PE10	149760	31,10
MB PE11	149760	31,36
MB PE12	149760	30,93
MB PE13	149760	31,29
MB PE14	149760	31,37
Bank1	266352	26,13
Bank2	271283	25,65
Bank3	268965	25,68
Bank4	277764	25,48

TABLE 8 performance monitoring results Solution2 : Line interleaved

Solution3	packets	latency
MB PE0	149760	20,16
MB PE1	149760	20,28
MB PE2	149760	20,31
MB PE3	149760	20,36
MB PE4	149760	20,37
MB PE5	149760	20,28
MB PE6	149760	20,24
MB PE7	149760	20,36
MB PE8	149760	18,29
MB PE9	149760	18,13
MB PE10	149760	18,09
MB PE11	149760	18,21
MB PE12	149760	18,29
MB PE13	149760	18,13
MB PE14	149760	18,18
Bank1	257682	19,38
Bank2	271508	18,76
Bank3	266720	18,97
Bank4	271517	18,91

TABLE 9 performance monitoring results Solution3 : Line interleaved with shift access

Firstly we naturally store each matrix in a separate memory bank. As shown in Figure 36, Matrix A is stored in Bank1; Matrix B is in Bank2 and Bank3 for Matrix C. We executed the application and used performance monitoring to record the number of packets and the average latency for each monitoring probe. PPC PE Tile report the results to programmer through PCIe interface. As shown in TABLE 7, the latency of bank3 and the number of packets are exceptional high, which means Bank 3 is highly charged by conflicted traffic. This also shows an unbalanced traffic distribution for the four memory banks. To equilibrate the data distribution we changed the scheme shown in Figure 37. PE

Tiles read the matrix B with an order access. PE tiles firstly read the first column of matrix B which is stored in Bank0, and then read the second column in Bank1. The monitoring results are shown in TABLE 8. We can see that the latencies are smaller than the first memory scheme. Then for the second memory distribution scheme, we used another data access strategy: shift access. we shift their data access order for first loading of Matrix B: PE Tile 1,5,9,13 read the first column in Bank0, PE Tile 2,6,10,14 read the second column in Bank1, PE Tile 3,7,11,15 read the third column in Bank2, PE Tile 4,8,12,16 read the fourth column in Bank3. Our performance monitoring system record the number of packets and latency of all monitoring points and the results are shown in TABLE 9. The latency of the third data distribution scheme is much smaller than the other schemes. Figure 39 shows the comparison of execution cycles for three schemes. We can clearly see the number of execution cycles is significantly reduced.

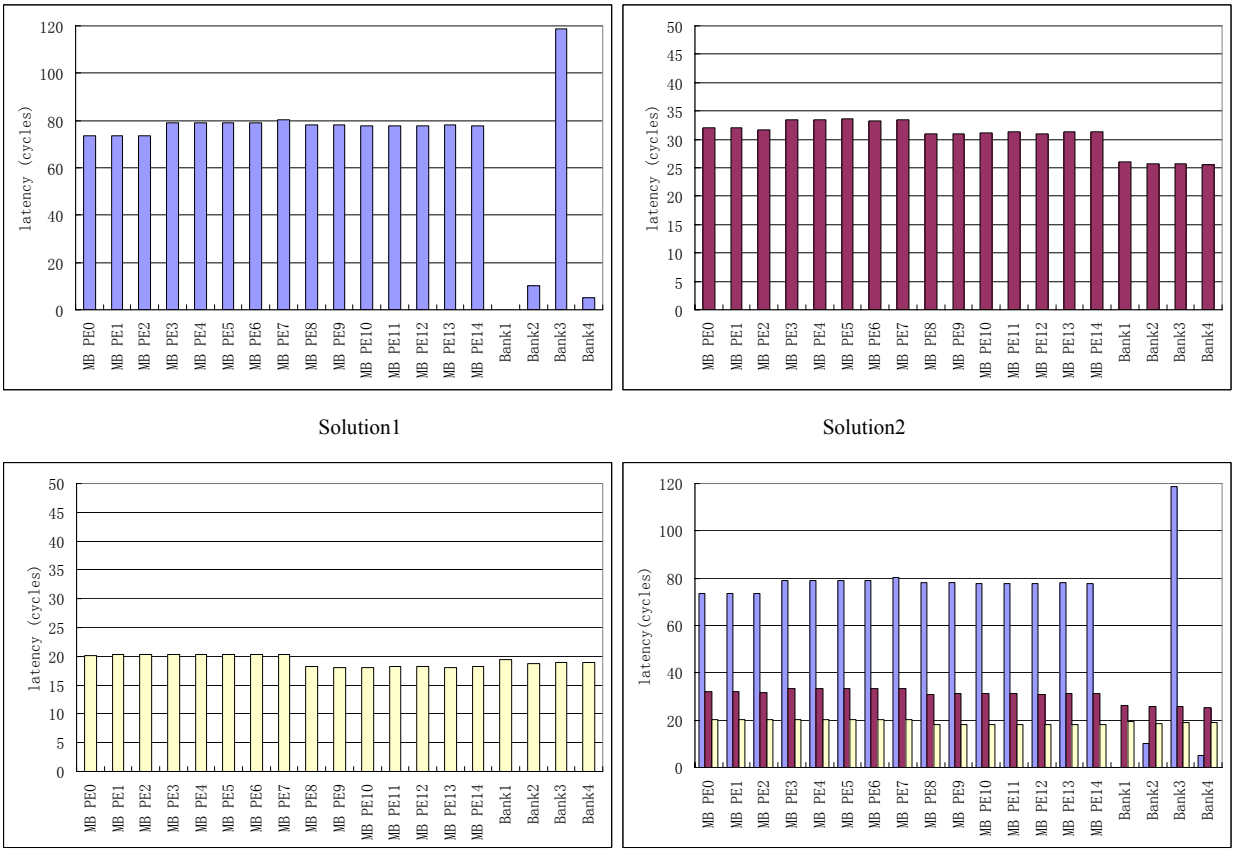


Figure 38 Latencies of three data access scheme.

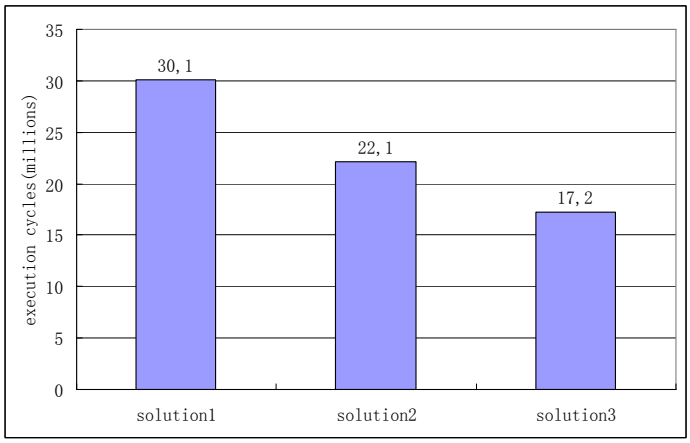
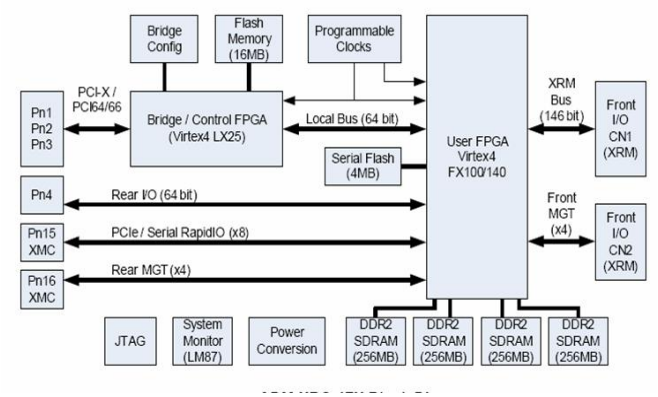


Figure 39 comparison of the execution cycles for three data access scheme.

3.5 Implementation



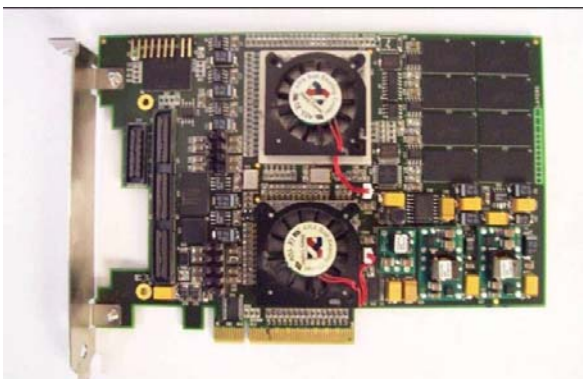


Figure 40 Block diagram of Alpha-Data FPGA platform card ADPe-XRC-4

The whole multiprocessor system is implemented on Alpha-Data FPGA platform card ADPe-XRC-4. As shown in Figure 40, the ADPe-XRC-4 is a high performance reconfigurable PCI Express card based on the Xilinx FPGA Virtex-4 FX140. The FPGA is

connected to four banks of DDR2 memory, while each 256MBytes DDR2 bank is comprised of two Micron MT47H64M16. The MPSoC is synthesized, placed and routed by Xilinx tools: EDK 9.2 and ISE 9.2, and the results of FPGA resource utilization are shown in TABLE 10. The multiprocessor system used about 90% slices of FPGA and 65% Blocked RAM. Each MicroBlaze takes 3 DSP48 blocks, while 15 MicroBlaze processors use 45 DSP48 blocks. Digital Clock Managers (DCM) are dispersively distributed in FPGA to generate different clock frequencies for different clock domains. 16 PE tiles, Data NoC, Synchronization NoC and DDR2 are separated as 19 clock domains and need totally 19 DCMs. Regarding the slices utilization, Figure 41illustrates the percentage of slice utilization. 16 PE tiles take about a half of total Slices. Data-NoC and synchronization NoC occupy about 29.5% and 9.5% respectively. About 8% of slices are used for DDR2 controllers.

Number Of DCM_ADVs	19 out of 20	95%
Number of DSP48s	45 out of 192	23%
Number of RAMB16s	357 out of 552	65%
Number of Slices	57344 out of 63168	90%
Number of PPC405s	1 out of 2	50%

TABLE 10 FPGA resource utilization

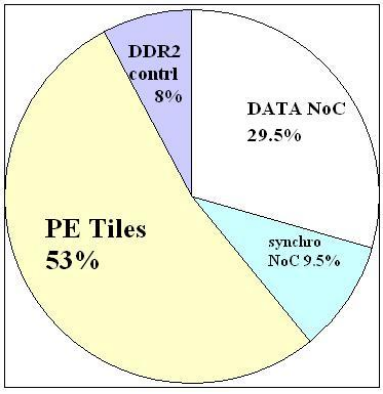


Figure 41 the percentage for FPGA Slices utilization

	Number	Area (slices)	Percentage
Master NIUs	16	3504	16%
Slave NIUs	4	1740	8%
Statistic collectors	5	9976	45%
Switches	16	6896	31%

TABLE 11 data-NOC resource utilization

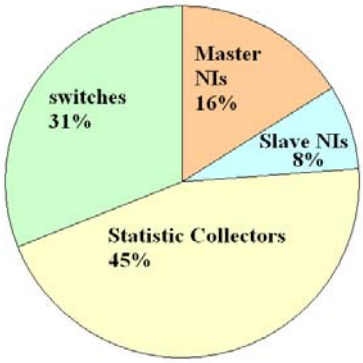
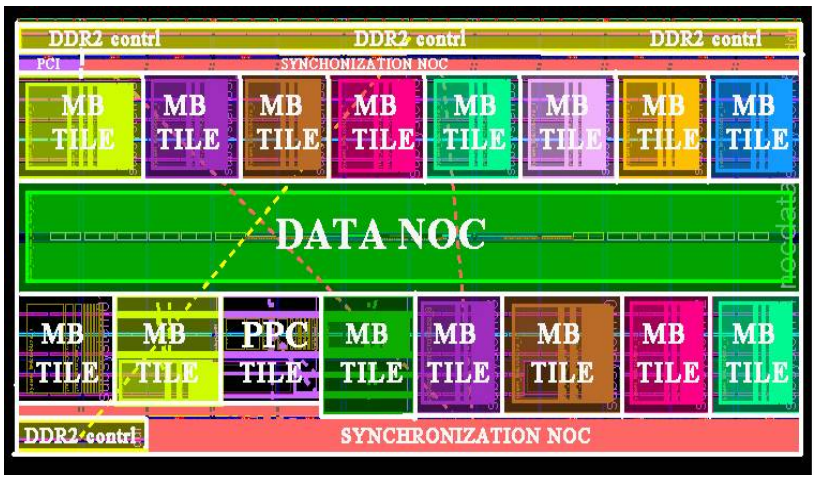


Figure 42 the Area percentage of Data-NOC



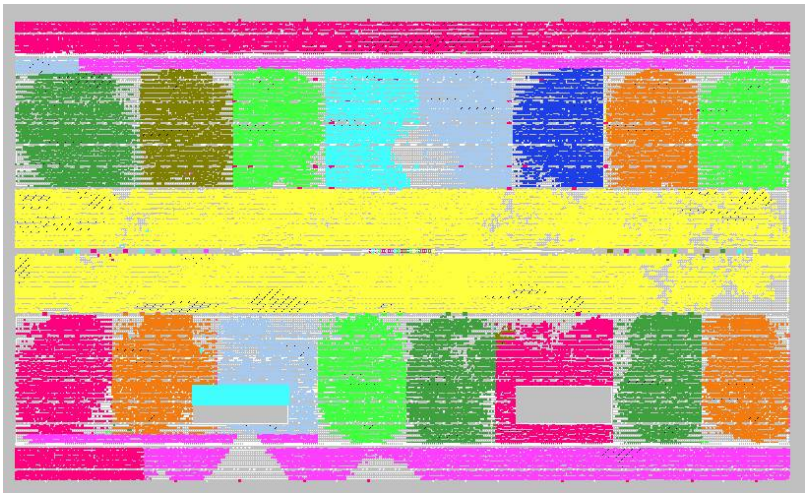


Figure 43 Floor planing and placed FPGA of platform

3.6 Conclusion

In this chapter we presented an FPGA implementation of a NoC-based multiprocessor embedded system. Three NoCs provide different functionalities for sixteen PE tiles. The cascading Data-NoC connects PE Tiles and DDR2 memories with a high bandwidth; synchronization-NoC offers two synchronization modes for PE tiles. And users can check and configure IPs of Data-NoC through our service NoC. We also demonstrated the use of our performance monitoring system for software debugging and tuning. With the bi-synchronous FIFO method, our GALS architecture successfully solves the long clock signal distribution problem and allows that each clock domain can run at its own clock frequency. Xilinx Virtex4 FX140 FPGA was selected to provide large logic resources with quick implementation and testing. FPGA design environment can offer large number of IP to reduce design efforts and decrease the pressure of time-to-market. For example, in our system a number of IP can be found from the Xilinx EDK library, such as MicroBlaze processor, LMB, FSL, BRAM. As future work, we will increase the number of tiles and all the system will be implemented in ASIC. Benefiting our OCP interface we can change the type of processor without difficulty.

Chapter 4 MPSoC Performance Evaluation

4.1 Using encryption as evaluation application

Data security plays a important role in the design of embedded system today, many embedded application rely heavily on security mechanism. Two commonly used cryptographic algorithms: the AES (Advanced Encryption Standard) [84] and the TDES (Triple Data Encryption Standard)[83] have been chosen as our application to evaluate our multiprocessor platform.

A common implementation that favors performance of these algorithms is LUT (LookUp Tables) where one or more of the operations in the algorithm are pre-calculated for all possible inputs and stored in tables. We use this implementation in our paper because it is simple to implement, fast and it fits into our architecture (area is already fixed by the choice of processing elements).

The TDES is built in a way that encryption and encryption uses the same function but with inverse order in rounds keys which leads to small footprint. The AES on the other hand have an Encrypt and Decrypt functions that uses different forward and inverse operations which means that the AES have a bigger footprint in terms of code, where 2 functions must be implemented instead of one for encryption/decryption, and in terms of the size of lookup tables as we need different tables for these two operations, however the AES is faster in execution and provides a much better security.

4.2 The algorithm

The TDES is a symmetric block cipher that has a block size of 64 bit and can accept a 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 was introduced by IBM in early 70's then was adopted by the US government as encryption standard for official use, it is a block cipher with 64bit block size and 54 bit key.

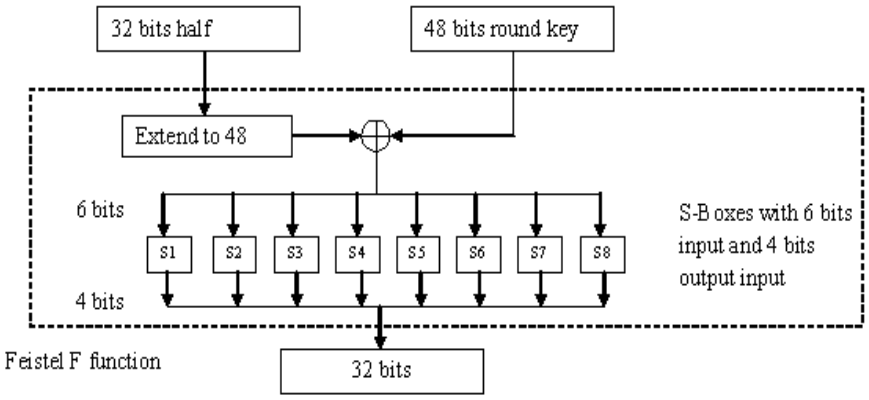


Figure 44: Feistel function F (SBoxes)

The TDES starts by dividing the data block into two 32bits 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 45, 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 44 shows the Feistel F function.

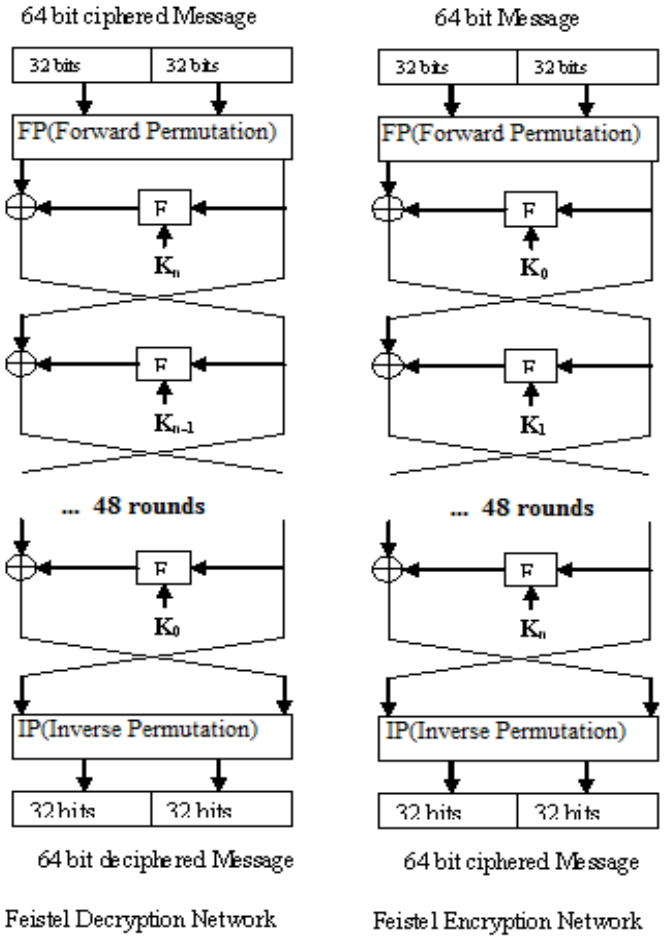


Figure 45 TDES encryption and Decryption schemes (Feistel Network)

4.2.1 The AES (Advanced Encryption Standard)

The AES is the successor of DES and was adopted also by US government for official use as a successor of the aging DES after a contest designate at determining best all around algorithm. The wining algorithm called Rijndael was slightly modified and adopted, a full description of the AES can be found in [84].

The AES algorithms is a block cipher algorithm that takes 128 bit sized block with a key of 128, 192 or 256 bit length, the algorithm executes from 10 to 13 identical rounds plus one final different round, the number of rounds is determined by the key length, a key expansion (also known as key scheduling) is executed prior to encryption or decryption, key expansion for encryption is different from key expansion for Decryption) increasing the footprint of implementation also. Both encryption and decryption in the AES implementation were arranged in an identical

manner and requires the same amount of operations. Figure 46 shows general operations flow of the AES.

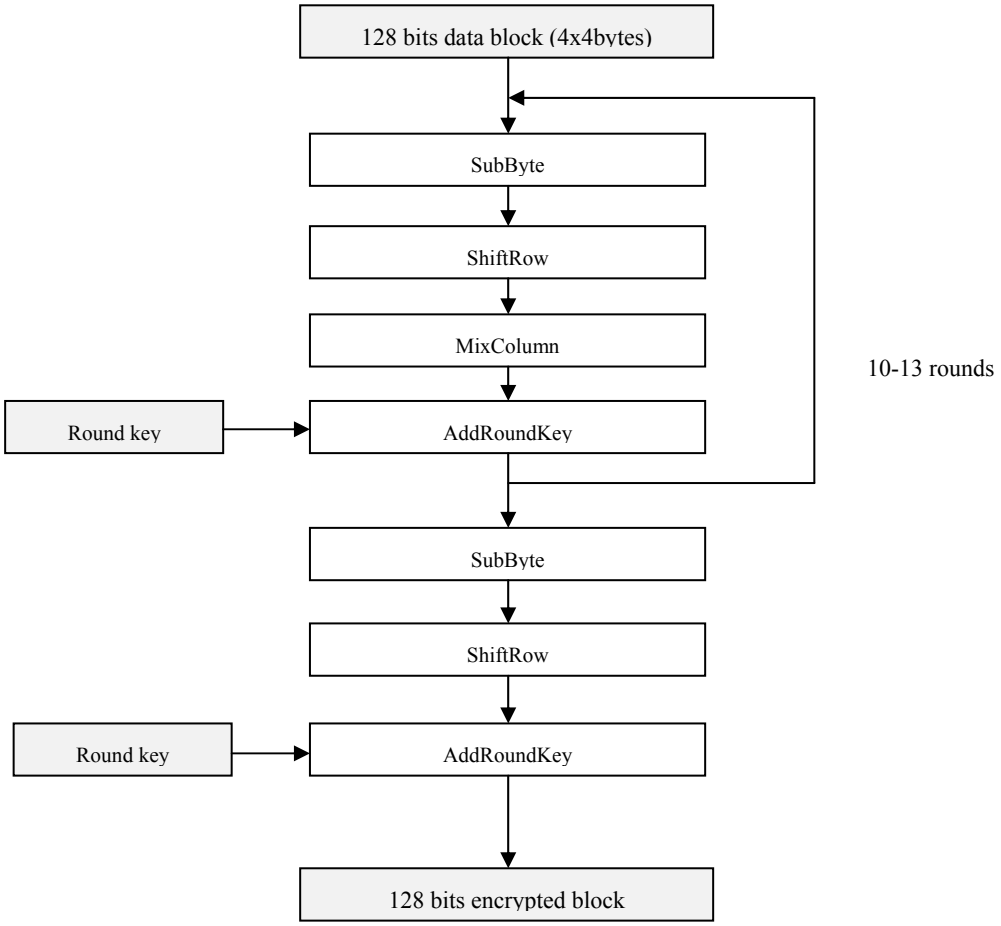


Figure 46: AES general flow

The 128bit data block is represented in a 4x4 matrix with 8bit width for each cell, this matrix is called the State variable. Operations are carried out on State. First there are 13 identical rounds (in our case with 256 bit key) then a final round. Each round consists of

- SubByte: each byte in the matrix is substituted with another byte defined by a multiplicative in $G(2^{8})$ field to assure non linearity.
- ShiftRow: a cyclic shift of the rows of the matrix is executed i^{th} row shifted I times starting with no shift at row 0.
- MixColumn: the four elements of each column of the matrix are multiplied in $G(2^{8})$.by the polynomial $3x^3 + x^2 + x + 2$. Modulo $x^4 + 1$.
- AddRoundKey: finally mixing the matrix with the round key generated earlier in the corresponding KeySetup function.

The last round contains all steps except for the column mixing. Both Byte substitution and column mixing are operations over G(28) field and they are usually implemented in lookup tables consisting of 4 arrays each is of 256 word with 32bit word width totaling 4096 bytes needed storage another 4096 bytes are needed for the decryption containing the inverse operation in the G(28) field. Other operations are basically simple operations like shifts on indexes

4.3 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 47.

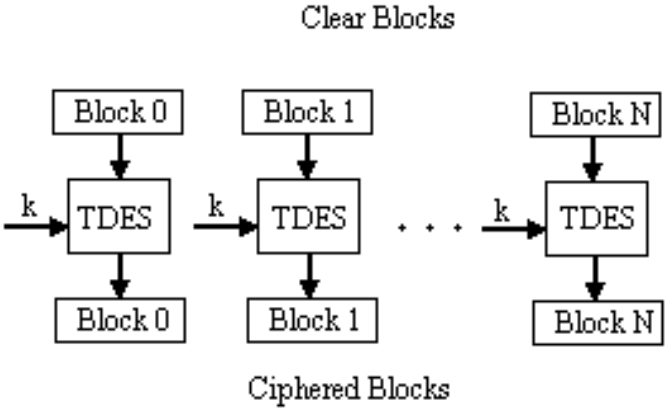


Figure 47 ECB operation mode for the TDES block cipher

The problem in ECB mode is when using it to encrypt identical data blocks with the same key the encrypted outputs will also be identical which means that if we encrypt an image the outlines of that image will be visible when visualizing the encrypted output this effect is illustrated in Figure 48

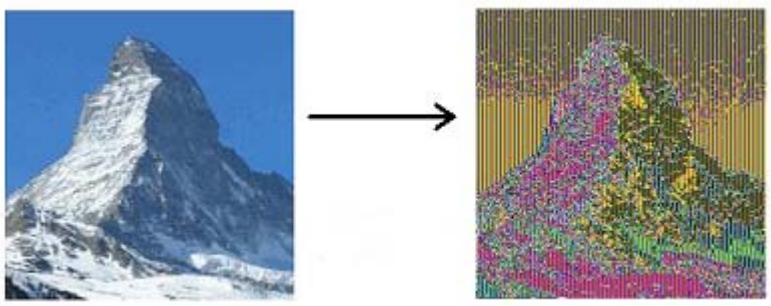


Figure 48:ECB operation mode

A solution to such a problem is achieved by chaining blocks by mixing the encrypted output with the next block in the message. The CBC (Cipher Block Chaining) is a strait forward chaining where the first block is mixed with an Initial Victor IV then the output of this block is mixed with the next block to be cipher and so on as illustrated by Figure 49 whereas Figure 50 shows the visualization of the encrypted image.

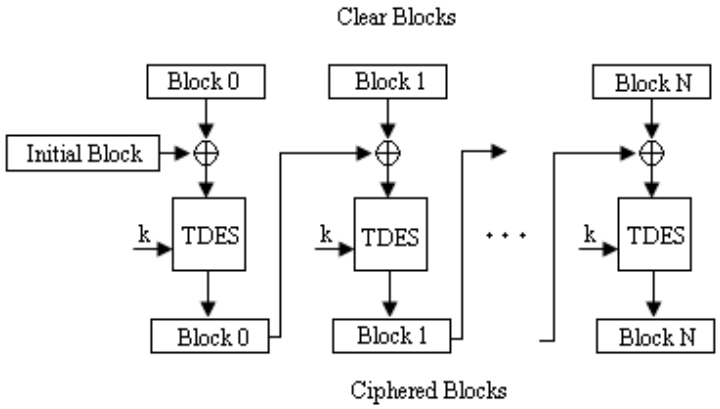


Figure 49: CBC operation mode for the TDES block cipher

There are other modes of chaining that are introduced for different considerations that can be presented, from implementation point of view, by the CBC like CFB (Cipher FeedBack).and OFB (Output FeedBack).

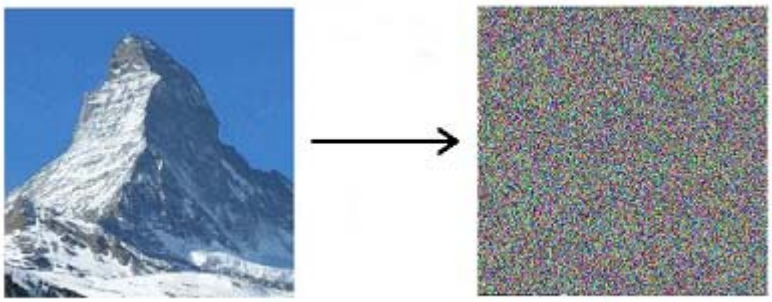


Figure 50: CBC operation mode

4.3.1 TDES Parallelization

We base our work on the C implementation from NIST, 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.

4.3.2 Data parallelization approach

4.3.2.1 Design Description

In this method, data are divided between the 16 processing elements. Two implementations were realized both in CBC mode (illustrated in Figure 50) and in ECB mode as well.

External DDR memory is physically segmented into 4 banks, further more and for both implementations memory is divided into 16 equal segments 4 segments in each banks, 1 segment per processor. The repartition of memory between processors is arranged to avoid collision at processors side switches as illustrated in Figure 49 to allow the simultaneous use of all 4 switches channels.

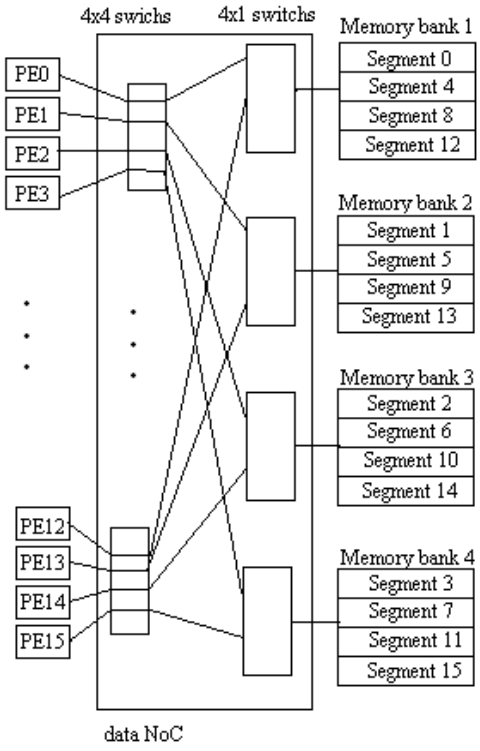


Figure 51: external memory segments repartition between PEs (segment number n is dedicated to PE number n)

For the CBC mode, parallelization is based on splitting the CBC chain into N chain with N the number of involved processors, each chain starts with a different IV (Initial Vector).

Initial Vectors are generated according to recommendations using a counter that is encrypted with the same key as a random IV generator as illustrated in Figure 51.

One processing element (PE0), is programmed to be a master, it has the task of preparing data and key and storing them in the external DDR memory, encryption, managing PEs synchronization and measuring performance. PE0 divide and copy data into PEs memory segments in an even sizes (as possible), calculates round keys then stores these data, keys and data size into corresponding segments, after that PE0 gives a start signal to the other 15 processing elements signaling valid data availability, after that all 16 processors starts encrypting/decrypting that data in its own memory segment, upon finish each PE signals job finished to PE0 by writing a flag to the synchronization memory.

The master PE0 first receives the key via the PCIe then applies key scheduling and stores rounds keys in known location in memory, then it reads data and stores it in equal shares in the 16 memory segments and stores data size in

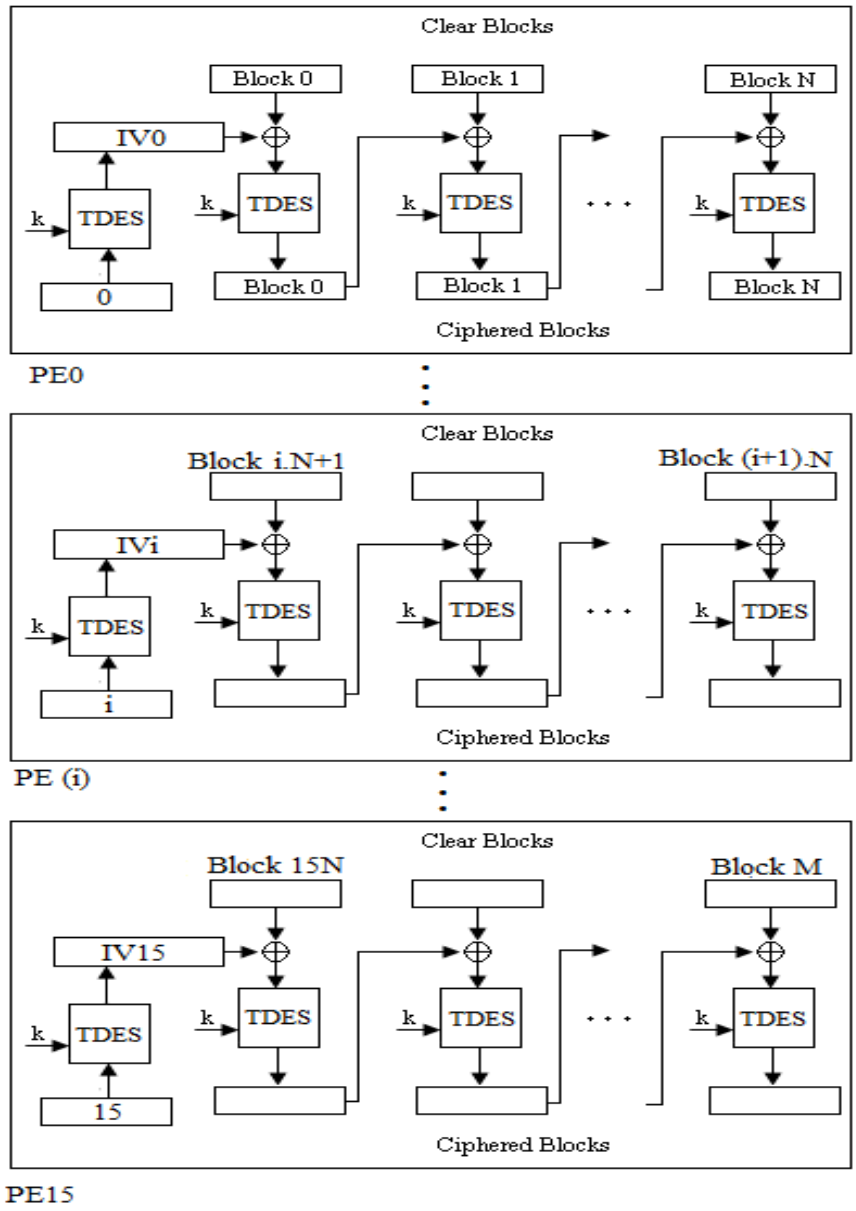


Figure 52: CBC mode mapping into the 16 Processing Elements platform

known location also, once this preparation phase ends, master processor starts a timer then gives launch signal to other processors via the synchronization memory which is being monitored by slave processors for start signal. After the launch signal is given each processor reads rounds keys and data size then starts encrypting (or decrypting) data located in its corresponding segment, the ciphered text is overwritten

on clear text. Once all data are processed each processor writes a flag in the synchronization memory to signal that it has finished job, while master PE0 waits, after it finished its job, until all processors have finished then captures timer time in number of clock cycles.

4.3.2.2 Timing and memory access schemes

In this layout data memory access conflict can occurs at the Go signal given by PE0 where all processors starts by reading first block in there segments, processors using the same data bank will have an access conflict one time, this will be transformed into order memory request arrival next time because all processors executed a task of equal time T0 that is considerably longer than memory access time as illustrated in Figure 53.

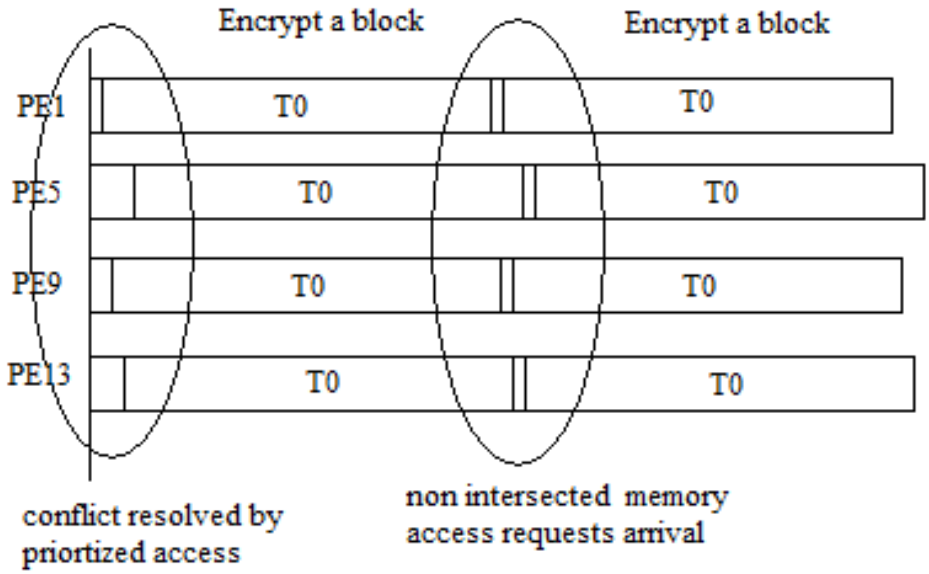


Figure 53: Timing and memory conflict in Data parallel method

This means that memory bandwidth will cause a slow down when the following inequity is true

$$N \times T_m > T_0$$

Where

N is the number of Processors per memory bank

Tm is the memory access time needed to read on data block

T0 is the time needed to encrypt one block

This illustrates the interest of memory segmentation.(more banks mean smaller N).

4.3.3 Pipelined approach

4.3.3.1 Design description

In this method the algorithm execution is divided between the 16 processing elements. However and because of the architecture with no possibility for direct connection between processors, a connection is carried out via the DDR memory while the synchronization memory is used to assure data validity.

To assure balance in computation loads between processors we measured the number of cycles necessary to execute each one of the basic functional entities (FP,F and IP) on one processor, the profiling results are illustrated in table 1, based on these numbers, a repartition of tasks between processors, a most balanced distribution at this level of granularity, is adopted. This distribution is illustrated in Figure 52. We can see that there is a bottle neck at processors PE1 and PE2 with maximum computation time, for overall time we must add data memory access time and synchronization memory access time, TABLE 12 shows the task overall number of cycles without synchronization mechanism, this will allow the measurement of average delay introduced by this mechanism.

Function	Cycles
FP (Forward Permutation)	220
F(Feistel function and Network	168
IP(Inverse Permutation)	196

TABLE 12 basic functions of TDES profiling on one processor

The synchronization is achieved using one common location between each two adjacent processors, this location is set by the processor i to signal the presence of valid data and clear by the i+1 processor to signal that the data has been read and common data location between the two processors can be used again to write data for

the next block. This means that the i-th processor have to check this flag to make sure not to overwrite data that is not read.

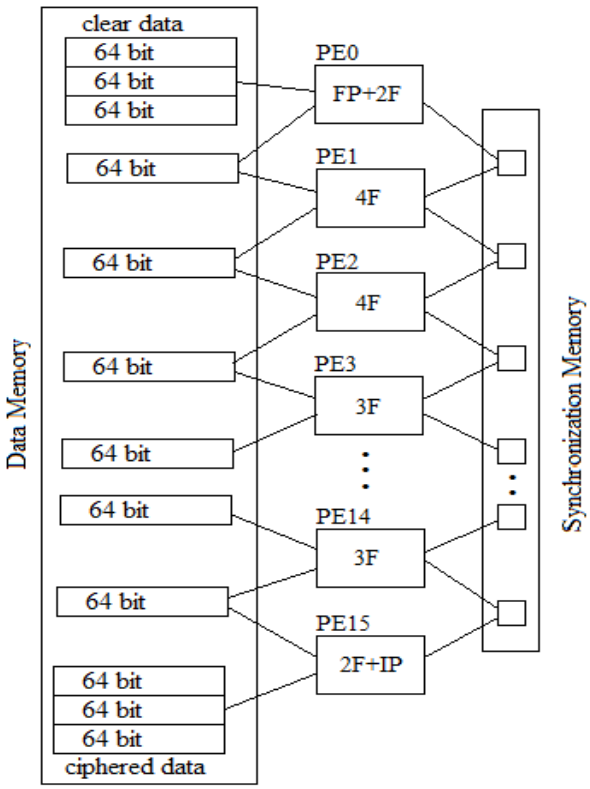


Figure 54: Pipeline Like method tasks repartition and memory sharing

From TABLE 13 and we can see that the bottle neck is at PE1 and PE2, a maximum average of number of cycles per block is governed by this bottle neck that will set the pace for all other processors.

Task	Cycles (approximate)
FP + 2F	680
4F	800
3F	630
2F + IP	660

TABLE 13 tasks executed on Processing Elements in cycles without synchronization overhead

4.3.3.2 Timing and memory access scheme

In this pipeline like approach tasks executed by different processors are not equal and memory access is considerably more important than the data parallel method where intermediate results are stored in memory as well causing considerable loss in performance for each PE memory access takes between 14-17%, data memory access however is regulated by synchronization mechanism, this mechanism is optimized to minimize traffic in the synchronization NoC in order to minimize synchronization overhead, this is carried out by introducing a carefully calculated delay to lineup different processes in PEs as illustrated in Figure 55, these delays are experimentally tuned for minimum overhead so we can get on average one read and one write on the synchronization memory per block.

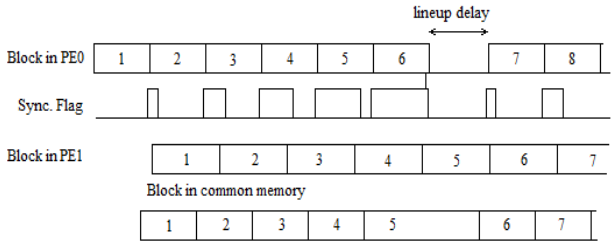


Figure 55: timing and synchronization between PE0 and PE1

No data memory conflict exists because shared memory is repartitioned in the 4 memory banks, this allows along with the shifted access caused by the synchronization to keep the data memory access without conflict.

4.4 Results and Discussion

We passed the same data size for both algorithms, this meant the blocks number passed to the TDES were the double of the blocks passed to AES because of the block size of each algorithms, the overall number of cycles taken to achieve the calculation of all data, as well as other resulting figures are illustrated, also we tested the speedup varying the number of processors.

We tested both algorithms with and without barrel shift in the processor element, obtained the results for the AES for single and 16 processors are presented in TABLE 14

AES (block size 128 bits)

Test Mode	overall number of Cycles	number of encrypted blocks	throughput at 100MHz (Kbytes/sec)	Speedup	cycles/block
single Processor	107 418 660	16 784	244,1	-	6 400
16 processors	6 721 423	16 784	3 901,7	15,98	400

TABLE 14 AES executed cycles to encrypt 16784 blocks on PEs without barrel shifters

The results for the single and 16 processors provided with barrel shift are provided in TABLE 15

AES (block size 128 bits) with barrel shift

Test Mode	overall number of Cycles	number of encrypted blocks	throughput at 100MHz (Kbytes/sec)	Speedup	cycles/block
single Processor	67 028 740	16 784	391,3	-	3 994
16 processors	4 194 311	16 784	6 252,5	15,98	250

TABLE 15 AES executed cycles to encrypt 16784 blocks on PEs with barrel shifters

Results for TDES algorithm single and 16 processor are presented in TABLE 16

TDES (block size 64 bits)

Test Mode	overall number of Cycles	number of encrypted blocks	throughput at 100MHz (Kbytes/sec)	Speedup	cycles/block
single Processor	424 111 884	33 568	61,8	-	12 634
16 processors	26 531 072	33 568	988,5	15,98	790

TABLE 16 TDES executed cycles to encrypt 16784 blocks on PEs without barrel shifters

While in TABLE 17 the results for TDES algorithm for single and 16 processor with barrel shift

TDES (block size 64 bits) with barrel shift

Test Mode	overall number of Cycles	number of encrypted blocks	throughput at 100MHz (Kbytes/sec)	Speedup	cycles/block
single Processor	277 854 154	33 568	94,4	-	8 277
16 processors	17 381 660	33 568	1 508,8	15,98	518

TABLE 17 TDES executed cycles to encrypt 16784 blocks on PEs with barrel shifters

We note that the use of barrel shift has improved performance by 38% in the case of AES and by 35% in the case of Triple DES.

The achieved speedup is linear which reflects the independence of executed blocks and the minimum communications between processor in this method (synchronization is required only at start and end).

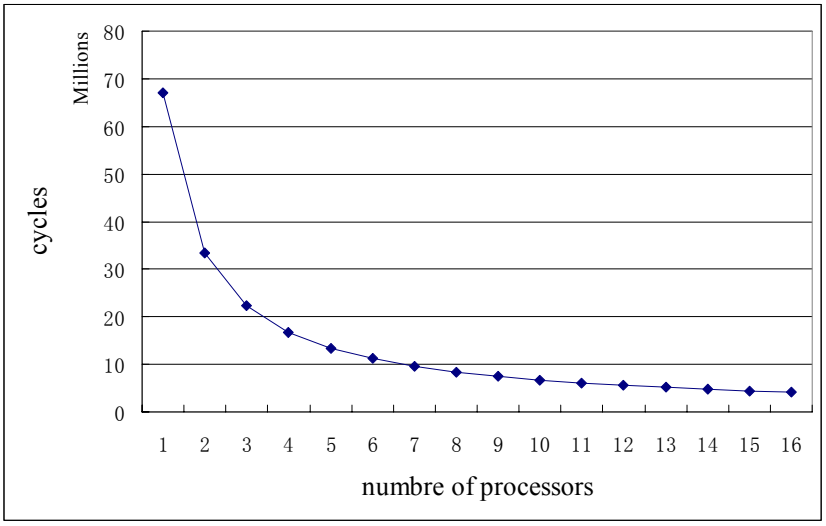


Figure 56: the number of cycles vs. the number of processors for AES

Figure 56 shows the overall number of cycles to encrypt 16784 blocks in the AES algorithms using different number of processors.

The detailed results along with the speedup and throughput at 100 MHz for different number of processors with barrel shifters are shown in TABLE 18

Number of processors	Cycles	throughput at 100MHz (Kbytes/s)	Speedup
1	67,028,740	391	1.00
2	33,501,403	783	2.00
3	22,364,912	1,173	3.00
4	16,772,947	1,564	4.00
5	13,419,492	1,954	5.00
6	11,185,268	2,345	5.99
7	9,586,196	2,736	6.99
8	8,387,108	3,127	7.99
9	7,455,626	3,518	8.99

10	6,712,291	3,907	9.99
11	6,100,736	4,299	10.99
12	5,593,368	4,689	11.99
13	5,161,574	5,081	12.99
14	4,793,823	5,471	13.98
15	4,474,090	5,862	14.98
16	4,194,418	6,253	15.98

TABLE 18 TDES executed cycles to encrypt 16784 blocks on PEs with barrel shifters

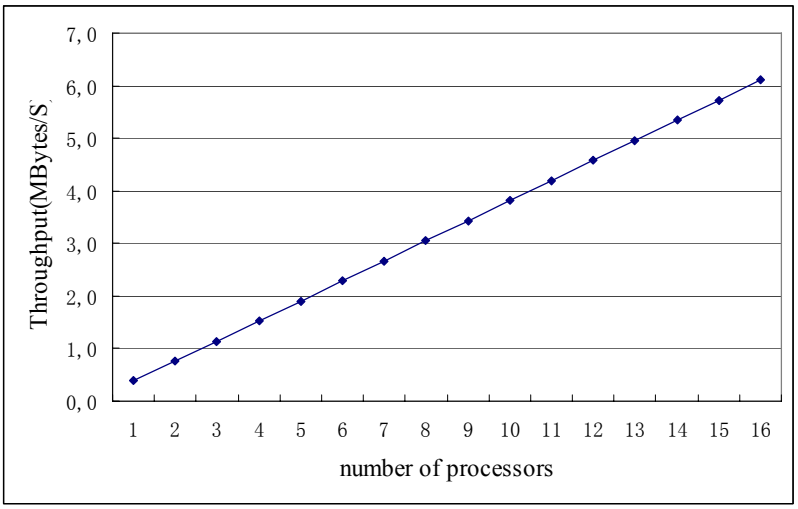


Figure 57: the throughput at 100MHz vs. the number of processors for the AES algorithm with barrel shifters

The speedup in function of number of processors is illustrated in Figure 57 expressed by the throughput of the system

The same outlook is correct for the TDES algorithm with synchronization at the beginning and at the end of the process and no communication during it

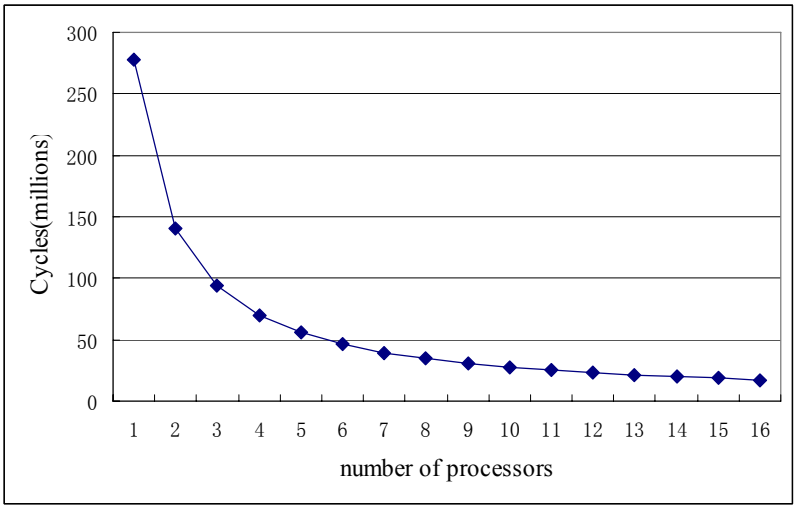


Figure 58: the number of cycles vs. the number of processors for TDES

The Figure 58 shows the total number of cycles to encrypt 16784 blocks in the TDES algorithms for different numbers of used processors with barrel shifters.

TABLE 19 shows detailed results for TDES encryption total number of cycles, throughput and speedup for different number of processor with barrel shifters

Number of processors	Cycles	throughput (Kbytes/s)	Speedup
1	277,854,154	391	1.00
2	140,979,916	783	2.00
3	94,182,002	1,173	3.00
4	69,522,290	1,564	4.00
5	55,621,353	1,954	5.00
6	46,351,593	2,345	5.99
7	39,732,401	2,736	6.99
8	34,761,884	3,127	7.99
9	30,901,447	3,518	8.99
10	27,811,587	3,907	9.99
11	25,284,793	4,299	10.99
12	23,180,586	4,689	11.99
13	21,391,118	5,081	12.99
14	19,866,930	5,471	13.98
15	18,541,469	5,862	14.98
16	17,381,660	6,253	15.98

TABLE 19 TDES executed cycles to encrypt 16784 blocks on PEs with barrel shifters

While Figure 59 shows throughput at 100 MHz of the TDES for different number of processors with barrel shifters.

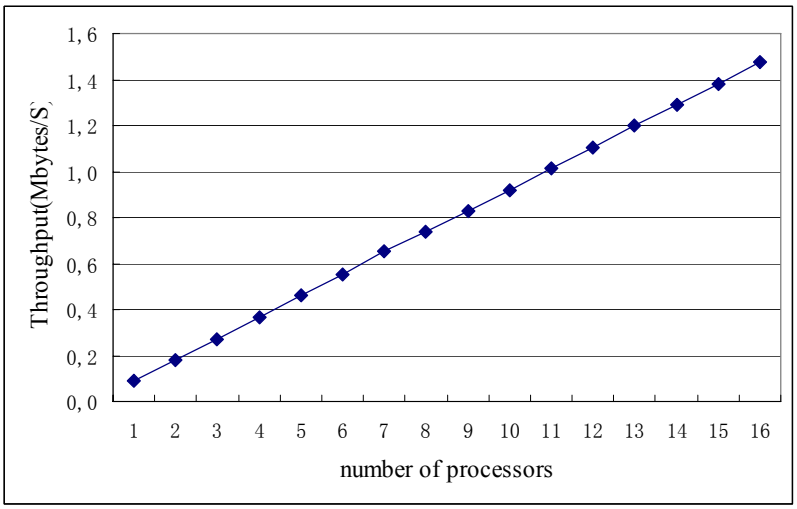


Figure 59: the throughput at 100MHz vs. the number of processors with barrel shifters for the TDES algorithm

Results shows that for the data parallel mode a ECB mode is 1.3% faster than the CBC mode and they show similar characteristics. TABLE 20 shows the convergence of the number of cycles on average to encrypt or decrypt one block, the average number decreases as the overall number of treated blocks.

Number of	pipelined		Data Parallel	
Blocks	total cycles	cycles/block	total cycles	cycles/block
1	13,891	13,891	8,487	8,487
2	14,530	7,265	8,611	4,306
3	15,361	5,120	8,598	2,866
4	16,461	4,115	8,669	2,167
5	17,319	3,464	8,688	1,738
6	18,288	3,048	8,917	1,486
7	18,927	2,704	8,967	1,281
8	20,032	2,504	9,011	1,126
9	20,952	2,328	9,048	1,005
20	30,526	1,526	18,311	916
100	98,944	989	58,402	584
1,000	869,078	869	522,633	523
10,000	8,573,123	857	5,177,715	518

33,568	28,740,720	856	17,381,660	518

TABLE 20 in of overall and average number of cycles in function of number of treated blocks

The pipelined method converges to an average of 856 while the data parallel converges towards 518 showing better performance by 40%. The pipelined method shows an average overhead of 56 cycles for synchronization (average 856 – 800 for bottleneck stage) this is equivalent of one read and one write average time for synchronization memory.

The speedup achieved on average is shown in TABLE 21 while TABLE 22 shows average throughput at 100 MHz clock.

	Pipelined	Data Parallel
speedup	9.67	15.98

TABLE 21 average speedup of the TDES

	Pipelined	Data Parallel
Kbytes/s	913	1508

TABLE 22 average throughput at 100 MHz clock

The speedup depends on the number of treated blocks, Figure 60 shows average cycles per block reflecting the speedup and achievable throughput for small number of encrypted blocks

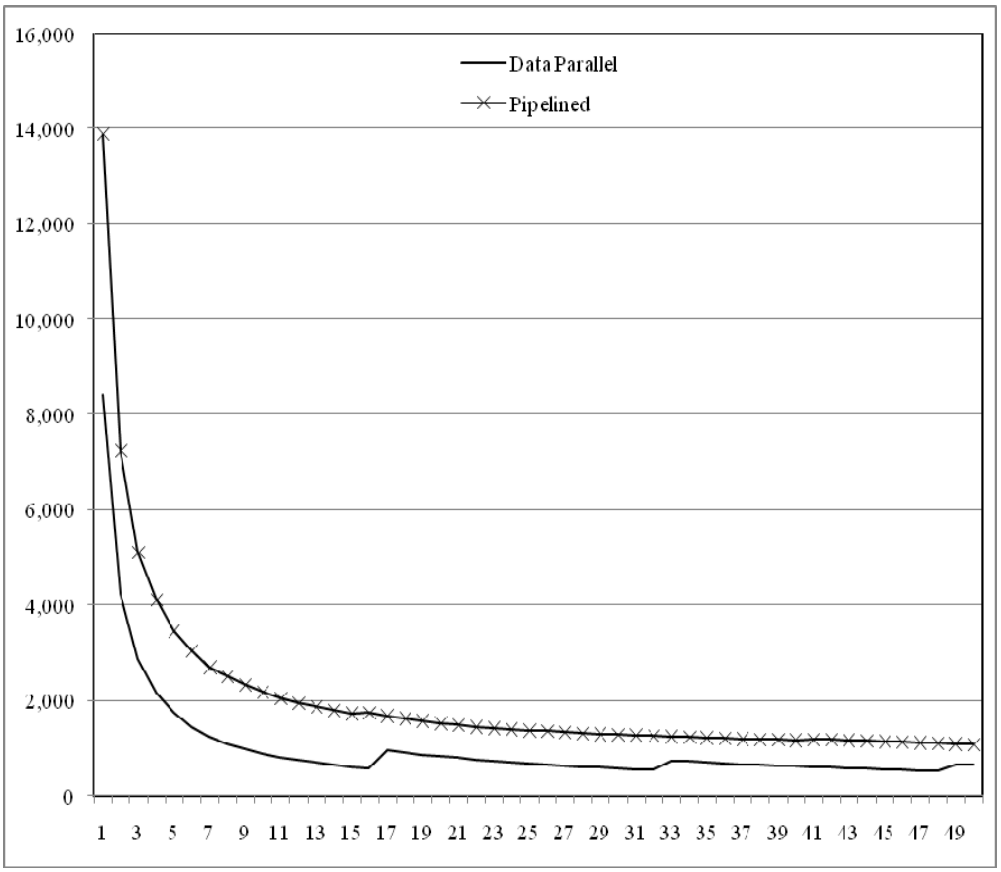


Figure 60: Average cycles/block in function of number of encrypted blocks

In this paper we explored two implementations of the TDES on a 16 processors MPSoC using two methods; the first is by dividing data between processors and the second is by dividing the execution of the TDES, the architecture was implemented on Virtex4 based board. Our implemented architecture connected processing elements via two NoCs, such architecture basically suites heavy memory access and allow scalability in the system construction.

Results shows better performance for data parallel method favored by the architecture, this result can be reversed in a pipelined architecture. Another conclusion shows that data parallel method can be applied with a greater number of processors, with different operation modes including CBC, because it has a dominating computation time than memory access time allowing a very effective linear speedup, we can predict as well that physically segmenting memory into banks helps pushing this parallelization to attain even a higher figure in processers number although adding more complexity to the connection between processors.

On the other hand our results shows that our architecture does not favor pipelined implementation because of lack of direct link between processor which heavily impacting performance because of the need to a synchronized memory access to exchange data. We note also that going to finer level of granularity may enhance performance.

4.5 Conclusion

we explored two implementations of the TDES on our 16 processors MPSoC using two methods; the first is by dividing data between processors and the second is by dividing the execution of the TDES. Our implemented architecture connected processing elements via two NoCs, such architecture basically suites heavy memory access and allow scalability in the system construction.

Results shows better performance for data parallel method favored by the architecture, this result can be reversed in a pipelined architecture. Another conclusion shows that data parallel method can be applied with a greater number of processors, with different operation modes including CBC, because it has a dominating computation time than memory access time allowing a very effective linear speedup, we can predict as well that physically segmenting memory into banks helps pushing this parallelization to attain even a higher figure in processors number although adding more complexity to the connection between processors.

On the other hand our results shows that our architecture does not favor pipelined implementation because of lack of direct link between processor which heavily impacting performance because of the need to a synchronized memory access to exchange data. We note also that going to finer level of granularity may enhance performance.

Chapter 5 MPSOC ASIC Design

5.1 MPSoC ASIC Design Introduction

Multiprocessor Systems-on-Chip (MPSoCs) have become the standard for implementing embedded systems [104]-[107]. The conventional interconnecting modes, such as buses and crossbars, cannot satisfy MPSOC's requirements of performance, area as well as scalability, reliability. On-Chip Network (OCN) or Network on Chip (NoC), has been proposed as a systematic approach to deal with the communication-centric design challenge [98][99]. Modular structure of NoC makes multiprocessor architecture highly scalable and improves reliability and operation frequency of on chip modules. Although these key advantages of NoCs have been widely discussed [98][99], the practical implementation of NoCs in very deep submicron technology (65 nm and below) is still an open challenge [114]..

Recent research has focused on efficient synthesis methods for NoC-based interconnects and comparisons with bus-based SoCs as well as design of application-specific NoC architectures and tool chains for known communication patterns . We build on previous work to address the problem of NoC topology design, accounting for technology and back-end tooling effects at the 65-nm and 45-nm nodes. Compared to the previous work, here we add several more studies and experiments on issues such as the impact of technology scaling on performances from 65-nm to

45-nm as well as migration from FPGA to ASIC. We use here a complete design flow integrated with standard industrial tool chains to perform accurate physical implementations of the NoCs.

We present experiment results of exploration on different NoC configurations and different submicron technologies with fully working FPGA and 65-nm/45-nm ASIC NoC designs.

5.2 standard cell ASIC design

Due to the growing complexity and increasing functionality of the embedded System on Chip, today's on chip system comprise of billions of gates. Standard cell based methodology makes it possible for designers to scale ASICs from comparatively simple single function ICs to complex multi-million or multi-billion gate system on chip. A standard cell library is a set of boolean logic functional blocks (e.g., AND, OR, XOR, XNOR, inverters) or storage functional blocks (flipflop or latch). Standard cell methodology is a method of ASIC design using standard cells developed previously by ASIC manufacturer. A standard cell has two parts of description: boolean logic function called logical view and physical layout view. The cell's function behavior in form of a truth table and the cell's electrical characteristics, such as propagation delay, capacitance, could be represented in different EDA tools. In modern ASIC design, the standard cell library contains the same logic function with different physical layout implementations in term of area, speed and power consumption. Different layout implementation could be used depending on the requirements of system, designers can find a implementation tradeoffs for their applications. In the STmicroelectronics standard cell library, the 65nm_HVT (High Threshold Voltage), 65nm_SVT (Standard Threshold Voltage) and 65nm_LVT (Low Threshold Voltage) represent different layout implementations for 65 nm technology, while the 45nm_LS (LowPower StandCell) and 45nm_HD (High Density).are two physical implementation in 45 nm technology.

5.3 ASIC 65nm and 45nm Semiconductor

An important design dependency is the specific technology library used. A single ‘‘technology library’’ no longer exists for standard cell design, especially at the 65-nm and 45-nm nodes. In fact, manufacturing technologies are spreading across a variety of libraries optimized for specific uses, such as low power or high performance, with several intermediate levels featuring, for example, different threshold voltage values. If very low-power libraries are selected by the designer, the size and speed of the buffers interleaved along wires become dramatically inferior, resulting in much tighter constraints on operation frequency or length. What we experienced with the 45-nm node is that timing performances are almost the same as 65-nm node, with less area and power!

We have used for our ASIC experiments, the three 65-nm libraries and two 45-nm libraries from STMicroelectronics, that is the 65nm_HVT (for High Threshold Voltage), the 65nm_SVT (for Standard Threshold Voltage), the 65nm_LVT (for Low Threshold Voltage), the 45nm_LS (for LowPower StandCell) and the 45nm_HD (for High Density). Here are their main characteristics:

CMOS 65nm (CMOS065-SOI) from STMicroelectronics

- Gate length : 65nm drawn poly length
- Dual or triple Vt MOS transistors
- Dual or triple gate oxide
- Dedicated process flavors for high performance or low power
- Dual-damascene copper for interconnect.
- Low-k (k = 2.9) dielectric.
- 6 or 7 metal layers dor interconnect.
- 0.20um metallization pitch.
- Analog / RF capabilities.
- Various power supplies supported : 2.5V, 1.8V, 1.2V, 1V
- Triple standard cell libraries (more than 800kgates/mm2).

- Embedded memory (Single port RAM / ROM / Double Port RAM

CMOS 45nm (CMOS045) from STMicroelectronics

- Gate length : 45nm drawn poly length
- Dual or triple Vt MOS transistors
- Dual or triple gate oxide
- Dedicated process flavors for high performance or low power
- Dual-damascene copper for interconnect.
- 7 metal layers for interconnect.
- 0.14um metallization pitch.
- Various power supplies supported : 1.8V, 1.1V, 0.9V
- Triple standard cell libraries (more than 1.6 Mgates/mm2).
- Embedded memory (Single port RAM / ROM / Double Port RAM
- Embedded memory (Single port RAM / ROM / Double Port RAM

5.4 ASIC Design Flow

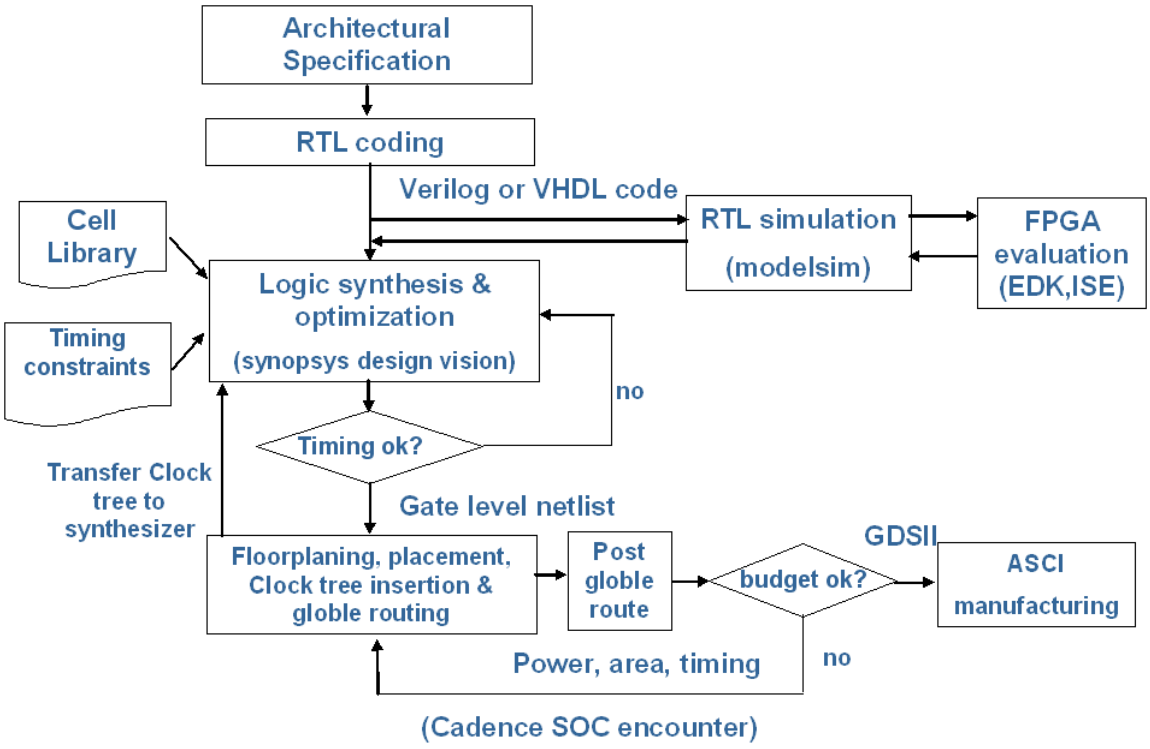


Figure 61 MPSoC FPGA emulation to ASIC implementation migration

The flow used and presented in Figure 61comprises several seamlessly integrated tools, which span core interconnection to physical layout. Scripts automate the front-end NoC design process. Next, the scripts automate the architecture generation phase, leveraging the library of NoC components and generating the RTL code for the desired topology. Finally, several industrial tools handle the back-end processes, logic synthesis, and physical design.

EDA Tools
Synopys Design Compiler
Cadence SOCencounter
Xilinx ISE/EDK

TABLE 23 EDA tools for physical design

5.5 MPSoC implementation case study

The block diagram of the overall multiprocessor architecture used in our experiments is illustrated in Figure 62. The multiprocessor system comprises 16 Processing Elements (PE), including 15 MicroBlaze based MB-PE and 1 PowerPC based PPC-PE. Each PE is a powerful computing system that can independently run its own program code and operating system. High throughputs Data-NoC connects 16 PEs to four DDR2 controllers which in turn connect to the respective off-chip 256MBytes DDR2 memory. One shared on-chip memory is attached to the synchronization NoC, which establishes a synchronization media for the PEs. Service network is also integrated in the system to provide functionalities like error management, IP status check, reconfiguration services and performance monitoring at run time.

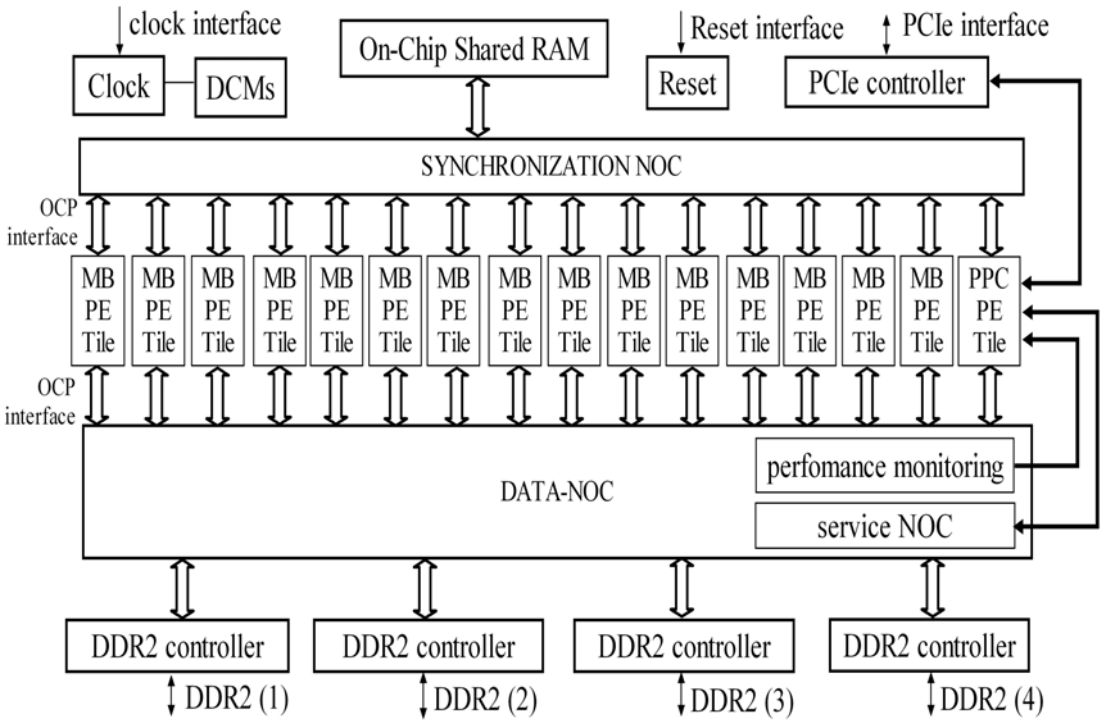


Figure 62 MPSoC implementation case study

This MPSoC have been implemented on FPGA and been introduced in Chapter 4. The floorplanning and placed FPGA is shown in Figure 43. In this chapter we will introduce the ASIC implementation of OCN of this MPSoC architecture.

5.5.1 ASIC implementation of OCN

Our OCN is comprised of a request portion and a response portion, and the request and response transactions are exchanged between Master NI and Slave NI. The OCN protocol, NTTP, is a three-layered approach comprising transaction, transport and physical layers. NTTP uses the packet-based wormhole scheduling technique.

As shown in Figure 63, the OCN is a cascading multistage interconnection network (MIN), which contains 8 switches for request as well as 8 switches for response. 16 OCP-to-NTTP NIs and 4 NTTP-to-OCP NIs are integrated at the boundary of OCN. four Master NIUs are placed as a group connected to one switch of the first stage, while the first stage are comprised of four 4IN*4OUT switches for 16

Master NIUs. Each switch in first stage distributes four output ports, which are respectively connected to the switches in second stage. The second stage contains four 4IN*1OUT switches. Each output port of switches in second stage is connected to a Slave NIU, which in turn connects to DDR2 memory controller

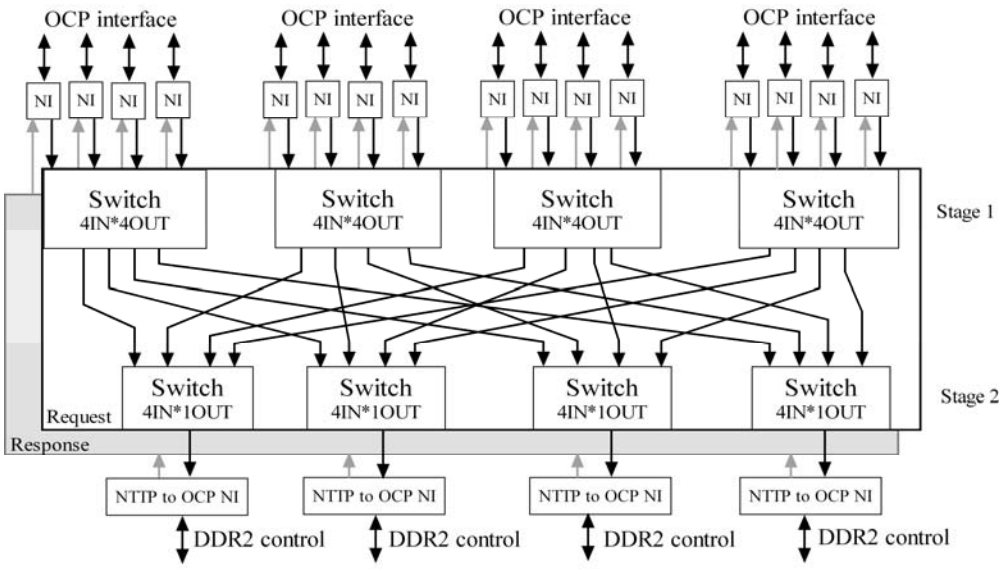


Figure 63 OCN case for ASIC implementation

5.5.2 Backend design flow of our OCN ASIC implementation

The front-end design and back-end design are two important and indispensable steps of ASIC design.

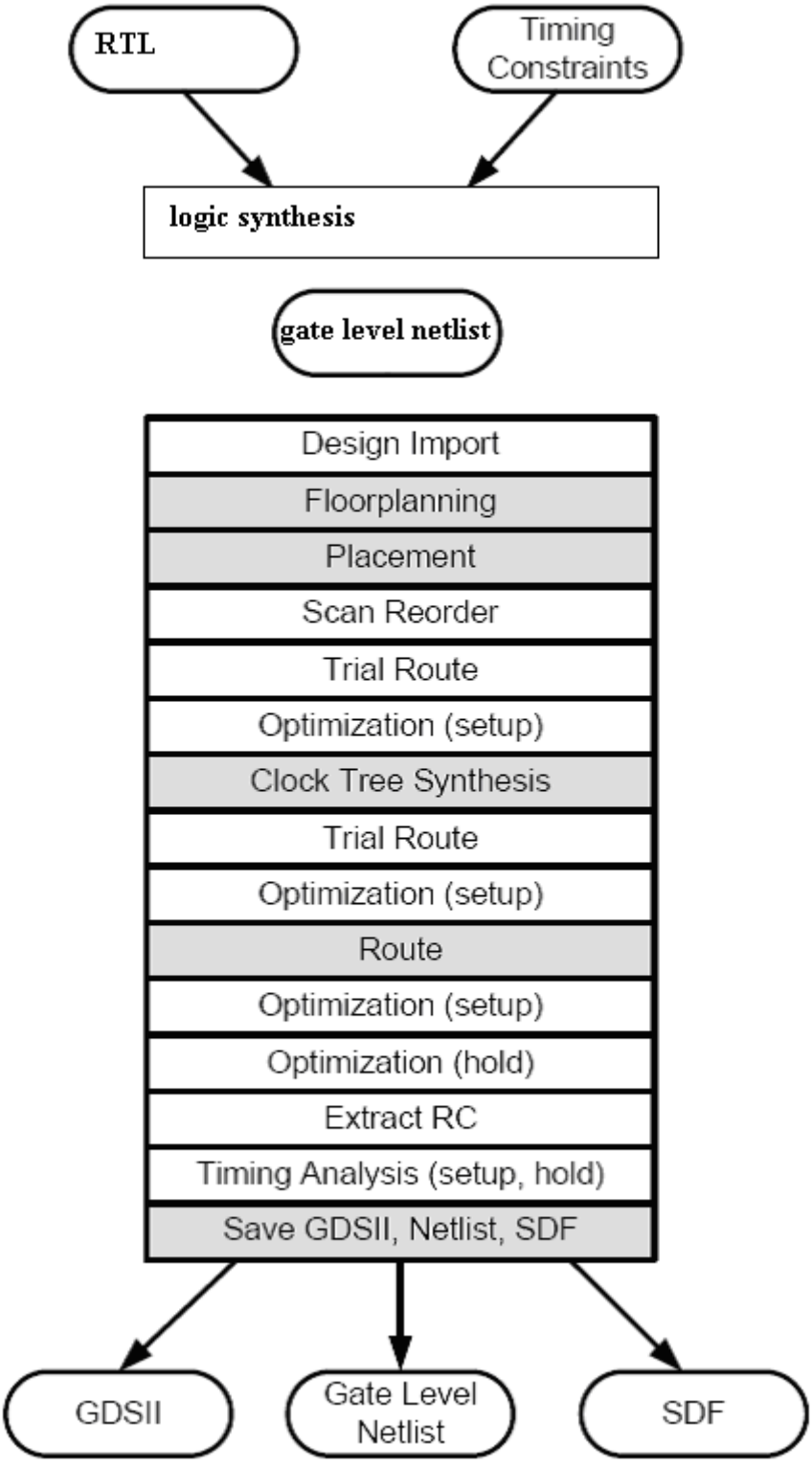


Figure 64 backend design flow for our NoC ASIC implementation

Using the technology library's cell logical view, on the front-end the Logic Synthesis tool performs the process of mathematically transforming the ASIC's register-transfer level (RTL) description into a technology-dependent gate level netlist.

In our case, the Synopsys' Design Compiler reads the RTL code of design, the timing constraints and using the cell library, synthesizes the code to structural level; thereby producing a mapped gate level netlist. The netlist is the standard cell representation of the ASIC design, at the logical view level. It consists of instances of the standard-cell library gates, and port-connectivity between gates. Proper synthesis techniques ensure mathematical equivalency between the synthesized netlist and original RTL description.

The synthesized gate-level netlist and the timing constraints are the input files for the back-end. The steps of back-end are listed as follows:

Design Import: It is the first step for back-end. It load the gate-level netlist and the timing constraints file (sdc) for import. It is used to setup the environment with the correct technology libraries, and to configure the EDA tool with the necessary information (buffers/inverters to be used, cells to be do not used, …)

- **Floorplanning**: On this step, the geometric definition of the circuit (width/height) must be done. It is used to define the limitations of to place the input/output/power pins of the circuit, to define the power and grand lines and stripes of the circuit, to define the location and orientation of the hard macros. And for the complex circuit, the blocking regions of some modules or all modules can be defined.

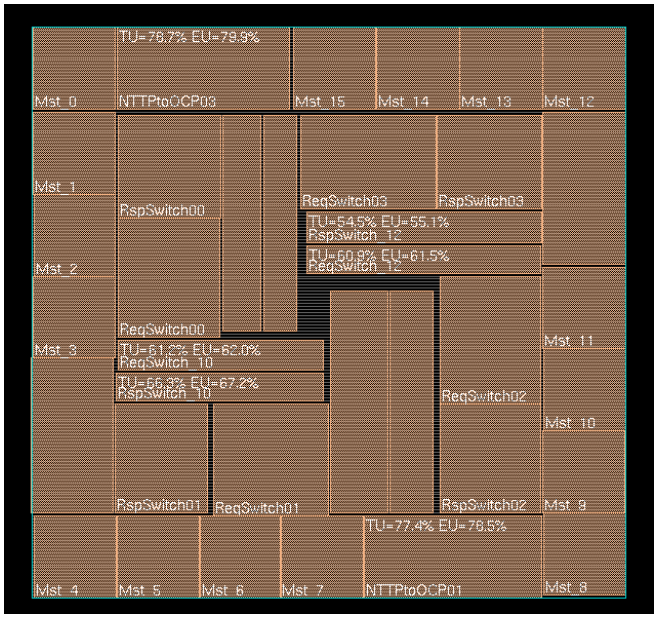


Figure 65 floorplaning of NoC

Placement: On this step, the tool places all the unplaced cells of the circuit and tries to optimize the placement for the timing constraints while respecting a targeted maximum density. The target density limits the placement density to avoid the wiring congestion of the circuit. Moreover, the tool tries to respect the regions defined on the floorplanning. Hence, all the cells of a module defined with a region are tried to be placed inside its region.

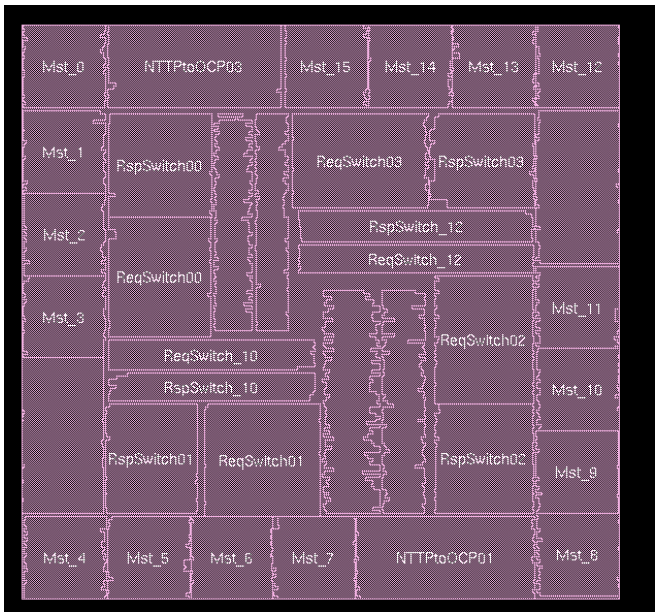


Figure 66 placement of NoC

- **Scan Reorder**: The scan chains are reordered to simplify the wiring length and so reduce the wiring congestion.

Trial Route: The circuit is routed with a simplified router to perform a first approximation of the routing complexity.

Optimization: The circuit is optimized on setup/hold time. Therefore, some cells are moved, others are resized, and long wires are buffered.

Clock Tree Synthesis: The clock signal distribution network (clock tree) is synthesized using a configuration script that defines the maximum insertion delay, the maximum skew, and other configuration parameters.

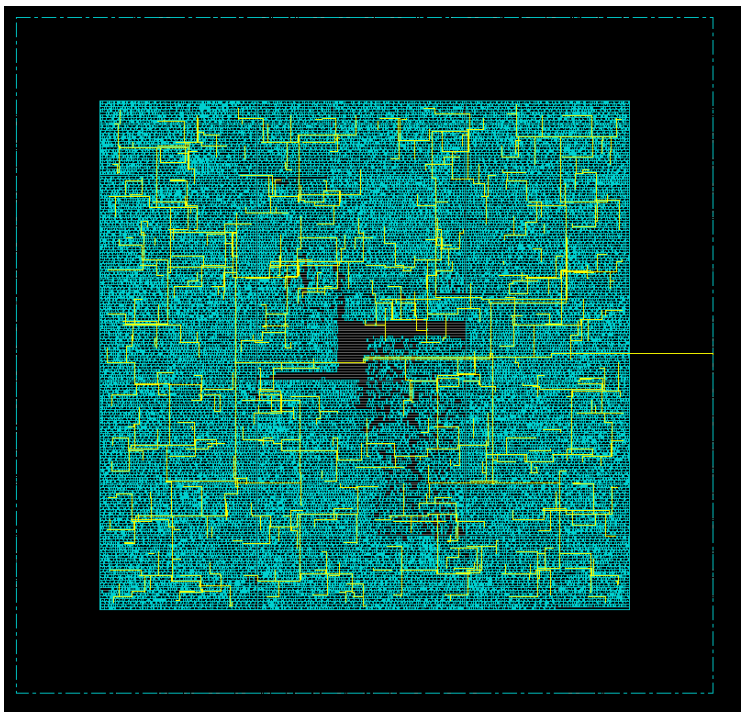


Figure 67 clock tree of NoC

- **Route**: The circuit is routed respecting the DRC rules and minimizing the signal integrity issues.

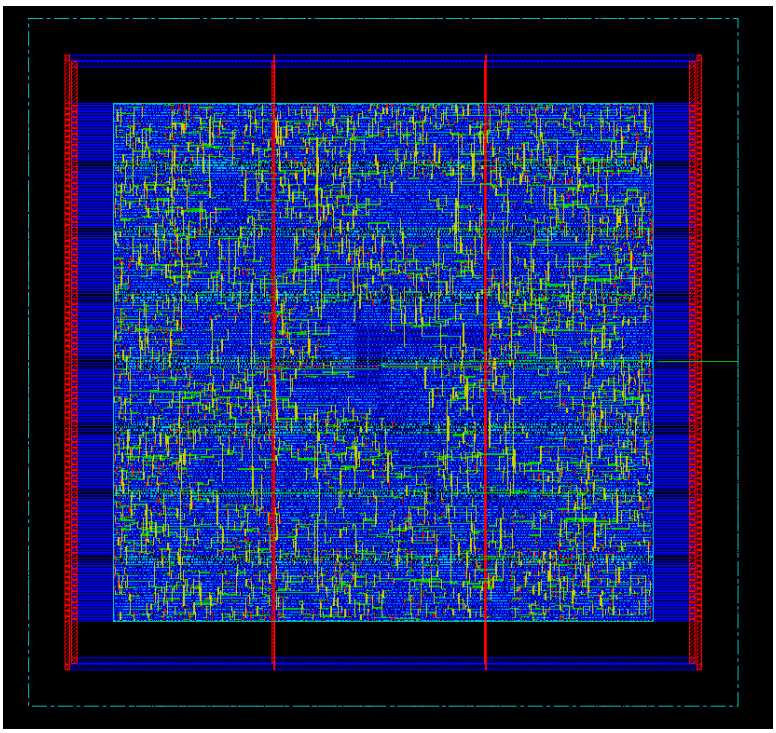


Figure 68 power and GND routing

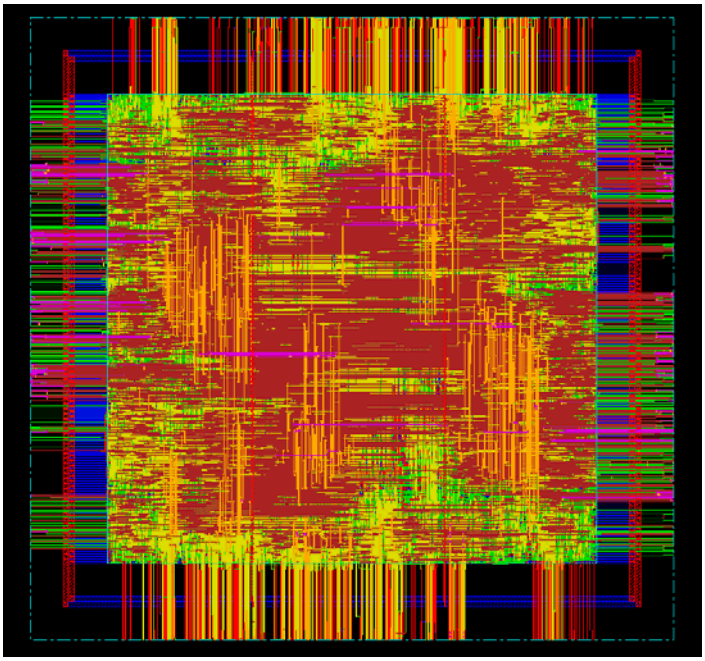


Figure 69 routed network

. **Extract RC**: The resistance and capacitance of the circuit are extracted.

Timing Analysis: The timing analysis is analyzed using the RC extracted data and the timing constraints. The analysis can be performed for the setup or hold time.

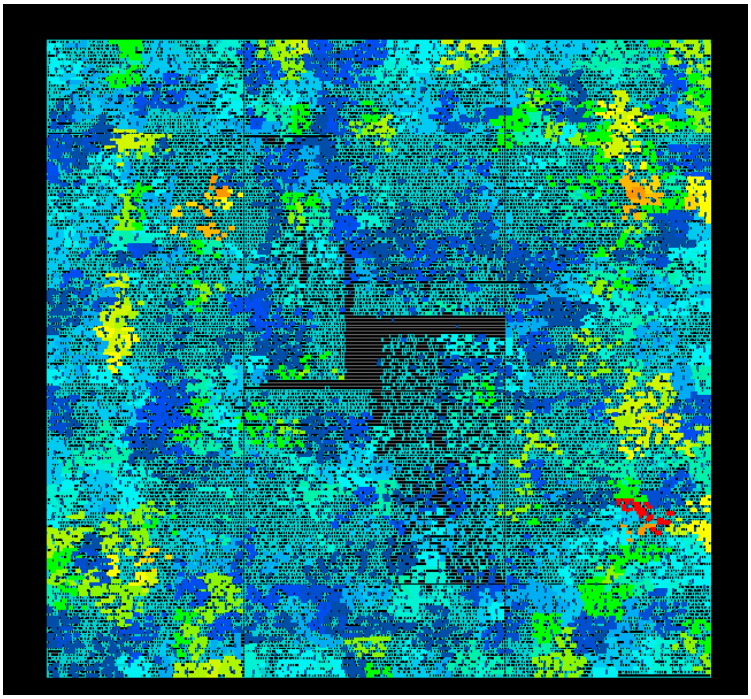


Figure 70 timing analysis

Save GDSII, Netlist, SDF: The output files are generated. The GDSII is the layout database for the mask generation. The gate-level netlist of the implemented circuit after Back-End optimization is saved. The SDF file is the timing data used for back annotation simulations.

The clock tree is the clock distribution network for a synchronous domain. In a SoC, thousands of synchronous flip-flops are clocked by the same clock signal. Hence, the clock signal has to be distributed and balanced to guarantee a maximum tolerable skew between any pair of flip-flops. The clock tree network is a collection of clock buffers and inverters interconnected in a tree manner. Moreover, all the elements on the clock tree network have to be properly balanced in terms of fan-in 32 and fan-out to respect the same rising and falling time. The input clock signal arrives to the root of the clock tree while the flip-flops are connected on the leaves. The clock tree network can be characterized in terms of insertion delay, maximum skew, and power

consumption. The insertion delay is the time it takes an event to propagate from the root to the leaves of the clock tree. It depends principally on the number of intermediate buffers/inverters between the root and the leaves and the area covered by the tree. The maximum skew is the maximum difference on time between any pairs of leaves of the clock tree. The lower the skew, the higher complexity of the clock tree and the higher the power consumed. Mesochronous clock tree distributions are well suited for low power consumption and low area. On the other hand, a fully synchronous clock tree networks can consume from 15% to over 45% of the total system power.

5.6 switch ASIC synthesis results vs placement and route results

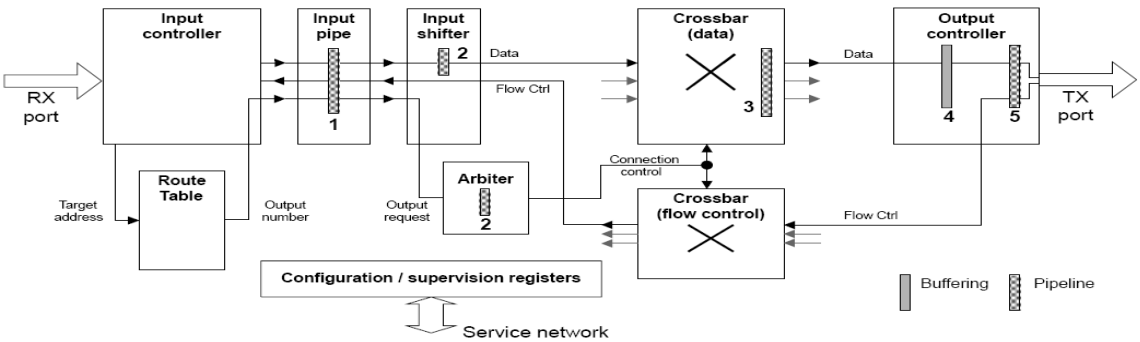


Figure 71 switch block diagram

The first case is a 5-input 3-output switch from Arteris' Danube library. As shown in Figure 71, switch includes several principal elements: input controller, route table, input pipe, input shifter, crossbar, arbiter and output controller. The full crossbar can transfer up to one data word per port and per cycle. The switch also support lock operation and pressure for enhanced arbitration decision making.

This 5IN 3OUT switch is synthesized in Synopys Design Compiler with the 90nm and 65nm technology of STmicronelectronics. The synthesized switch is illustrated in Synopys Design Compiler, as shown in Figure 72.

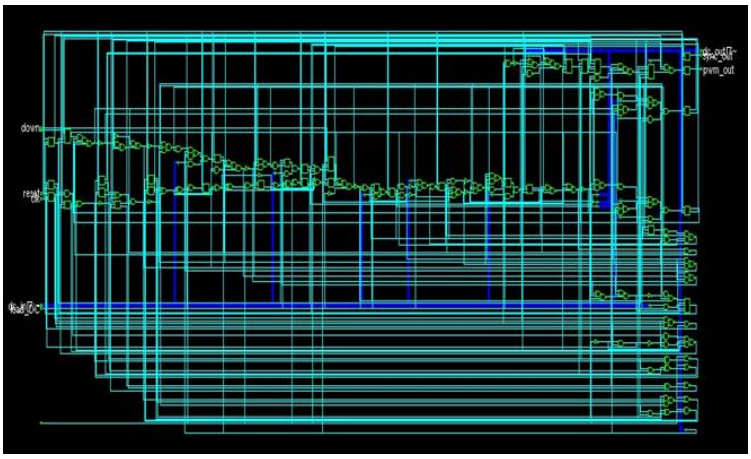


Figure 72 The synthesized switch in Synopys Design Compiler

As shown in Figure 64; in the backend design flow, there are two inputs, RTL design resource and timing constraints. In our implementation, we use the same switch HDL code and set different timing constraints. The different synthesis results of different timing constraints are shown in TABLE 24

90nm LVT library

Voltage and temperature condition: 0.9v 110c

Timing constraint (ns)	Area(nm2)	Total Dynamic Power(mW)	Cell Leakage Power(uW)
3	**25414.927734**	**6.1259**	**831.8013**
2.5	**26569.119141**	**7.9341**	**836.9886**
2.25	**27748.597656**	**8.3321**	**852.2310**

TABLE 24 synthesis results for different timing constraints

From this table we can see, when timing constraints is more strict, the switch will take more area and use more power, both in dynamic power and leakage power. With the different timing constraints, the same switch RTL was synthesized as different netlists. The different netlists were implemented with cadence SOCencounter, the placed and routed switch is shown in Figure 73. the place and routing results of three netlist from different timing constraints are shown in TABLE 25.

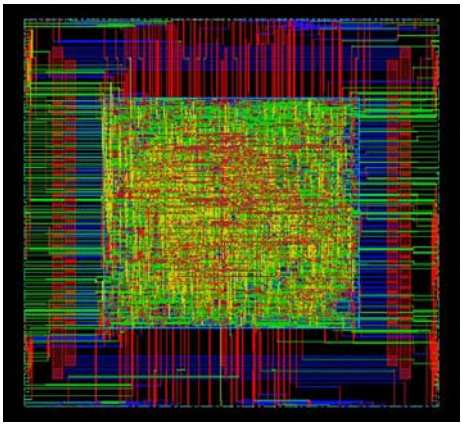


Figure 73 Switch place and routing

Timing constraint (ns)	Area(nm2)	Total-Power (Watts)	Leakage-Power (Watts)
3	27524.5152	7.289523 e-03	0.8928891e-03
2.5	28074.5112	9.282897 e-03	0.8995627e-03
2.25	30002.7660	9.848557	0.9295627e-03

TABLE 25 P&R results for different timing constraints

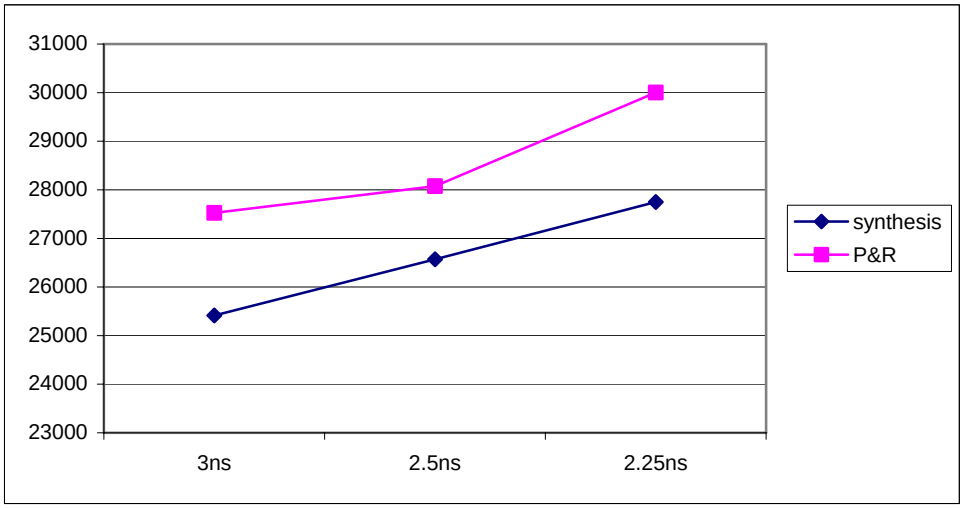


Figure 74 Switch AREA results: synthesis vs P&R

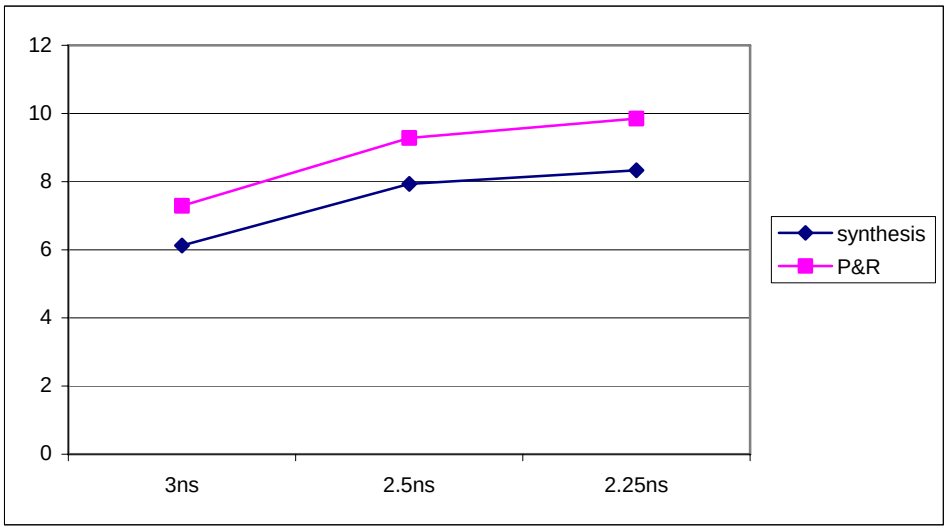


Figure 75 Switch Total Dynamic Power result: synthesis vs P&R

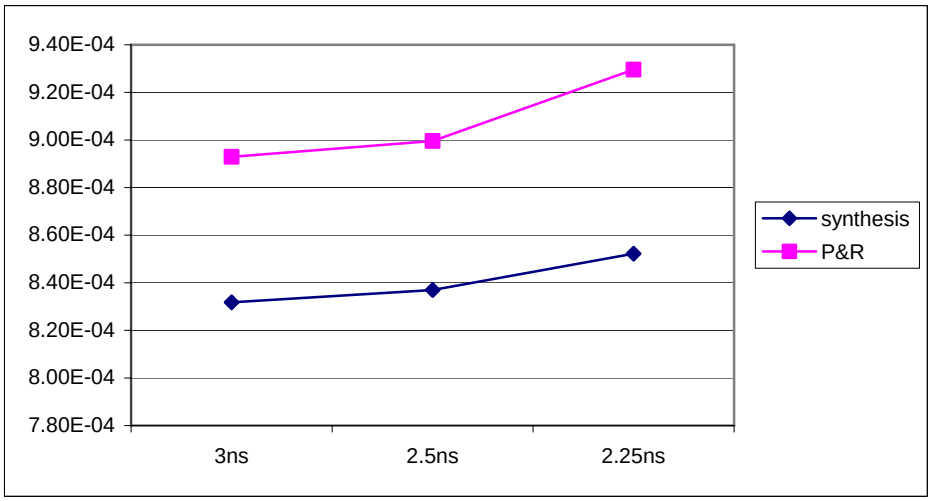


Figure 76 Switch Leakage Power result: synthesis vs P&R

From the three figures, we can see after placement and routing the results of area, dynamic power, leakage power are different from the results of synthesis. But the relationship for three different timing constraints is not change. As shown in Figure 74, Figure 75 and Figure 76, the results of synthesis and results of placement & routing show the same fact: when timing constraints is more strict, the switch will take more area and use more power, both in dynamic power and leakage power.

5.7 Design Space Exploration

In this design space exploration, we explored the technology: two technology, 65nm and 45nm. The libraries are also explored, the three 65-nm libraries and two 45-nm libraries from STMicroelectronics, that is the 65nm_HVT (for High Threshold Voltage), the 65nm_SVT (for Standard Threshold Voltage), the 65nm_LVT (for Low Threshold Voltage), the 45nm_LS (for LowPower StandCell) and the 45nm_HD (for High Density). the supply voltage and temperature of library is one input of our exploration.

The different experiments presented here are with physical synthesis and without place & route. But our place & route experiments showed that physical synthesis estimations were accurate for our design thus not mandatory for our explorations.

5.7.1 ASIC Power vs Clock Frequency

The first exploration presented Figure 78 and Figure 77 concerns the static and dynamic power for different supply voltages for the 65nm HVT node and according to the timing constraint provided, from 4 ns to 1.4 ns clock period. For one timing constraint, if the results showed that the slack is positive, that means actual timing constraint can not be arrived, the points in the timing constraint is not appeared.

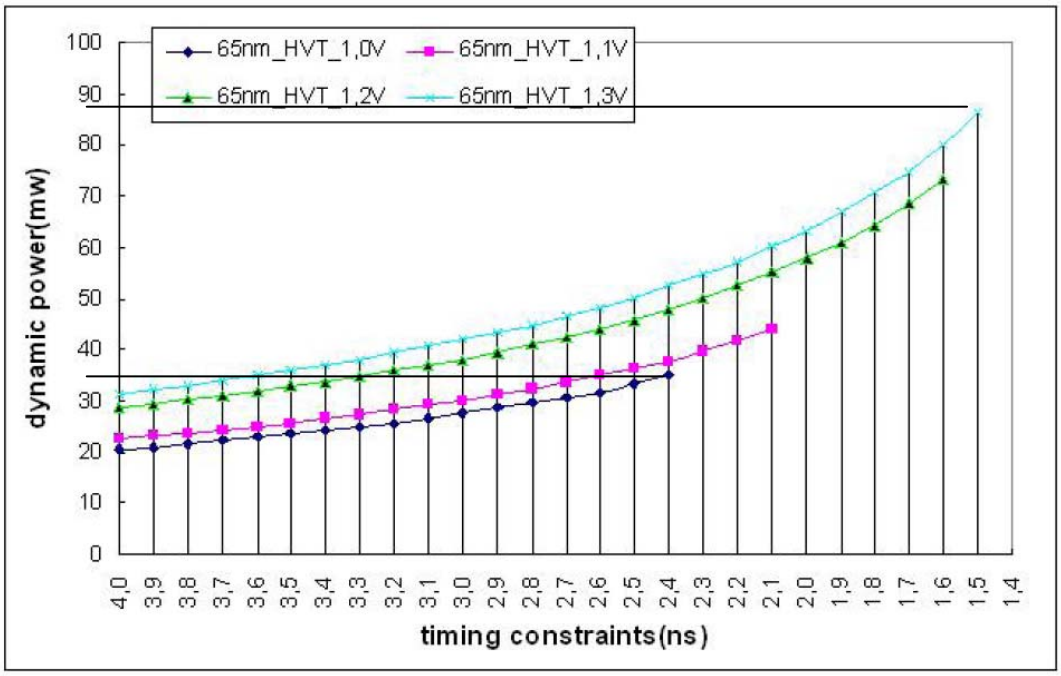


Figure 77 ASIC Dynamic power exploration

From the Figure 78, we can see when supply voltage is 1.0V, the minimum clock period is 2.4ns. the minimum clock period is 2.1ns for 1.1V supply. 1.6ns for 1.2V supply and 1.5ns for 1.3V supply. When supply voltage is higher, design can run at a higher frequency. But at the same time design will consume more dynamic power, from 1.1V to 1.3V, design increased 86% dynamic power and reduced 41% execution time.

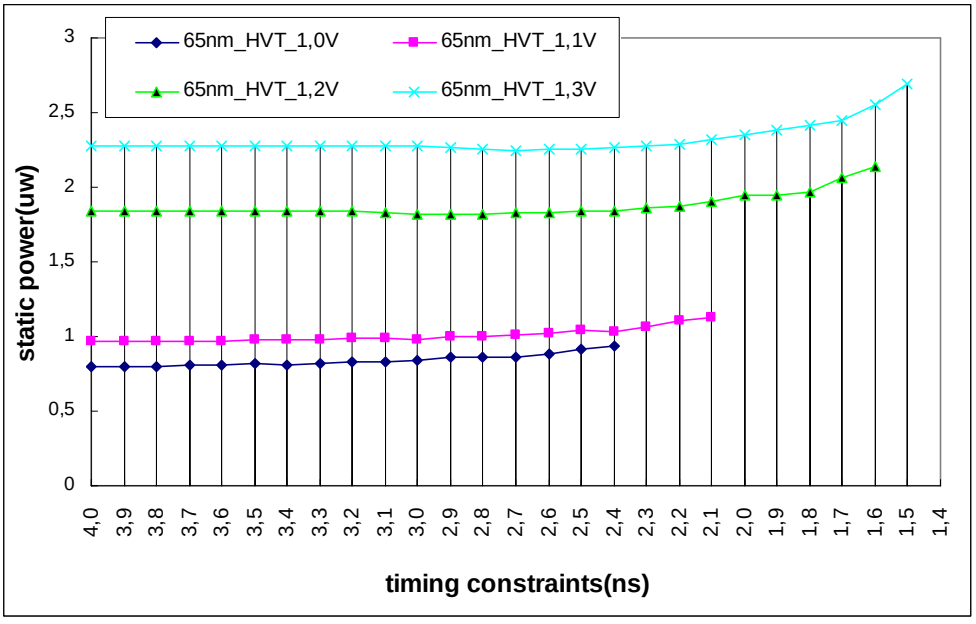


Figure 78 ASIC static power exploration

An important variability is obtained from 1.5 to 3 times for both static and dynamic power, and with a dynamic power much more important than the static power. Also the dynamic power is not a linear function of the timing performances.

The second exploration presented in Figure 79 and Figure 80 concerns the static and dynamic power comparison between 65-nm and 45-nm nodes according to the timing constraint provided, from 4 ns to 1.3 ns clock period.

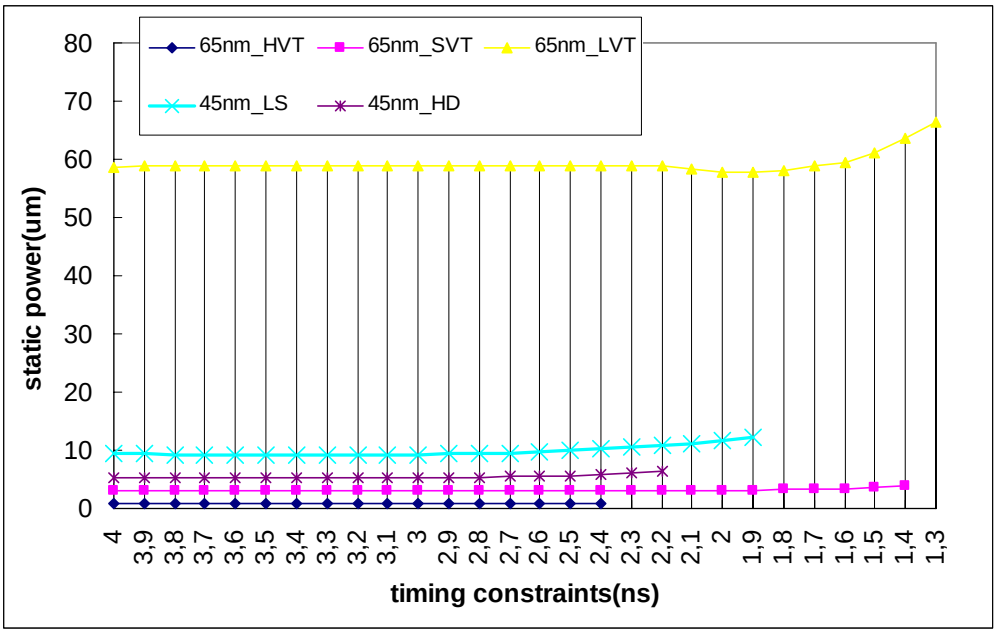


Figure 79 ASIC static power exploration: 65 nm vs 45 nm

Leakage power is often mentioned as a major problem in deep-submicron design. Our experiences with a 45-nm NoC switch implementations tend to contradict this assumption, as Figure 79 and Figure 80 shows. What we can see is that static power for the 45-nm is not very much higher than for the 65-nm, and the dynamic power for the 45-nm is still much more important than the static power.

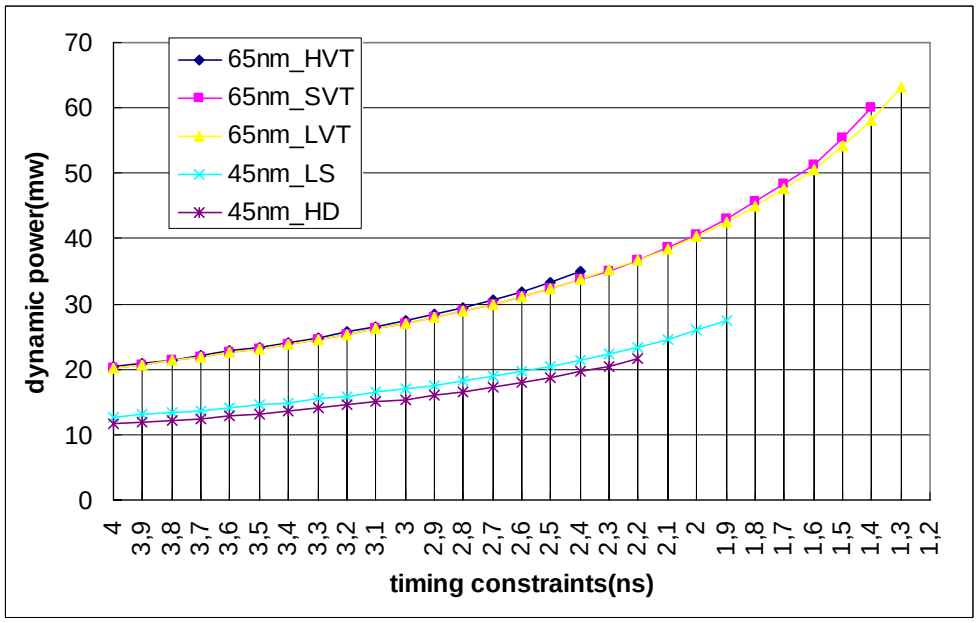


Figure 80 ASIC Dynamic power exploration: 65 nm vs 45 nm

5.7.2 ASIC Area vs Clock Frequency

The third exploration presented in Figure 81 concerns the area evolution for different supply voltages for the 65nm HVT node and according to the timing constraint provided, from 4 ns to 1.5 ns clock period.

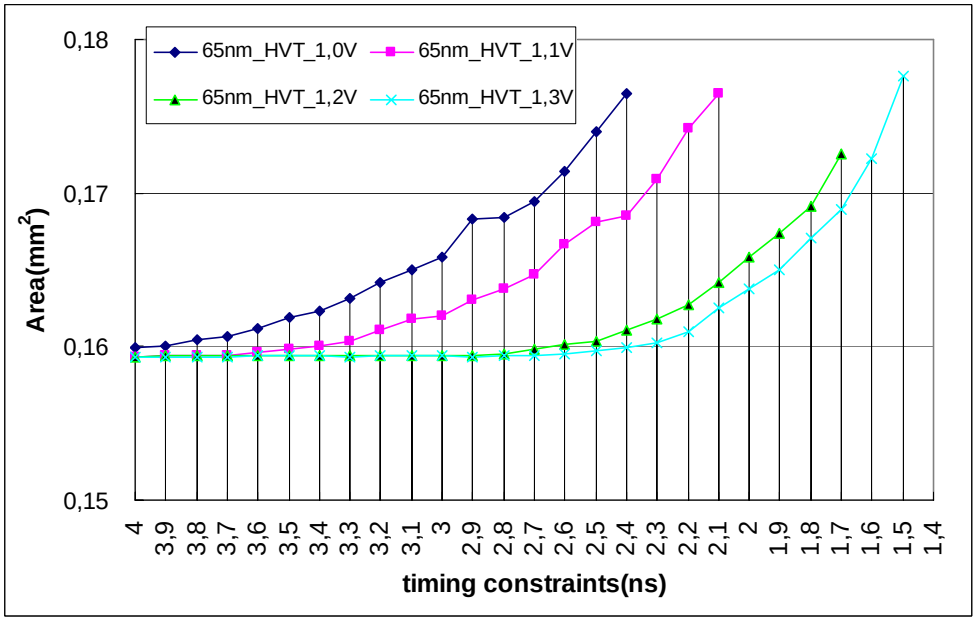


Figure 81 ASIC Area exploration

Here the area growing impact is not important (less than 13%) compared to the timing performance gain!

The foorth exploration presented in Figure 82 concerns the area comparison between 65-nm and 45-nm nodes according to the timing constraint provided, from 4 ns to 1.5 ns clock period.

We obtain here almost a 2 factor between the two nodes.

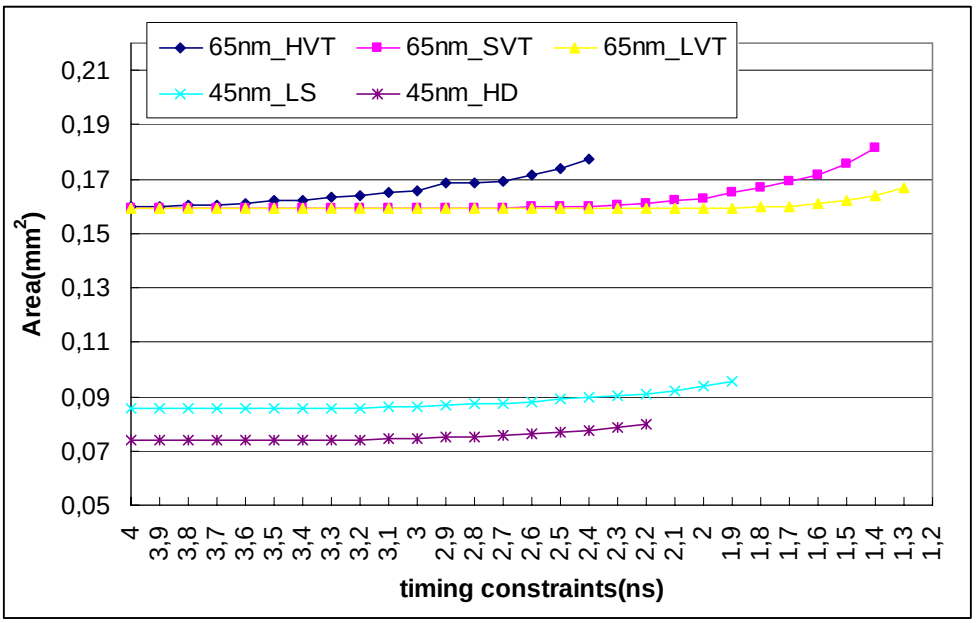


Figure 82 ASIC Area exploration: 65 nm vs 45 nm

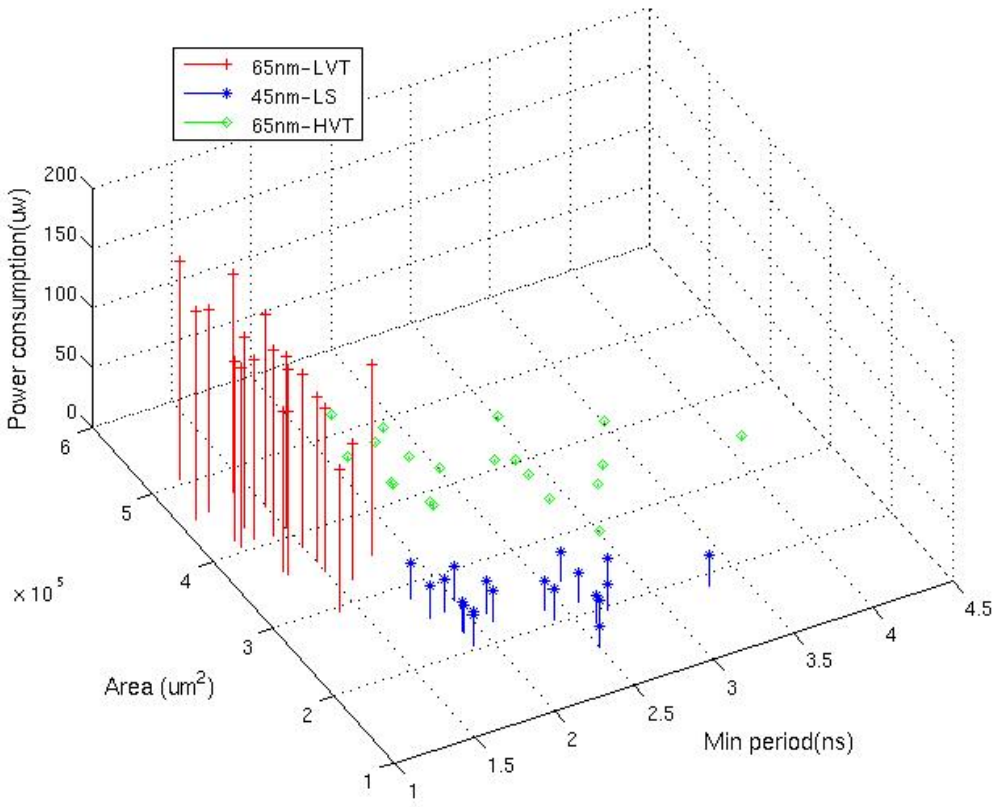


Figure 83 area, power consumption and min period comparison

The Figure 83 shows a 3D image comparing with area, power consumption and minimum period for three libraries: 65nm-LVT, 45nm-LS, 65nm-HVT. From the figure, we can clearly see with 45nm-LS, design takes about 50% AREA comparing with the two 65nm libraries. 65-LVT can achieve the minimum clock period, but the level of power consumption is large higher than the other two. The 65-HVT consumes less power and can run at the same level of frequency with 45nm-LS, but at the same axe of min period, the 65nm-HVT must take about 3 times of area.

5.7.3 FPGA-ASIC exploration

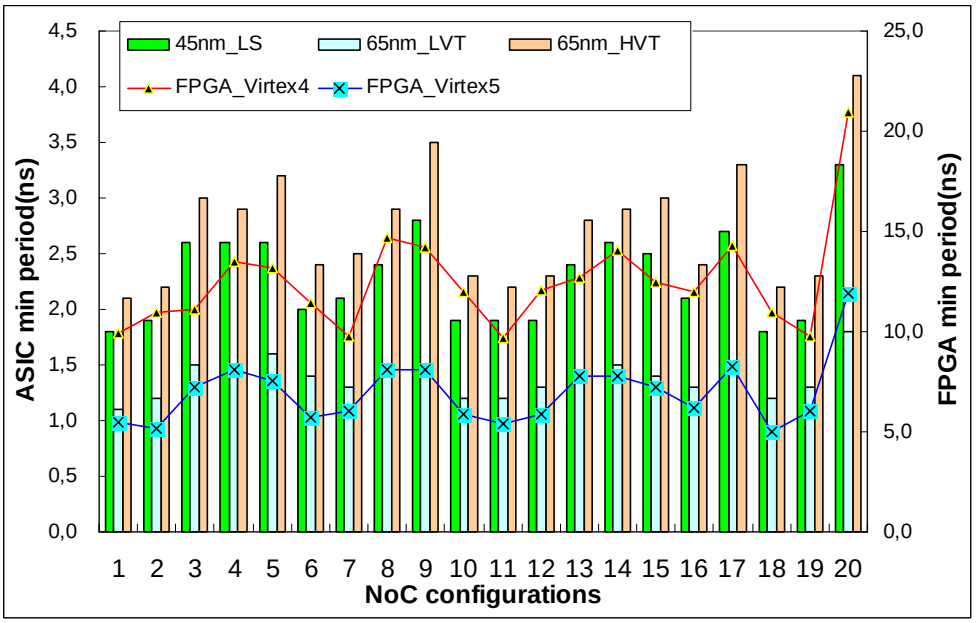


Figure 84 ASIC-FPGA Area exploration

We have conducted several explorations for different configurations of our NoC. We have considered 4 arbitration algorithms (Random, LRU, FIFO, Round-Robin) and different buffer (pipeline) mechanisms at the input and output of the switchs. We have selected 20 representative configurations. The impact of migrating from one technology to another is presented in Figure 84 and Figure 85 for area and frequency respectively. We compare here three ASIC technologies and two FPGA families. The main result from those two figures is that the curves have the same shape for both area and frequencies. When we consider a migration, the context of the target technology allows different architectural compromises and thus design configurations. So the new configuration selected to best fit the target technology context can be prototyped or emulated on the initial FPGA with a high confidence as the relative area/performances can be extrapolated from the results presented here. The main conclusion is that exploration can be conducted on FPGA with a high confidence before ASIC implementation. This is the same for a migration from 65-nm node to 45-nm node!

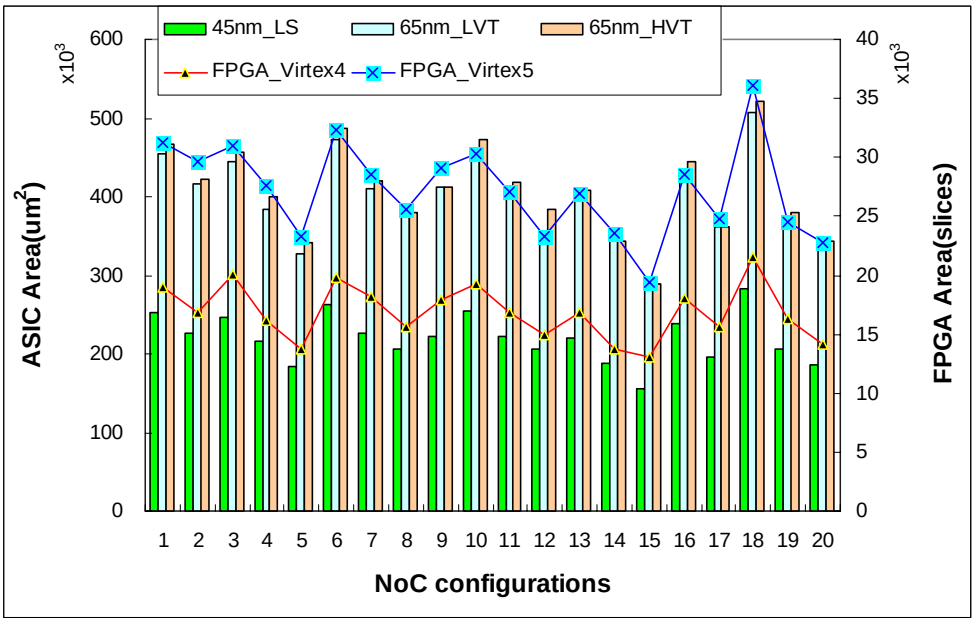


Figure 85 ASIC-FPGA Frequency exploration

5.8 NoC and switch opportunities with technologies change: impact on parallel software programming portability

Moving a design from FPGA to ASIC questions the gains and benefits which can be achieved both at an architectural level but also at the parallel programming level. Our results show although we can naturally expect an area gain the working frequency is not as significantly augmented. This suggests that performance improvement can not be achieved by technology alone and area advantage should be exploited by selecting network on chip components with more aggressive features. For example switch architecture and performance are modified through the use of different arbitration policies as well as the use of pipelining. The area performance tradeoff being more area beneficial, improvements should be extracted from these features. A

carefully tuned parallel application is the result of a good computation -communication balance. This communication performance comes from minimizing network on chip bottlenecks and packets conflicts with the most appropriate switch architecture under area constraints. Relaxing area constraints by moving from FPGA to 65nm or from 65nm to 45nm suggests to reconsider the network on chip architectural choices and in particular switch features. Porting a carefully tuned parallel application on a specific network on chip technology while changing both technology and network on chip components features obviously affects communication patterns and therefore performance. This paper advocates that a single design environment is needed which includes both parallel programming environment and FPGA/ASIC design with performance feedback to parallel programming. Also parallel programming for high performances embedded System on Chip needs tunable parallel code and libraries in order to explore different architecture configurations to obtain a fully optimized compromise at system level.

5.9 Conclusion

Multiprocessor on chip with network on chip are strongly emerging as prime candidates for complex embedded applications. Several commercial products are available for various application domains from wireless to consumer electronics. The end product in these cost sensitive markets are implemented using ASIC technologies. In a general ESL design methodology and for significant size designs the use of prototyping through FPGA is necessary for intensive validation and test as well as careful design space exploration. Moving a design from FPGA to ASIC questions the gains and benefits which can be achieved both at an architectural level but as well at the parallel programming. In this chapter we analyzed the migration of an implemented, validated and tested single FPGA chip multiprocessor with network on chip towards 65nm and 45nm ASIC technologies. Our results show although we can naturally expect an area gain the working frequency is not as significantly augmented.

This suggests that performance improvement can not be achieved by technology alone and area advantage should be exploited by selecting network on chip components with more aggressive features.

Chapter 6 Potential use of High level synthesis in MPSoC platform

6.1 Design Productivity and High Level synthesis

Embedded systems are increasingly complex to design, validate and implement [1]. System composition of embedded processors and hardware accelerators lead to the HW/SW co-design methodologies.

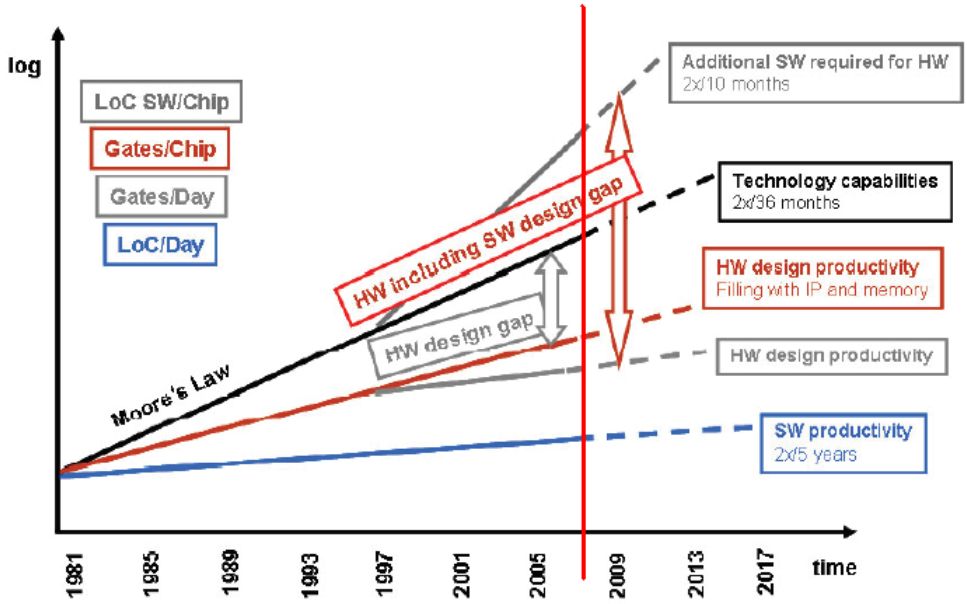


Figure 86 Design productivity challenges

Traditional co-design methodologies require that hardware and software are specified and designed independently. Hardware and software validations are thus done separately without interface validations. The partitioning is therefore decided in

advance and as changes to the partition can necessitate extensive redesign elsewhere in the system, this decision is adhered as much as possible. It can lead to sub-optimal designs as partitioning techniques often rely on the designer's experience. This lack of an unified hardware-software representation can lead to difficulties in verification of the entire system, and hence to incompatibilities across the hardware/software boundary. Using a single language for both simplifies the migration task and ensures an entire system verification. Important uses of a unified design language in addition to synthesis are validation and algorithm exploration (including an efficient partitioning). C-like languages are much more compelling for these tasks, and one in particular SystemC[42] is now widely used for system level design[43][44][45], as are many ad-hoc variants. Rapid C synthesis allows also fast emulation techniques of hardware and software used for architecture exploration. C-based synthesis fundamentals emerged from these HW/SW co-design methodologies with the objective of reducing the design productivity challenge [1]. However, although raising the system design level of abstraction contribute to reduce the design complexity reliable and predictable silicon implementation remains mandatory which support high level design handoff. The question is then on the impact of C-based behavioural synthesis on C-based specification and modelling framework for embedded systems.

We evaluate through a case study the performance and efficiency of several high-level description languages (SystemC, Handel-C) for FPGA-based embedded systems. The rest of the chapter is organized as follows: the 6.3 presents the related work while the 6.4 review main issues regarding C-based synthesis System design methodology. Design case studies along with performance evaluation studies and results are fully described in 6.5 and 6.6. At the end of Chapter, 6.8 presents the conclusions.

6.2 Embedded Processor Coprocessing Support

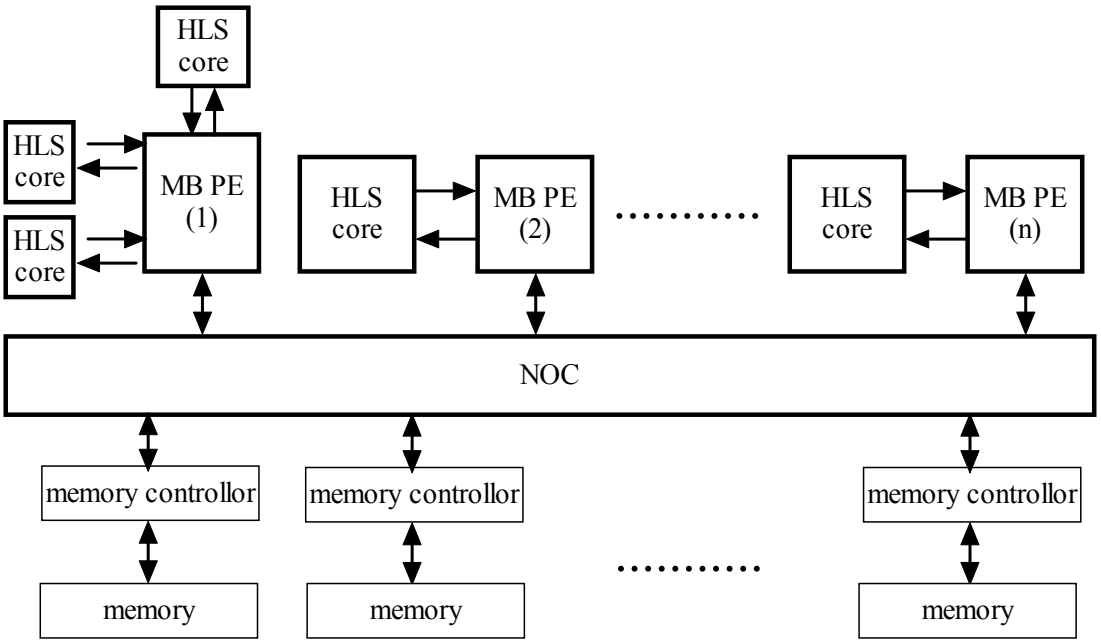


Figure 87 Flexible MPSOC platform with HLS Generated Coprocessors

As shown in Figure 87, the HLS core work as coprocessor in the multiprocessor system on chip platform based architecture.

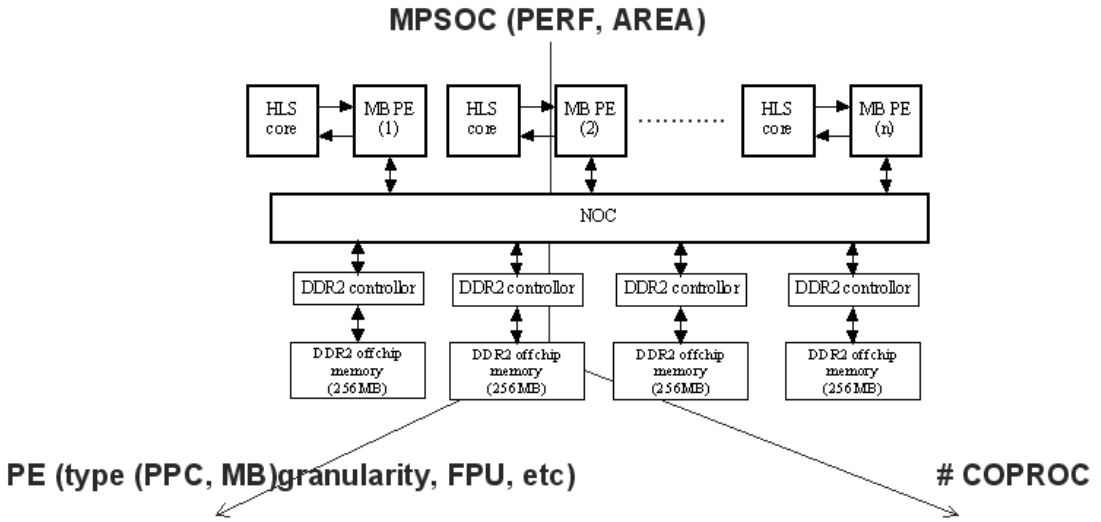


Figure 88 DSE coproc vs PE in MPSOC platform

The Figure 88 shows the possible design space exploration: changing the number of co-processors and the granularity of PE can obtain different performance, area and power consumption of system.

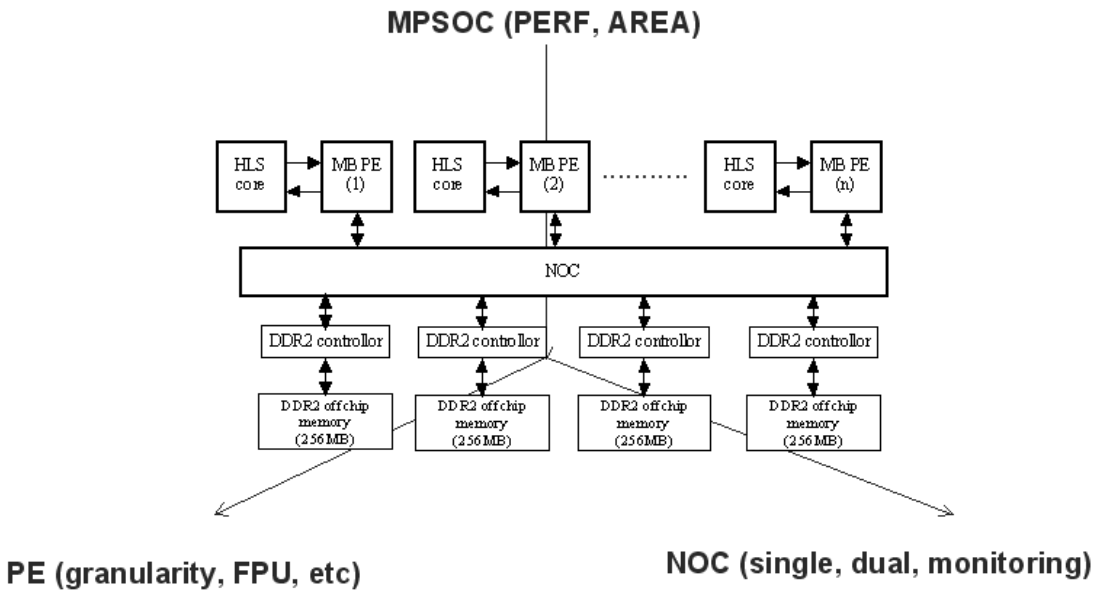


Figure 89 DSE PE vs NOCs in MPSOC platform

The Figure 89 shows the possible design space exploration: changing the number of on chip network and the granularity of PE can obtain different performance, area and power consumption of system.

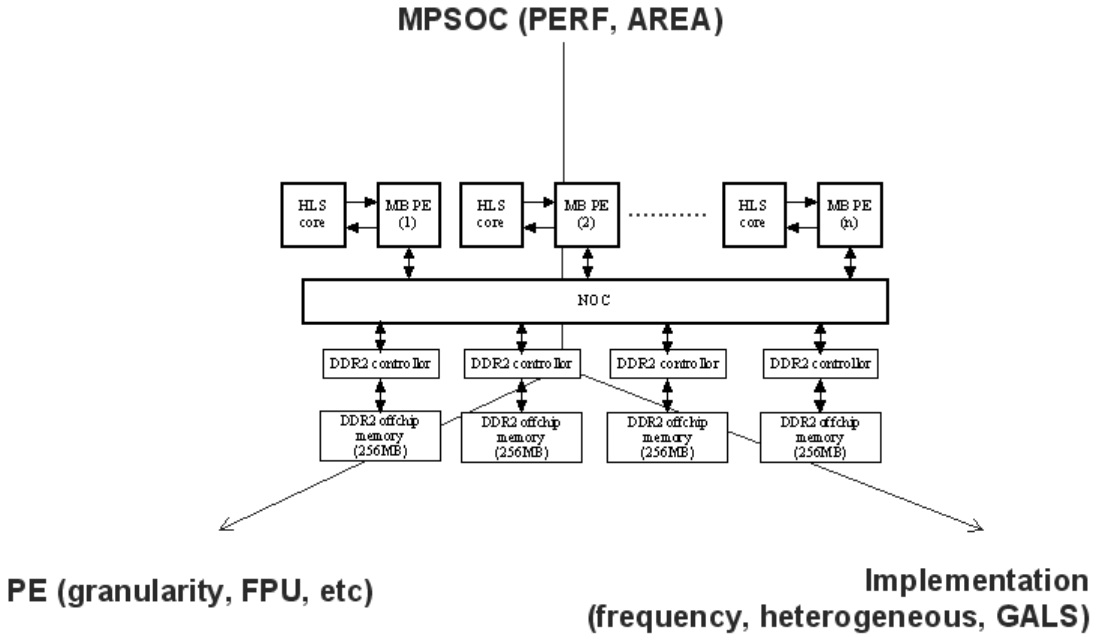


Figure 90 DSE PE vs Implementation in MPSOC platform

The Figure 90 shows the possible design space exploration: changing the frequencies of on chip modules and the granularity of PE can obtain different performance, area and power consumption of system.

Designing coprocessors without taking into account the implementation environment constraints ignore the mutual effects of design. We consider the coprocessing model where coprocessors are connected to a main embedded processor through bisynchronous FIFOs. A typical example is given in the Figure 87 where a Xilinx embedded processor Microblaze is connected to N coprocessors through bisynchronous FIFO: FSL (Fast Simplex Link).

The coprocessing design may be addressed from various aspects such as the concurrency model or the processor-coprocessors interface. However in an environment where those issues are fixed remain the specifications of the various clocks of the system and their mutual relationships. In Figure 91 the Microblaze is connected to clock clk1 while every other component has its own clock. Clock to clock variations affect the performance of the system and coprocessors clocks are the result of C-based synthesis.

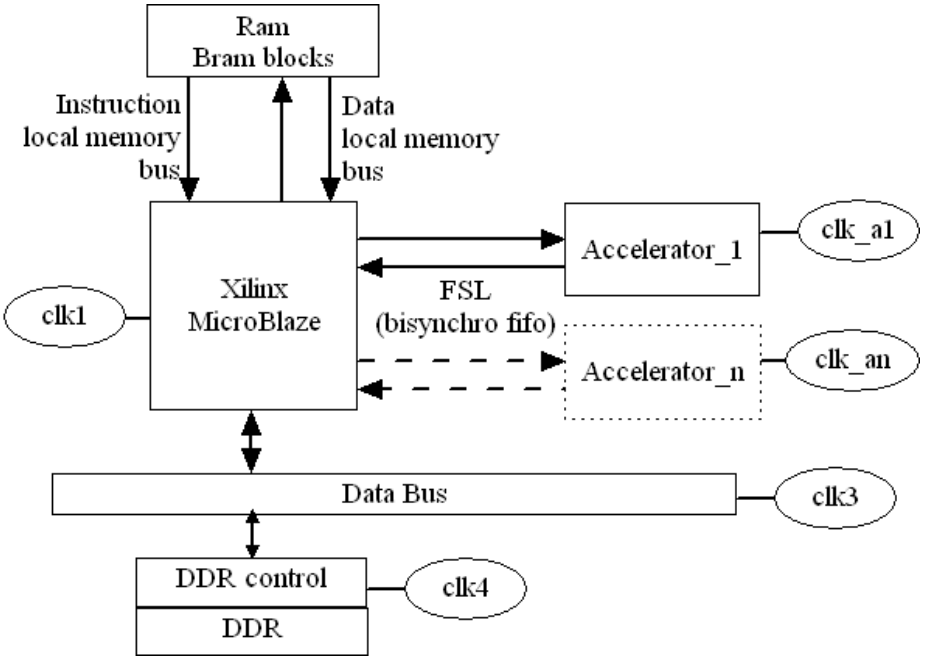


Figure 91 Block Diagram of Accelerator Connection Forms

6.2.1 C-Based Synthesis and Hardware Accelerator Design Workflow

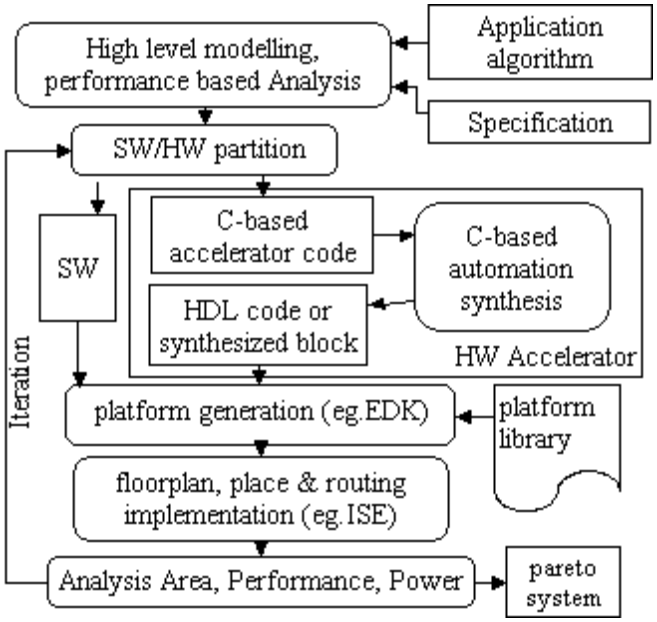


Figure 92 C-based HW Accelerated System Design Workflow

We propose a C-based HW accelerator design flow which evaluates C-based synthesized coprocessor in the final environment in order to take into account multiple clock effects. Feedback of low level implementation effects in this workflow is necessary to send to higher level. SW/HW partitioning should be repartitioned following the performance, area and power analysis in implementation level. Each HW/SW partition solution should be explored with different synthesis, placement and routing options, at the end of workflow, a system area, performance and power Pareto curve is provided. Designer can select the suitable system implementation solution from this Pareto line depending the application budgets and constraints. In this flow the quality impact of C-based synthesis tool is essential. Various C-based hardware description languages have been proposed over the past decade. Familiarity, using the same language in both SW and HW, possible full system verification, all these advantages of C-to-hardware method make more and more engineer adopt this new methodology for HW accelerated embedded system Design. But some C language characteristics which are different from traditional HDLs, are yet to be mastered when synthesizing hardware from C. Underlying concurrency model, data types, timing specification, memory and communication patterns, hints and constraints, all

characteristics must be considered when designing C-based system. We selected Three C-based synthesis tools , presented in TABLE 27

6.3 State of the art of c-based synthesis

Intensive research has been conducted on C-based behavioral synthesis. With the evolution of system level design languages, the interest in efficient hardware synthesis based on behavioral description of a hardware module has also been visible. For a system designer, the behavioral synthesis is very attractive for hardware modeling as it leads to significant productivity improvements. Important work has been done in academia on behavioral synthesis with C/SystemC [47]-[55]. For example, [47] have presented a synthesis environment and the corresponding synthesis methodology. It is based on traditional compiler generation techniques, which incorporate SystemC, VHDL and Verilog to transform existing algorithmic software models into hardware system implementations. In [53] authors address the problem of deriving exact Finite State Machines (FSMs) from SystemC descriptions which is the first part of behavioral synthesis methodologies. In an effort to extend synthesis to object oriented constructs of C++ language, [51] presents an approach to object-oriented hardware design and synthesis based on SystemC. Behavior synthesis problem is multi-objective in nature where synthesis tools allow the hardware designers to customize the behavioral synthesis process through various options which constitute a huge design space. In this situation, solution exists of a set of optimized results represented in the form of Pareto curves and Pareto surfaces. In [55], a methodology that allows the designers to generate and analyze the best synthesis results based on area, performance and power consumption estimation through an automatic exploration of synthesis results is presented. ROCCC [56] and Spark [52] are C-to-VHDL high-level synthesis academic frameworks.

Some tools for behavioral SystemC/C synthesis are available in the market. Synopsys SystemC compiler [57] was perhaps the first commercial effort to synthesize behavioral code written in ESL languages. Celoxica's Agility [58] and

Forte Design can synthesize a SystemC behavioral description of hardware modules. They also give the area estimation requirements for various ASIC and FPGA-based architectures. Orinoco Dale [60], ImpulseC [61] and CatapultC [63] estimate the area and the energy for a C description of an application.

C-based behavioral synthesis can be used at system level design in order to guide system partitioning. [44] presents a framework for the generation of embedded hardware/software from SystemC. In [45] area and energy are estimated for various IP components in the system before actually modeling the system in TLM. The area and energy estimation information is fed into the TLM model of the system where we partition the system by automatically exploring the system design space based on the given information. This design flow uses the TLM modeling for system design space exploration. It includes area and energy estimation information during the partitioning process. This work represents a good layout foundation for addressing the impact of implementation on embedded systems. To the best of our knowledge our study is the first attempt to the contribution of benchmarking these tools for performance comparison purposes and to analyze the interaction with place and route tools options.

6.4 SoC methodology of C-based synthesis

6.4.1 C-Language Fundamentals

Some C-language characteristics are troubling when synthesizing hardware from C. Edwards listed in [46] the key challenges of a successful hardware specification language: concurrency model, types, timing specification, memory and communication patterns, hints and constraints. The C-language has no support for parallelism. Either the synthesis tool is responsible for finding it or the designer must modify the algorithm by inserting explicit parallelism. C-hardware language designers adopted different parallelism strategies. Communication patterns depend on the parallel programming model provided by the C-hardware languages. These communication patterns do not exist in C-language. The C-language is also mute on the subject of time. It guarantees causality among most sequences of statements but

says nothing about the amount of time it takes to execute each. It is essential to find reasonable techniques for specifying hardware needs and mechanisms for specifying and achieving timing constraints. Data types are another major difference between hardware and software languages. The most fundamental type in hardware is a single bit travelling through a memory-less wire. Variable width integer are natural in hardware yet C does not support variable width. C's memory model is a large undifferentiated array of bytes, yet many small varied memories are most effective in hardware. All these characteristics must be considered when designing C-like hardware languages. All characteristics related to the considered tools are analyzed in the rest of the paper.

6.4.2 HLS Approaches and Tools

Various C-based hardware description languages have been proposed over the past decade. These tools use different approaches for timing, parallelism, data types and communication modeling. These approaches can be either automatic or manual.

Concurrency model: Coarse grain and fine grain parallelism are available for most of the approaches. The communication between tasks is the coarse grain parallelism. The fine grain parallelism is the operator parallelism and pipeline. This parallelism is either explicit or implicit. Constructs dedicated to the parallelism are added to the C-language for the explicit parallelism programming. Additional constructs are not required for the implicit parallelism. The level of parallelism can be handled with constraints and compile options.

Types: All approaches ensure hardware bit-true data type manipulation and declaration with additional data types. The size of the data can be precisely controlled for each operating stage.

Timing specification. All approaches take the latency and the throughput into account. The approaches are either explicit or implicit. Each clock cycle is precisely described for the explicit approach. The timing constraint only concerns the clock period. Tools with an implicit timing approach generate the scheduling and the parallelism of operators to meet the latency and the throughput timing constraints.

Memory: internal memories are either RAM or registers. They can be either implicitly selected or explicitly specified.

Communication patterns: predefined communication and memory protocols are used for the implicit approach. For the explicit approach the protocols are described in details in the code with dedicated instructions.

The chosen approach can lead to substantial code modifications.

Each tool use specific approaches for concurrency model, types, timing specification, memory and communication pattern. several tools cannot be studied in this paper but they can be classified in the following table (TABLE 26).

Features	Approaches	Language	Tools
Concurrency model	Implicit Explicit	C HandelC, SystemC	Spark, Impulse CoDeveloper™, CatapultC, DK Design Suite, Forte Cynthesizer, Agility
Types	Explicit	All languages	All tools
Timing specification	Implicit Explicit	C HandelC, SystemC	Spark, Impulse CoDeveloper™, CapatultC, DK Design Suite, Forte Cynthesizer, Agility
Memory	Implicit Explicit	C HandelC, C	Impulse CoDeveloper™ DK Design Suite
Communication patterns	Implicit Explicit	C HandelC	Spark DK Design Suite, Impulse CoDeveloper™, Agility

TABLE 26 Approaches used for C-based hardware description languages

Several tools can be evaluated with the presented approaches. Three selected tools given in TABLE 27 are selected on the basis of our own design experience and the used approaches. It is indeed of importance that a significant amount of design experience is needed in order to compare the design approaches. with the design. Each tool has different approaches for concurrency, data types, timing specifications, memory and communication.

Tool	Language	Company
Impulse CoDeveloper™	Impulse C™	Impulse Accelerated Technologies
DK suite	Handel-C	Celoxica
Agility SystemC Compiler	SystemC	Celoxica

TABLE 27 Case study selected C-based environments

Impulse C:

Impulse CoDeveloper™ is an ANSI C synthesizer [61] 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 coprocessing 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:** the 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 controled.
- **Memory**: all arrays are stored either in RAM or in a set of registers according to a compilation option.
- **Communication patterns:** streams (FIFO) with different formats are available as well as signals and shared memories interface.

DK Design Suite tool:

DK Design Suite is a complete Electronic System Level (ESL) environment supporting the Handel-C language [62]. It provides the user with a complete flow:

from specification to implementation such as architecture-optimized EDIF Netlist for FPGA's RTL Verilog or VHDL used for alternative synthesis flows and other hardware targets including ASIC designs.

Celoxica's Handel-C is a language for digital logic design that has many similarities to ANSI-C. Handel-C is a variant that extends the language with constructs for parallel statements and OCCAM –like rendez-vous communication.

Handel-C language and the IDE tool introduced by Celoxica, provides both simulation and synthesis capabilities.

- **Concurrency model**: the application is written with sequential programs in Handel-C. Programs written in Handel-C are implicitly sequential: writing one command after another indicates that those instructions should be executed in that exact order. Parallel constructs are possible with the par keyword to gain maximum benefit in performance from the target hardware .
- **Types:** this language adopts the manual customisation of numerous representations. Handel-C types are not limited to specific widths. Any defined Handel-C variable can be specified with the minimum width required to minimise hardware usage.
- **Timing specification:** DK Design Suite includes a cycle accurate multithread symbolic debugger. Handel-C timing model is uniquely simple: any assignments or delay takes one clock cycle.
- **Memory:** each data storage is explicitly specified in a RAM or a set of registers by the programmer.
- **Communication patterns:** any external specification with any type of communication protocol can be described. Handel-C allows you to target components such as memory, ports, buses and wires.

Agility Compiler:

The Agility compiler from Celoxica is described below [58]. The Agility Compiler provides a behavioral design and synthesis for SystemC. It is a single

solution for FPGA design and ASIC/SoC prototyping. Early SystemC models can be quickly realized in working silicon yielding accurate design metrics and RTL for Physical design.

- **Concurrency model:** Explicit multi-process is possible to manually manage parallelism. Each stage of the pipeline can be manually described with the use of wait() instructions in a RTL-like style.
- **Types:** ANSI C types and operators such as int and char can be used .Agility Compiler also accepts hardware types such as sc_uint<8>, sc_int<20> and fixed point sc_fix<>.
- **Timing specification:** the systemC timing model is uniquely simple: all assignments between two wait() take one clock cycle. This modelling style is similar to the VHDL "wait until rising_edge(clk)" style.
- **Memory:** SystemC provides supports for interfacing to on-chip RAMs and ROMs using dedicated array keywords. If not used, arrays are stored in a set of registers.
- **Communication patterns** : any external specification with any type of communication protocol can be described.

6.5 Exploration of C-based synthesis of coprocessor design

In order to evaluate the synthesis efficiency of the previously described tools the use of commonly accepted benchmarks for C-based synthesis would have been useful. However, so far no benchmarks have been released from the OSCI Synthesis Working Group which defined the synthesizable subset of SystemC nor by any other body. Therefore we decided to compose our own case studies which are basic and simple functions to ensure reproducibility.

6.5.1 Designs examples

Evaluation consists in studying the efficiency of the synthesis from C-based hardware descriptions. Common benchmarks are used for the evaluation of the previously described tools.

Our own case studies consists in a set of short and simple functions to allow reproducibility. The selected cases are two 3x3 image filters [64] , the FFT and an octree traversal algorithm (Ray Casting in Projective Geometry RCPG) [65].

A1. Linear Filter

A linear filter and a non linear filter are chosen. The two filtering benchmarks are based on a 3*3 window core processing. The linear filter is the mean filter . The mean filter is the simplest type of low-pass filter. The Mean or Average filter is used to soften an image by averaging surrounding pixel values in a 3x3 window. This filter is often used to smooth images prior to processing. It can be used to reduce pixel flicker due to overhead fluorescent lights.

A2. Median Filter

The second filter is the median filter based on the bubble sort of the 3*3 neighboring pixels . The median filter is a non-linear digital filter which is able to preserve sharp signal changes and is very effective in removing impulse noise. This filter is widely used in digital signal and image/video processing applications. For the median filter, pixels are first sorted based on intensity. The center pixel would be the middle value of the sorted list of pixels. We present an example of ImpulseC code for the mean filter in Figure 93.

```
void FIL_MEAN( unsigned char image[][n], unsigned
char vimage[][vn])
 {     short x,y;
 unsigned int sum;
       for (x=1; x<vn-1; x++)
          for (y=1; y<vn-1; y++)
          {
               sum    = vimage[x][y];
               sum += vimage[x-1][y];
               sum += vimage[x+1][y];
               sum += vimage[x-1][y-1];
               sum += vimage[x][y-1];
               sum += vimage[x+1][y-1];
               sum += vimage[x-1][y+1];
               sum += vimage[x][y+1];
               sum += vimage[x+1][y+1];
               image[x-1][y-1] = sum/9;
          }
```

(a) original C code

```
while (co_stream_read(r0,&p0,sizeof(int32)) == co_err_none) {
#pragma CO PIPELINE
#pragma CO set stageDelay 32
#pragma CO nonrecursive image0
#pragma CO nonrecursive image1
#pragma CO nonrecursive image2
co_stream_read(r1,&p1,sizeof(int32));
co_stream_read(r2,&p2,sizeof(int32));
for(k=2;k<6;k++) {
#pragma CO UNROLL
image0[k]=p0 & 255; p0=p0>>8; image1[k]=p1 & 255; p1=p1>>8;
image2[k]=p2 & 255;
p2=p2>>8; }
res=0;
for(k=1;k<5;k++) {
#pragma CO UNROLL
res=((image0[k-1]+image0[k]+image0[k+1]+image1[k-1]+image1[k]+image1[k
+1]+
image2[k-1]+image2[k]+image2[k+1])>>3)+(res<<8);
}

for(k=0;k<2;k++) {
#pragma CO UNROLL
temp=image0[k+4];
image0[k]=temp;
temp=image1[k+4];
image1[k]=temp;
temp=image2[k+4];
image2[k]=temp; }
co_stream_write(output_stream,&res,sizeof(int32));
```

(b) Impulse C code

Figure 93 the C and ImpulseC codes for a 3*3 mean filter

The filters are implemented by sliding a window of odd size (a 3*3 window) over an N*M image. At each window position the sampled values are sorted and the resulting value of the sample replaces the sample in the center of the window. Three 32-bit streaming inputs provide four pixels for each line for each clock cycle. The size of the internal storage is 6 * 3 pixels. Three internal storage solutions are implemented and evaluated. The first one is a sequential one with RAM as internal storage. The

second one is a parallel/pipeline solution with RAM as internal storage. Three separate RAMs are used to allow parallelism between the three inputs. The third solution is a parallel/pipeline solution that use registers as internal storage. Figure 94- Structure of external signals for both filters. Three streaming inputs are used to provide 4 pixels. Each input corresponds to one line of the image.

A3. FFT

The third benchmark is the radix-4 FFT on 256 complex values (16-bit).

A4. Ray Casting in Projective Geometry (RCPG)

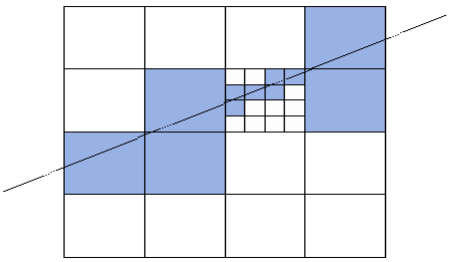


Figure 94 2D recursive octree grid traversal principle.

The Ray Casting in Projective Geometry (RCPG) is an iterative octree traversal algorithm (Figure 94). In a regular grid, from a current cell crossed by the ray, the next cell is defined by a minimization of a cost function which is iteratively updated. At each step, the ray is propagated in 3D along the directions x, y and z. The direction depends on the face where the ray and the current cell intersect (Figure 94). The parameters of the intersection between the ray and each face are progressively stored and updated. The data structure is an octree structure: a cell can contain a data pointer to a higher resolution grid. Thus at each cell, the algorithm continues to the next one or descents in the sub-grid. When it reaches a sub-grid border it mounts to the upper level of the grid.

For this design case, the sequential and pipeline algorithms are described.

The chosen cases are described in the TABLE 28.

Benchmark	Description	No IP
Linear digital filter (mean)	pipeline using registers	IP1
	pipeline using memory	IP2
	sequential	IP3
Non linear digital filter (median)	pipeline using registers	IP4
	pipeline using memory	IP5
	sequential	IP6
FFT	sequential	IP7
	pipeline	IP8
RCPG	sequential	IP9
	pipeline	IP10

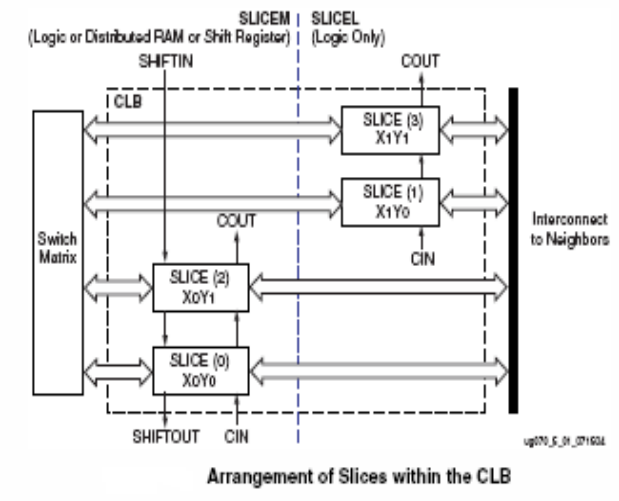
TABLE 28 Core Case studies

6.5.2 Target Platform

The previously described case studies are intended to be synthesized, placed and routed on a target technology in order to evaluate how the selected C-based design environment outputs as inputs to a same synthesis, place and route tool will be processed. The metrics to be considered will be performance and area.

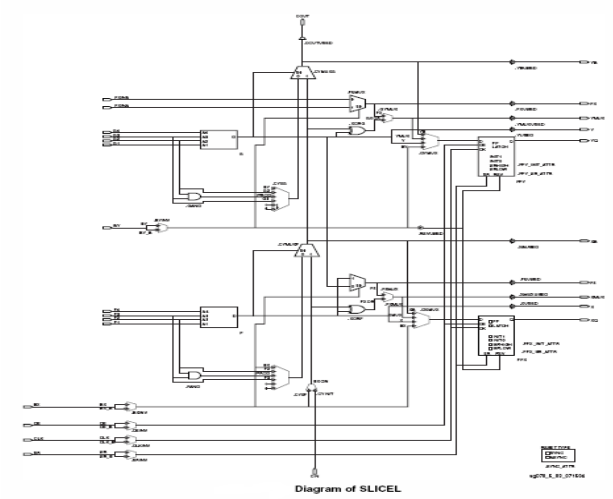
6.5.2.1 Target technology

We selected the FPGA Xilinx Virtex-4 technology [75] as the target technology in this case study. The main reason for the choice of an FPGA technology was to allow a quick implementation and verification of all the IPs through actual execution on chip. The Virtex-4 technology based on a 90nm Copper CMOS process has a fixed number of hardcores resources such as DSP, embedded RAM and fixed interconnections. The Xilinx Virtex-4 can be described as a matrix of CLB each of them being composed of several slices (Figure 95). The Xilinx Virtex-4 proposes memory-oriented slices **SLICEM** and logic-oriented slices **SLICEL** (Figure 96). Also the embedded memory BRAM is a dual port 18kb memory array. Hard cores in the FX family includes embedded processors – IBM PowerPC – and 18x18 two's complement signed multipliers (DSP blocs).



Source: Xilinx Virtex-4 User GuideUG070 (v2.3.)

Figure 95 Virtex-4 Slice structure



Source: Xilinx Virtex-4 User GuideUG070 (v2.3.)

Figure 96 Virtex-4 Slice L Structure

The FPGA structure represents an additional challenge for C and SystemC based synthesis tools due to the higher granularity and heterogeneity of FPGA compared to ASIC. The variety of FPGA resources makes the resource selection more difficult for the compiler tools to synthesize high-level C constructs. Several similar resources can

be good candidates for one C-construct. The compiler tool has to select the most appropriate resource among all these candidate resources.

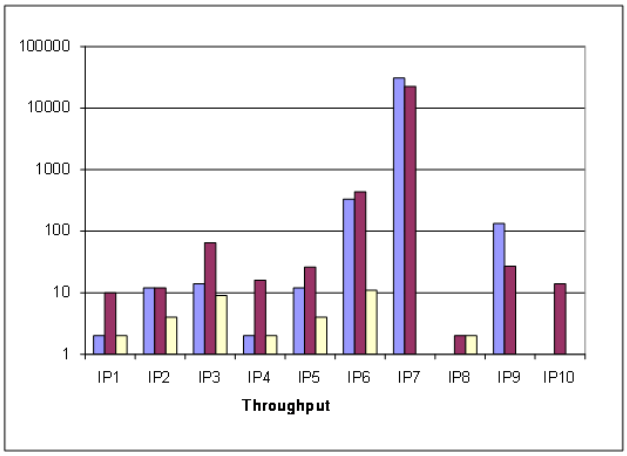
6.5.2.2 Tools and Options Combinations

Physical synthesis have been conducted using Xilinx tools ISE XST[77]. An automatic exploration of physical synthesis (synthesis, place and route) options spanning a wide range from area oriented towards speed oriented with optimization effort, density factor varied at the different steps was conducted with actual execution of the IPs on an FPGA board. This physical level design space exploration comes as a complement to high level optimization techniques used by C-based synthesis such as for example speculative execution (pipeline). The mutual effects - potentially inhibitory - of C-based synthesis followed by a VHDL physical synthesis are unspecified in any of the tool documentations. Different combinations of synthesis and place/route options on the different cases are explored in order to evaluate the possible interactions. The options used for VHDL synthesis and place and route are the global optimization options for area and speed with the level of optimization:

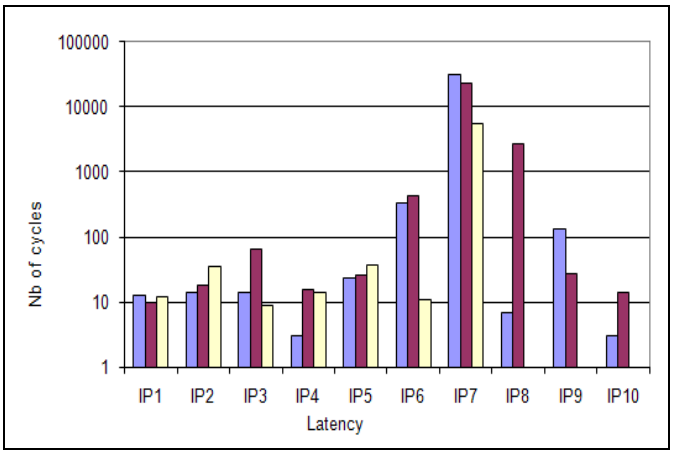
1. XST VHDL synthesis option: -opt_mode Speed or Area
2. mapping option: -cm balanced or speed or area
3. place and route option: -ol std or med or high or -ol high -xe n
4. This results in 24 combinations of synthesis, place and route options for each IP.

6.6 Results of synthesis and place & route.

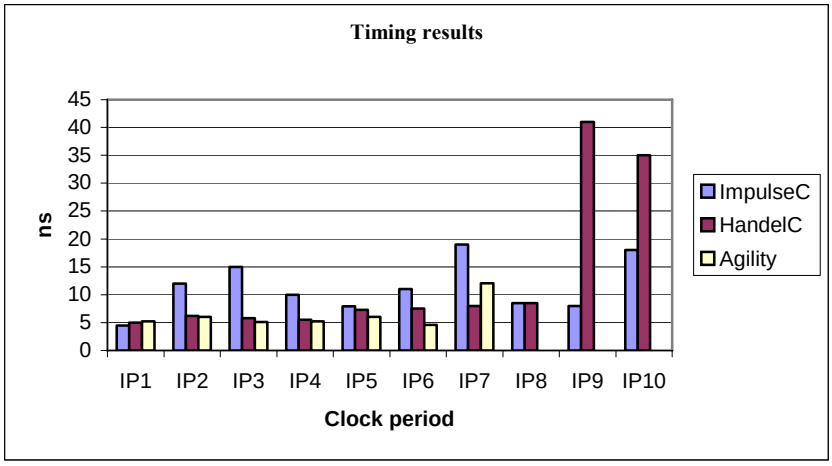
Timing results are presented in Figure 97. The timing results are obtained for each IP with the use of the previously presented tools. The timing metrics are the clock frequency, the latency and the number of cycle per result.



(a)



(b)



(c)

Figure 97 Timing results for throughput, latency and clock period

The variability of results between the tools comes from different reasons. Firstly, the RAM implementation is a direct implementation with no multiplexing of resources. The three RAM of the filters are accessed with one access per clock cycle. It results in a limitation of the pipeline rate of twelve cycles per data produced. Secondly the analysis of the results can be divided in two points. The first point concerns the different approaches the used tools. For SystemC and Handel-C tools pipeline needs to be explicitly described. The C-code is functional with no specific programming with the ImpulseC tool. The number of stages of the pipeline is not precisely controlled with ImpulseC but indirectly through timing constraints. The

automatic exploration of different options and constraints is the only solution to obtain the best compromise between the different constraints as the impact of the throughput/latency of the pipeline on the area/frequency is not straightforward. The difference of throughput between a pure sequential solution and a fully pipeline solution can be more than two orders of magnitude. This is the main source of performance/area trade off at this level. This difference is increased with the implementation variability.

6.6.1 Area results

The area results have been obtained through VHDL generation of the various case studies followed by synthesis and place and route using Xilinx XST tool. They are presented in Figure 98. Our area metrics contain various resources present in the Xilinx Virtex-4 that is slices, Flip-Flop, Look-Up-Table, RAM. The synthesis and place and route options used are here the default options.

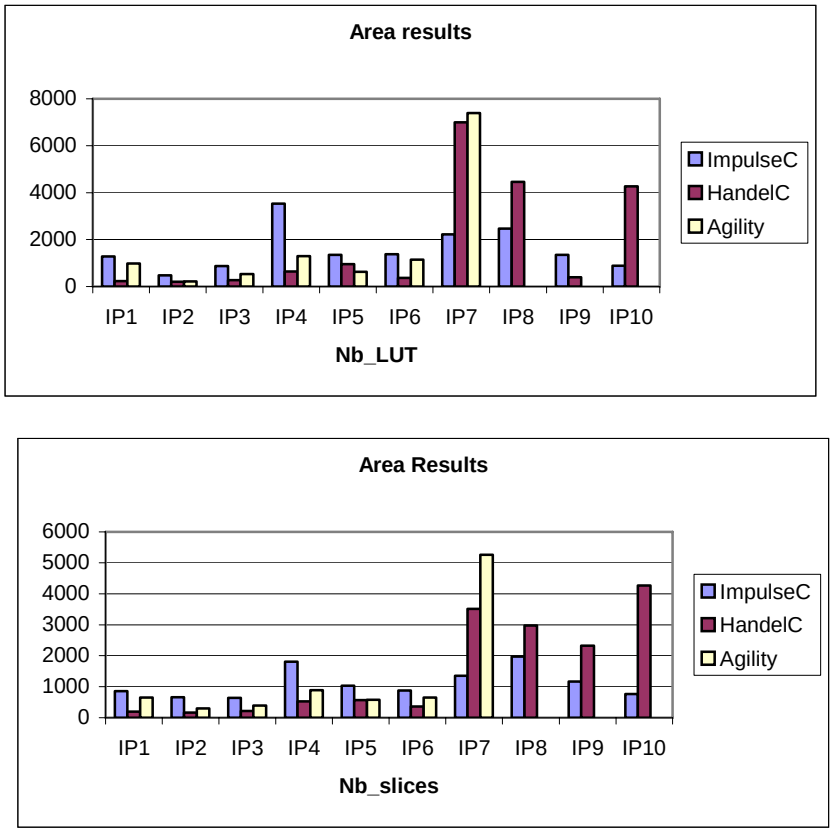


Figure 98 Area results for logic elements

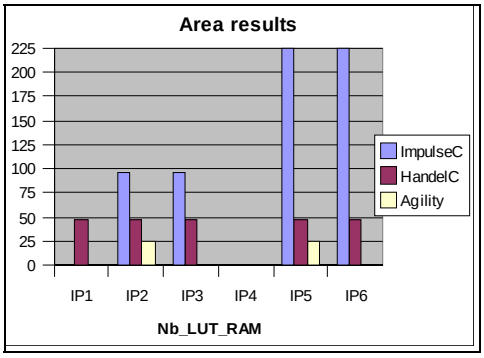


Figure 99 Number of LUT BRAM for storage elements

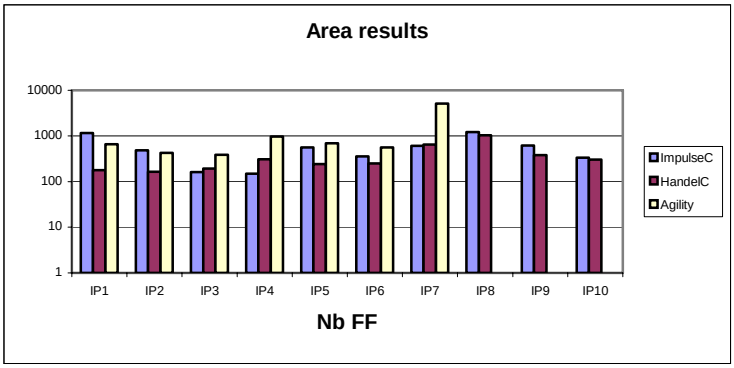


Figure 100 Number of flip flop for storage elements.

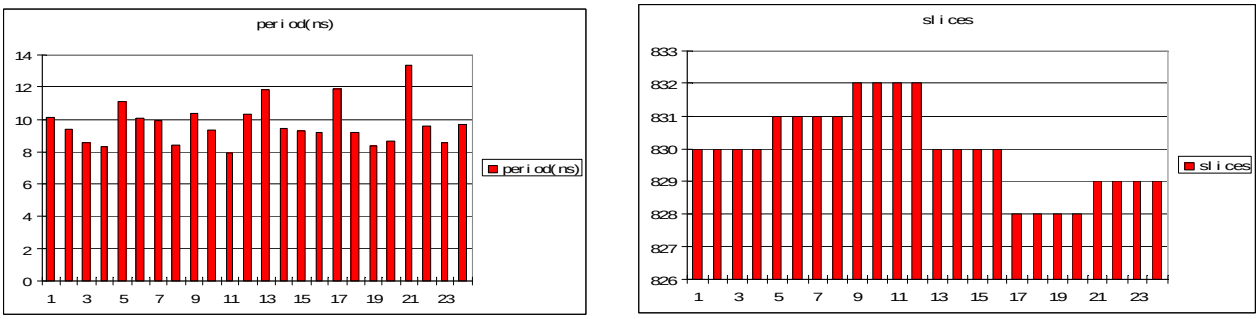
The Figure 99 shows the number of LUT BRAM for storage elements, the number is zero for IP 7 to IP 10. the results of number of Flip Flop synthesized of storage element is shown in Figure 100.

A first observation is that behavioral C-Based approach with ImpulseC produces not always but often more logic and storage elements and a higher clock period than the other approaches. On the other side, ImpulseC brings the advantage of abstraction and genericity. This difference is not critical as throughput and latencies are similar.

6.6.2 Variability of results with compilation options

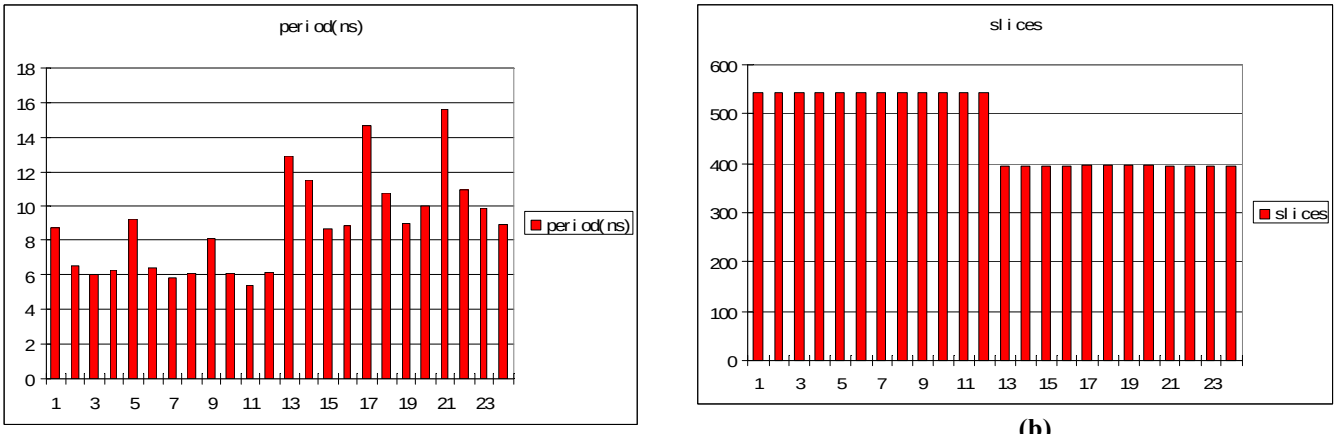
The results presented Figure 101 to Figure 105 show a variability of timing and area results according to the options used for synthesis and place and route. This variability depends on the front-end tool used (Agility, DKDesign, ImpulseC Codevelopper). The results presented in Figure 98 to Figure 100 should be revisited for each option. These important clock period variations up to 150% are obtained with a

variation of area cost between 100 and 200 slices, that is 10 to 20% of area variation. The impact of tool options has a significant impact on timings compared to the impact on the size. Another major observation is the variability of results between options. This variability can be more significant than the variability between front-end tools. For instance the clock period variation is about 10 ns for the pipeline example (Figure 102 and Figure 104). Thus ImpulseC gives better results with one option, for example option 11 but not with another option. Agility gives better results with option 1 (Figure 102). In fact the best trade-off is found by a careful analyze of the area/timing results. The area and timing results are not always linked as we can see with the configurations 19 and 11 in Figure 101. for both configurations the clock period is small but the configuration 11 provides a higher area result compared to the configuration 19. These variations also exist with placement constraints. The timing results can be either better or even worst when constraining area placement. Figure 105 compares the 24 configurations exploration with (left) and without (right) placement constraints. The improvement here is at most 0.15 ns on the clock period which is not significant. Results were even worst for the FFT. Thus a manual floorplanning becomes really difficult for heterogeneous hardware architectures such as FPGA. It should be reminded that obviously synthesis and place and route can incur large variations if no constraints are imposed and if large chips are selected. With large chips the design can be spread without constraints conducting to higher delays. In our case first the synthesize and place and route stages are done without constraints. Then a constrained Place and Route is used for the other configurations tested. The results are better with constraints.



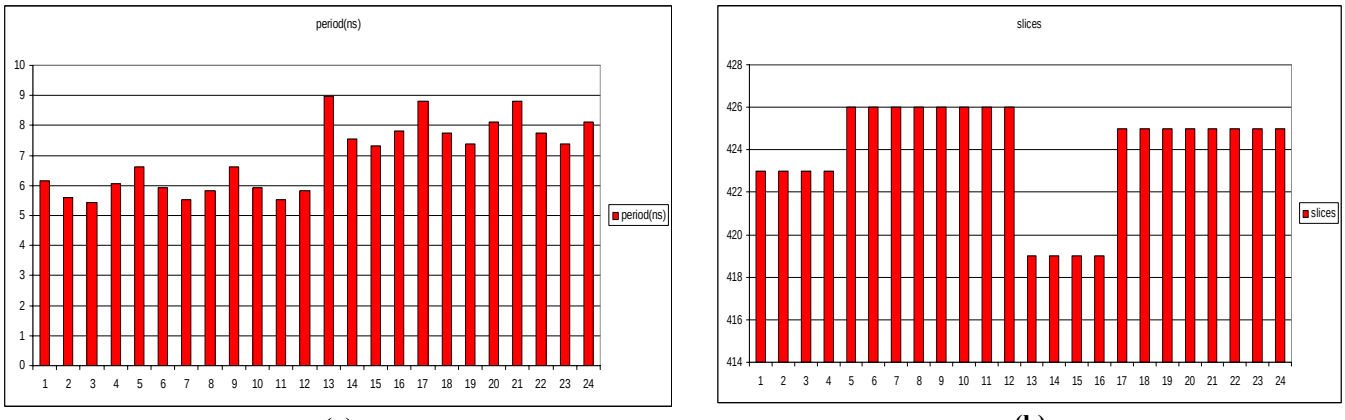
(a) **(b)**

Figure 101 Sequential Mean Filter with ImpulseC- XST VHDL Synthesis tool variation (a) period (b) slices



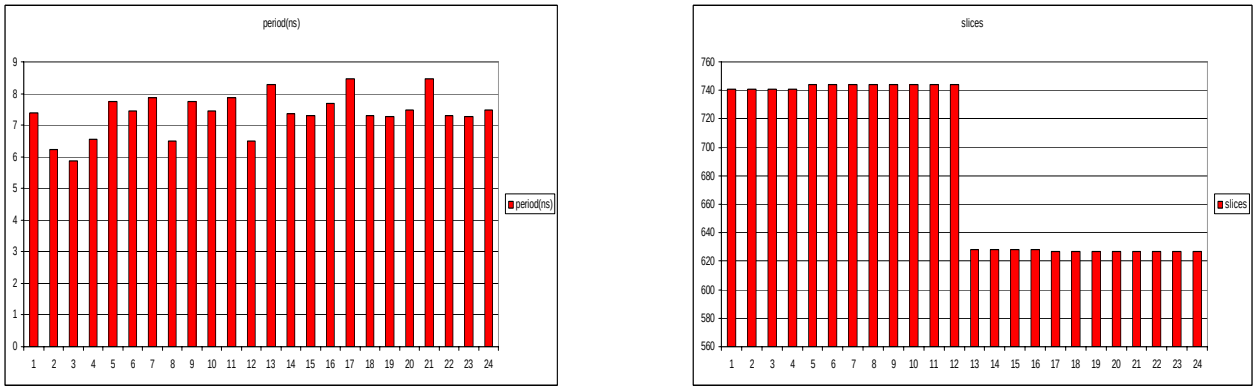
(a) **(b)**

Figure 102 Pipeline Mean Filter with ImpulseC - XST VHDL Synthesis tool variation (a) period (b) slices



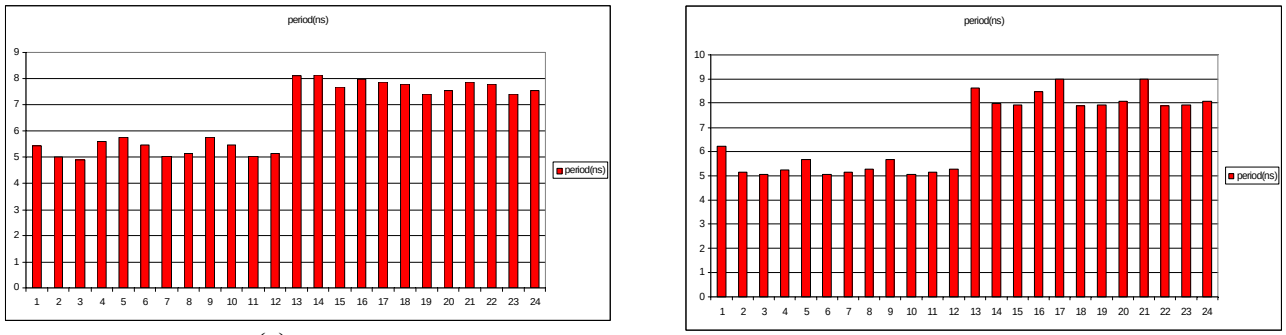
(a) **(b)**

Figure 103 Sequential Mean Filter with Agility - XST VHDL Synthesis tool variation (a) period (b) slices



(a) (b)

Figure 104 Pipeline Mean Filter with Agility - XST VHDL Synthesis tool variation (a) period (b) slices



(a) **(b)**

Figure 105 Pipeline Median Filter with Agility - XST VHDL timing variation with and without placement constraints

The last point concerns the impact on place and route described on several examples in Figure 106 to Figure 109 from best to worst. Best solutions are less spreaded and thus have reduced delays and higher operating frequencies.

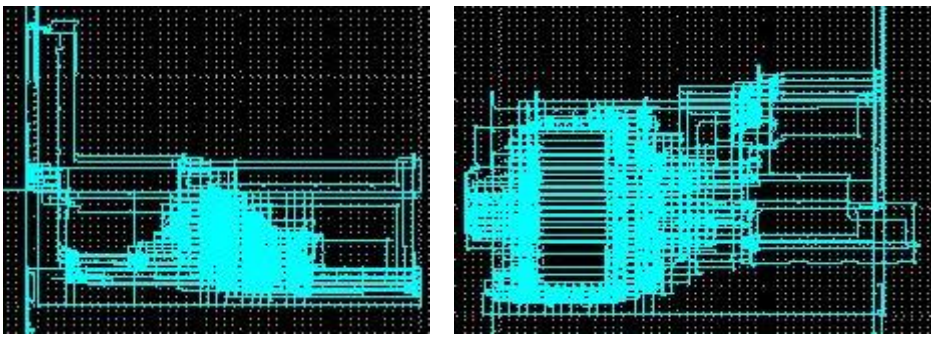


Figure 106 Sequential Mean Filter- Place and Route variations from best (left) to worst (right)

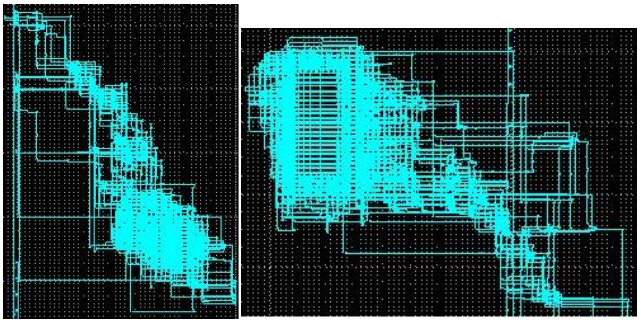


Figure 107 Pipeline Mean Filter -Place and Route variation from best (left) to worst (right)

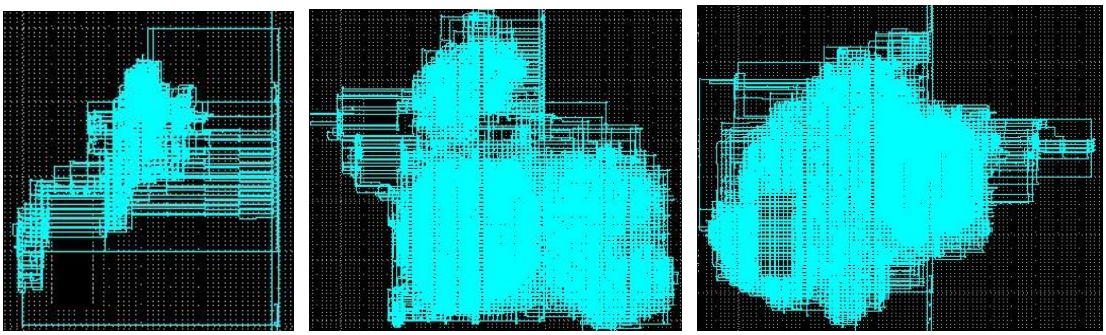


Figure 108 pipeline FFT - Place and Route variations from best(left) to worst (right)

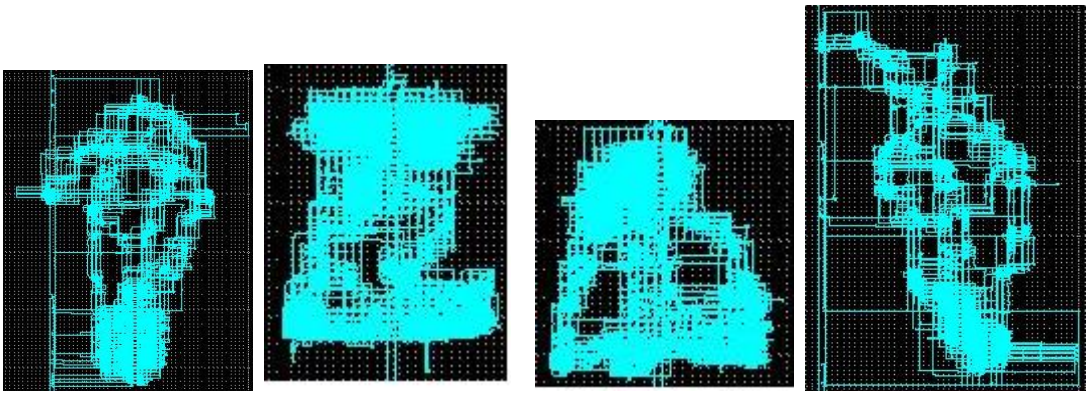


Figure 109 Pipeline Median Filter -Place and Route variations from best (left) to worst (right)

The variations in terms of area and resources are obvious from the above figures. This points out that C-based synthesis may generate very different implementations resulting from C-based high level modeling and the strong impact of back end tools. It should be noted that with heterogeneous devices such as Virtex-4 where hard cores are embedded the place and route tools may decide to implement a function in the vicinity of such embedded cores even if no interaction exists. This can be easily

observed in Figure 107 and Figure 108(right part). This affects the quality of the implementation as the logic is spread out on the circuit. This clearly shows that there is a missing link between the system modeling level and the physical implementation. A feedback is necessary to help joint optimization.

6.7 Discussion

The C-to-hardware compilers considered here take two approaches to concurrency. The first approach chosen by Handel-C and Agility Compiler adds parallel constructs to the language. It forces the programmer to expose most concurrency that is not a difficult task in major cases. Handel-C provides specific constructs that dispatch collections of instructions in parallel. These additional statement constructs can be used by any programmer. The compilers considered here use a variety of techniques for inserting clock cycle boundaries. Handel-C and Agility use fixed implicit rules for inserting clocks and are very simple. Assignments and delay statements each takes one cycle in HandelC and instructions between two wait() statements take one cycle in Agility SystemC. All the instructions inserted in a par statement are executed in one clock cycle in HandelC. For all the implemented filters, adding manually parallelism is an easy task that can be achieved by any programmer. On the other hand, pipeline extraction can become a tricky task as algorithm must be written in that way. An example was the FFT algorithm implementation: adding pipeline from a sequential code can take a long time and changes are important to make. It is even more difficult to express pipeline with HandleC than SystemC as dependencies between instructions imposes the use of different cycles. The precise control of logic/operators per clock cycle is difficult with HandelC: Either all the instructions in one stage are independent and the pipeline clock can be of one clock cycle per result or reuse is possible that makes the number of clocks per result

proportional to the reuse rate. Another solution is to use a higher frequency and divide the processing in elementary cycles (one per code line). SystemC Agility Compiler representation becomes therefore almost an RTL level representation allowing optimization at the clock cycle level.

The second approach lets the compiler identifies parallelism helped with pragmas in the source code. ImpulseC compiler allows automatic pipelining through pragmas but only for inner loops. Loop unrolling is used to obtain full pipelining. Precise control of the number of stages is difficult with such pragmas. These simple rules can make it difficult to achieve a particular timing constraint. It is difficult to predict in advance when a second input data can be inserted, that is the throughput. Several synthesis cycles must be operated to converge to the best solution. The tool helps this exploration by automating the use of VHDL synthesis tool in the loop. Pipeline exploration is conducted automatically with VHDL synthesis on different solutions providing a frequency graph function of the latency/rate of the pipeline. This helps to obtain the higher rate/latency pipeline but with no considerations of the area. It is thus difficult to make a compromise between timing and area constraints.

The IP interfaces provided by ImpulseC are FIFO or memory which is better adapted to stream processing. In fact, it is difficult to design specific protocols at RTL level. Also considering local memory storage, RAM/register inference selection is only obtained through a compilation option of ImpulseC, that is for all the design and not separately for each array, which is really limiting as registers are a limited resource in FPGA. The two other tools provide pragmas to define precisely which way to store arrays of data, with registers or RAM.

According to the data types, C-based Design tools considered several approaches. The first approach neither modify nor augment C's types but allow the compiler to adjust the width of the integer types outside the language. The second approach is to add hardware types to the C-language. Handel-C and ImpulseC compilers chose the data customization. The programmer cannot cast a variable or expression to a type with a different width, that makes the code more difficult to write. For the filter implementation, arrays are of several sizes and the indexes' size are different. The

programmer must often use the concatenation operator to zero pad or sign extend a variable to a given width that make the programming time longer.

Most of the HandelC debugging time was spent in adjusting the size of data and manually programming the pipeline optimization. The programmers must carefully analyze the code to specify all the widths and it can quickly be tiresome. A parser for automatic adjusting of the size of any used variable according to the type of the operators used can be considered.

One main argument to chose an approach to use is the level of description needed at the interface level of the design. If FIFO or memory-like protocols are sufficient, behavioral C-based HLS is now a mature solution with equivalent performances and area results than a more precise almost RTL level C-based solution like Agility Compiler or HandelC. Futhermore, behavioral C-based HLS provides abstraction and genericity of the pipeline allowing easy retargeting of hardware in different timing/area constraints without redesigning. This criteria is fundamental in streaming applications where throughput is the key performance parameter.

6.8 Design Space Exploration Coprocessors in MPSOC

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 16-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. 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. The 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. Instances of the 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 110: 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.

	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

TABLE 29: HLS based TDES IP vs optimized IPs

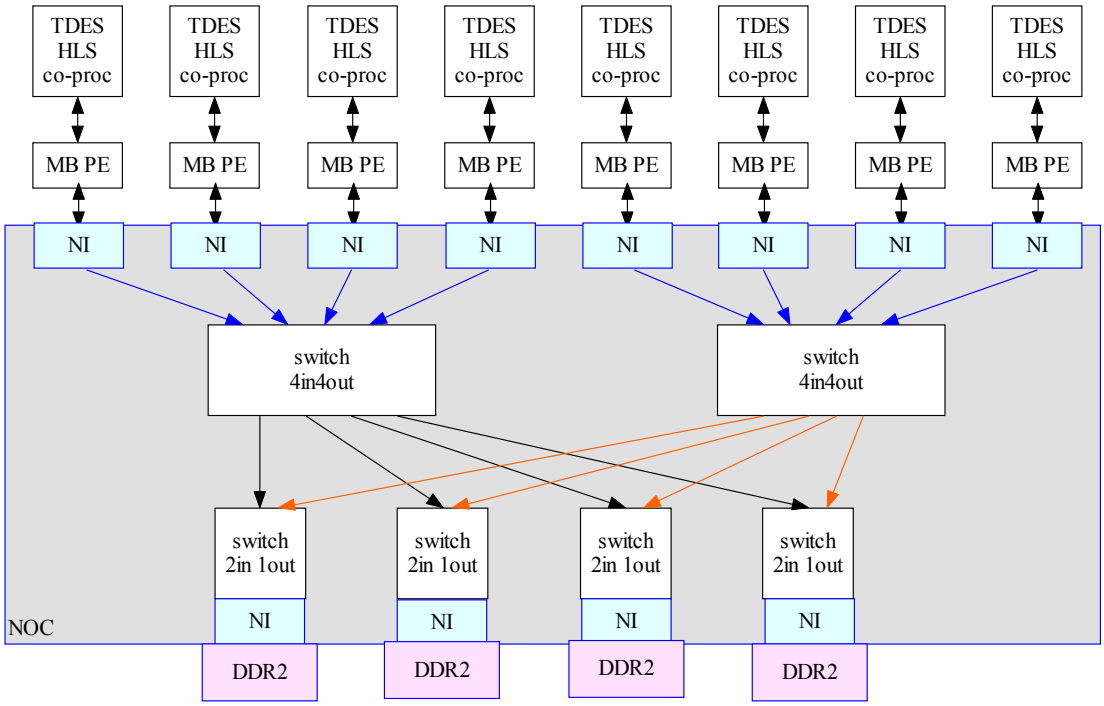


Figure 111: TDES HLS co-processor connected to MPSOC based platform

6.9 Conclusion

We have conducted a case study on the evaluation of C-based high level synthesis systems. The objective was to assess a potential higher use of these in area constrained high level system multi-objective partitioning and how system decisions could be impacted. Indeed, although growing system complexity calls for high level abstract modeling it is still mandatory to take into account precise implementation feedback to improve performances. This puts into question the capacity of C-based tools to meet this challenge. Evidences on case studies of significant result variations among the high level synthesis tools and their emphasis through physical synthesis options exploration challenge the use of C-based multi-objective modeling methodologies for system design. In a multi-objective approach area-performance system design tradeoffs should be based on as accurate as possible data otherwise inappropriate design decisions could be made. We argue that implementation issues

(area, frequency, floorplan) for large scale complex systems should be taken into account when using C-based high level modeling since currently the tools do not guarantee that high level concurrency semantics properties are preserved. Indeed, extracted concurrency at high level is challenged through code and representation transformation as well as resources constraints.

Solution to this come through an integrated flow with concurrency properties propagated as constraints as well as concurrency feedback to the highest level.

Chapter 7 Conclusion

Due to increasingly complex embedded applications, Multi Processor System on Chip design and implementation is the focus of considerable academic and industrial research. Main research directions include MPSOC design flows, MPSOC architecture, MPSOC modelling/performance evaluation techniques and implementation. It remains that despite considerable research there are very few existing implementations. In addition, most of existing implementations present in general a few processors. Intuitively, it seems obvious that issues and solutions in small size MPSOC will be different from larger size MPSOC. Therefore there is a need to design and build MPSOC platforms with a significant number of processors as experimental tool to feedback the current MPSOC design flows. The main focus of this thesis is on implementation in order to better extract physical level design properties which could be used in higher level design while at the same time provides valuable experimental platforms. Indeed, with the advent of high level abstraction modelling and design techniques such as SystemC TLM it appears that there is need to enrich the accuracy of these models with physical implementation. Besides, complex applications communication traffic accurate monitoring is needed to understand how to tune future parallel applications.

We addressed several issues in this thesis:

1. High level synthesis flow and the potential use for MPSOC design,
2. FPGA implementation of shared memory homogeneous and heterogeneous MPSOC,

3. NOC hardware monitoring,

With regard to high-level synthesis, C-based hardware-accelerator or co-processor have been proposed for the architecture based rapid application to implementation solution. We have conducted a case study on the evaluation of C-based high level synthesis. The objective was to assess a potential higher use of these in area constrained high level system multi-objective partitioning and how system decisions could be impacted. Indeed, although growing system complexity calls for high level abstract modeling it is still mandatory to take into account precise implementation feedback to improve performances. This puts into question the capacity of C-based tools to meet this challenge. Evidences on case studies of significant result variations among the high level synthesis tools and their emphasis through physical synthesis options exploration challenge the use of C-based multi-objective modeling methodologies for system design. In a multi-objective approach area-performance system design tradeoffs should be based on as accurate as possible data otherwise inappropriate design decisions could be made. We argue that implementation issues (area, frequency, floorplan) for large scale complex systems should be taken into account when using C-based high level modeling since currently the tools do not guarantee that high level concurrency semantics properties are preserved. Indeed, extracted concurrency at high level is challenged through code and representation transformation as well as resources constraints. This conclusion affects the potential use of high level synthesis for area constrained MPSOC. However; for the design of coprocessor within MPSOC HLS is appropriate. We need to focus on MPSOC platform based design issues such that customization can be achieved with high level synthesized coprocessor while keeping the main MPSOC structure.

We implemented a family of NoC based MPSOC with varying number of processor (2- 24 PE), homogeneous and heterogeneous (Microblaze PPc) on single FPGA chip. Three NoCs, provide different functionalities for 16 PE tiles while on single chip increasing the number of processors reduce the number of NOC to one. Clearly, with area resource constraints a tradeoff exists between an architecture with

fewer processors but with strong communication support and performance monitoring and an architecture with 50% more processors and less communication support This Phd thesis have clearly asserted place and route based the impact of NOC layout on MPSOC design. The cascading Data-NoC connects PE Tiles and DDR2 memories with a high bandwidth; synchronization-NoC offers two synchronization modes for PE tiles. And users can check and configure IPs of Data-NoC through our service NoC. We also demonstrated the use of our performance monitoring system for software debugging and tuning. With the bi-synchronous FIFO method, our GALS architecture successfully solves the long clock signal distribution problem and allows that each clock domain can run at its own clock frequency. Xilinx Virtex4 FX140 FPGA was selected to provide large logic resources with quick implementation and testing. FPGA design environment can offer large number of IP to reduce design efforts and decrease the pressure of time-to-market. For example, in our system a number of IP can be found from the Xilinx EDK library, such as MicroBlaze processor, LMB, FSL, BRAM.

No MPSOC benchmarks existed at the time of this Phd. It is only recently that EEMBC multi-core v.1.0 were available to our laboratory therefore we have conducted performance evaluation studies on our own application. For evaluating our MPSoC we successfully implemented AES and TDES block cipher cryptographic algorithms on our platform using two methods. the first is by dividing data between processors and the second is by dividing the execution of the TDES. Our implemented architecture connected processing elements via two NoCs, such architecture basically suites heavy memory access and allow scalability in the system construction. Results shows better performance for data parallel method favored by the architecture, this result can be reversed in a pipelined architecture. Another conclusion shows that data parallel method can be applied with a greater number of processors, with different operation modes including CBC, because it has a dominating computation time than memory access time allowing a very effective linear speedup, we can predict as well that physically segmenting memory into banks helps pushing this parallelization to attain even a higher figure in processors number although adding more complexity to

the connection between processors. On the other hand our results shows that our architecture does not favor pipelined implementation because of lack of direct link between processor which heavily impacting performance because of the need to a synchronized memory access to exchange data. We note also that going to finer level of granularity may enhance performance. Moving a design from FPGA to ASIC questions the gains and benefits which can be achieved both at an architectural level but as well at the parallel programming. The Network on Chip of our MPSoC has been implemented on ASIC technology. and has been explored with different timing constraints and different library categories of STmicroelectronics' 65nm/45nm technologies. The experimental results of ASIC and FPGA are compared, and our results show although we can naturally expect an area gain the working frequency is not as significantly augmented. This suggests that performance improvement can not be achieved by technology alone and area advantage should be exploited by selecting network on chip components with more aggressive features.

This thesis is a preliminary step in the direction of accurate physical aware MPSoC design flows on both FPGA and ASIC technology. Further research is needed with regards to DFM, automatic translation from FPGA to ASIC with layout sensitive constraints, coupled design space exploration. Platforms as designed in this Phd will contribute to better parallel programming techniques through accurate and real time NoC monitoring and with the hardware support of multiple NOCs, benchmarking methodologies for MPSoC should become a major research topic as a full Design Space Exploration flow can benefit from down the NOC monitoring to high level parallel programming and automatic parallelization. It is clear that the trend to manycore implementations are emerging [118] and there is a need to address large scale platform. The laboratory have contributed in this area through large scale multi-FPGA emulation. Besides, physical constraints of single chip (MPSOC) have triggered a new research direction in 3D ICs [119-121] and new activity have started in this regard in the laboratory. Finally, it appears necessary for large scale multiprocessor manycore to consider the potential of photonics NOC [122-123]. The

use of photonics NOC at least between cluster of MPSOC as described in this phd thesis seems reachable and very promising.

REFERENCES

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

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

[3] Benini, L. ; De Micheli, G., “Networks on Chips: Technology and Tools”, Morgan Kaufmann, 2006

[4] Wolf, W.; Jerraya, A.A.; Martin, G.; “Multiprocessor System-on-Chip (MPSoC) Technology, ” Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 27, Issue 10, Oct. 2008, Page(s):1701 - 1713

[5] Atienza, D.; Angiolini, F.; Murali, S.; Pullini, A.; Benini, L.; De Micheli, G, Network-on-Chip design and synthesis outlook, Integration, the VLSI Journal, Volume 41, Issue 3, May 2008, Pages 340-359

[6] Ogras, U.Y.; Marcillescu, R.; Hyung Gyu Lee; Choudhary, P.; Marculescu, D.; Kaufman, M.; Nelson, P.; “Challenges and Promising Results in NoC Prototyping Using FPGAs” IEEE Micro journal, pp. 86-95, September 2007

[7] C. Bartels, J. Huisken, K. Goossens, P. Groeneveld, and J. Meerbergen,“Comparison of An Athereal Network on Chip and A Traditional Interconnect for A Multi-Processor DVB-T System on Chip,”in Proc. IFIP Int'l Conference on Very Large Scale Integration (VLSI-SoC), October 2006

[8] Pullini, A.; Angiolini, F.; Murali, S.; Atienza, D.; De Micheli, G.; Benini, L.;” Bringing NoCs to 65 nm” Micro, IEEE; Volume 27, Issue 5, Sept.-Oct. 2007 Page(s):75 – 85

[9] P. Guerrier and A. Greiner, “A generic architecture for on-chip packet-switched interconnections,” Proc. Design Automation and Test in Europe (DATE’00), pp. 250-256, Mars 2000

[10] Andriahantenaina and A. Greiner “Micro-network for SoC: Implementation of a 32-port SPIN network,” Design Automation and Test in Europe (DATE 2003) pp. 1128-1129, March 2003.

[11] K. Goossens, J. van Meerbergen, A. Peeters and P. Wielage “Networks on Silicon: Combining Best-Effort and Guaranteed Services,” Design Automation and Test in Europe (DATE’02), 2002.

[12] K. Goossens, J. Dielissen, and A. Radulescu “The AEthereal network on chip: Concepts, architectures, and implementations,” IEEE Design and Test ofComputers, Vol 22, pp. 414-421, Sept-Oct 2005.

[13] K. Goossens, J. Dielissen, O. P. Gangwal, S. Gonzalez Pestana, A. Radulescu, and E. Rijpkema “A Design Flow for Application-Specific Networks on Chip with Guaranteed Performance to Accelerate SOC Design and Verification,” Proc. Of Design, Automation and Test Conference in Europe (DATE05), March 2005.

[14] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," in IEEE International Solid-State Circuits Conference San Francisco, CA, USA: Digest of Technical Papers, 2007, pp. 5-7.

[15] T. Bjerregaard and J. Sparso, "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip," in Proceedings of the conference on Design, Automation and Test in Europe - Volume 2: IEEE Computer Society, 2005.

[16] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," J. Syst. Archit., vol. 50, pp. 105-128, 2004.

[17] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in Proceedings of the conference on Design, automation and test in Europe - Volume 2: IEEE Computer Society, 2004.

[18] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Albenes Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network," in Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum - Volume 2: IEEE Computer Society, 2003.

[19] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-onchip," ACM Comput. Surv., vol. 38, p. 1, 2006.

[20] Senouci, B.; Kouadri M, A.M.; Rousseau, F.; Petrot, F. “Multi-CPU/FPGA Platform Based Heterogeneous Multiprocessor Prototyping: New Challenges for Embedded Software Designers” 19th IEEE/IFIP, pp. 41–47, June 2008

[21] Wentzlaff, D.; Griffin, P.; Hoffmann, H.; Liewei Bao; Edwards, B.; Ramey, C.; Mattina, M.; Chyi-Chang Miao; Brown, J.F.; Agarwal, A.; "On-Chip Interconnection Architecture of the Tile Processor" IEEE Micro journal, pp. 15-31, September 2007

[22] Ito, M.; Hattori, T.; Yoshida, Y.; Hayase, K.; Hayashi, T.; Nishii, O.; Yasu, Y.; Hasegawa, A.; Takada, M.; Mizuno, H.; Uchiyama, K.; Odaka, T.; Shirako, J.; Mase, M.; Kimura, K.; Kasahara, H.;, An 8640 MIPS SoC with Independent Power-Off Control of 8 CPUs and 8 RAMs by An Automatic Parallelizing Compiler Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International 3-7 Feb. 2008 Page(s):90 – 598

[23] ARM 11 MPCore http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html

[24] MIPS32® 1004K™ Core http://www.mips.com/products/processors/32-64-bit-cores/mips32-1004k/

[25] 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.

[26] M.Butts, A.M.Jones, TeraOPS Hardware & Software: A New Massively-Parallel, MIMD Computing Fabric IC, Hot Chips 18th, Aug. 20-22 2006, Stanford, USA. http://www.ambric.com

[27] Texas Instruments Multicore Fact Sheet SC-07175

[28] Texas Instruments TMS320C6474 Multicore DSP SPRS552 – Oct. 2008

[29] Texas Instruments TMS320VC5441 Fixed-Point DSP data manual SPRS122F – Oct. 2008

[30] QorIQ™ P4080 Communications Processor http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=0162468rH3bTdG25E4

[31] 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/ *also "A Power Performance Scalable 8 cores Media Processor for Mobile Multimedia Applications"*, IEEE Journal of Solid State Circuits, Vol.44, No.11, Nov. 2009.

[32] 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/

[33] S.Kumar and al., Architectural Support for Fine-Grained Parallelism on Multi-core Architectures, Vol. 11 Issue 3 (August 2007) Tera-scale Computing , Intel technology Journal.

[34] Mouhoub, R.B.; Hammami, O.; "NoC Monitoring Hardware Support for Fast NoC Design Space Exploration and Potential NoC Partial Dynamic Reconfiguration" IES'06, pp. 1–10, Oct 2006

[35] Krstic, M.; Grass, E.; Gurkaynak, F.K.; Vivet, P.; "Globally Asynchronous, Locally Synchronous Circuits: Overview and Outlook" IEEE Design and Test of Computers, pp. 430-441 September 2007

[36] Trong-Yen Lee; Yang-Hsin Fan; Yu-Min Cheng; Chia-Chun Tsai; Rong-Shue Hsiao "Enhancement of Hardware-Software Partition for Embedded Multiprocessor FPGA Systems", IIHMSP 2007, pp. 19-22, Nov 2007

[37] Hoskote, Y.; Vangal, S.; Singh, A.; Borkar, N.; Borkar, S.; "A 5-GHz Mesh Interconnect for a Teraflops Processor", IEEE Micro journal, pp. 51-61, September 2007

[38] Lukovic,S; Fiorin,L, "An Automated Design Flow for NoC-based MPSoCs on FPGA" 19th IEEE/IFIP, pp. 58-64, June 2008

[39] Trong-Yen Lee; Yang-Hsin Fan; Yu-Min Cheng; Chia-Chun Tsai; Rong-Shue Hsiao; "Hardware-oriented Partition for Embedded Multiprocessor FPGA Systems", 2th ICICIC, pp. 65-65 September 2007

[40] J.Goodacre and A.N.Sloss, "Parallelism and the ARM instruction set architecture", Computer, vol.38, no.7, pp.42-52, Jul.2005

[41] Intel. www.intel.com/Xeon/

[42] IEEE 1666 Standard SystemC Language Reference Manual www.systemc.org

[43] C.Haubelt, J.Falk, J.Keinert, T.Schlichter, M.Streubühr, A.Deyhle, A.Hadert, and J.Teich,"A SystemC-Based Design Methodology for Digital Signal Processing Systems", EURASIP Journal on Embedded Systems, Volume 2007 (2007), Article ID 47580, 22 pages

[44] S. Ouadjaout, D. Houzet "Generation of Embedded Hardware/Software from SystemC", EURASIP Journal on Embedded Systems, Volume 2006 Article ID 18526, 11 pages, 2006.

[45] M. O.Cheema, L. Lacassagne, and O. Hammami, "System-Platforms-Based SystemC TLM Design of Image Processing Chains for Embedded Applications", EURASIP Journal on Embedded Systems, Volume 2007 Article ID 71043, 14 pages, 2007

[46] S. Edwards “The Challenges of Synthesizing Hardware from C-Like language” IEEE Design and Test, Vol. 23, No.5, pp. 375-386, Sept.-Oct.2006.

[47] D. Galloway “The transmogrifier C hardware description language and compiler for FPGAs”. Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines FCCM, pp 136-144, Napa California, April 1995.

[48] K., Wakabayashi “C-based synthesis experiences with a behavior synthesizer, Cyber”, Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings, 9-12 March 1999 Page(s):390 – 393

[49] D.C Ku, G.De Micheli, ”Hardware C: a language for hardware design”. Technical report CSTL-TR 90-419, Computer System Lab, Stanford University, v2.0, August 2000.

[50] T. Kambe, A. Yamada, K.Nishida, K. Okada, M. Ohnishi, A. Kay, P. Boca, V. Zammit, T. Nomura “A C-based synthesis system, Bach and its application”. Proceedings of the Asia South Pacific Design Automation Conference, pp151-155, Yokohama, Japan, 2001

[51] E.Grimpe ,F.Oppenheimer ,"Extending the SystemC synthesis subset by object-oriented features", Proceedings of ISSS+CODES 2003, Page25-30, 2003

[52] S. Gupta, N.D. Dutt, R.K. Gupta, A. Nicolau “SPARK : A High-Level Synthesis Framework For Applying Parallelizing Compiler Transformations”. International Conference on VLSI Design, January 2003.

[53] V, Singh Saun; Preeti Ranjan Panda, "Extracting exact finite state machines from behavioral SystemC descriptions", Proceedings of International Conference on VLSI Design, Page 280-285, 2005

[54] Patel, H.D, Shukla, S.K., Bergamaschi, R.A., Heterogeneous Behavioral Hierarchy Extensions for SystemC, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 26, Issue 4, April 2007 Page(s):765 – 780

[55] S Chtourou, O Hammami, “SystemC Space Exploration of Behavioral Synthesis Options on Area, Performance and Power consumption”, IEEE International Conference on Microelectronics (ICM Islamabad) 2005.

[56] ROCCC http://www.cs.ucr.edu/~roccc/

[57] Synopsys. Behavioral Compiler User Guide Version 1999.10, 1999

[58] Agility : http://www.celoxica.com/products/agility/default.asp

[59] Forte Design www.forteds.com

[60] Orinoco Dale http://www.chipvision.com/company/index.php

[61] ImpulseC Inc “Co-developper’s user guide” www.impulseC.com, 2007

[62] Handel-C http://www.celoxica.com/

[63] CatapultC : www.mentor.com

[64] R. C. Gonzalez, R.E.Woods, “Digital Image Processing “, 3rd Ed.,Prentice Hall, Aug. 2007

[65] J. Revelles, C. Urena, M. Lastra “An efficient parametric algorithm for octree traversal”. ICCGV’2000, Czech Republic, Feb. 2000.

[66] AMD http://developer.amd.com/ZONES/BARCELONA/Pages/default.aspx

[67] NoC Solution 1.12, NoC Compiler user’s guide, o918v7, April 2008

[68] Klimm, A.; Braun, L.; Becker, J.; “An adaptive and scalable multiprocessor system For Xilinx FPGAs using minimal sized processor cores” IPDPS2008, April 2008

[69] Joven, J.; Font-Bach, O.; Castells-Rufas, D.; Martinez, R.; Teres, L.; Carrabina, J.; “xENoC - An eXperimental Network-On-Chip Environment for Parallel Distributed Computing on NoC-based MPSoC Architectures” PDP 2008, pp. 141-148 march 2008

[70] OCP-IP OCP-IPOpenCoreProtocolSpecification2.2.pdf, www.ocpip.org, 2008

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

[72] NoC Solution 1.12, NoC NTTP technical reference, o3446v8, April 2008

[73] Arteris Danube 1.12, Packet Transport Units technical reference, o4277v11, April 2008

[74] Alpha-data ADPe-XRC-4 FPGA card http://www.alpha-data.com/adpe-xrc-4.html

[75] Xilinx Virtex-4 www.xilinx.com

[76] Xilinx EDK 9.2 www.xilinx.com

[77] Xilinx ISE 9.2 www.xilinx.com

[78] Kumar, R.; Tullsen, D.M.; Jouppi, N.P.;, Heterogeneous chip multiprocessors, Computer Volume 38, Issue 11, Nov. 2005 Page(s):32 – 38

[79] Moraes, F.; Calazans, N.; Mello, A.; Moller, L.; Ost, L.HERMES: an infrastructure for low area overhead packet-switching networks on chip, Integration, the VLSI Journal, Volume 38, Issue 1, October 2004, Pages 69-93

[80] Raymond R. Hoare, Zhu Ding, Shenchih Tung, Rami Melhem, Alex K. Jones , A framework for the design, synthesis and cycle-accurate simulation of multiprocessor networks, Journal of Parallel and Distributed Computing, Volume 65, Issue 10, October 2005, Pages 1237-1252

[81] Matteo Monchiero, Gianluca Palermo, Cristina Silvano, Oreste Villa, Exploration of distributed shared memory architectures for NoC-based multiprocessors, Journal of Systems Architecture, Volume 53, Issue 10, October 2007, Pages 719-732

[82] Joan Daemen and Vincent Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard." Springer-Verlag, 2002

[83] FIPS 46-3: The official document describing the DES standard

[84] FIPS 197: The official document describing the AES standard http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[85] Patrick Crowley “The future in your pocket” March 2008 SIGCOMM Computer Communication Review , Volume 38 Issue 2 ACM

[86] 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

[87] 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

[88] 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.

[89] 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.

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

[91] Klimm, A.; Braun, L.; Becker, J.; “An adaptive and scalable multiprocessor system For Xilinx FPGAs using minimal sized processor cores” IPDPS2008, April 2008

[92] “Recommendation for Block Cipher Modes of Operation-Methods and Techniques”, NIST Special Publication 800-38A 2001 Edition

[93] Hutton, M.; Yuan, R.; Schleicher, J.; Baeckler, G.; Cheung, S.; Kar Keng Chua; Hee Kong Phoon;, A Methodology for FPGA to Structured-ASIC Synthesis and Verification**,** Design, Automation and Test in Europe, 2006. DATE '06. Proceedings Volume 2, 6-10 March 2006 Page(s):1 – 6.

[94] Bautista, T.; Nunez, A.;, Quantitative study of the impact of design and synthesis options on processor core performance , Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , Volume: 9 Issue: 3 , June 2001 Page(s): 461 -473

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

[96] Benini, L. ; De Micheli, G., "Networks on Chips: Technology and Tools", Morgan Kaufmann, 2006.

[97] David. J. Frank, Ruchir Puri, Dorel Toma , Design and CAD challenges in 45nm CMOS and beyond, Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design, November 2006.

[98] J. W. McPherson, Reliability challenges for 45nm and beyond, Proceedings of the 43rd annual conference on Design automation, July 2006.

[99] Andrew B. Kahng, Design challenges at 65nm and beyond, Proceedings of the conference on Design, automation and test in Europe, April 2007.

[100] Jamil Kawa, Charles Chiang, DFM issues for 65nm and beyond, Proceedings of the 17th ACM Great Lakes symposium on VLSI, March 2007.

[101] Jinwen Xi, Peixin Zhong , A Transaction-Level NoC Simulation Platform with Architecture-Level Dynamic and Leakage Energy Models, Proceedings of the 16th ACM Great Lakes symposium on VLSI, April 2006.

[102] Love Singhal, Sejong Oh, Eli Bozorgzadeh , Yield maximization for system-level task assignment and configuration selection of configurable multiprocessors, Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis, October 2008.

[103] Matt Nowak, Jose Corleto, Christopher Chun, Riko Radojcic , Holistic pathfinding: virtual wireless chip design for advanced technology and design exploration Proceedings of the 45th annual conference on Design automation, June 2008.

[104] James Balfour, William J. Dally , Design tradeoffs for tiled CMP on-chip networks, Proceedings of the 20th annual international conference on Supercomputing, June 2006.

[105] Timothy G. Mattson, Rob Van der Wijngaart, Michael Frumkin , Programming the Intel 80-core network-on-a-chip terascale processor, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 2008.

[106] Jacob Leverich Hideho Arakida Alex Solomatnikov Amin Firoozshahian Mark Horowitz Christos Kozyrakis, Comparative evaluation of memory models for chip multiprocessors, Transactions on Architecture and Code Optimization (TACO) , Volume 5 Issue 3 , November 2008

[107] Henry Wong, Pangaea: a tightly-coupled IA32 heterogeneous chip multiprocessor, Proceedings of the 17th international conference on Parallel architectures and compilation techniques, October 2008.

[108] Christof Pitter, Time-predictable memory arbitration for a Java chip-multiprocessor, Proceedings of the 6th international workshop on Java technologies for real-time and embedded systems, September 2008.

[109] CMP http://cmp.imag.fr/

[110] Hee Kong Phoon; Yap, M.; Chuan Khye Chai; A Highly Compatible Architecture Design for Optimum FPGA to Structured-ASIC Migration**,** Semiconductor Electronics, 2006. ICSE '06. IEEE International Conference on Oct. 29 2006-Dec. 1 2006 Page(s):506 – 510

[111] Pistorius, J.; Hutton, M.; Schleicher, J.; Iotov, M.; Julias, E.; Tharmalingam, K.; Equivalence Verification of FPGA and Structured ASIC Implementations**,** Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on 27-29 Aug. 2007 Page(s):423 – 428

[112] Compton, K.; Hauck, S.; Automatic Design of Area-Efficient Configurable ASIC Cores, Computers, IEEE Transactions on Volume 56, Issue 5, May 2007 Page(s):662 – 672.

[113] Kuon, I.; Rose, J.;, Measuring the Gap Between FPGAs and ASICs**,** Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 26, Issue 2, Feb. 2007 Page(s):203 – 215.

[114] Pullini, A.; Angiolini, F.; Murali, S.; Atienza, D.; De Micheli, G.; Benini, L.; Bringing NoCs to 65 nm Micro, IEEE, Volume 27, Issue 5, Sept.-Oct. 2007 Page(s):75 – 85

[115] 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.

[116] A.Hanson, K.Goossens,M.Bekooij and J.Huisken,"CoMPSoC: A template for composable and predictable multi-processor system on chips", ACM Transactions on Design Automation of Electronic Systems (TODAES) Volume 14 , Issue 1, Jan. 2009.

[117] E.S.Chung and al, ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs, ACM Transactions on Reconfigurable Technology and Systems (TRETS),Volume 2 , Issue 2 (June 2009).

[118] D.N. Truong and al.," A 167-Processor Computational Platform in 65 nm CMOS", IEEE Journal of Solid State Circuits, pp.1130 – 1144, Vol.44, No.4, April 2009.

[119] P.Emma and E.Kursun, "Opportunities and Challenges for 3D Systems and Their Design", IEEE Design & Test of Computers, pp.6-14, September/October 2009.

[120] H.Sun and al.,"3D DRAM Design and Application to 3D Multicore Systems", IEEE Design & Test of Computers, pp.36-46, September/October 2009.

[120] Pavlidis, V.F.; Friedman, E.G.; "3-D Topologies for Networks-on-Chip", Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Vol. 15, Issue 10, Oct. 2007 Page(s):1081 – 1090.

[121] Feero, B.S.; Pande, P.P.; "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation", Computers, IEEE Transactions on Volume 58, Issue 1, Jan. 2009 Page(s):32 – 45.

[122] C.Batten and al.,"Building Many-core Processor-To-DRAM Networks with Monolithic CMOS Silicon Photonics", IEEE Micro, pp.8-21, July-Aug. 2009.

[123] M.Petracca and al.,"Photonic NOCs: System Level Design Space Exploration", IEEE Micro, pp.74-84, July-Aug. 2009.

PUBLICATIONS

Journal papers

1. O.Hammami, Z.Wang, V.Fresse and D.Houzet, “A Case Study: Quantitative Evaluation of C-based High Level Synthesis Systems”, EURASIP Journal on Embedded Systems, 2008.
2. Z. Wang, O.Hammami, “A NoC based heterogeneous Multiprocessor System on Chip Implementation” *submitted Elsevier Microelectronics Journal*
3. Z.Wang, O.Hammami and D.Houzet, “Architecture and Parallel Programming Implications of Migrating Single FPGA Chip Multiprocessor with Network on Chip to 65nm and 45nm ASIC” submitted *Elsevier Microelectronics Journal*
4. I.Taj, Z.Wang, O.Hammami, M.Akil and K.Huggins, "Performance Evaluation and Enhancement Mechanisms for Image Processing Algorithms on 16 PE NOC Based Multi-core Embedded Platform" *submitted to Journal of Real-Time Image Processing*

Conferences papers

5. Hammami, Z.Wang, V. Fresse and D. Houzet “A Quantitative Evaluation of C-Based Synthesis on Heterogeneous Embedded Systems Design”, published in IEEE ISCAS 2008, Seattle, USA.
6. Z. Wang, O. Hammami, “C-Based Hardware-Accelerator Coprocessing for SOC An Quantitative Area-Performance Evaluation”, IEEE ICECS 2008, Malta.
7. Z. Wang, O. Hammami, “A Twenty-four Processors System on Chip FPGA Design with On-Chip Network Connection”, IP SOC 2008, France.
8. Z. Wang, O. Hammami, “External DDR2-Constrained NOC-Based 24-Processors MPSOC Design and Implementation on Single FPGA”, IEEE International Design and Test Workshop 2008, Tunisia.
9. M.KHaddour, Z.Wang and O.Hammami "Cryptography on Multiprocessor Platform (The Case of AES)" , Sixth IEEE International Multi-Conference on Systems, Signals and Devices SSD09
10. M.KHaddour, Z.Wang and O.Hammami "Implementing Block Cipher on Multiprocessor Platform”, International Conference on Multimedia Computing and Systems ICMCS 09
11. M.KHaddour, Z.Wang and O.Hammami “Performance Evaluation and Analysis of Parallel Software Implementations of TDES on a16-PE Embedded Multiprocessor Platform”, IFIP network and service security conference, Paris 2009.

Thèse préparée dans Laboratoire Electronique et Informatique d'ENSTA