System-on-chip design flow for Software Defined Radio

Guangye Tian

=$N° D'ORDRE

THÈSE DE DOCTORAT

SPECIALITE : INFORMATIQUE

Ecole Doctorale « Informatique »

Présentée par : Mr. Guangye Tian

Sujet :

Flot de Conception Système sur Puce pour Radio Logicielle

Soutenue le 28/06/2011 .devant les membres du jury :

M ……Alain MERIGOT, Président……

M…… Guy GOGNIAT, Rapporteur ……

M ……Smail NIAR, Rapporteur……

M…… Daniel ETIEMBLE, Directeur de thèse…

M…… Omar HAMMAMI, Co-directeur de thèse

ACKNOWLEDGMENT

Je remercie vivement mon directeur de thèse, Processeur Daniel Etiemble, Professeur d'Université Paris Sud, pour tous ses conseils et discussions techniques.

J'exprime ma profonde reconnaissance à mon co-directeur de thèse, Processeur Omar Hammami, processeur de l'ENSTA ParisTech, pour toute la confiance qu'il m'a témoigné 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 donné, 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 Messieurs Guy Gogniat, professeur à l'université de Bretagne Sud et Smail Niar, professeur à l'université de Valenciennes, pour le temps qu'ils ont consacré à l'examen de mes travaux et à la rédaction du rapport de thèse.

Je remercie Monsieur Alain Mérigot, Professeur à l'université Paris Sud, pour avoir accepté de présider le jury.

Je remercie, bien évidement, Xinyu Li, Zhoukun Wang, Mazen Khaddour, Muhammad Imran Taj et tous mes amis, collèges de l'ENSTA pour leur soutien pendant le moment le plus difficile.

Enfin, et sur tout, je remercie ma famille pour son soutien, sa compréhension et ses encouragement.

ABSTRACT

The Software Defined Radio (SDR) is a reconfigurable radio whose functionality is controlled by software, which greatly enhances the reusability and flexibility of waveform applications. The system update is also made easily achievable through software update instead of hardware replacement. The Software Communication Architecture (SCA), on the other hand, is an open architecture framework which specifies an Operating Environment (OE) in which waveform applications are executed. A SCA compliant SDR greatly improves the portability, reusability and interoperability of waveforms applications between different SDR implementations.

The multiprocessor system on chip (MPSoC) consisting of large, heterogeneous sets of embedded processors, reconfiguration hardware and network-on-chip (NoC) interconnection is emerging as a potential solution for the continued increase in the data processing bandwidth, as well as expenses for the manufacturing and design of nanoscale system-on-chip (SoC) in the face of continued time-to-market pressures.

We studied the challenges of efficiently deploying a SCA compliant platform on an MPSoC. We conclude that for realizing efficiently an SDR system with high data bandwidth requirement, a design flow with systematic design space exploration and optimization, and an efficient programming model are necessary. We propose a hybrid programming model combining distributed client/server model and parallel shared memory model. A design flow is proposed which also integrates a NoC topology synthesis engine for applications that are to be accelerated with parallel programming and multiple processing elements (PEs).

We prototyped an integrated SW/HW development environment in which a CORBA based integrated distributed system is developed which depends on the network-on-chip for protocol/packet routing, and software components are deployed with unified interface despite the underlying heterogeneous architecture and os; while the hardware components (processors, IPs, etc) are integrated through interface conforming to the Open Core Protocol (OCP).

Key works: Hybrid programming model, parallel programming, Network-on-chip, FPGA, SDR, and SCA

Résumé

1. Motivation

Depuis la première commercialisation de systèmes mobiles cellulaires début des années 1980, l'industrie des communications sans fil a connu un développement croissant de normes de communication de la technologie première génération (1G) au standard de quatrième génération (4G). La technologie 1G a été introduite début des années 1980 et complétée début des années 1990. La technologie sans fil 1G est analogique. La technologie 2G, dont le développement a commencé fin des années 1980 et s'est terminé dans les années 1990, est souvent qualifié de « numérique », et a remplacé la technologie 1G en utilisant des signaux et des réseaux numériques. Entre la technologie 2G et la 3G, on a pu observer le déploiement intermédiaire, une technologie 2.5G, numérique mais avec des possibilités de transfert de données limitées telles que les services de SMS. Les systèmes de troisième génération 3G, développés dans les années 1990, ne se limitent plus à la seule transmission de la parole comme en 2G, mais permettent l'utilisation simultanée de la parole et de services de données à un débit plus élevé. Ainsi, les réseaux 3G permettent aux opérateurs de réseau d'offrir aux utilisateurs une gamme plus large de services plus avancés, bien qu'à des débits réseau accrus, grâce à une meilleure utilisation du spectre alloué. Successeur de la télécommunication 3G, la technologie $4^{ème}$ génération (4G) fournira aux utilisateurs des services de transmission de la parole, de données, et de multimédias en temps réel, à un débit encore plus élevé. Elle offrira également une meilleure qualité de service (QoS), la sécurité et la possibilité d'interface avec des réseaux filaires constituant l'épine dorsale de l'architecture du réseau.

Afin de supporter les besoins et les contraintes des différents réseaux, de très nombreuses normes sont apparues. Les opérateurs réseaux doivent se conformer à l'ensemble de ces normes, des premières de la technologie 2G à celles attendues concernant la 3G.

Vu que chaque norme est différente et utilise même parfois des fréquences porteuses différentes, des stations ou des handsets doivent être développés, déployés et maintenus, entraînant des codes très lourds et un développement lent. Comte tenu du rythme auquel les nouvelles normes sont publiées, la conformité à ces normes pour un coût acceptable en temps

de développement et en taille de puce devient vite un un cauchemar pour toute personne impliquée dans les systèmes de communication.

Le concept de Radio Logicielle a été proposé, la première fois, par Joseph Mitola III pour faire face à une telle crise. Dans cette approche, les transformations de la forme d'onde, modulation, démodulation des signaux d'un système radio sont mises en œuvre par du logiciel plutôt que par du matériel à fonctionnalité spécifique. Les composants développés en logiciel sont ensuite implantés dans les dispositifs modernes programmables/reconfigurables, tels que les GPP, DSP, FPGA, ou ASIP. Avec de tels dispositifs, l'adaptation du système à une autre norme de communication, ou même l'évolution vers une technologie plus récente peuvent être réalisés par mise à jour du logiciel sans remplacement du matériel qui serait long et coûteux. Compte tenu des progrès de la technologie des semi-conducteurs et de la technologie sans fil fournissant un accès haut-débit à Internet fiable, les mises à jour de logiciels et la reconfiguration du système peuvent être réalisées en temps réel avec des données de configuration téléchargées via Internet. De cette façon, un dispositif unique peut être rendu compatible avec tout un ensemble de normes, par exemple, ZigBee, Bluetooth, 802.11 a/b/g/n, 3G, etc. Il est possible de réaliser le passage d'un protocole à l'autre sans dégradation de qualité de service, si la conception est rigoureuse.

La réalisation de ces fonctions concernant la radio, par logiciel, présente un avantage sous réserve de :

1. réutilisabilité, portabilité et d'interopérabilité des applications
2. La plateforme et le support de modèle de programmation permettant de maintenir la complexité de la programmation à un niveau raisonnable

La première condition est primordiale : elle est à l'origine même du concept de radio logicielle. Les avantages de la flexibilité, idée maîtresse de ce concept, ne sont effectifs que si l'on peut librement ajouter, mettre à jour ou améliorer les capacités fonctionnelles d'un système radio réalisé sous forme de modules logiciels. Idéalement, les traitements concernant les formes d'onde conçue pour une plate-forme SDR peuvent être facilement transposés à une

autre plate-forme ; de même, les traitements développés par une entreprise peuvent fonctionner conjointement avec ceux d'une autre entreprise. Pour atteindre cet objectif, il faut qu'un framework ouvert et standardisé définisse des interfaces homogènes et les services auxquels une application doit se conformer.

L'architecture de communication logicielle (Software Communication Architecture, SCA), est une architecture ouverte largement acceptée pour les projets de SDR. Elle est développée par le Département de la Défense des Etats-Unis (DoD) pour la réalisation, pour un coût abordable, d'une famille de systèmes radio tactiques de haute capacité offrant des services réseaux évolutifs. La spécification SCA définit un environnement d'exploitation (Operation Environment, OE) dans lequel on exécute les applications. L'OE est constitué d'un cadre de base (Core Framework), d'un middleware minimal conforme à CORBA, et d'un système d'exploitation conforme à POSIX. La norme POSIX minimise le coût de portage des applications car elle fournit une couche d'abstraction qui rend transparentes les méthodes spécifiques de chaque système d'exploitation. CORBA permet un certain niveau de transparence et l'indépendance vis-à-vis du langage de programmation. Dans cette thèse, on s'intéresse au développement et à la programmation d'une plateforme SDR conforme à SCA.

D'autre part, beaucoup d'applications haut-débit ont besoin d'une puissance de traitement et d'une bande passante I/O supérieures à celles fournies par les systèmes traditionnels composés d'un mono-processeur accompagné de certains IPs matériels. Les nouvelles plateformes de SDR sont en général implémentées sur des plateformes multiprocesseurs système sur puce (MPSoC) exploitant ses importantes ressources de calculs avec une bonne efficacité énergique. Il existe déjà des systèmes intégrant des dizaines cœurs, des matériaux reconfigurables et le réseau sur puce. Les possibilités d'un rapide développement, déploiement et vérification des logiciels embarqués parallèles sur ces nouvelles plateformes MPSoC sont autant de points clés pour satisfaire les objectifs de performance tout en respectant les délais de mise à disposition sur le marché et le coût de développement.

Le déploiement de SDR à base de SCA sur une plateforme moderne MPSoC implique la combinaison de deux paradigmes de programmation : le modèle distribué à base de CORBA,

et le modèle parallèle utilisant la programmation SMP. La conception de SDR à base standard manque de flots de conception et d'un modèle de programmation efficace pour tirer parti de riches ressources de calcul de MPSoC de manière systématique. Dans cette thèse, nous proposons un flot de conception de SDR avec une exploration architecturale et une optimisation systématique basé sur un modèle de programmation hybride (le modèle distribué client/serveur et le modèle parallèle).

Nous nous sommes intéressés à la partie traitement de bande de base d'un système radio. Les fonctionnalités de bande de base radio sont représentées dans un réseau de processus de Kahn. Un système distribué sans contraintes de ressources est produit par un engin de générateur de système distribué d'un premier niveau. Les nœuds générés sont analysés et classés afin de déterminer ceux qui ont un important besoin en performance de calcul. Ces nœuds sont ensuite regroupés dans une sous-branche pour être parallélisés. A la sortie du flot, une voie de rétroaction globale est fournie pour permettre l'optimisation des ressources, et l'ajustement de fréquence tout en répondant aux exigences de performance du système.

2. L'Etat de l'art de la radio logicielle et de la radio cognitive

Il existe des efforts sur l'implémentation de la plateforme radio logicielle ainsi que le développement d'un framework complet pour le développement et le déploiement des applications waveforms.

Le SCARI-OPEN est une implémentation de la JTRS Architecture de la communication de logiciel (SCA v2.2). Il a été certifié par le JTRS-JPO. Le projet est effectué par la Centre de Recherche de Communications (CRC) de Canada avec l'objectif de développer une référence d'implémentation (RI) afin de : 1. réduire le niveau d'ambigüité de la spécification SCA ; 2. augmenter l'interopérabilité des applications ; 3. Augmenter la compréhension de l'architecture par un exemple ; 4. accélérer l'émergence de SDRs par la disponibilité d'une implémentation ; 5. réduire le coût de développement et le délai de mise sur le marché.

L'OSSIE (Open Source SCA Implementation :: Embedded) est un core framework basé sur SCA et un outil pour le développement rapide de SDR. Il est développé à Virginia Tech et la

dernière version est la version 0.8.0 sortie en 2010. L'OSSIE contient un core framework et une suite d'outils orientés GUI (Interface Utilisateur Graphique) qui est capable de générer automatiquement les codes sources conformes à SCA et les fichiers de support permettant aux développeurs de se concentrer sur les fonctionnalités de traitement de signal.

La SDR-4000 est une plateforme émetteur-récepteur développée par Spectrum. Elle offre des COTS (Composant pris sur étagère) matériels, logiciels et services pour accélérer le développement et le déploiement des solutions pour le modem sans fil. La SDR-4000 contient deux composants principaux, l'engin de traitement modem PRO-4600 et l'émetteur-récepteur à deux canaux XMC-3321. Les deux composants ensemble constituent un modem sans fil supportant deux canaux par slot.

L'IDROMel est un projet de l'Agence Nationale de la Recherche (ANR) de France visant à définir et valider une SDR reconfigurable et une plateforme de CR. La plateforme combine les technologies les plus récentes, comme : 1. le traitement bande de base flexible ; 2. un système intégré basé sur un réseau sur puce ; 3. un support de reconfiguration partielle utilisant un FPGA ; 4. une bande RF très large de 200 MHz à 7.5 GHz ; 5. un support de 4x4 MIMO ; 6. une conception flexible de MAC pour le support de handover vertical.

Le WiNC2R développé par l'Université Rutgers est un prototype de plateforme de radio cognitive. L'Annabelle développé par l'Université de Twente, propose une architecture multiprocesseur système-sur-puce (MPSoC) pour le traitement bande de base de la radio cognitive. La SDR LSI est une solution mono-puce pour le traitement bande de base de SDR développé par Fujitsu.

Dans les chapitres suivants, nous nous intéresserons à l'aspect de traitement de bande de base de la SDR.

3. L'implémentation et l'optimisation d'un système embarqué pour la SDR

Dans ce chapitre, plusieurs méthodologies de conception de multiprocesseur système sur puce (MPSoC) sont présentées. On a proposé un flot de conception de MPSoC avec l'aide d'un

paralléliseur automatique, l'outil PLuTo. PLUTO effectue des transformations source-à-source automatiques basées sur la modélisation polyédrique. Il est capable d'optimiser les séquences de boucles imbriquées pour le parallélisme à grain gros et la localité de cache simultanément. Après la transformation, un code parallèle OpenMP est généré qui peut être ensuite exécuté sur les plateformes multi-cœurs. Par contre, l'exécution du code OpenMP dépend des APIs OpenMP, du compilateur et du support runtime de l'OS qui sont rarement présents dans un système embarqué. On a donc conçu un adaptateur OpenMP vers l'environnement embarqué qui est intégré dans le flot de conception d'accélérateur.

Une étude de cas est présentée dans laquelle on a programmé et évalué une plateforme multiprocesseur système sur puce à base de réseau sur puce développée par le laboratoire. Le système est composé de 16 processeurs de type Microblaze et les communications inter-processeur se font à travers un réseau sur puce avec un modèle de programmation à mémoire partagé.

Nous avons étudié le potentiel de la parallélisation automatique sur le système multi-cœur avec 16 processeurs élémentaires (PE) interconnectés par un réseau sur puce (NoC). L'implémentation effective de matériel nous a permis d'aborder les trois sujets suivants : (1) l'efficacité du support matériel des primitives de synchronisation, (2) la performance de la parallélisation automatique, (3) les avantages de la multiprogrammation. Avec le paralléliseur PLUTO, on a fait des expériences de programmation parallèle sur la plateforme MPSoC. On a noté que plusieurs éléments clé existent qui influent sur l'efficacité de la parallélisation. Certains de ces éléments sont inhérents à l'application, tandis que d'autres dépendent de l'architecture. Une compréhension détaillée des caractéristiques aussi bien de l'application que de l'architecture est essentielle pour obtenir une performance satisfaisante. On a programmé la multiplication de matrices, Seidel, la DCT, et Jocobi 1d. La multiplication de matrices et la DCT présentent de bonnes caractéristiques pour la parallélisation et la performance évalue linéairement quand le nombre de processeurs augmente. La performance pour Seidel atteint un pallier quand le nombre de processeur dépasse 8. Jocobi 1d n'expose aucun parallélisme. Il n'y a donc aucun intérêt à essayer de paralléliser cette application.

Pour la plupart des applications, les ressources des processeurs ne peuvent pas être totalement exploitées. Nous sommes naturellement conduits à la solution multiprogrammation où les ressources processeurs sont partagées par plusieurs applications. Nous constatons que le nombre de 8 processeurs est souvent un point critique au-delà duquel l'augmentation de performance avec l'augmentation de nombre de processeurs s'arrête. Une combinaison judicieuse d'applications peut effectivement améliorer la performance globale. Une solution est de partager les ressources entre plusieurs applications. On a donc fait des expériences de multiprogrammation sur la même plateforme. Les résultats montrent une meilleure utilisation de ressources.

L'Unité d'Interface de Réseau (Network Interface Unit, NIU) du MPSoC en question est basée sur le Protocole Open Core (OCP). OCP est un protocole non propriétaire. Il établit un standard commun pour l'intégration des propriétés intellectuelles (IPs) à la façon « plug et play ». Le protocole OCP est basé sur le modèle maître-esclave point-à-point. Nous nous sommes intéressés à deux des mécanismes de synchronisation fournies par le protocole OCP, plus précisément : la synchronisation exclusive, et la synchronisation paresseuse. On a développé un benchmark de synchronisation de type barrière pour tester la performance des deux mécanismes sous différentes hiérarchies de mémoire ainsi que différentes types de mémoire. Les résultats montrent que la performance de la synchronisation exclusive dépasse celle de la synchronisation paresseuse de 50% quand les variables de synchronisation sont centralisés. Quand il s'agit du même mécanisme de synchronisation avec différents types de mémoire, on a constaté que lorsque la variable de synchronisation est placée dans la mémoire sur puce BRAM, la performance est meilleure que lorsqu'elle est dans la DDR.

4. Le mapping de middleware sur un système embarqué distribué à base de réseau

Il y a de plus en plus de systèmes qui sont composés d'une collection de composants divers interconnectés par un réseau où chaque composant exécute des fonctionnalités qui impliquent à la fois l'interaction locale et distante avec d'autres composants du système. Stimulée par l'augmentation du nombre d'applications à base de réseau, la technologie middleware est devenue de plus en plus importante. Dans un système distribué, le middleware est défini comme une couche de logiciel qui se situe entre le système d'exploitation et les applications.

Par cacher l'hétérogénéité de l'architecture, l'OS sous-jacent et le langage de programmation, le middleware facilite l'intégration d'application, améliore la portabilité des composants logiciels et l'interopérabilité des applications développées par différentes entreprises.

Dans ce chapitre, on a introduit la spécification middleware du Groupe de Management d'Objets (Object Management Group, OMG) : c'est Common Object Request Broker Architecture (CORBA) et sa version embarquée, l'eCORBA.

Il existe de nombreuses implémentations de CORBA académiques ou commerciales. On a présenté omniORB, qui est développé par le Laboratoire AT & T de Cambridge. OmniORB sera plus tard choisi comme middleware dans notre système distribué.

Nous avons construit un système embarqué distribué utilisant plusieurs cartes FPGA comme plateforme de preuve de concept. Les cartes sont interconnectées via un commutateur Ethernet. Chaque carte contient un système de calcul à base de processeur PowerPC405 disposant d'un système d'exploitation Linux avec une pile TCP/IP. Les applications de communications sont développées à l'aide du middleware CORBA conforme à la spécification SCA. La performance du middleware est évaluée à l'aide de micro-benchmarks d'évaluation. Les effets de l'augmentation de fréquence sur la performance globale du système sont examinés pour chaque composant du système (Client, Serveur, ou Services communs). Les résultats donnent de bonnes indications sur le domaine de fréquences qui minimise la consommation d'énergie.

A la fin de ce chapitre, on a proposé un flot de conception pour la SDR avec l'exploration architecturale systématique et l'optimisation multi-objective utilisant le modèle de programmation hybride (distribué client/serveur + parallèle).

5. Le mapping de middleware sur un mono-puce multiprocesseur système

L'objectif ultime est d'intégrer l'ensemble sur une seule puce pour fournir une plateforme de SDR bande de base conforme à SCA. La plateforme hybride, basée sur un commutateur Ethernet, dont on a parlé, tout en permettant une preuve-de-concept rapide et pertinente, a ses

limites comme la bande passante du réseau et la flexibilité de configuration en raison de l'isolement des nœuds. Afin de bien tirer parti de l'interopérabilité et de la portabilité des applications à base de CORBA, la conception de plateforme SDR conforme à SCA sur une mono puce implique d'effectuer l'adaptation du mécanisme de transmission de CORBA de GIOP/IIOP à une couche de communication sur puce propriétaire. Dans notre cas, on va utiliser la bibliothèque Danube d'Arteris [65] pour l'interconnexion de plusieurs dispositifs de calcul, des mémoires, et des IPs via une interface standard, le Open Core Protocol (OCP). Le mécanisme de transport de CORBA est par défaut TCP/IP via internet. On va garder TCP/IP comme protocole de la couche transport et de la couche réseau de CORBA. Par contre, on va modifier la couche MAC en remplaçant Ethernet par OCP/NTTP. NTTP est un protocole de transport de paquet propriétaire implémenté dans les composants de Danube. Avec cet empilage de protocole, la couche de communication de CORBA peut rester largement inchangée et un driver gérant l'interface OCP doit être inclus au noyau Linux afin de traiter les interruptions générées par la couche OCP et router correctement les paquets entre les couches de protocoles. Avec cette solution mono-puce, les ressources peuvent être librement allouées aux nœuds nécessitant des calculs intensifs qui peuvent alors utiliser des dispositifs de calcul parallèles afin d'accélérer le calcul.

Le modèle de programmation distribué sur puce est inspiré de l'approche traditionnelle pour les grands systèmes. Mais il faut aussi prendre en compte les contraintes et les opportunités que permettent les SoCs. Par exemple, la taille maximale de transmission (MTU) doit être adaptée aux ressources mémoire de l'interface MAC, et la longueur maximale d'une écriture en mode burst du réseau sur puce afin de pouvoir envoyer un paquet de données de manière atomique. D'autre part, contrairement au réseau informatique, le réseau sur puce offre un meilleur taux de succès de transmission, et divers services comme le contrôle de flux, l'accusé de transmission fourni par le Network Interface Unit (NIU). Par conséquent, les services correspondants fournis par la couche transport (TCP) peuvent être économisés.

Une autre partie de la thèse porte sur la synthèse de topologie de réseau-sur-puce (PSTRP) pour la sous-branche de parallélisation du flot de conception. Le problème de la synthèse de la topologie du réseau-sur-puce peut se modéliser sous forme de programme linéaire en nombres

entiers. On a étudié deux modèles de communications, le passage de messages et la mémoire partagée. Les résultats montrent que les contraintes d'implémentation, comme la hiérarchie du réseau sur puce, doivent être prises en compte pour obtenir un résultat à la fois mathématiquement optimisé et électroniquement réalisable.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Since the first commercialization of mobile cellular systems in the early 1980's, the wireless communication industry has exhibited a rapid evolution of communication standards from first generation (1G) technology to the forth generation (4G) standard. The first-generation (1G) technology system was introduced in the early 1980s and completed in the early 1990s. 1G wireless used analog technology. The second generation (2G) technology, fielded in the late 1980s and finished in the late 1990s and often referred to as "digital", replaced the 1G technology by using digital signals and digital networks. During the transition from 2G to 3G there exists an interim deployment of 2.5G digital technology with limited data capabilities, such as short messaging services. The third-generation systems was developed in the late 1990s, which extended the voice-only digital from 2G (as enhanced), and allowed simultaneous use of speech and data services and higher data rates. Thus, 3G networks enable network operators to offer users a wider range of more advanced services while achieving greater network capacity through improved spectral efficiency. The successor to the 3G mobile telecommunication technology is the 4th generation (4G) technology that provides voice, data and streamed multimedia to users at even higher data rates, higher Quality of Service (QoS), security and interface with wire-line backbone networks.

Table 1 summarizes the advancement of wireless technology generations in terms of steady growth of data rate and new services requiring high throughput for handling Internet and multimedia content.

Table 1 Supported data rates for each wireless generation

Wireless generation	Data rate	Services
1G	2.4kbps	Voice only
2G	64kbps	Voice, limited data capacity
3G	125kbps ~ 2Mbps	Global roaming, superior voice quality, M-TV, Internet
4G	~ 1Gbps	Enhanced QoS, security, global roaming, wireline Internet backbone interface

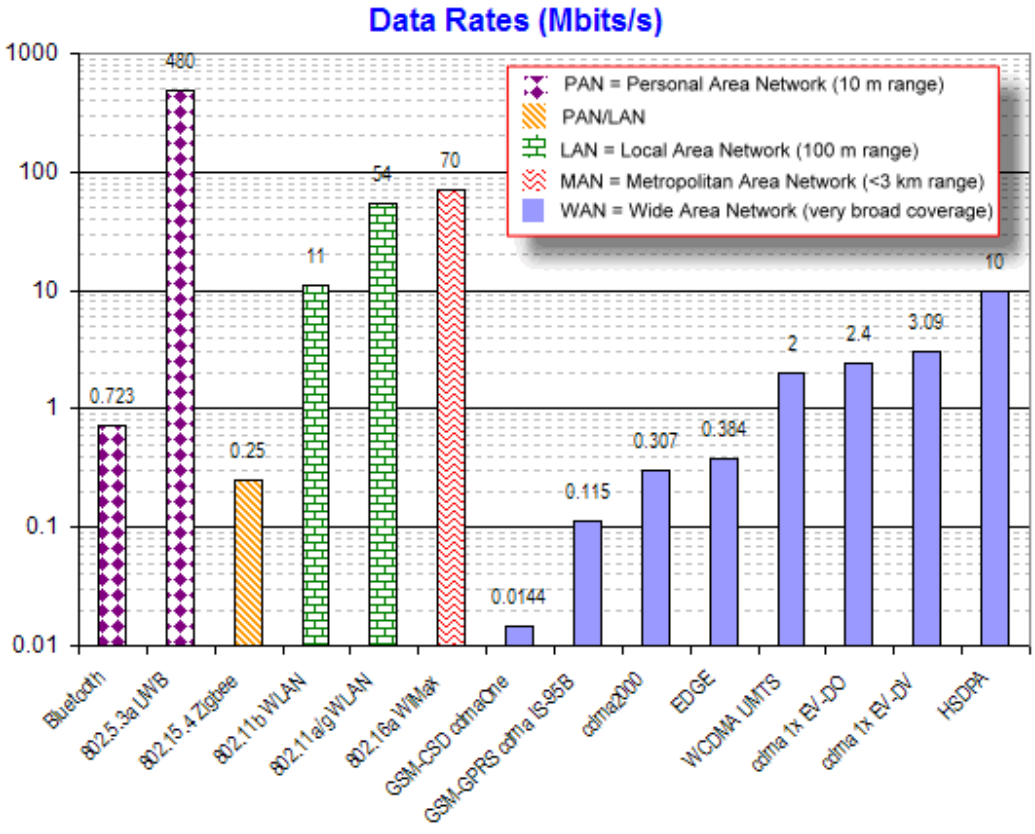

Figure 1 Wireless communication standards and their data rates

More over, in order to support the needs and constraints of various networks, a huge number of standards have appeared that operators are required to support, ranging from the early second generation to all the expected new third generation standards. Figure 1 shows the

different wireless communication standards and their corresponding data rates, while Table 2 gives more detailed information as connection, and modulation methods, for various wireless standards, including WiFi, WLAN, WiMax, WCDMA, GSM, EDGE, and ZigBee.

Table 2 Data rates for various wireless standards

Cellular Family	Standard	Peak Data Rate (kbits/s)	Typical Data Rate	Connection	Modulation
GSM	GSM-CSD	9.6/14.4	9.6	Circuit Switched	GMSK
	HS-CSD	28.8/43.2	28.8	Circuit Switched	GMSK
	GPRS	115/171	50	Packet Switched	GMSK
	EDGE	385/513	115	Packet Switched	8-PSK
UMTS	FDD	384/2000	144	Packet Switched	QPSK
	TDD	384/2000	144	Packet Switched	QPSK
CDMAOne	IS-95A	14.4	14.4	Circuit Switched	QPSK
	IS-95B	65/115	56	Packet Switched	QPSK
CDMA2000	IX	144/307	130	Packet Switched	QPSK
	IX EV	2000	N/K	Packet Switched	QPSK
TDMA	CSD	9.6	9.6	Circuit Switched	π/4 QPSK
PDC	i-mode	9.5	9.6	Packet Switched	π/4 QPSK

(Data obtained from Philips 2002 & 2004 Worldwide Wireless Telecommunication Standards chart)

Since each standard is different, sometimes even using different carrier frequency, specific stations or handsets have to be developed, deployed and maintained, implying very large codes and slow developments. Considering the pace at which new standards are being released, it quickly becomes a nightmare for anybody involved in the communication industry to support them all at an acceptable cost in terms of development time and chip area.

The idea of Software Defined Radio coined by Mitola Joseph III is proposed to cope with such a crisis. In such an approach, the channel modulation waveforms in a radio system are implemented in software instead of hardware with fixed functionality. The software defined

components are deployed on modern programmable/reconfigurable devices like GPP, DSP, ASIP or FPGA. Consequently, a demand of system adaptation to different communication standards or even an update to newer generation technology can be achieved by software update instead of the tedious and time/money consuming hardware replacement. With the improvement of semiconductor technology and availability of wireless technology providing reliable and high data rate Internet access, software updates and system reconfiguration can be done in a real-time manner with configuration data downloaded form air interface. In such a way, a unique device can be made compatible with a whole set of standards, for example ZigBee, Bluetooth, 802.11a/b/g/n, 3G, etc., and handovers between different protocols can be done without degradation with careful design.

Whether the adoption of software defined radio is beneficial, however, depends on two factors:

1. Software reusability, portability and interoperability.
2. Hardware platform and programming model support to achieve the performance requirements while keeping the programming difficulty at a reasonable level.

The first factor is intrinsically important which determines the usability of the SDR idea as a whole. The benefits of flexibility, which is the main idea represented by SDR, is only achievable if one can freely add, update, or enhance functional capabilities of the radio system having been realized in form of software modules. Ideally, waveform applications designed for one SDR platform can be easily ported to another platform; waveform applications developed by one enterprise can be interoperable with the waveform applications of another company. In order to achieve this goal, an open standardized framework is necessary which provides uniform definitions of interfaces, and services an application should conform to.

The Software Communication Architecture (SCA) is a largely accepted open architecture for SDR programs. It is developed by the US Department of Defense (DoD) for the development a family of affordable, high-capacity tactical radio systems that can provide scalable, interoperable wireless mobile network services. The SCA specification defines an Operating

Environment (OE) comprising of a Core Framework, a minimum CORBA compliant middleware and a POSIX compliant operating system in which waveform applications are executed. The POSIX standard minimizes the cost of porting waveform software because it provides an abstraction layer for operating system-specific methods. CORBA provides a level of transparency and program-language independence. The developments and programming of SCA compatible SDRs are the problems treated in this thesis.

On the other hand, many of the new high-bandwidth waveforms demand processing power and I/O bandwidth that exceeds that provided by traditional single processor systems combined by certain hardware IP. New SDR platforms are most likely deployed on multi- and many- core systems (MPSoC) leveraging its rich processing resources with energy efficiency. Systems exists which incorporate dozens, hundreds or even thousands of cores. [1][2][3][4][5] Rapid development, deployment and verification of parallel embedded software in these emerging MPSOC is key issue to ensure performance requirements under strong time to market (TTM) and development cost constraints.

The deployment of SCA based SDR and the modern MPSoC platform entails the combination of two programming paradigm: CORBA based distributed model, and SMP based parallel model. Standard based SDR design lacks explicit design flow and efficient programming model for leveraging the rich processing resources that an MPSoC platform provides in a systematic manner. In this thesis, we propose a SDR design flow with systematic architecture exploration and optimization based on a hybrid programming model (distributed client/server and parallel).

We are interested in the baseband processing part of the radio system. The radio baseband functions are represented in a Kahn Process Network. A distributed system with no resource constraints is generated with a first level distributed system generation engine. The resulted nodes are profiled and classified in order to determine the ones that have a high requirement of processing performance and are passed to a sub-branch to be parallelized. At the output of the flow, a global feedback path is provided to optimize resource utilization, and frequency scaling while meeting system performance requirement.

We built an embedded distributed system based on multiple FPGA cards as a proof-of-concept platform. The cards are connected via an Ethernet switch. Each card contains a PowerPC405 based computing system running Linux kernel with a TCP/IP stack. The communicating applications are realized by Common Object Request Broker Architecture (CORBA) middleware conforming to the SCA specification. The performance of the middleware is tested with micro-benchmarks. Frequency scaling effects on the overall system performance is examined on a participant-by-participant basis (Client, Server, or Common Service). The results give good clues for frequency configuration with the goal of minimizing consumption.

The parallelization sub-branch is based on an automatic parallelizer and a chain of parallel library transformation/customization tool and FPGA design tools. We studied the potential of automatic parallelization on a NoC-based 16 PE multi-core system which we designed and implemented on a single FPGA. We addressed three issues in the framework of NOC based MPSOC with actual hardware: (1) an efficient hardware support for synchronization primitives (2) the performance of automatic parallelization (3) the multiprogramming benefits.

The execution results of several parallelized code show us several key elements that influence the effectiveness of parallelization. Some of these elements are intrinsic in the application, while others are architecturally dependant. A comprehensive understanding of the characteristics of both the application and the architecture accompanied by an optimum combination of the two is necessary for a satisfying performance.

The Network Interface Unit (NIU) of the MPSoC is based on the Open Core Protocol (OCP) standard. The OCP protocol is an openly licensed, core-centric protocol intended to contemporary system level integration challenges. It provides a common standard for intellectual property (IP) core integration in a “plug and play” manner. The protocol is based on the master-slave point-to-point model. We focus on the two synchronization mechanisms provided by the protocol, namely: the Exclusive Synchronization and the Lazy

Synchronization. The results show the superiority of the blocked mechanism in the dedicated synchronization NOC with BRAM over LL-SC with BRAM or blocked with DDR in a single-lock case.

Single-application performance results show an under-exploited MPSoC platform lack of sufficient parallelizability. We are naturally led to the multiprogramming solution where processors resources are shared by multiple applications. We notice that 8 processors is usually a critical number beyond which the performance stops scaling linearly. A combination of applications and an efficient allocation of processor resources can effectively improve the overall performance.

The ultimate objective is to move everything on a single chip to provide a SCA compliant single-chip SDR baseband. The Ethernet switch based hybrid platform while serving as fast and pertinent proof-of-concept has limits like network bandwidth and configuration flexibility due to the isolation of nodes. Single chip design of SCA compliant SDR platform involves efforts as the mapping of transmission mechanism of CORBA from the GIOP/IIOP to GIOP + proprietary-on-chip-communication in order to fully leverage the interoperability and portability of CORBA based applications. In our case, we will leverage the Network-on-chip (NoC) Danube library of Arteris [65] for the interconnection of multiple processing elements, memory resources and IP integration via a standard interface, the Open Core Protocol (OCP). By default, the communication mechanism of CORBA is TCP/IP. We still use TCP/IP as the transport layer and internet layer protocol of COBRA. However, we modify the MAC layer by replacing Ethernet by OCP/NTTP. NTTP is the proprietary packet transport protocol implemented in the Danube NoC. An example architecture is shown in Figure 2 that is composed of two servers and two clients whose communication is based on a version of CORBA adjusted to OCP network. With this configuration, the communication layer of CORBA can remain largely unchanged and an OCP device driver should be registered in the Linux kernel in order to handle interruptions generated by the OCP layer and route the packet properly among the protocol layers. Extra resources can be flexibly allocated to nodes in charge of processing computing-intensive algorithms by synthesizing an array of parallel processing elements to assist the computation, as is shown in upper-right corner of Figure 2.

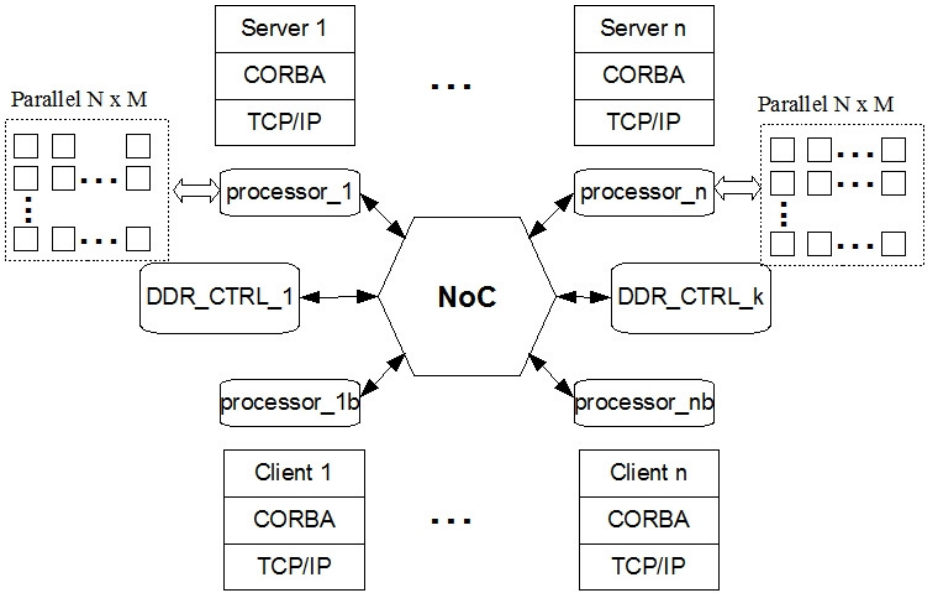

Figure 2 Single chip distributed system based on CORBA & NoC

Chapter 2

Software Defined Radio and Cognitive Radio State of the art

2.1 SDR Definition

The term “Software Defined Radio” was coined in 1991 by Joseph Mitola in his publication [23]. A Software Defined Radio (SDR) is a radio system, where components are implemented using software instead of hardware.

By realizing the main components in software, a SDR offers support for multiple standards, multiple bands, and seamless mode/band transitions by software update rather than hardware alternation. This greatly reduces the development and deployment cost of radio systems with the ever developing radio communications standards. SDR also have significant utility for the military area and cell phone services, both of which must serve a wide variety of changing radio protocols in real time.

There are several other important concepts that are closely related to SDR which should first be clarified, namely Digital Radio (DR), Software Radio (SR), and Cognitive Radio (CR). By the term Software Radio we refer to a transceiver whose functions are realized as programs running on a suitable processor. An SR transceiver comprises all the layers of a communication system. An ideal SR directly samples the antenna. Digital Radio is a radio system whose baseband signal processing functions are implemented on a Digital Signal Processor (DSP). A Software Defined Radio (SDR) is a presently realizable version of SR: instead of sampling directly antenna output, the received signals are sampled after a suitable

band selection filter. A Cognitive Radio (CR) [1] combines an SR with a Personal Digital Assistant (PDA) and connects its owner to Intelligent Networks (INs). [21]

2.2 SCA Specification

The Software Communication Architecture (SCA) [24] is an open architecture framework developed under the requirement of US Department of Defense (DoD) to maximize portability, interoperability, and configurability of the Software Defined Radio.

The SCA specifies an Operating Environment (OE) in which waveform applications are executed. In the context of SCA, a waveform is defined as the entire set of radio and/or communications functions that occur from the user input to the radio frequency output and vice versa.

The Operating Environment is composed of a Core Framework (CF), a minimum CORBA compliant middleware and a POSIX compliant Operating System (OS). [24] The OS running the SCA must provide services and interfaces that are defined as mandatory in the Application Environment Profile (AEP) of the SCA. Figure 3 depicts the main building blocks and the hierarchy of the SCA. [25]

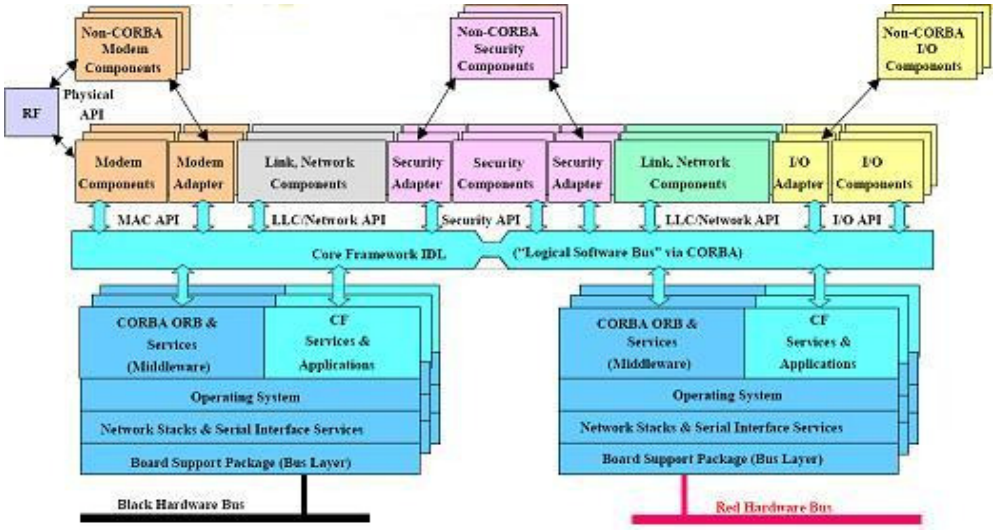

Figure 3 Software Communication Architecture

The Core Framework is a set of open application-layer interfaces and services which provide an abstraction of the underlying system software and hardware for software application designers. [21] The CF consists of four parts:

- Base Application Interfaces: provide the management and control interfaces for all system software components. The interfaces in this group are: Port, LifeCycle, TestableObject, PropertySet, PortSupplier, ResourceFactory and Resource.

- Base Device Interfaces: realize the management and control of hardware devices within the system through their software interface. The interfaces in this group are: Device, LoadableDevice, ExecutableDevice, and AggregateDevice.
- Framework Control Interfaces: control the instantiation, management, and destruction/removal of software from the system. The interfaces in this group are: Application, ApplicationFactory, DomainManager, and DeviceManager.

- Framework Services Interfaces: provide additional support functions and services such as file system management. The interfaces in this group are: File, FileSystem, and FileManager.

In Figure 4, the SCA core framework components and their interfaces are represented in UML form.

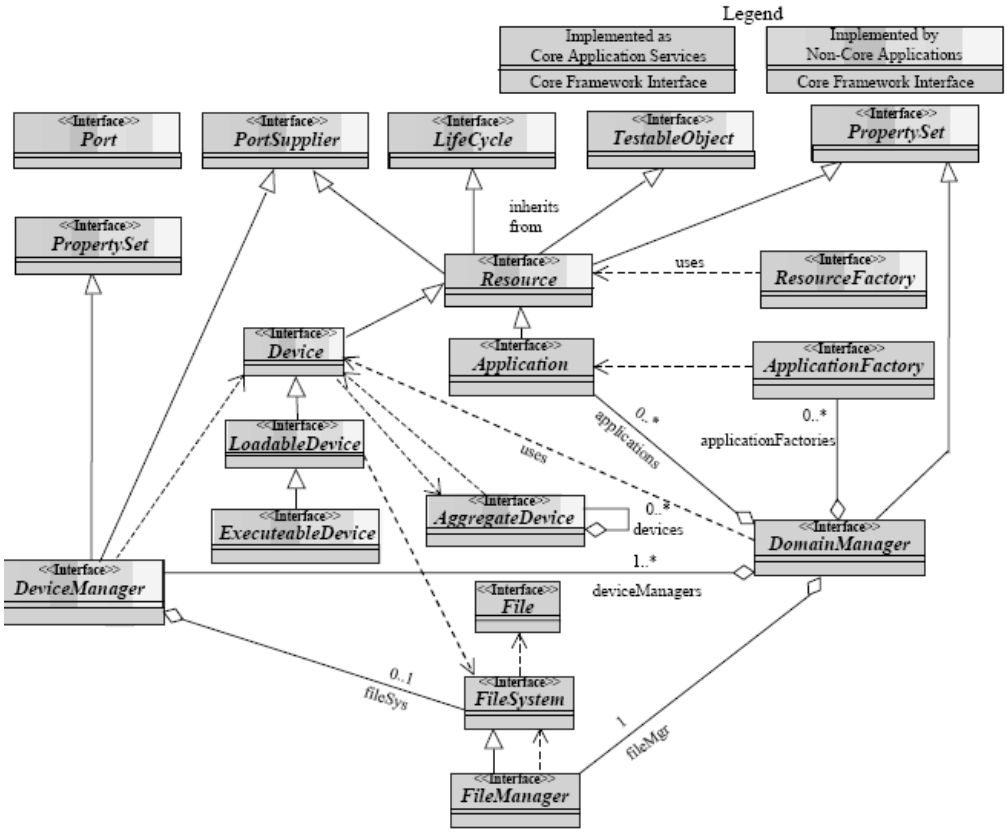

Figure 4 SCA Core framework in UML

The CF uses a Domain Profile to describe the components in the system. The software component characteristics are contained in the Software Package Descriptor (SPD), Software Component Descriptor (SCD) and Software Assembly Descriptor (SAD).The hardware device characteristics are stored in the Device Package Descriptor (DPD) and Device Configuration Descriptor (DCD). The Properties Descriptor contains information about the properties of a hardware device or software component. The Profile Descripter contains an absolute file name for either a Device Configuration Descriptor, or a Software Package Descriptor or a Software Assembly Descriptor. Finally, the DomainManager Configuration Descriptor (DMD) contains the configuration information for the DomainManager.

The reconfiguration of radio usually concerns the installation/uninstallation of SCA applications as well as connection/disconnection of ports. These operations are accomplished by a series of function calls to the SCA Core Framework.

The following scenario depicts the steps and interfaces utilized when a client application tries to install a new application within a certain system domain. It needs to invoke the create operation provided by the ApplicationFactory interface. ApplicationFactory then refers to the Domain Profile for available devices that meet the application's memory and processor requirements, available dependant applications, and dependant libraries needed by the application. If the requirements are met, an Application instance is created and the memory and processor are allocated. The application software module is then loaded on the devices using the appropriate Device interface. Then connect the resources' ports. Finally the Application object reference in the context of CORBA Naming Service is returned.

Distributed processing is a fundamental aspect of SCA and OMG CORBA [26] is used as the middleware that provides the standardized message passing technique in a client/server model. Using CORBA allows software objects to communicate with each other through a standardized interface description language (IDL). CORBA is designed to be both language and platform independent, which simplifies the development and deployment of communication software. All CF interfaces are defined in Interface Definition Language (IDL). The CORBA handles the message marshalling and delivering.

2.3 CR Definition and theoretical issues

Cognitive radio is a paradigm for wireless communication in which either a network or a wireless node changes its transmission or reception parameters to communicate efficiently avoiding interference with licensed or unlicensed users. This alteration of parameters is based on the active monitoring of several factors in the external and internal radio environment, such as radio frequency spectrum, user behavior and network state. The term "Cognitive Radio (CR)" was coined by Joseph Mitola III in October 1998 to represent the integration of substantial computational intelligence – particularly machine learning, vision, and natural language processing – into software defined radio (SDR). [33] CR embeds a RF-domain intelligent agent as a radio and information access proxy for the user, making a myriad of detailed radio use decisions on behalf of the user (not necessarily of the network) to use the radio spectrum more effectively.

Although cognitive radio was initially thought of as a software-defined radio extension, most of the research work is currently focusing on Spectrum Sensing Cognitive Radio, particularly in the TV bands. The main problem of Spectrum Sensing Cognitive Radio is in designing high quality spectrum sensing devices and algorithms for exchanging spectrum sensing data between nodes. It has been shown that a simple energy detector cannot guarantee the accurate detection of signal presence, calling for more sophisticated spectrum sensing techniques and requiring information about spectrum sensing to be exchanged between nodes regularly. Increasing the number of cooperating sensing nodes decreases the probability of false detection. [144] Filling free radio frequency bands adaptively using OFDMA is a possible approach. Applications of Spectrum Sensing Cognitive Radio include emergency networks and WLAN higher throughput and transmission distance extensions.

2.4 Academic SCA based SDR (OSSIE and SCARI)

1. SCARI (CRC 2004)

The SCARI-OPEN is an implementation of the JTRS Software Communication Architecture SCAv2.2 and certified by the JTRS-JPO. The project is carried out at the Canada's Communications Research Center (CRC) and was launched in 2001 under a contract between CRC and SDR Forum to develop a reference implementation (RI) aiming at: [28]

- Reduce the level of ambiguity of the SCA specification documents
- Increase the potential for interoperability by allowing implementers to customize the RI instead of rewriting the whole architecture
- Increase understanding of the architecture through an example
- Accelerate the emergence of SDRs through the availability of an implementation
- Reduce the cost and time-to-market for SDRs

SCARI-OPEN is an open source implementation written in Java. The RI provides the mandatory components of the SCA core framework, along with support for the most used features, including Service Interfaces, Core Framework with the XML Domain Profile,

related tools to operate the radio and simple waveform applications to demonstrate the operation of radio.

The SCARI++ core framework is a new generation core framework of CRC implemented in C++. It supports an exceptional number of operating environments. Some of them are especially designed for real-time embedded systems.

2. OSSIE

OSSIE, acronym for Open Source SCA Implementation::Embedded, is an open source SCA-based core framework and rapid development tool for SDR developed at Virginia Tech. [27] Its latest version 0.8.0 was released in January, 2010.

OSSIE is targeted for use in wireless communications curricula and research efforts. OSSIE includes a core framework as well as a suite of graphical user interface-oriented tools. The tools are capable of auto-generation of SCA-specific component source codes and supporting files, leaving the developer the task to specify the signal processing functionalities.

2.5 Commercial SCA based SDR (Spectrum Signal)

Spectrum SDR-4000 [29]

SDR-4000 is a SDR small form factor transceiver platform development by Spectrum. SDR-4000 offers the commercial off the shelf (COTS) hardware, software and services to accelerate the development and deployment of black-side wireless modem solutions for tactical military communications system. The SDR-4000 consists of two major component level products: the PRO-4600 SDR modem processing engine and the XMC-3321 dual transceiver I/O mezzanine card. The two components together provide a wireless modem that supports up to two channels per slot. Figure 5 illustrates the two components and an example of data flow.

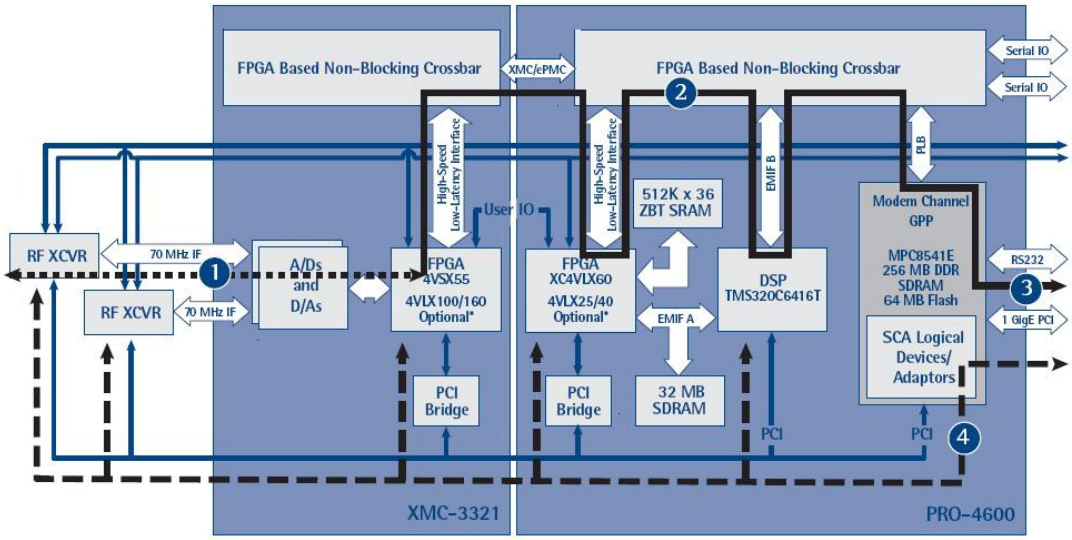

Figure 5 PRO-4600/XMC-3321 example of data flow

The PRO-4600 component employs a combination of heterogeneous processors and FPGA: Xilinx Virtex-4 FPGA, TMS320C6416T DSP and MPC8541E GPP, which fulfills the size, weight and power-limited requirements of SDR applications.

The XMC-3321 is dual channel transceiver module optimized to operate with the PRO-4600 for SDR applications. The XMC-3321 supports 10.6, 21.4 and 70 MHz IF frequencies through the use of dual 14-bit A/D converters sampling at up to 105 MSPS and dual 14-bit D/A converters sampling at up to 300 MSPS.

Figure 6 shows the standards-based software operating environment of the SDR-4000 platform. It supports real-time operating systems such as Integrity of Green Hills or Wind River VxWorks. The SCARI Core Framework of CRC is supported by the SDR-4000, which maximizes the real-time performance of embedded platforms by providing a full implementation of all the SCA Core Framework interfaces and implementing exceptional features that minimize the boot time of an SCA system.

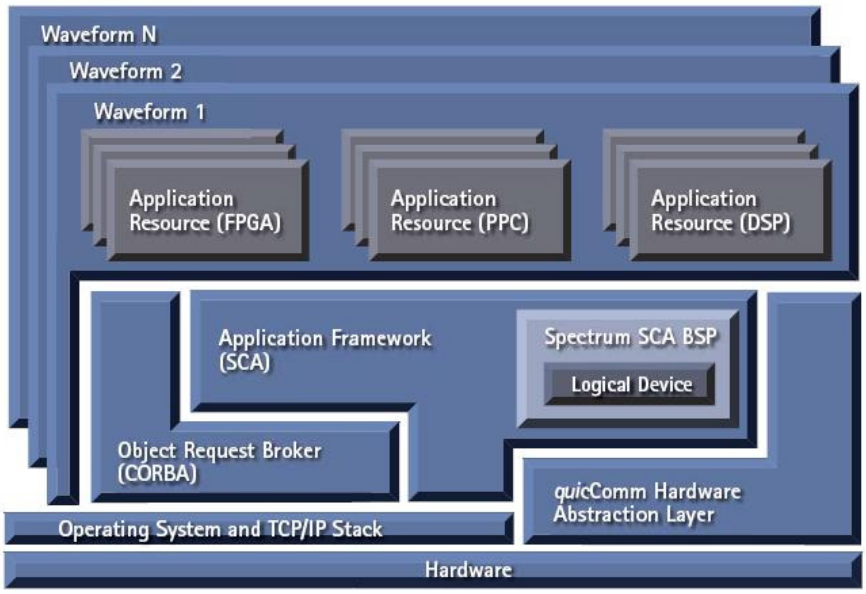

Figure 6 Software Operating Environment

2.6 Other efforts in SDR implementations

1. IDROMel (ANR project, France, 2009)

The project IDROMel [30] is a French National Agency for Research (ANR) project aiming at defining, developing, and validating a reconfigurable SDR and Cognitive Radio platform. The platform combines the latest technologies, such as:

- Flexible baseband processing
- Network on Chip based integration
- FPGA partial reconfiguration support
- Very wide band RF from 200 MHz to 7.5 GHz agility
- 4 X 4 MIMO support
- Flexible MAC design for vertical handover support

The platform permits various SDR or CR scenarios like vertical handovers in a heterogeneous network including multiple Radio Access Technologies (RATs) (with different QoS parameters, frequency bands and bandwidths). The selected RATs are UMTS and WiMax. The baseband processing part of the platform features a hierarchical heterogeneous

architecture. The implementation is based on two FPGAs. The first FPGA, Virtex-5LXT110 from Xilinx, implements a 32-bit microcontroller that is responsible for communications with the host-PC via a PCI-express link and for the global control of the second FPGA. The second FPGA, which is a Xilinx Vertex-5LX330 board, consists of 7 DSP blocks interconnected by a crossbar responsible for various signal processing functions and interface with the RF front end, as shown in Figure 7. The main characteristics are summarized in Table 3.

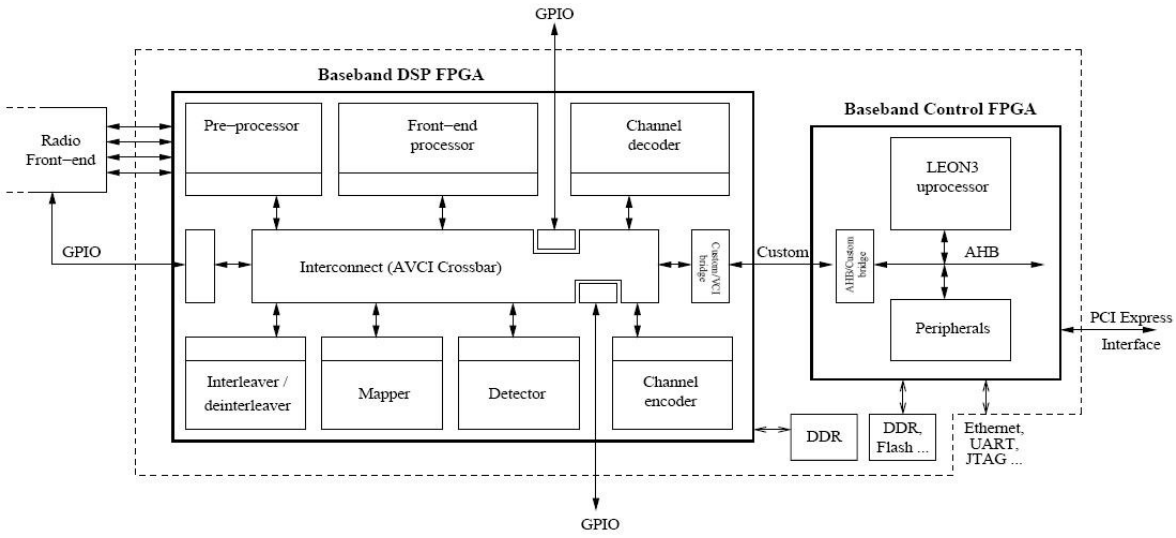

Figure 7 IDROMel baseband architecture

Table 3 IDROMel summary

Standards	UMTS, WiMAX
Technology	Xilinx Virtex-5110LXT control; Virtex-5 330LX processing
IP Core	DFT, Generic modulator BPSK to QAM256, generic channel coder (conventional, cyclic, M-sequence), generic channel decoder (Viterbi, turbo), generic interleaver/deinterleaver
Processor	8 bit uC for each IP Core

Partial reconfiguration of FPGA is a new feature that is capable of extending SDR perspectives by bringing the highest flexibility to the hardware level. The results show an interesting reconfiguration overhead which is as little as 700 µs/Partial Reconfiguration.

2. WiNC2R (WINLAB, Rutgers University, 2008)

In [31], a prototype of a Cognitive Radio hardware platform – the WiNC2R is described. The platform is based on the FPGA technologies featuring rich logic resources. The flexible processing elements provide the designers a large exploration space to find the best performance/power/area tradeoff. The architecture is composed of three parts, the RF module, the baseband module and the networking module. While the RF module is mainly composed of analog circuits, the baseband module and the networking module are all implemented with FPGAs. The baseband module is implemented in Xilinx Virtex-4SX series of FPGA, which features rich DSP resources and is geared towards high-performance digital signal processing applications. The network module is implemented in the Xilinx Virtex-4FX series of FPGA, which is targeted for embedded control intensive applications. DMA engines and hardware accelerators are used to accelerate some computation-intensive PHY layer functions, like FFT, Viterbi decoding, ECC, etc, which are dynamically configurable on a per-packet basis to cover multiple standards. Figure 8 shows the baseband and network module of the WiNC2R while Table 4 lists the main baseband characteristics.

Table 4 WiNC2R baseband summary

Standard	OFDM, QPSK/DSSS
Technology	Xilinx Virtex-4SX35, Virtex-4FX12
IP Core	ECC, FFT, Viterbi decoding, Reed-Solomon (RS) encoding & decoding
Processor	Data Processor (DP) (MAC & higher layer), Cognitive processor (CP)

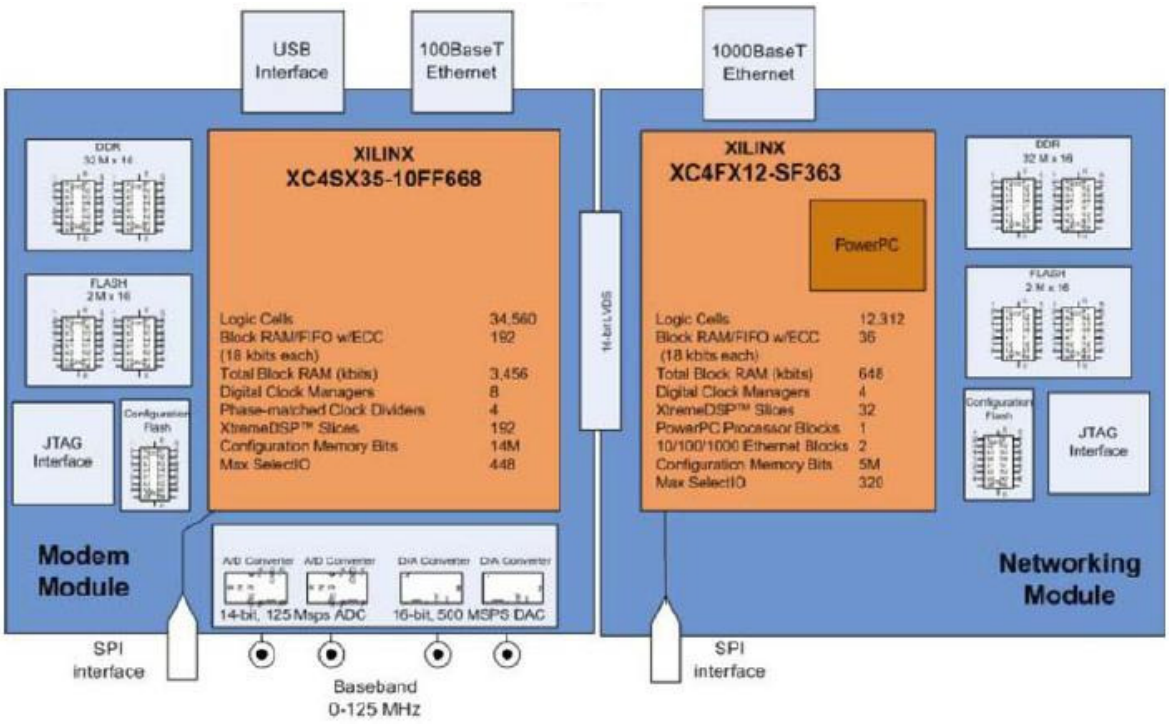

Figure 8 Baseband and network modules

3. Annabelle (University of Twente, Netherlands, 2007)

Annabelle [32] is a multiprocessor system-on-chip (MPSoC) cognitive radio platform. It focuses on the baseband processing aspects of the Cognitive Radio. A Cognitive Radio needs an adaptive physical layer that must be supported by a reconfigurable baseband processing platform. As illustrated in Figure 9, the Annabelle architecture is an ARM core based heterogeneous MPSoC. The baseband processing functions and reconfiguration are carried out on an array of Montium reconfigurable DSP processors interconnected via a circuit-switched Network-on-chip (NoC). A summary of the main characteristics of Annabelle platform is given in Table 5.

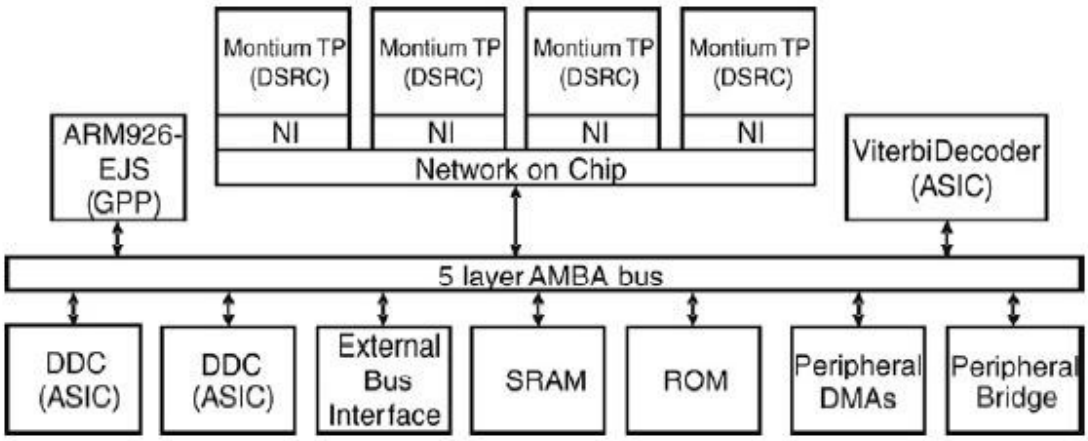

Figure 9 Block diagram of Annabelle base band

Table 5 Annabelle baseband summary

Standard	OFDM
Technology	ATMEL 130nm process
IP Core	Viterbi decoder (ASIC)
Processor	ARM926-EJS, Montium DSP (sparse FFT, filter bank, DCFD)

4. SDR LSI (Fujitsu 2006)

SDR LSI is a single-chip solution for SDR baseband. It is developed for programmable wireless communications systems. As shown in Figure 10, SDR LSI features a hybrid architecture consisting of reconfigurable signal processors and parametric accelerator circuits for baseband processing. The reconfigurable signal processors (RSPs) are arranged in cluster structure that improves the mapping efficiency and minimizes the processing time. The main characteristics and supported standards of SDR LSI are summarized in Table 6.

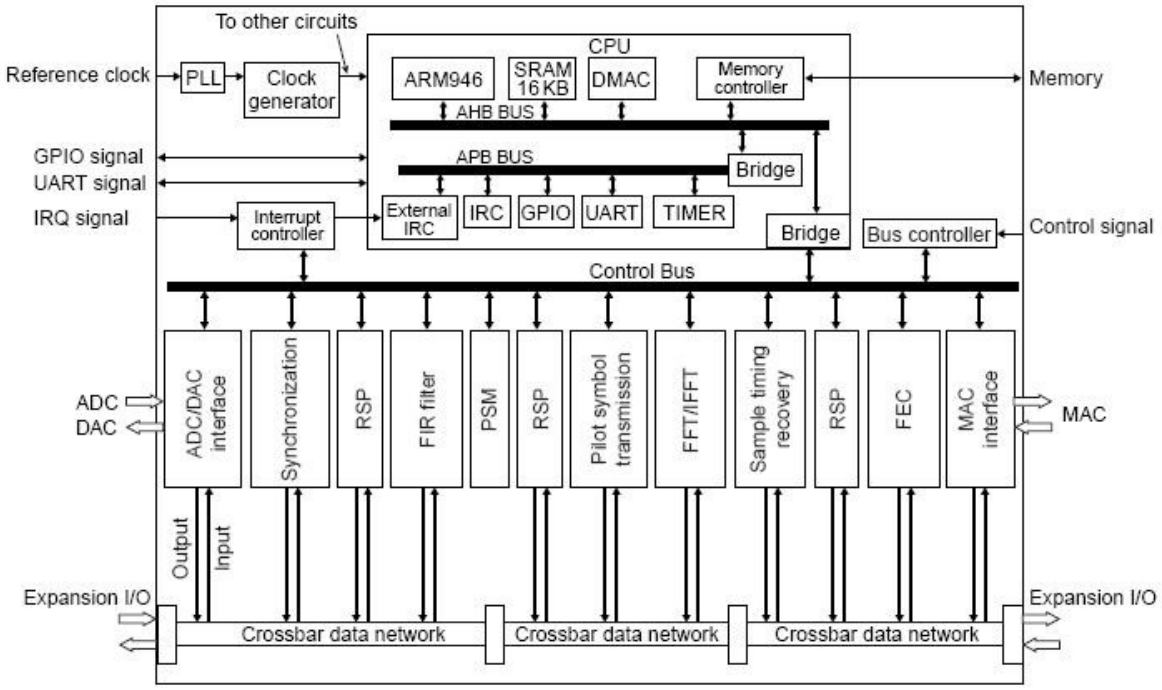

Figure 10 SDR LSI architecture

Table 6 SDR LSI baseband summary

Standard	802.11a, b
IP Core	FFT, Viterbi decoder, programmable flip-flop array (Scrambler/descrambler, CRC, Convolution encoder), FIR
Processor	ARM946, RSP (reconfigurable signal processor)

2.7 Academic CR major projects and achievements

Cognitive radio technologies have been proposed in order to identify and exploit unused spectrum while controlling the interference caused to licensed users. Local regulatory authority around the world license frequency bands to primary users (PU). However, primary users typically leave parts of their allocated spectrum underutilized. According to extensive measurement campaigns, radio resources are utilized from 15 percent to 85 percent depending

on location, frequency band and time of day [126]. This allows opportunistic communication by exploiting unoccupied frequency bands.

Authors in [125] present an FPGA implementation of a feature detector for OFDM-based primary user signals. The paper compares different spectrum sensing techniques that have been proposed and chooses an autocorrelation based OFDM signal detection algorithm due to its performance despite a relatively more complex implementation. Implementation is realized on a Xilinx Virtex-5 FPGA. Simulations with Matlab and Modelsim indicate that the detector works well in above SNR of -5dB.

The opportunistic radio (OR) is a narrower definition of Cognitive radio where the environmental awareness is limited to the spectrum knowledge. The study in [127] proposes an OR decision making framework including the flow of context information as an input process to the decision making engine, the context filtering and the reasoning mechanisms in which the decision optimization is achieved using a genetic algorithm (GA)-based approach. The experimental study is performed on the Ettus USRP (Universal Software Radio Peripheral) hardware and the GNU Radio open source software. The test results show the OR ability to perform spectrum sensing in the 2.4GHz ISM band and provide evidence that the proposed framework enables the OR terminal to detect spectrum opportunity and provide the best solution for a suitable channel allocation.

Authors in [128] describes a hardware demonstrator of an OR system detecting and using temporal opportunities. They present an exclusive implementation of a cyclostationarity sensing algorithm, and propose a low complexity decision-making algorithm, which performs real-time regulation of the OR communications. The demonstrator operates in the 2.4GHz band and is validated by sharing the spectrum resource with a standard IEEE 802.11g primary system (PS) running a video streaming application without perceptible impact of the OR system.

2.8 Conclusion

This chapter introduces the notion of Software Defined Radio (SDR). The advantages introduced the SDR and its implementation challenges are discussed. We are especially interested in the group of SDR that conforms to the Software Communication Architecture (SCA). SDRs that are compatible with the SCA open framework maximize the portability, reusability and interoperability of its waveform applications that are desirable features under current context of rapid advancement of communication standards and hardware platform. The definition of Cognitive Radio (CR) extends the Software Defined Radio by the integration of substantial computation intelligence - particularly machine learning, vision, and natural language processing.

Then the academic and industrial efforts in the development of Software Defined Radio and Cognitive Radio are summarized. In the case of Software Defined Radio, many platforms and software tool kits have been developed for the fast prototyping, test and verification of a SDR system, such as the SDR4000, and SCARI. Some focus on the architectural design of the baseband processing part based on modern multiprocessor system on chip (MPSoC), such as Annabelle and SDR LSI. Other platforms leverage the flexibility provided by modern FPGAs to realize dynamic reconfiguration aspect of the SDR to conform to different communication standards without perceptible performance degradation during protocol handoff, like IDROMel. When it comes to the Cognitive Radio, researches focus more on a narrower definition by confining the environmental awareness of a Cognitive Radio to the spectrum knowledge. Algorithms for spectrum sensing and decision making are proposed and hardware platforms are also developed as a proof-of-concept.

As mentioned in the introduction, we are more interested in the baseband processing aspects of the Software Defined Radio system. In all the mentioned works, no one ever proposed a SDR design flow for systematic and automatic system generation, and the programming paradigm faced by the SDR community is also of its own specificities. In our work, we propose a SDR design flow with systematic architecture exploration and optimization based on a hybrid programming model (distributed client/server + parallel).

Chapter 3

Embedded System Implementation and Optimization for SDR

This chapter first introduces the state of the art the MPSoC and FPGA design flows. Then an automatic parallelizer based automatic MPSoC design flow is proposed. The automatic parallelizer tool PLuTo is described. A NoC based multiprocessor architecture is designed and implemented. Some performance analyses were carried out on this platform to evaluate the design flow. The synchronization performance the OCP (Open Core Protocol) is also studied.

3.1 MPSoC and FPGA Design Flow

In [124], the authors propose a design methodology to generate and program MPSoC designs in a systematic and automated way for multiple applications. The architecture is automatically inferred from the application specifications, and customized for it. The flow is illustrated in Figure 11. The applications are described in the form of Synchronous Data Flow (SDF) graphs, which are used to generate the hardware topology. The software project for each core is produced to model the applications behavior. The project files specific to the target architecture are also produced to link the software and hardware topology. The final MPSoC platform is then generated.

Article [96] leverages the performance and energy efficiency provided by the single-chip heterogeneous multiprocessors, where different processors are customized for the tasks they perform. However, a primary bottleneck is the development of programming paradigms and tools to alleviate the design complexity. The authors proposed a multilevel custom multiprocessor-synthesis methodology to perform the assignment and scheduling of the

applications tasks on the various processors together with the processor customization in an integrated manner. The author makes use of the Application Specific Instruction set Processor (ASIP) technology to customize instruction set depending on specific task characteristics. The ASIP technology customizes the instruction set according to specific application characteristics which can give a more energy efficient implementation for the same performance level.

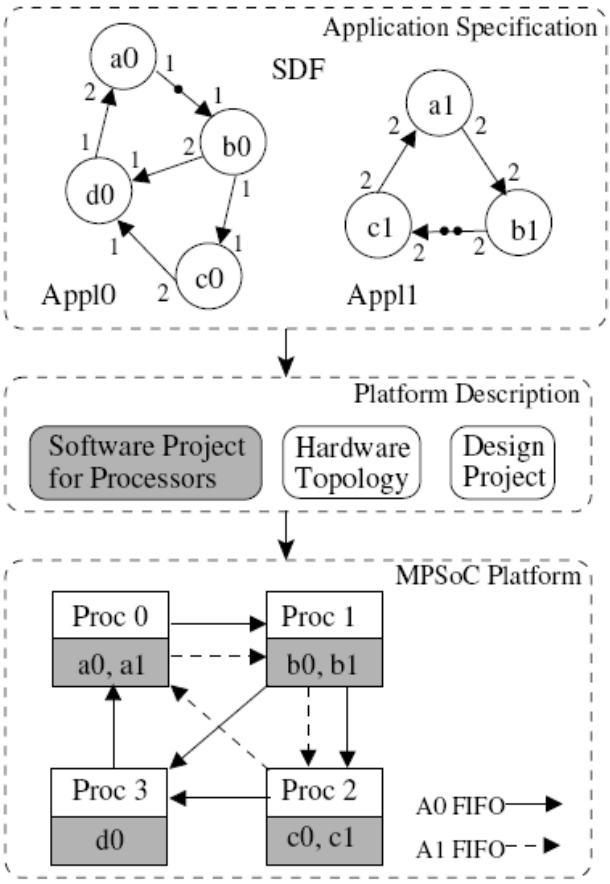

Figure 11 Multiprocessor Synthesis Design flow [124]

The heterogeneous multiprocessor-synthesis problem is abstracted by the author as: A task graph is composed of n tasks and each task t_i has m_i custom-instruction versions. $Cycle_{ij}$ corresponds to the execution cycle of an instruction version of a task and $Area_{ij}$ corresponds to the area consumed ($1 \leq i \leq n$, $1 \leq j \leq m_i$). Given p initially homogeneous processors in a multiprocessor system, and a total area budget AB for all custom instructions, assign and schedule the tasks on these processors with a set of custom instructions such that the total

execution time of the task graph is minimized while the total area of all the custom instructions is within AB. Figure 12 illustrates the overall design flow. A task graph is generated from the application. Each task is profiled and a performance-area tradeoff curve is generated in for different levels of instruction set customization. More custom instructions are added, more performance will be got in sacrifice of chip area. Then the task graphs are scheduled. The tasks that appear on the critical path are candidates for more instruction set customizations while others can be relaxed to save silicon resources. Then the tasks are re-scheduled and eventually a new critical path appears. This operation is repeated until a satisfying performance-area trade-off is achieved.

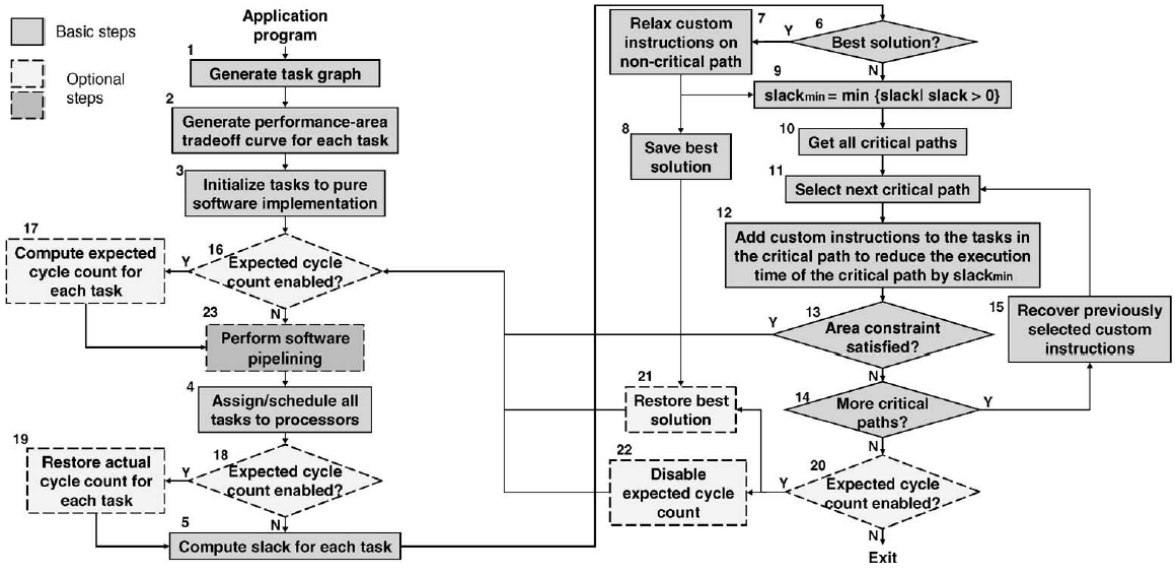

Figure 12 Synthesis methodology for heterogeneous multiprocessors [96]

The ever-increasing complexity of applications and platforms makes the tradition RTL level approach of SoC design error-prone and time-consuming and thus impractical. Authors in [98] argue the importance of high level of abstraction in the SoC design in order to tackle this problem. Moving up to higher levels of abstraction opens a gap that the authors name the Implementation Gap. Tools are needed to close this gap in a systematic and automated way. [98] The paper proposes a methodology and techniques implemented in a tool called ESPAM (Embedded System-level Platform Synthesis and Application Mapping) for automated multiprocessor system design and implementation, as illustrated in Figure 13.

The flow is composed of three levels of specification: System Level, RTL level and Gate Level. The System-level specification is given as input to ESPAM. First, ESPAM constructs a platform instance following the platform specification. Second, ESPAM refines the abstract platform model to an elaborate parameterized RTL model ready for implementation. Finally, ESPAM generates the program code for each processor in the multiprocessor system in accordance with the application and mapping specifications.

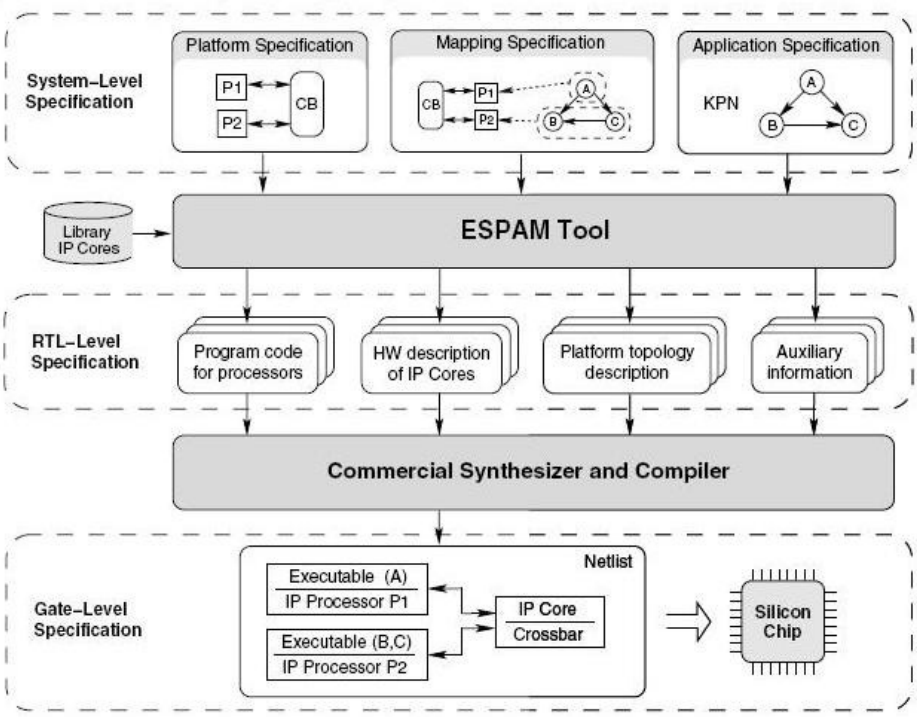

Figure 13 ESPAM system design flow

With the increasing design complexity, the gap between ESL (Electronic System Level) design to RTL synthesis becomes more crucial for industrial projects. In [138], the authors present a SystemC-based ESL tool, SystemCoDesigner, to carry out automatic multi-objective optimization for a hardware/software SoC implementation. This tool combines the behavioral synthesis with automatic software generation. The design flow using SystemCoDesigner is illustrated in Figure 14. Starting from a SystemC behavioral model, SystemCoDesigner automatically extracts the mathematical model, performs behavioral synthesis step, and explores the multi-objective design space. During the design space exploration, a single point

is evaluated by simulating highly accurate performance models, which are automatically generated from the SystemC behavioral model and the behavioral synthesis results. SystemCoDesigner then automatically generates the bit stream for FPGA targets from any previously optimized implementation.

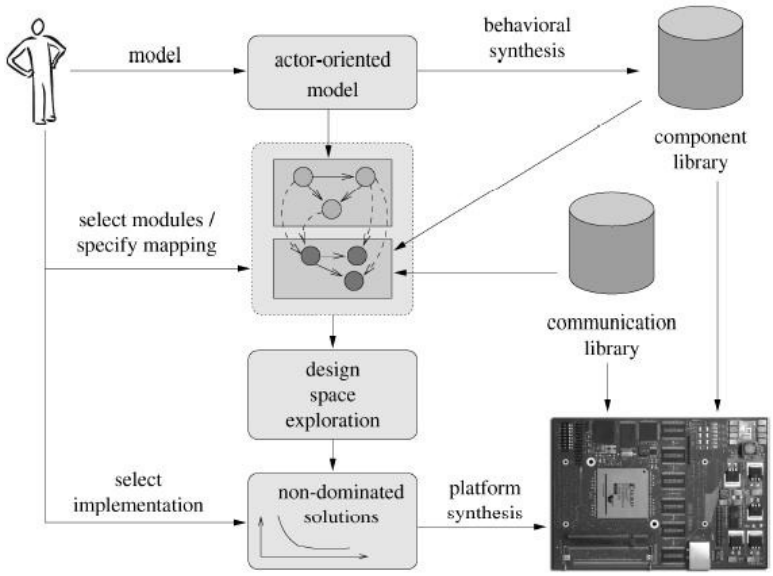

Figure 14 ESL Design Flow using SystemCoDesigner

In [43], authors present the Daedalus system-level design flow for the design of MPSoC based embedded multimedia systems. The design flow is shown in Figure 15. It offers a fully integrated tool-flow in which design space exploration, system-level synthesis, application mapping, and system prototyping of MPSoCs are highly automated. The Daedalus aims at composable MPSoC design in which MPSoCs are strictly composed of IP library components including a variety of programmable and dedicated processors, memories, and interconnects. The input to the flow is a sequential multimedia application specification in C. The KPNgen tool automatically converts the sequential application into a parallel Kahn Process Network (KPN) specification which is subsequently used by the Sesame modeling and simulation environment to perform the system-level architectural design space exploration. The resulting system designs are then passed to the ESPAM tool to generate synthesizable VHDL that implements the candidate MPSoC platform architecture. In addition, C codes are generated at this step for the applications processes that are mapped onto programmable cores. Using

commercial synthesis tools and compilers, this implementation can be readily mapped onto an FPGA for prototyping.

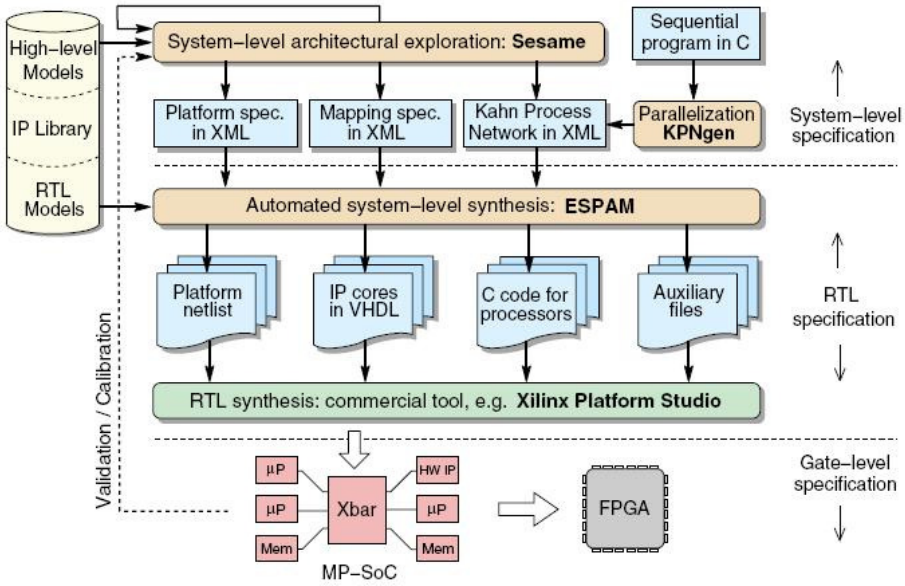

Figure 15 The Daedalus system-level design framework

In [134], the authors address the design space exploration (DSE) problem in order to find out Multi-Processor System-on-Chip architectures for a given multi-task signal processing application aiming to minimize the system cost while satisfying the real-time constraints. They propose a two step design architecture exploration to solve the three sub-problems, which are the processing elements selection, the application mapping and the synthesis of the communication architecture. The design flow is illustrated in Figure 16.

The flow inputs are the behavioral specification, an architecture template and a block performance database. The behavioral specification is represented in synchronous data flow (SDF) graphs where each node represents a coarse grain function block whose body is described in C code, an arc represents a FIFO channel that carries a stream of data from a source node to a destination node. The block performance database recodes the information on how long it takes for each PE to execute a functional block. The co-synthesis loop is performed as the first loop of the proposed exploration framework. When the PE selection and the mapping decision have been made, a HW/SW co-simulation is carried out to obtain the memory traces of all PEs. The communication architecture exploration loop then follows. The

design flow has also a global loop that updates the communication costs used in the co-synthesis loop with those obtained after the communication architecture is determined from the communication DSE loop. The experimental results with various random graphs and the 4-channel DVR application validated the efficiency and the viability of the proposed exploration method.

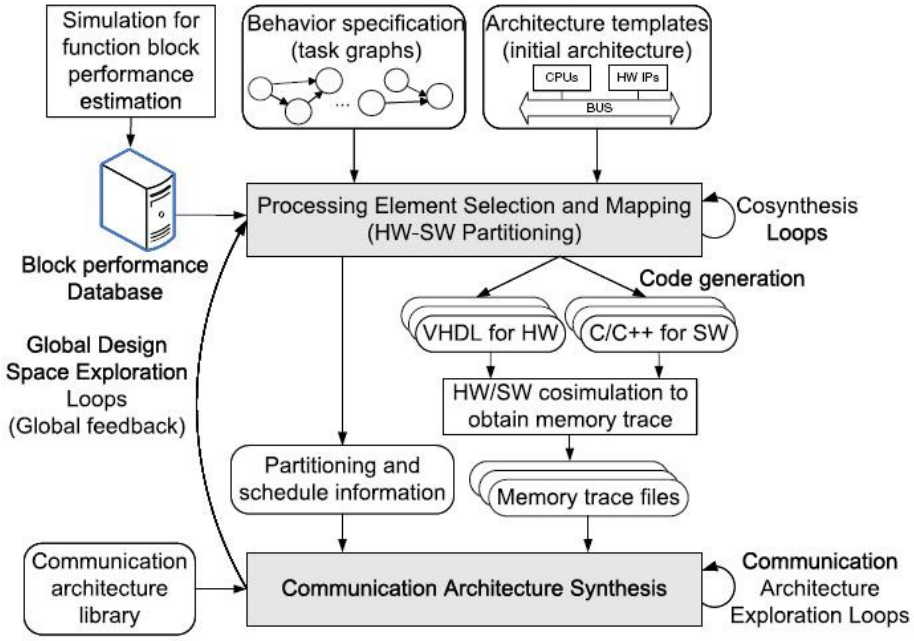

Figure 16 Two step design architecture exploration

3.2 Optimization Based Design Flows

Authors in [41] focus on the synthesis bus matrix based communication architecture for the high bandwidth MPSoC design. They propose an automated approach, named bus matrix synthesis (BMSYN), for synthesizing a bus matrix communication architecture, which satisfies all performance constraints in the design and minimizes wire congestion in the matrix.

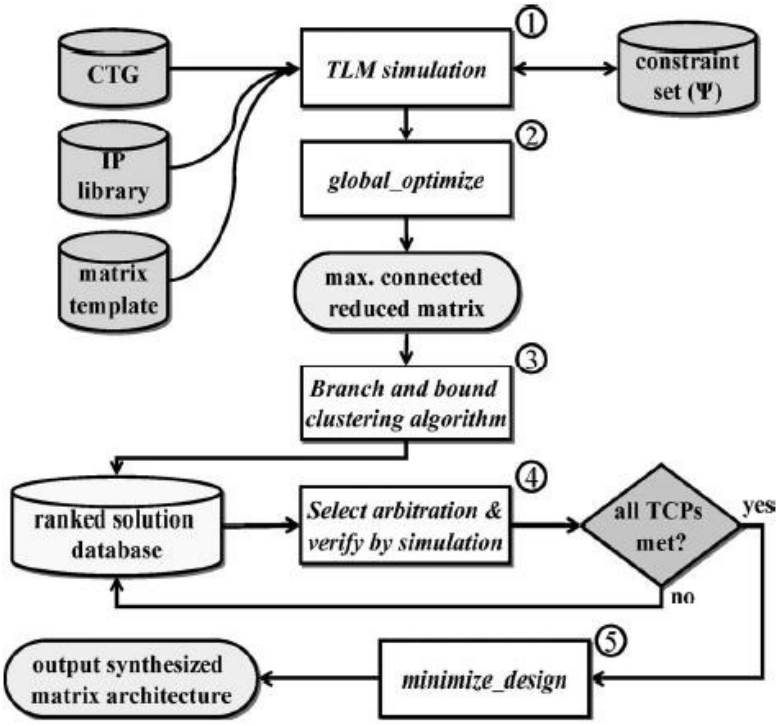

Figure 17 BMSYN automated flow

Figure 17 shows the automated BMSYN flow. The flow inputs include a common through graph (CTG) representing the performance constraints of the system, a library of IP models, a target bus matrix template, and the communication parameter constraint set. First of all, a fast transaction-level model (TLM) simulation of the system is carried out to determine the application-specific data traffic statistics. The information is then passed to the global optimization phase to reduce the full bus matrix architecture by removing unused busses and local slave components from the matrix. The resulting matrix is called a maximally connected reduced matrix. In the next step, an optimization engine based on a static branch and bound algorithm is used to cluster slave components which further reduces the number of busses in the matrix. The resulting architecture is then passed to a fast bus cycle accurate simulation engine to validate and select the best solution that meets all the performance constraints, determine slave arbitration schemes, optimize the design to minimize bus speeds and OO buffer sizes and then finally output the optimal synthesized bus matrix architecture. The results from the synthesis of an AMBA3, AXI-based bus matrix for four MPSoC applications from the networking domain show a significant reduction in bus numbers in the synthesized

matrix when compared with a full bus matrix (up to 9 x) and a maximally connected reduced matrix (up to 3.2x).

3.3 Automatic Parallelization State of the Art: The case of PluTo

For the purpose of this work we have selected an open source automatic parallelizer PLUTO. PLUTO [58] is a polyhedral automatic source-to-source transformer that can optimize nested loop sequences for coarse-grained parallelism and cache locality simultaneously. OpenMP parallel code for multicores can be generated from very regular C program sections. The effectiveness of the tool is based on the observation that a long running program often spends most of its time in nested loops. This is particularly common in scientific applications. Therefore a sub-optimized nested loop hinders the efficiency of a program in such aspects as inefficiency of cache access, unnecessary data dependence, overhead of synchronization point, etc. The polyhedral model is used in PLUTO for program representation and transformation. The polyhedral model provides powerful abstractions to optimize loop nests with regular accesses for parallel execution. Affine transformations in this model capture a complex sequence of execution-reordering loop transformations that improve performance by parallelization as well as better locality. The polyhedral model provides a powerful abstraction to reason about transformations on such loop nests by viewing a dynamic instance (iteration) of each statement as an integral point in a well defined space, which is the statement's polyhedral. Below we list some basic mathematic representations of the polyhedral model:

Loops are represented using iteration vectors:

$$\vec{x} = (i_1, i_2, \cdots i_n)^T$$

The iteration domain D defined as the set of values for which the statement is executed are represented as:

$$D = \left\{ \vec{x} \middle| \vec{x} \in Z^n, A\vec{x} \geq \vec{c} \right\}$$

where $\vec{x}$ is the iteration vector, A is a integer matrix and $\vec{c}$ is a constant vector (possibly parametric).

With such a representation for each statement and a precise characterization of inter and intra-statement dependence, it is possible to determine the correctness and goodness of a sequence

of complex loop transformations using the machinery from Linear Algebra and Integer Linear Programming. The polyhedral model is applicable to loop nests in which the data access functions and loop bounds are affine combinations of the enclosing loop variables and parameters. The task of program optimizations in the polyhedral model involves mainly three phases: (1) static dependence analysis of the input program, (2) transformations in the polyhedral abstraction, (3) generation of efficient loop code. There have been significant recent advances in dependence analysis and code generation that demonstrated the applicability of the polyhedral model to real applications. However current state-of-the-art polyhedral implementations still require manual efforts and expertise for determining the best set of transformations. For example, an import issue is the choice of transformations from the huge space of valid transforms. PLUTO addresses this problem by formulating a way to obtain good transformations fully automatically.

One of the key transformations involved in the PLUTO automatic transformation framework is tiling. It is studied in two perspectives: data locality and parallelization. Tiling for locality requires grouping points in the iteration space into smaller blocks (tiles) which allows data reuse in multiple directions when the block fits in a faster memory (register, L1 or L2 cache). Tiling for parallelism involves partitioning the iteration space into tiles that may be concurrently executed on multiple processors with minimum frequency and volume of inter-processor communications. PLUTO develops a cost function for looking for good tiling hyperplanes.

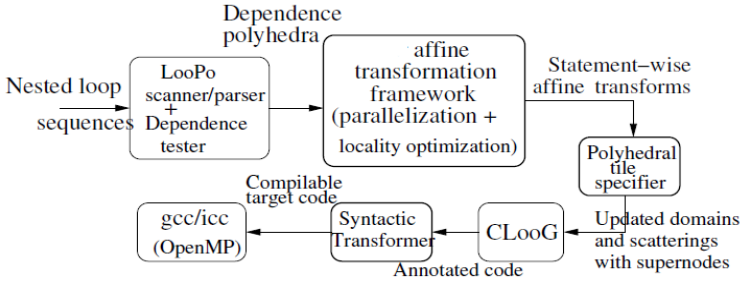

Figure 18 PLuTo workflow

Figure 18 shows the entire tool-chain of the PLUTO tool. The PLUTO tool utilizes LooPo [81] infrastructure for program scanner/parser and dependence tester. LooPo is a project of the University of Passau whose purpose is to develop a prototype implementation of loop

parallelization methods based on the polyhedral model. PLUTO uses PipLib [80] as ILP solver and CLooG [79] for code generation. CLooG provides software and library generating loops for scanning Z-polyhedra. It finds the code or pseudo-code with which each integral point of one or more parameterized polyhedra, or union of parameterized polyhedra is reached. CLooG is designed to avoid control overhead and produce very effective code. Figure 19 presents an example of PLUTO transformed code.

```
#pragma scop
 for(i = 0; I < M; i++)
  for(j = 0; j < N; j++)
   for(k = 0; k < K; k++)
    C[i][j] = beta * C[i][j] + alpha * A[i][k] * B[k][j];
#pragma endscop
```

(a)

```
#pragma omp parallel for shared(lb1,ub1) private(t0,t1,t2,t3,t4,t5)
 for(t0 = lb1; t0 <= ub1; t0++) {
  for(t1 = 0; t1 <= floord(N-1,128); t1++) {
   for(t2 = 0; t2 <= floord(K-1,8); t2++) {
    for(t3 = max(8*t0,0); t3 <= min(M-1,8*t0+7); t3++) {
     for(t4 = max(0,8*t2); t4 <= min(8*t2+7,K-1); t4++) {
      for(t5 = max(0,128*t1); t5 <= min(128*t1+127,N-1); t5++) {
       C[t3][t5] = beta * C[t3][t5] + alpha * A[t3][t4] * B[t4][t5];
      }}}}}}
```

(b)

Figure 19 PLUTO transformation: (a) sequential code, (b) parallel

The PLuTo parallelizer allows multiple options. The options are described in Table 7. In particular, classical unrolling and unrolling factors are proposed to the user. Vectorization is also proposed as part of the currency extraction potential.

Table 7 polycc command-line options

Pluto options	**Description**
--tile	Tile code
--l2tile	Tile also the L2 Cache
--parallel	Parallelize code using OpenMP
--multipipe	Extraction of multiple degree of parallelism
--smartfuse	Fuse between strongly-connected components
--unroll	Unroll up to two loops
--ufactor=<f>	Unrolling factor
--prevector	Vectorization

Pluto has been applied in various studies [58][104]. In [104], hybrid iterative and model-driven optimizations have been successfully proposed and applied.

3.4 Automatic parallelizer based MPSoC design flow

The PLUTO parallelized code depends on the OpenMP API, compiler and OS run time support to realize task partition. However, such support is rarely available in an embedded context where OS is not always present. We proposed an automatic accelerator generation flow that integrates PLUTO and adapts an application targeting the general purpose processor to an embedded environment. The flow is illustrated in Figure 20.

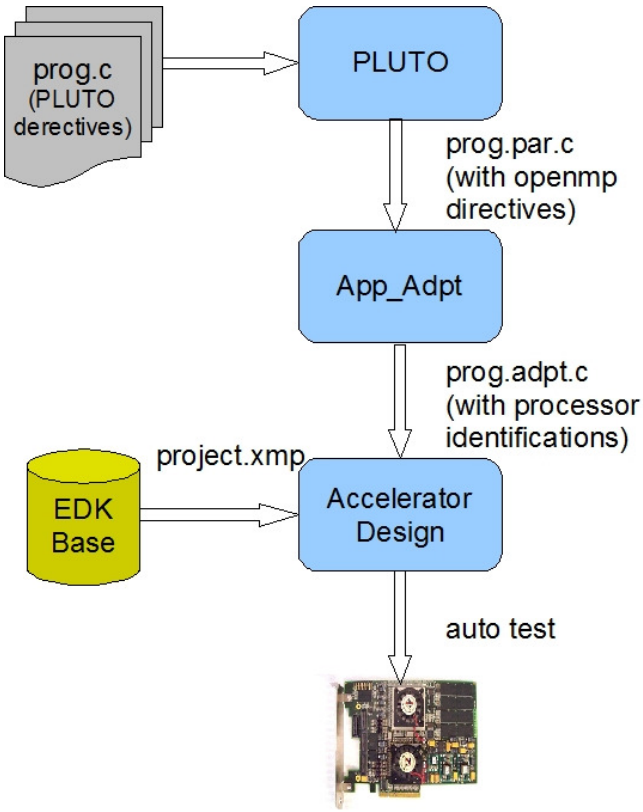

Figure 20 Automatic parallelizer based accelerator design flow

The input source file (prog.c) is marked with PLUTO directives indicating the loop nests to be parallelized and the accompanying configuration file containing platform information, such as memory hierarchy. PLUTO parallelizer then analyzes the code and generates the parallel version in the form of OpenMP (prog.par.c). An automatic application adaptor (App_Adpt) then replaces the OpenMP directives with platform specific identifiers so that workload can

be partitioned and identified by their corresponding processor. Necessary synchronization functions are also inserted at this step based on the semantics of the specific OpenMP directives. After preparing the application, the accelerator generator (Accelerator Gen) generates the platform in the form of Xilinx Microprocessor Project file (.xmp) with the help of a library of EDK project basic construction components. Then the platform is synthesized, place and routed (ISE) to generate the final downloadable bit stream. Finally, the bit stream is downloaded and executed on the target evaluation board.

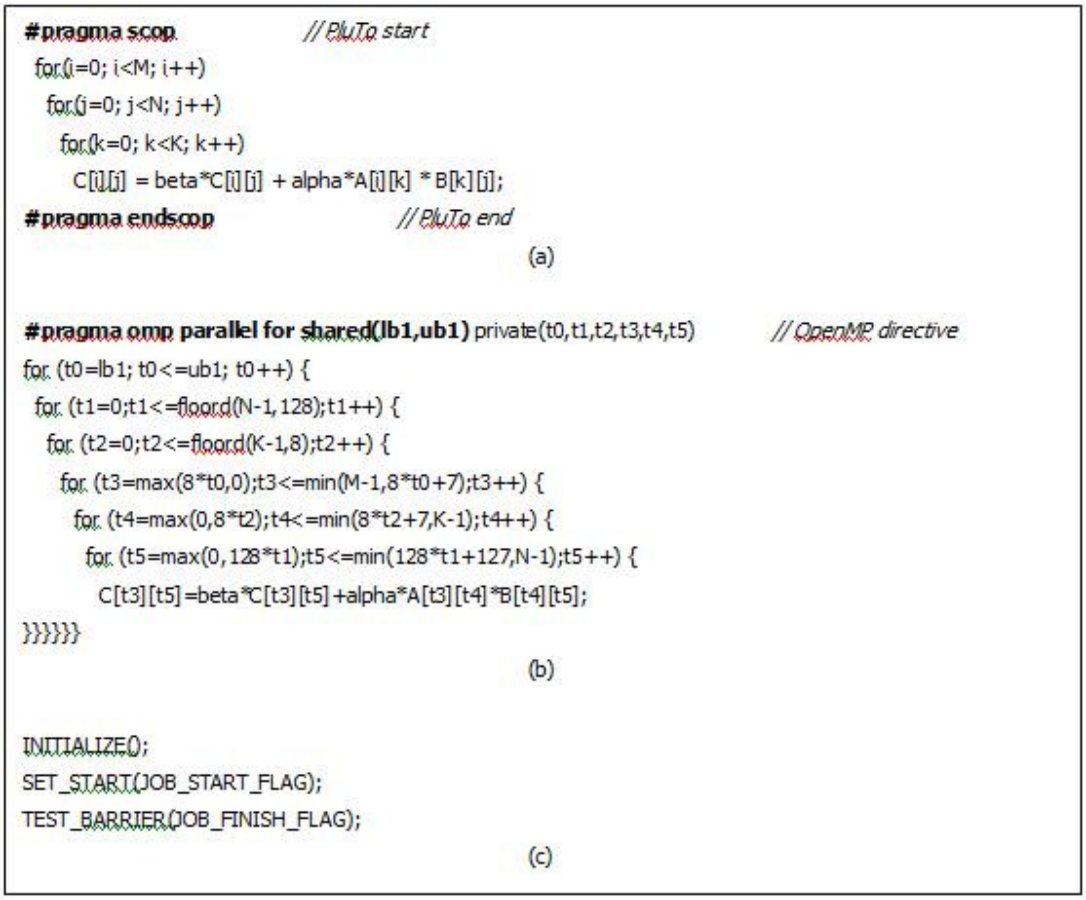

Figure 21 Code example of the Design flow

This flow is readily adaptable to other platforms or CAD tools. Figure 21 is an example showing the evolution of the form of source code. (a) is the original sequential code marked with PLUTO directives which is passed to the PLUTO tool to generate (b), parallel code in the form of OpenMP. Then the semantics of the OpenMP is analyzed and replaced with processor identification functions and synchronization mechanisms to generate (c) and (d) which are ready to be compiled and executed on the target platform. Wherein, code (c) is the

master code which initializes the DDR2 memory and takes charge of the global synchronization, while code (d) does the actual calculation.

3.5 Case study: A NoC based MPSoC programming and optimization

3.5.1 MPSOC platform

We designed and implemented a single FPGA chip 16PE shared memory MPSOC using a 2 stage NOC [46]. The multiprocessor platform is in general a shared memory architecture which integrates 16 Processing Elements (PE) Tiles of type MicroBlaze v7.0, as illustrated in Figure 22. Each tile is a powerful computing system with 64KB local memory that can independently run its own program code. PE tiles are connected to four DDR2 controllers via the Data-NoC which in turn control four off-chip DDR2 memory banks each having 256MB of capacity. This is where data storage and communication take place. A piece of small (1KB) on-chip block memory (BRAM) is implemented and connected to the 16 PEs via the synchronization NoC providing different synchronization facilities with minimum latency. One of the PEs is also connected to a PCI-Express controller that serves as console output when the platform is in a standalone mode or as high bandwidth data transfer channel configured in the host/coprocessor mode.

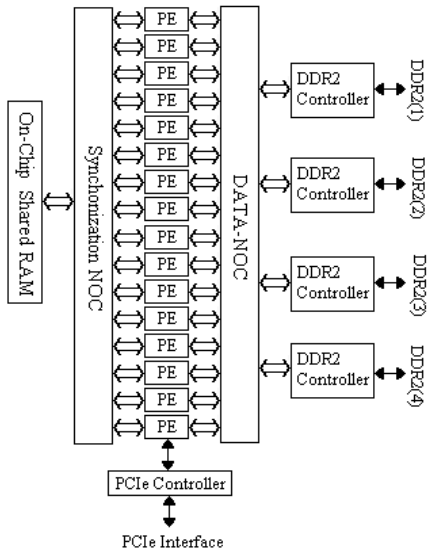

Figure 22 Architecture of NoC-based multi-core

(1) Data & Synchronization NoC

The communication infrastructure is built on the Network-on-Chip technology. Arteris™ Danube NoC Intellectual Property Library [22] provides highly configurable IP blocks, switch, network interface unit (NIU), routing table, etc., that manage on-chip communications between IP cores in System-on-Chip (SOC) designs like processors, DSPs, memories, I/O peripherals and so on. Data and control are transferred through different components of the NoC in form of packet conforming to the Arteris proprietary NoC Transaction and Transport Protocol (NTTP). IP integration is facilitated by the Network Interface Unit (NIU) of different standardized bus protocols, for example, AHB, AXI, OCP, and so on. We utilized the OCP protocol for the network work interface unit to which we will return in the next sub-section.

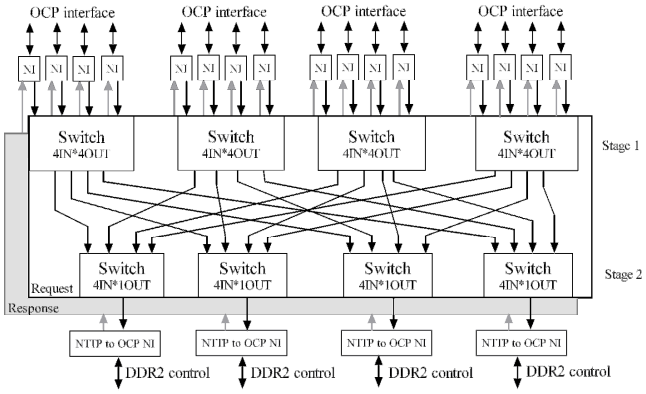

Figure 23 Architecture of Data NoC

Figure 23 shows the internal design of the Data NoC. It is a two-stage packet switched network comprised of a request network and a response network. The first stage is composed of 4 switches with 4-input-4-output each while the second one is composed of 4 switches with 4-input-1-output. Processors and memory controllers are integrated via the OCP-NTTP and NTTP-OCP NIU respectively, which realizes protocol conversion between IP core native transactions and NoC.

The architecture of the Synchronization NoC is shown in Figure 24. It differs from that of the Data NoC in the second stage of switches that contains only one 4-input-1-output switch directing traffic onto an on-chip RAM (BRAM). The Exclusive Access Manager inserted between the last-stage switch and the output NIU is an optional unit that can be included to realize the Load-Linked Store-Conditional (LL-SC) synchronization mechanism defined in the OCP protocol.

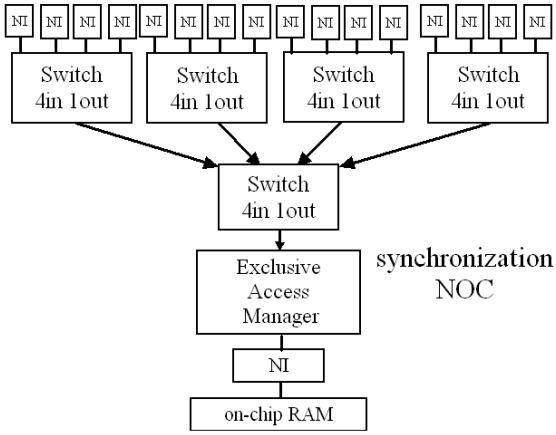

Figure 24 Architecture of Synchronization NoC

(2) Processing elements

In our system we used the Xilinx Microblaze v7.00 soft-core microprocessor as the processing element. Figure 25 shows the block diagram of the microblaze architecture. The Microblaze processor is a 32bit Reduced Instruction Set Computer (RISC). It is implemented with Harvard memory architecture. The Microblaze processor is highly configurable and is optimized for FPGA implementation. A set of parameters and execution units can be configured at design time to fit design requirement, such as the number of pipeline stages, the cache sizes, a selectable Barrel Shifter (BS), a Floating Point Unit (FPU), a Hardware Divider (HWD), a Hardware Multiplier (HWM) and a Memory Management Unit (MMU). The performance and the maximum execution frequency varie depending on the processor configuration. For 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 in its simple version, which contains a 5 stage pipelines, a 32 bit integer HWM, and the Machine Status Register Instructions are enabled, as well as the pattern comparator. The Microblaze based element contains an Instruction-side Local Memory Bus (ILMB), a Data-side Local Memory Bus (DLMB), an ILMB BRAM interface controller, a DLMB BRAM interface controller and a BRAM based 32KByte local on-chip memory. The local memory is connected to the processor through the LMB interface controller and the LMB memory bus. The FSL interface of the Microblaze is directly connected to the OCP Synchronization

Adapter and the OCP Data Adapter. The 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 consumed by the Data-NoC and Synchronization NoC.

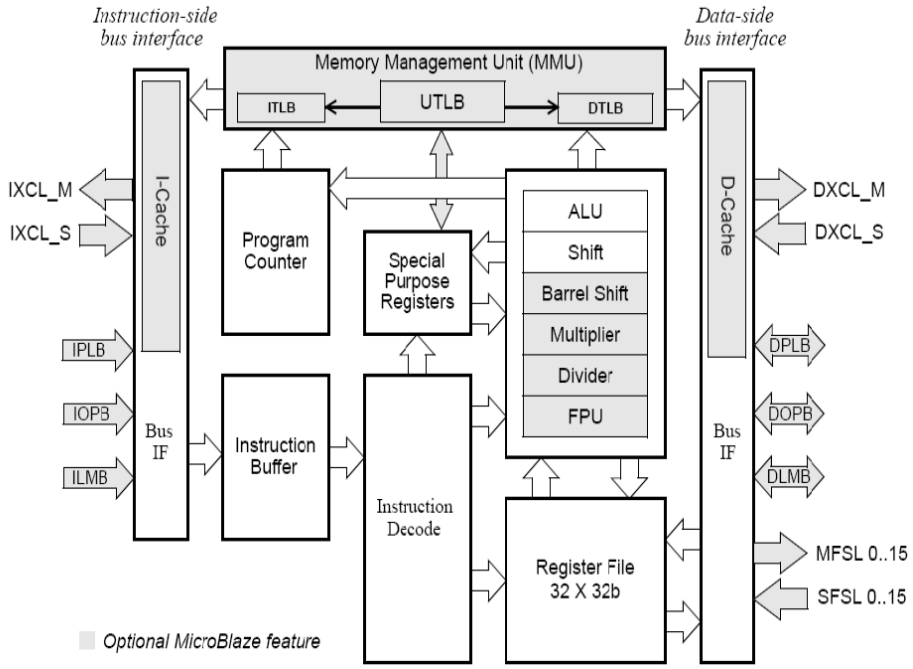

Figure 25 Block diagram of the microblaze architecture

3.5.2 OCP-IP Specification

The Open Core Protocol (OCP) is an openly licensed, core-centric protocol defined by the OCP International Partnership (OCP-IP) [66]. It provides a common standard for intellectual property (IP) core integration in a "plug-and -lay" manner. The protocol is based on the master-slave point-to-point model.

Today's IP cores have custom, tight coupling interface logic that is difficult to design and difficult to modify. Consider a scenario in which we need to connect M master cores to N slave cores each having a custom interface. M x N bridges must be designed in order to connect each master core to every slave core. (Figure 26 (a)) OCP uses a network socket approach to separate communication from the design of the individual IP core interface design

which allows differing IP cores to effectively make use of the on-chip network. As a consequence, the M-master-N-slave scenario requires only (M + N) bridges with every core integrating the OCP interface. (Figure 26 (b))

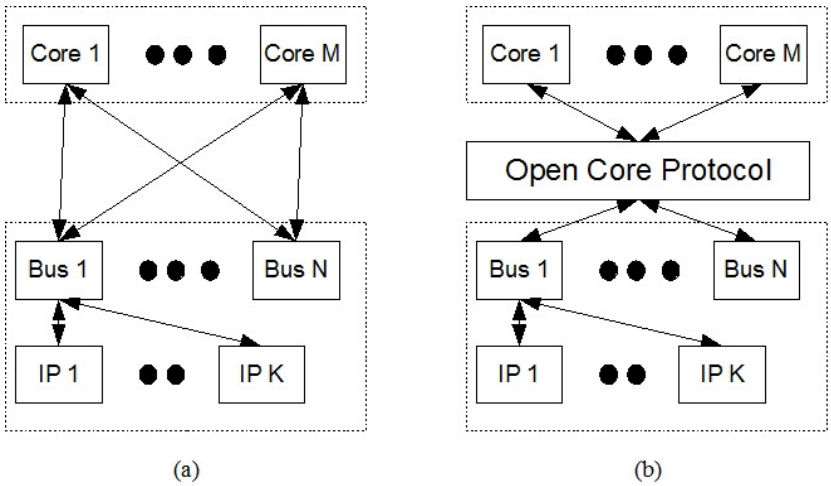

Figure 26 System integration with a custom interface. (a) System integration with an OCP protocol (b)

OCP provides a master/slave connection between two cores. One core, called the OCP initiator core has an OCP master interface. A master interface enables a core to generate OCP requests such as READ or WRITE and receive the READ responses. The other core, called the OCP target core, has an OCP slave interface that allows it to receive and respond to requests.

OCP interface signals are grouped into dataflow, sideband, and test signals. The dataflow signals are divided into basic signals, simple extensions, burst extensions, and thread extensions. [67] The OCP is a synchronous interface with a single clock signal. All OCP signals are driven with respect to, and sampled by the rising edge of the OCP clock. Except for clock, OCP signals are strictly point-to-point and unidirectional. The complete set of OCP signals is shown in Figure 27. The encoding of different Master Commands (MCmd) is summarized in

Table 8.

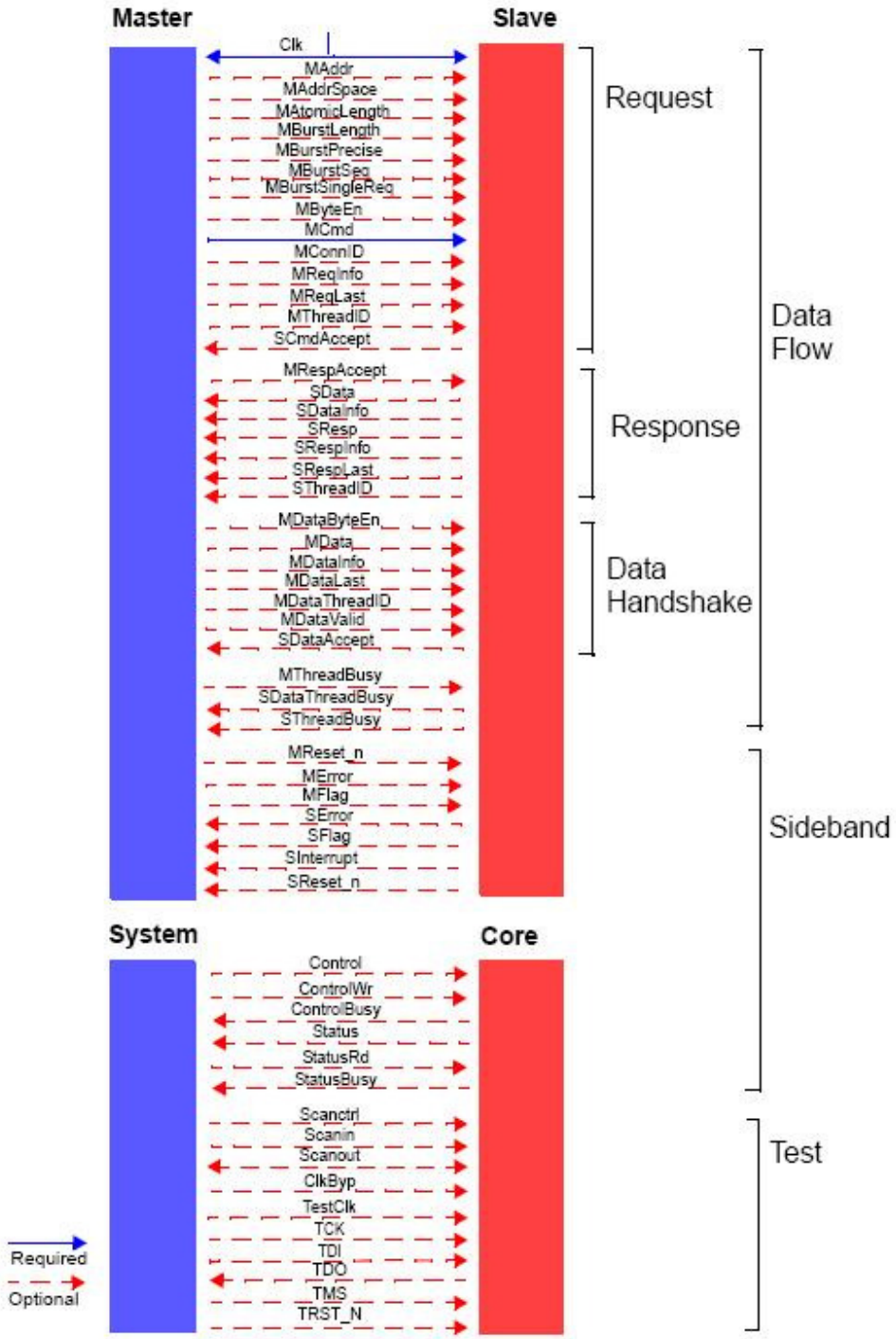

Figure 27 OCP signals

Table 8 OCP MCmd

MCmd [2:0]			Command	Type
0	0	0	Idle	(none)
0	0	1	Write	write
0	1	0	Read	read
0	1	1	ReadEx	read
1	0	0	ReadLinked	read
1	0	1	WriteNonPost	write
1	1	0	WriteConditional	write
1	1	1	Broadcast	write

3.5.3 Synchronization with OCP-IP

We are interested in the synchronization mechanisms defined in the OCP protocol. There are 3 major steps in a synchronization event: (1) acquire method, (2) waiting algorithm (3) release method [14]. There are 2 main choices for the waiting algorithm: busy waiting and blocking. Busy-waiting means that the process spins in a loop that repeatedly tests for a variable to change its value. Blocking the process does not spin but simply blocks itself and releases the processor if it finds that it needs to wait. Busy-waiting is likely to be better when the waiting period is short whereas blocking is better if the waiting period is long. Synchronization mechanisms should present: (1) low latency, (2) low traffic, (3) scalability (4) low storage cost and (5) fairness. Two common ways of implementing synchronization are: read-modify-write and LL-SC. The OCP protocol [21] supports these two ways of synchronization among OCP masters by encoding of different Master Commands MCmd as listed in

Table 8.

The first one is Locked synchronization, which is a read-modify-write style atomic transfer. OCP initiator uses the ReadExclusive (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. 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. The OCP master modifies the data and sends it to the slave by a Write or a WritreNonPost command. When the NIU receives the Write command, it writes the data to require the NTTP slave by a STORE request packet and then releases the NoC by an Unlock request packet. The other competing OCP masters cannot 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. The Load-Linked Store-Conditional (LLSC) synchronization is realized by first issuing a ReadLinked (RDL) and then a normal Write or WriteNonpost command. If in between the two operations, another write operation occurred at the same position, or put it another way, the linked resource is modified, the write operation fails, thus realizing the atomic semantic. The Arteris NoC library embeds the Locked Synchronization implementation in the switch component, while it requires a specialized component called Exclusive Access Manager for the LLSC mechanism. Parallel software implementation involves complex tradeoffs including partitioning and load balancing, the granularity of communication, working set and the overhead of synchronization [15-16]. Poor synchronization primitives and hardware may greatly impact the parallel program performance. In this paper, we will present the performance analysis on hardware support for synchronization.

The ADPe-XRC-4 evaluation board of Alpha Data was chosen for the implementation of the system. It is designed with a Xilinx Virtex4 FX140 FPGA which features 63, 168 slices and 1.2 MB BRAM. The board has 4 independent banks of DDR2 SDRAM with a total capacity of 1GB memory. The block diagram and an actual illustration of the board are presented in Figure 28 and Figure 29 respectively.

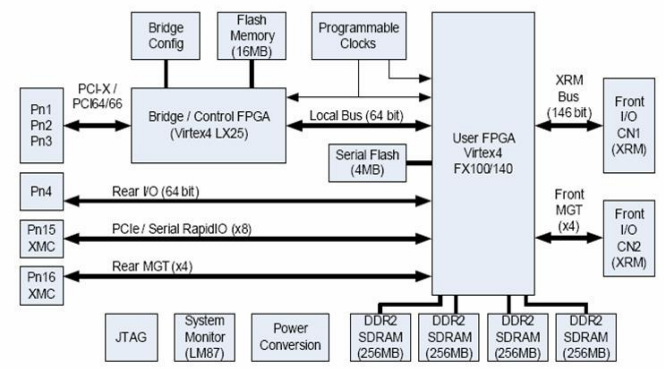

Figure 28 Block diagram of Alpha-Data FPGA

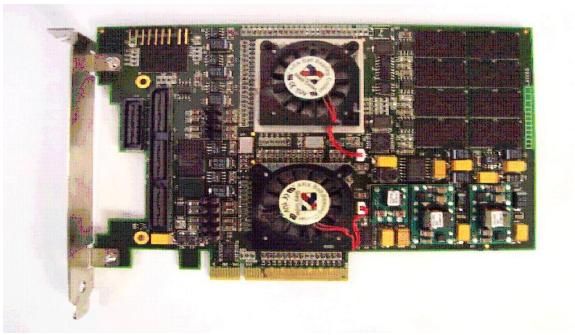

Figure 29 Alpha-data ADPe-XRC-4 FPGA board

3.5.4 Synchronization results and analysis

We have conducted synchronization performance evaluation experiments on the system. Although some efforts have been made for the benchmarking of NOC based multicore systems, NOC benchmarking for synchronization remains an open issue. We use the same approach as [68] through synchronization micro-benchmarks. The processors are divided into 1 master PE and 15 slave PEs. As illustrated in Figure 30, when entering the program, the master processor set the “Start” flag that triggers the execution of the other slave processors. It then turns to test the “Finish” flag which is updated by each slave processor upon task completion. There is zero workload for the slave processor between the “Start” flag test and the “Finish” flag updating to measure the number of clock cycles introduced by the synchronization.

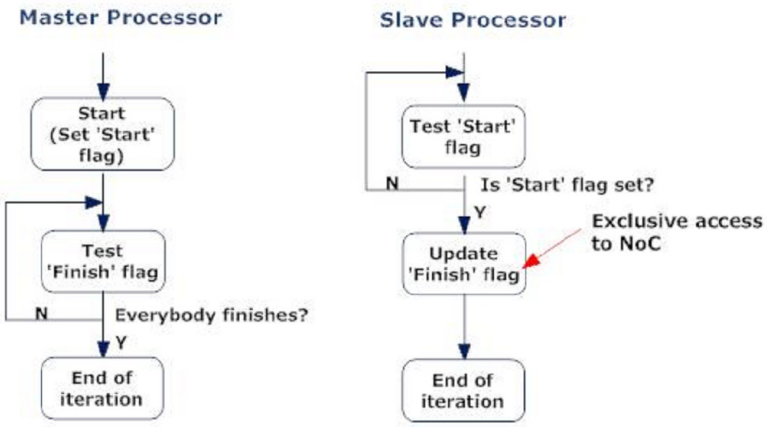

Figure 30 Synchronization micro benchmarks

Because of the concurrent write nature of the "Finish" flag, a synchronization mechanism is needed to assure the atomic read-modify-write operation. We implemented different synchronization mechanisms and compared their impact on the performance. The first mechanism is called the "block synchronization", which means that when one processor gets access to a protected memory area, the others cannot access the same memory area until the first one releases the memory by a write operation. The access is controlled by the NoC interface. The second mechanism is called LL-SC (Load linked Store conditional) as it uses the Load Linked and Store Conditional instructions that allows to implement an atomic operation without forbidding memory accesses between the two instructions. It means that the protected memory area is not exclusively owned by any processor at any time. If a processor wants to perform an atomic update, it should first read the contents of memory, updates the contents, and before writing back it should make sure that no other processor has modified the contents between the read and write operation. In a first step we evaluate the performance of blocked synchronization versus LL-SC by varying synchronization agents with the synchronization lock placed in the BRAM.

From the results shown in Figure 31, it should be noticed that the Locked Synchronization outperforms that of the LLSC by 50%. It can be explained by the overhead introduced in the LLSC when a conditional write is judged to either pass or fail depending on whether the tested variable is modified. More over, upon failure, the ocp packet containing the

WriteConditional command has to be reissued by the processor and routed through the NoC which introduces extra processor and packet routing (NIU to Exclusive Access Manager) cycles compared to the Locked mechanism for which the write packet blocks at some conflicting point on the NoC and passes once the lock is cleared. As a result, we conclude that the LLSC synchronization mechanism is not preferable in an environment where the shared memory is protected by an access-exclusive NoC. In a second step, the synchronization performance is compared between a dedicated NOC (Synchronization NoC) with access to a lock variable placed in an on-chip memory (BRAM) and a shared NoC (Data NoC) with the lock variable placed in the DDR2 memory. The results are shown in Figure 32.

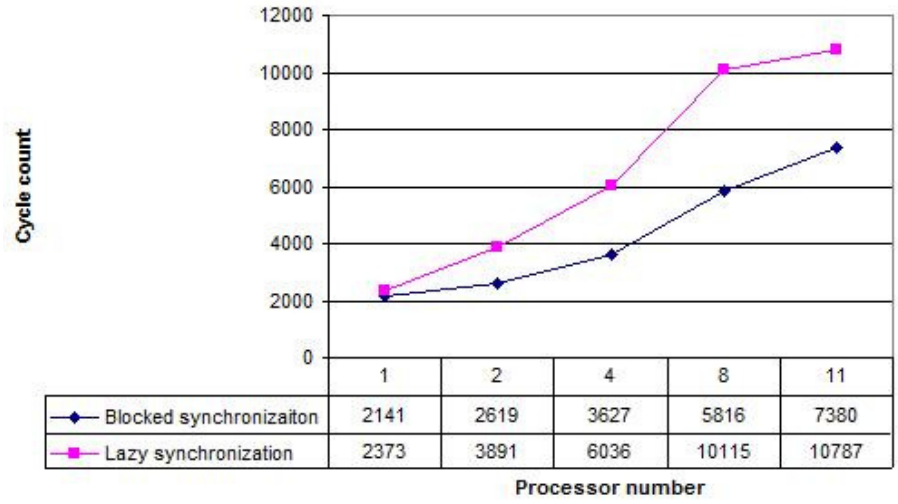

Figure 31 Synchronization performance: Locked vs. LLSC

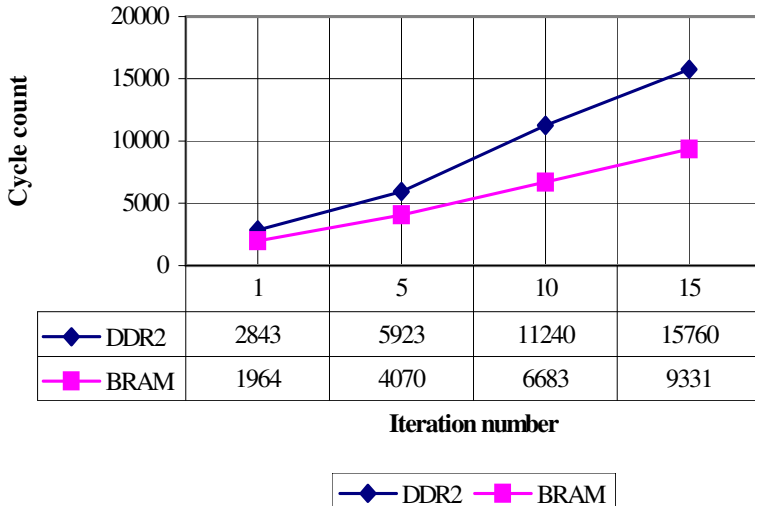

Figure 32 Synchronization performance: BRAM vs. DDR2

Clearly, blocking on BRAM with dedicated NOC reduces by 40% the synchronization time for 15 processors. It indicates the advantage of using the blocked synchronization mechanism for centralized synchronization architectures.

The overall results show the superiority of the blocked mechanism in the dedicated synchronization NOC with BRAM over LL-SC with BRAM or blocked with DDR in a single-lock case. The synchronization only requires a small number of synchronization variables even for a great number of processors. So it is a good choice to sacrifice some on-chip resources as a dedicated synchronization memory.

3.5.5 Experiments of automatic parallelization

In examining the execution results of the PLUTO parallelized codes on our platform, we noted several key elements that influence the effectiveness of parallelization. Some of these elements are intrinsic in the application, while others are architecturally dependant. A comprehensive understanding of the characteristics of both the application and the architecture accompanied by an optimum combination of the two is necessary for a satisfying performance. We parallelized and tested several micro benchmarks from linear algebra and multimedia algorithms with each one of them highlighting one or a couple of the performance limiting aspects for parallel computation.

(1) Matrix multiplication 128 * 128

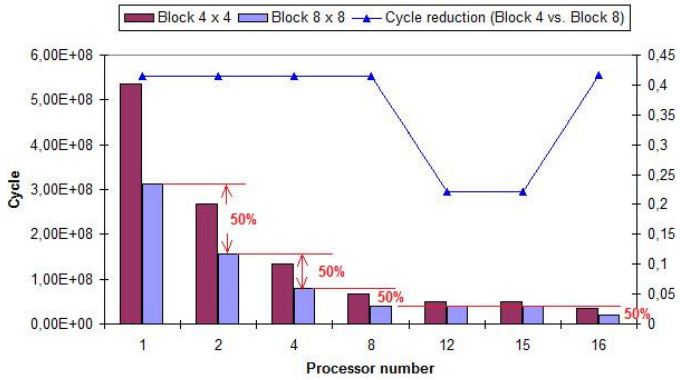

Figure 33 Execution results of Matrix Multiplications (128 * 128)

The timing results are obtained using low level timer register read/write instruction thus introducing only trivial overheads, in the order of several tens of cycles.

Matrix multiplication is among others the most parallelizable application because of its high data independency. The parallelized code generated by PLUTO is actually a block based matrix multiplication. The resulting matrix is divided into blocks. Each processor can simply take charge of one unfinished block to work on independently without having to stall waiting for data produced by other blocks. However the memory access efficiency becomes an important issue. For all hierarchical memory architectures, memory access efficiency is a key element for the overall performance. The better the local cached data is exploited, the better the performance will be. In our case, we defined two block sizes for the blocked based matrix multiplication algorithm, 4 * 4 and 8 * 8, respectively. From Figure 33 we can see that the cycle reduction in increasing the block size from 4 * 4 to 8 * 8 is 41% (as the triangle marked line illustrates). This is because with larger block sizes, there is a larger data reuse ratio. (This ratio differs from application to application and is approximately in the order of magnitude of O(n) for matrix multiplication, where n indicates block size.) However there are two exceptions on the curve where the processor number equals 12 and 15 respectively. This comes from an under-utilization of processor resources when the workloads are not divisible by the number of processors. We can deduce that using 16*16 block size will saturate even earlier the performance gain. The figure also shows that that the cycle counts don't scale from

8 processors to 12 or 15 processors. We will return to this issue in the other applications. Figure 33 shows a perfect performance scalability of the system as, when we double the number of processors the performance also doubles. The synchronization and communication overheads can be ignored with the block size we chose due to relatively high level of parallelization. The result shows, on one hand, that the NoC is far from saturation as proved by the nearly perfect scalability and still have space for even heavier traffic loads, and on the other hand, the possibility to hide the communication latency with computation is a promising technique for better performance.

(2) Seidel 128 * 128

There are two important aspects in a PLUTO generated code: data dependence analysis and data locality improvement. The data dependence analysis results can be readily exploited by dispatching independent parts of codes to different processors, and the correction of the operation is guaranteed by the PLUTO tool. The cycle counts for different numbers of processors show a relatively low but still satisfying parallelizability of the Seidel Algorithm compared to that of the Matrix Multiplication. The performance scaling keeps track of resource scaling until 8 nodes. The core of Seidel algorithm is the calculation of the average of each 3 x 3 window of 9 elements in a two dimensional array. If we partition the job on a line-by-line basis, the second processor can only start after the first one has finished calculating the first two pixels, the third waits similarly for the second processor. That's why when the number of processors increases beyond a certain level, the new coming should wait the total calculation of the first processor before being able to start execution.

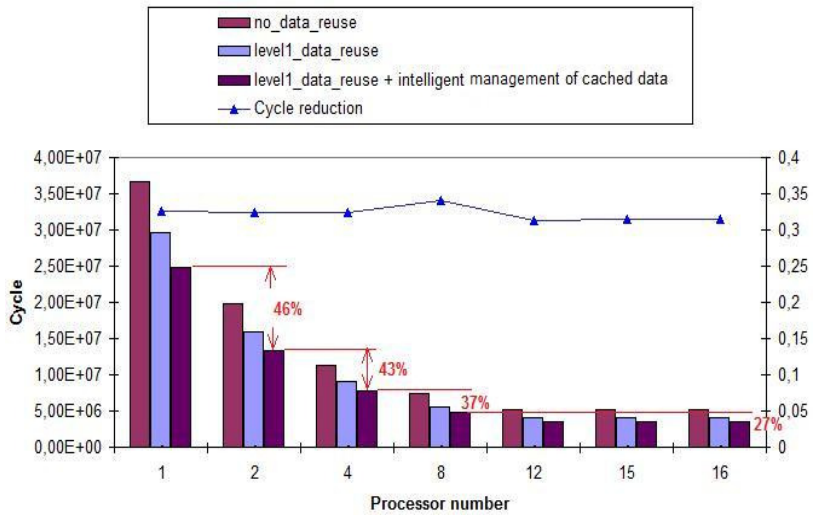

Figure 34 Execution results of Seidel 128 * 128

When passing from 8 nodes to 16, we only get 27% cycle reduction. Because of the lack of hardware support for cache control, we manually managed some software cache functionalities to improve data reuse. The data are should be first fetched from the DDR memory to the local Bram before the processor can proceed the calculation. We consider the local Bram as the equivalent of a software cache. Cache functionality is the reuse of partial fetched data instead of refetching the whole block in a later block calculation. This is done by software. However the data locality implemented by the compiling tool by tiling is not readily usable due to different architectural constraints. In our case, the lack of hardware support for cache (local memory) control makes it necessary to manage the cache protocol in software. PLUTO implements tiling in transforming long “for” loops into nests of small “for” loops so that the inner most loops can totally fit into the local memory, and cache miss happens only when the outer loops change. We manually coded some cache control protocols and compared their performance with a protocol without cache data management. We restricted manual coding in the innermost loop because of the code complexity and memory limits. The blue curve shows the cycle reduction of an intelligently cache data managed code compared with the original one and there’s a steady 33% cycle reduction for all configurations with different number of processors.

(3) DCT (Block size: 4 * 4)

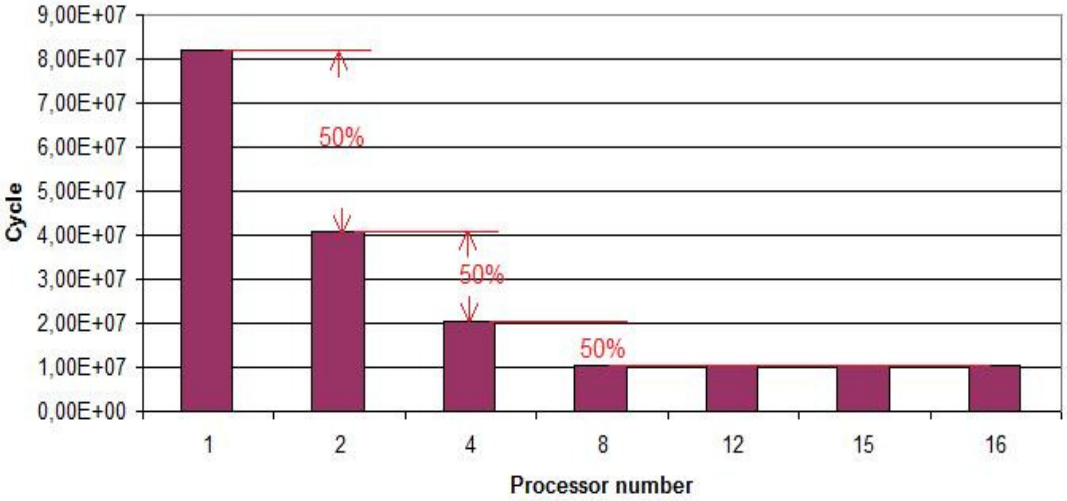

Figure 35 Execution results of DCT (32 * 32)

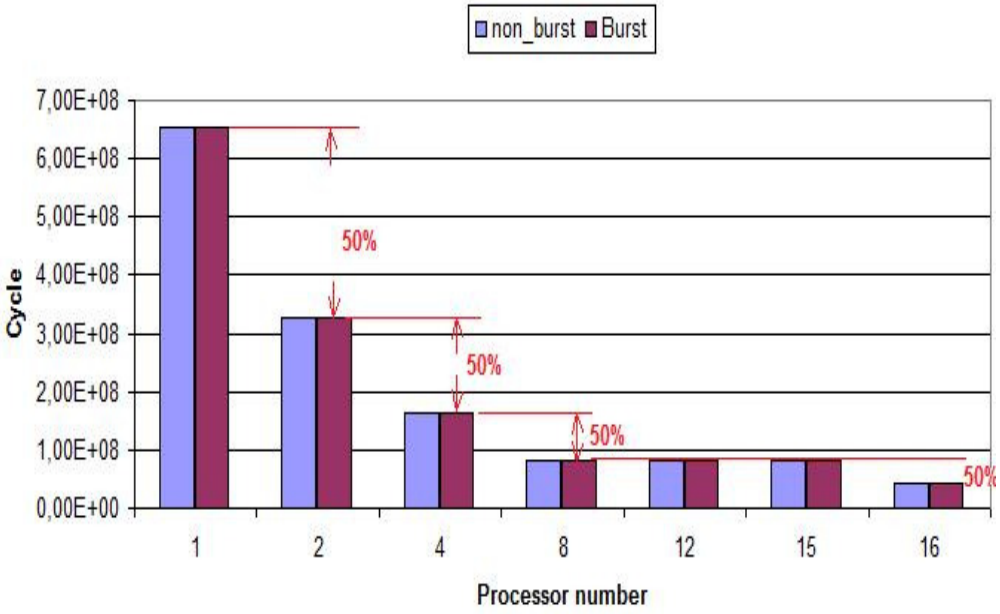

Figure 36 Execution results of DCT (64 * 64)

In this application, we fixed the size of data block to 4 * 4. From Figure 35 we note that beyond 8 nodes, additional processor doesn't introduce any performance improvement. This phenomenon results from the fact that we use a relatively large parallelization granularity (4 * 4) for a small workloads (32 * 32) in which case extra processor resources are in idle mode lack of available workload. When we increase the workload from 32 * 32 to 64 * 64, this phenomenon exists no longer and we get a perfect performance scaling until 16 nodes, as illustrated in Figure 36. We also compared the performance differences between burst and

non burst mode of ddr access. The resulting performance improvement is trivial and is hardly visible from the figures.

(4) Jacobi_1d (Vector size: 1000 Iteration: 2)

In this application, we noted a limitation of the PLUTO parallelizer. For this particular application and some other ones (LU decomposition), it can provide an efficient parallelization only when the number of iterations is great or when the workload is large. For jacobi_1d with 2 iterations, the parallelization efforts only introduce synchronization overheads and no performance improvement.

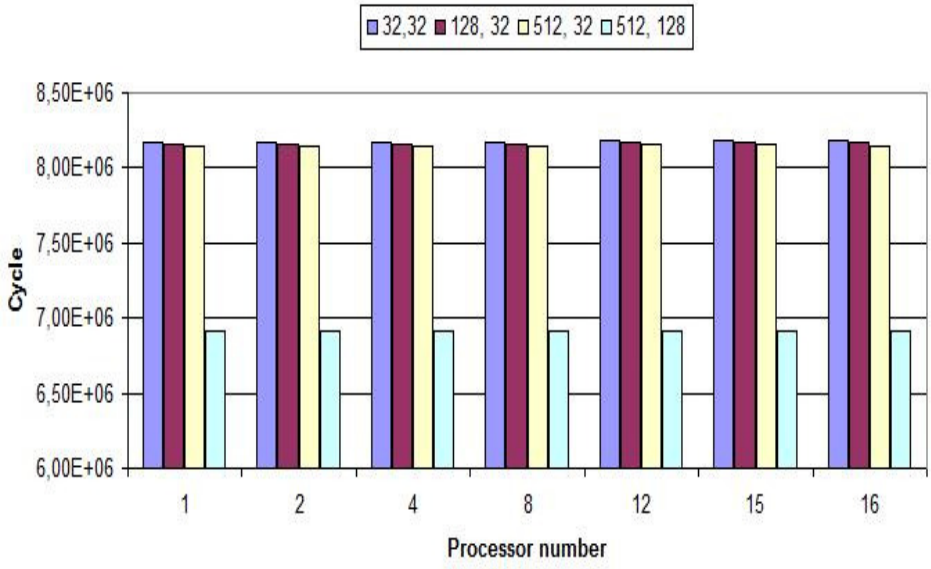

Figure 37 Execution results of Jacobi 1D

Instead of parallel execution, the processors take turns to execute different parts of the work bringing out the same performance as one processor taking charge of all the work. However, Figure 37 does show us another interesting point regarding the synchronization overhead. Even though this synchronization shouldn't exist at the first place if only one processor is activated, the fact that choosing the optimum tiling size, (512, 128), can greatly reduce synchronization calls and thus improve the performance, a factor of two compared to others in this case.

3.5.6 Multi-programming Experiments and Analysis

As it can be noticed from the above single-application parallelization performance results, the power of the parallel architecture is not always fully exploited due the application's intrinsic limitation in terms of parallelizability or the limitation of parallel compiler to fully exploit the application parallelism. Therefore we are naturally led to such a situation that the resources of the parallel architecture should be shared by multiple applications.

We did some experiments of the multi-programmed platform with different combinations of the above mentioned applications. We noticed that the number of 8 processors is usually a critical point beyond which performance stops improving linearly. So in our experiments, we divided the processors in two groups with each application being allocated 8 processors.

Table 9 Matrix Multiplication (128 * 128 Block size 8 * 8) / DCT (32 * 32 Block size 4 * 4)

Number of processors	Matrix	DCT	Total
16 (single application)	19,707,161	10,246,616	29,953,777
8 (single application)	39,339,984	10,250,044	49,590,028
8 each (multi-program)	N/A	N/A	40,278,067

Table 10 DCT (32*32 Block Size 4*4) / Seidel 128*128 Level1_Data_Reuse and intelligent management of cached data

Number of processors	DCT	Seidel	Total
16 (single application)	10,246,616	3,545,928	13,792,544
8 (single application)	10,250,044	4,837,995	15,088,039
8 each (multi-program)	N/A	N/A	10,408,323
8 DCT / 4 Seidel / 4 open	N/A	N/A	10,408,249
8 DCT / 2 Seidel / 6 open	N/A	N/A	13,609,924

From the result we notice that for the combination of matrix multiplication and DCT, the cycle count of the multi-programmed platform (40,278,067) is 34% greater than the accumulated (29,953,777) value of separated execution. The reason is that the matrix multiplication is the most parallelizable application and the performance improves linearly until 16 processors. So there's no need to optimize the resources utilization by sharing the computing resources with another application. And the inclusion of another parallel executed application add additional burden on the interconnection infrastructure and memory access conflicts.

As the figures in the previous section show, the performance scaling stops for DCT when the processor number reaches 8. The 16 processor to 8 processor improvement for Seidel is only 27%. So it makes sense to share processors between these two applications and the measured results justify this reasoning. The cycle count of the multi-programmed platform (10408323) is 25% less important than the accumulated (13792544) value of the separated executions. We should also notice that because of the data size chosen, there is a performance difference between the two applications in the order of ten. If the data size of Seidel increases, the performance improvement of the multi-programmed platform should become more important. So we kept reducing the processor allocation to the Seidel application. When the processor allocation is 8 (DCT), 4 (Seidel), 4 (open), the cycle count remains unchanged as the case where each application is allocated 8 processor. If we remove 2 more processors from Seidel application, which gives the configuration 8 (DCT), 2 (Seidel), 6 (open), the performance starts to deteriorate to the best achieved performance in a single programmed platform. So the most performance/resources optimized configuration for DCT/Seidel combination is 8 (DCT), 4 (Seidel), 4 (open). The remaining four idle processors are good candidates for compute bound applications.

The platform also provides means to realize design space exploration. The microblaze soft core is highly reconfigurable. It can be reconfigured to include different number of pipeline stages and a hardware multiplier and a barrel shifter. The architecture can also be extended by hardware accelerator via the FSL link. Brams are limited resources that should be used in an

efficient way. In a multiprogrammed environment, Bram resources should be customized for each processor according to application needs.

3.6 Conclusion

The design of multiprocessors on chip is strongly emerging in high performance embedded systems. The specific features of embedded multiprocessor on chip present new challenges for parallel applications mainly due to the limited on-chip memory available per processor and the rudimentary memory hierarchies. Time to market and development costs constraints in embedded systems do not always allow for carefully hand tuned parallel applications and the potential of automatic parallelization in this framework needs to be evaluated.

This paper addresses several issues in regard to this goal. First, we have analyzed and implemented various hardware supports for synchronization mechanisms each presenting different area-performance tradeoffs in order to select the most suitable synchronization which would best support parallelization.

It clearly appears that a dedicated synchronization NOC with dedicated on-chip memory for synchronization lock variables have shown the best performance. To the best of our knowledge this paper is the first paper to evaluate various synchronization mechanisms on an actually implemented 16PE multi-core with a network on chip.

Second, we have conducted several automatic parallelization experiments on a single chip embedded multi-core system. Our platform is composed of 16 PE with 4 external DDR. Experiments on the selected applications show that the automatic parallelization can hardly efficiently exploit more than 8 processors. The number of external DDR resulting from the single chip package pins constraints reduces memory access concurrency and cannot match the communication concurrency potential allowed by the NOC.

Third, consequently to the findings of point 2 we evaluated the potential of multiprogramming performance. Multiprogramming for the considered applications exhibits memory accesses, synchronization and communication patterns which allows a better use of the platform.

In our future work, the platform generation flow will be enriched by introducing a multi-objective optimization engine that takes into full consideration the performance/cost influential parameters, algorithmic or architectural, and works in an iterative manner for the generation of a cost-effective, application oriented high performance multi-core implementation.

Chapter 4

Mapping middleware on Distributed Networked Embedded Systems

An increasing number of systems are composed of a collection of various devices interconnected by a network, where each individual device performs functions that involve both local interaction and remote interaction with other devices of the system. On the other hand, the users interact with internet applications through a variety of devices, whose characteristics and performance figures span and increasingly wide range.

Stimulated by the growth of network-based applications, the middleware technologies are more and more important. In a distributed computing system, the middleware is defined as the software layer that lies between the operating system and the applications on each site of the system. By hiding the heterogeneity of the underlying architecture, the operating system, the programming language, the middleware facilitates software integration, enhances portability of software components and interoperability between applications developed by different enterprises.

4.1 CORBA, e/CORBA and OmniORB

CORBA is the acronym for Common Object Request Broker Architecture. It is a potential and wide-accepted middleware standard developed by the Object Management Group (OMG) [6]. It is OMG's showcase specification for application interoperability independent of platforms, operating systems, and programming languages - even of networks and protocols.

From the first publication of CORBA 1.0 in October 1991, the CORBA specification has evolved through various completion and modifications and arrived at version 3.0 in July 2002. The CORBA versions are usually referred to as CORBA 2 or CORBA3, which in fact are complete releases of the entire CORBA specification. Because OMG increments the major release number only when they make a significant addition to the architecture, these terms become shorthand for just the significant addition. So, "CORBA 2" sometimes refers to CORBA interoperability and IIOP protocol, and "CORBA 3" sometimes refers to the CORBA Component Model. For the CORBA transport mechanism discussion, the CORBA/IIOP Specification is the right place to go.

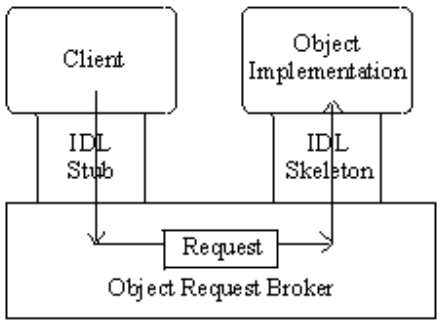

Figure 38 A client sending a request to an object implementation[6]

The CORBA architecture is built upon a collection of objects that provides services to clients. An object is an identifiable, encapsulated entity that provides one or more services, while a client of service is any entity capable of requesting the service. The requestors of services (clients) are isolated from the providers of services by a well defined encapsulation interface as shown in Figure 38. In this figure, the Client wants to perform an operation (request) on the object, whose code and data are implemented in the Object Implementation. The ORB is responsible for all of the mechanisms required to find the object implementation for the request, to prepare the object implementation to receive the request and communicate the data corresponding to the request.

The interface the client sees is completely independent of where the object is located, what programming language is implemented, or any other aspect that is not reflected in the object's interface. The interfaces the client calls and the object implementation provide are defined in the OMB Interface Definition Language (IDL). In particular, the clients are isolated from the

implementation of services as data representations and executable code provide for the location, language and architecture transparency.

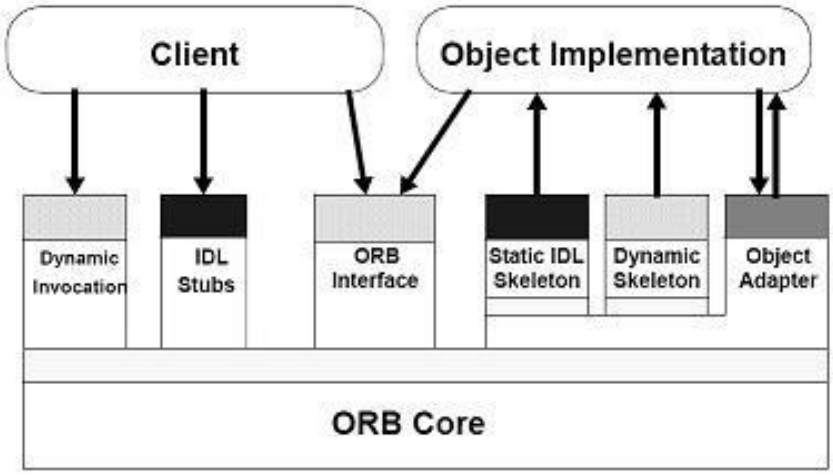

Figure 39 The structure of Object Request Interfaces

Figure 39 shows the structure of an individual Object Request Broker (ORB). The interfaces to the ORB are shown by striped boxes, and the arrows indicate whether the ORB is called or performs an up-call across the interface.

To make a request, the Client can use the Dynamic Invocation interface or an OMG IDL stub. The Client can also directly interact with the ORB for some functions. The Object Implementation receives a request as an up-call either through the OMG IDL generated skeleton or through a dynamic skeleton. The Object Implementation may call the Object Adapter and the ORB while processing a request or at other times.

The definitions of the interfaces to objects can be defined in two ways. The interfaces can be defined statically in an interface definition language, called the OMG Interface Definition Language (OMG IDL), which defines the types of objects according to the operations that may be performed on them and the parameters for those operations. Alternatively the interfaces can be added to an Interface Repository service; this service represents the components of an interface as objects, permitting run-time access to these components.

The client performs a request by having access to an Object Reference for an object. The client initiates the request by calling the stub routines or by constructing the request dynamically, as illustrated in Figure 40.

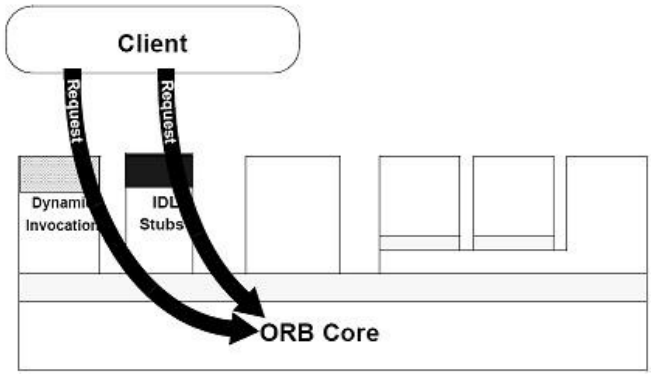

Figure 40 A client using the stub or dynamic invocation interface

The ORB locates the appropriate implementation code, transmits parameters, and transfers control to the Object Implementation through an IDL skeleton or a dynamic skeleton, as shown in Figure 41. When the request is complete, the control and output values are returned to the client.

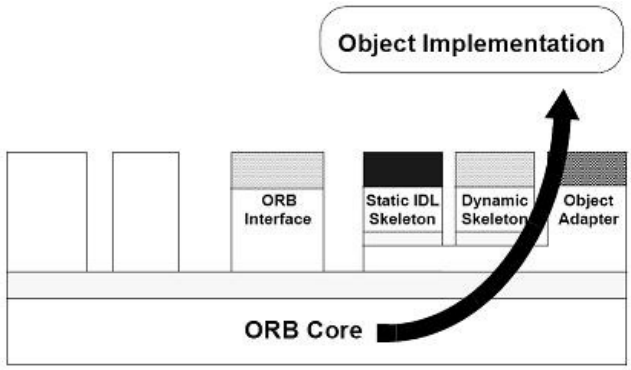

Figure 41 An Object Implementation receiving a request

Interoperability is another important specification of CORBA that offers support for networks of objects managed by multiple, heterogeneous CORBA-compliant ORBs.

4.1.1 CORBA interoperability and GIOP/IIOP

The ORB interoperability [7] specifies a comprehensive, flexible approach for supporting networks of objects that are distributed across and managed by multiple, heterogeneous

CORBA-compliant ORBs -- “interORBability”. The elements of interoperability are as follows:

- An ORB interoperability architecture
- An Inter-ORB bridge support
- General and Internet Inter-ORB Protocols (GIOPs and IIOPs)

The ORB Interoperability architecture provides a conceptual framework for defining the elements of interoperability and for identifying its compliance points. It also characterizes new mechanisms and specifies conventions necessary to achieve interoperability between independently produced ORBs.

The inter-ORB bridge support element specifies ORB APIs and conventions to enable the easy construction of interoperability bridges between ORB domains.

The General Inter-ORB Protocol (GIOP) element specifies a standard transfer syntax (low-level data representation) and a set of message formats for communications between ORBs. The GIOP is specifically built for ORB to ORB interactions and is designed to work directly over any connection-oriented transport protocol that meets a minimum set of assumptions. While versions of GIOP running on different transports would not be directly interoperable, their commonality would allow easy and efficient bridging between such networking domains.

The Internet Inter-ORB Protocol (IIOP)® element specifies how GIOP messages are exchanged using TCP/IP connections. The IIOP specifies a standardized interoperability protocol for Internet, providing “out of box” interoperation with other compatible ORBs based on the most popular product- and vendor-neutral transport layer.

The GIOP is designed to be implementable on a wide range of transport protocols. The objective is to draw analogy between the TCP/IP mapped GIOP transportation and a proprietary on-chip communication protocol used in the embedded context that will be

discussed later, and discuss the feasibility of mapping GIOP message transportation on this protocol.

4.1.2 CORBA/e

In today's world, the stand-alone systems are becoming a thing of the past. The embedded processor environments are networked and highly interconnected. The software must cope with the communications and interoperability issues, while delivering the same reliability and performance as the isolated embedded system of the past. Embedded system software development becomes a more and more expensive and time-consuming task. But with a solid middleware architecture, this investment can pay dividends across many generations of technology. For the developers of real-time and embedded systems, CORBA/e is ideally suited to the challenges of today's mission-critical environment.

CORAB/e is a specification targeted to applications that will be executing on an embedded processor with constrained resources and/or that require predictable real-time behavior. [8] The architecture and specifications described in the manual are aimed at software designers and developers of Distributed Real-Time Embedded (DRE) Systems who want to produce embedded applications that comply with OMG standards for the Object Request Broker (ORB). CORBA/e has been designed to have the best of both worlds: dramatically minimizing the footprint and overhead of typical middleware, while retaining the core elements of interoperability and real-time computing that support optimized distributed systems. There are two CORBA/e profiles, the CORBA/e Compact and the CORBA/e Micro Profile, separately tailored for minimal and single-chip environments.

The CORBA/e Compact Profile merges key features of standard CORBA suitable for resource-constrained static systems (no DII, DSI, Interface Repository, or Component support) and Real-time CORBA into a powerful yet compact middleware package that interoperates with other CORBA clients and servers of every size, executes with the deterministic characteristics required by a true real-time platform.

The CORBA/e Micro profile shrinks the footprint even more, small enough to fit low-powered microprocessors or digital signal processors. This profile further eliminates the Valuetype, the Any type, most of the POA options preserved in the Compact Profile, and all of the Real-time functions excepting only the Mutex interface. In exchange for these limitations, the profile defines a CORBA executable that vendors have fit into only tens of kilobytes - small enough to fit onto a high-end DSP or microprocessor of a hand-held device.

The developers of real-time embedded distributed system must pay special attention to resources utilization and to the predictability of system execution. In order to provide support for the development of real-time systems, CORBA/e provides handles for managing resources and end-to-end predictability.

To decide a priori if a real-time requirement is met, the system must behave predictably. This can only happen if all the parts of the system behave deterministically and if they "combine" in a predictable way. The real-time interfaces and mechanisms provided by CORBA/e facilitate a predictable combination of the ORB and the application. The application manages the resources by using real-time CORBA/e interfaces and the ORB's mechanisms coordinate the activities of the application. The real-time ORB relies upon the RTOS to schedule threads that represent activities being processed and to provide mutexes to handle any resource contention.

4.2 omniORB

There have been commercial as well as academic efforts for implementing ORB. Commercial ORBs include Orbix and Orbacus from Iona, Visibroker from Borland, and the ORBexpress series from OIS. On the academic side, there are TAO from Washington University [9], omniORB [10] from the former AT&T Laboratory in Cambridge, and the GOPI [11] from Lancaster University.

We chose the omniORB-4.1.3 [12] as the middleware implementation for the distributed application development. omniORB is an Object Request Broker (ORB) that implements the 2.6 specification of the OMG CORBA. Its various characteristics like light-weight, high

performance and the GPL license policy make it a potential candidate for the development of an embedded distributed system based on the proprietary transportation layer. We list below some features of the omniORB:

(1) Multithreading

OmniORB is fully multithreaded. With default policies, there is at most one call in flight in each communication channel between two address spaces at any one time. To maximize the level of concurrency, new channels connecting the two address spaces are created on demand and cached when there are concurrent calls in progress, while each channel is served by a dedicated thread. More over, the throughput is maximized in processing large call arguments by sending large data elements as soon as they are processed while the other arguments are being marshaled. From version 4.0 onwards, omniORB allows a flexible thread pooling policy and supports sending multiple interleaved calls on a single connection which allows omniORB to scale to large numbers of concurrent clients.

(2) Portability

OmniORB is designed to be portable. It runs on many flavors of Unix, Windows, several embedded operating systems, and less known systems such as OpenVMS and Fujitsu-Siemens BS2000. It is designed to be easy to port to new platforms. The IDL to C++ mapping for all target platforms is the same.

OmniORB uses true C++ exceptions and nested classes. It keeps to the CORBA specification's standard mapping as much as possible and does not use the alternative mappings for C++ dialects. The only exception is the mapping of IDL modules, which can use either namespaces or nested classes.

OmniORB relies on native thread libraries to provide multithreading capability and uses a small class library, namely omnithread, to encapsulate the APIs of the native thread libraries. It is easy to port omnithread to any platform that either supports the POSIX thread standard or has a thread package that supports similar capabilities. [12]

4.3 Analysis Case studies : Performance and Scalability

From now, we will present a case study in which a CORBA based distributed embedded systems developed. Four ML403 evaluation cards of Xilinx are deployed in the system. The omniORB-4.1.3 is used for the development of the distributed applications. The Client/Server communication is based on the IIOP protocol via the Ethernet. Several benchmarking tests are carried out to evaluate the performance of the system in terms of latency and throughput.

4.3.1 Distributed Embedded System Hardware Architecture

4.3.1.1 ML403 board

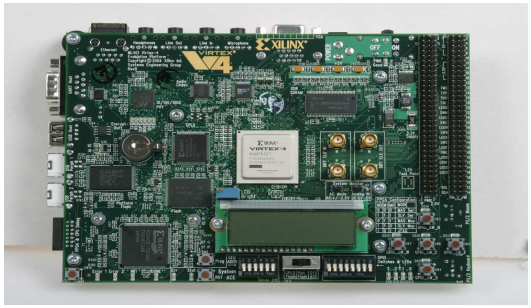

Figure 42 ML403 board from Xilinx

The embedded distributed system is based on four ML403 board from Xilinx as illustrated in Figure 42. The ML403 board features a Virtex-4FX12 FPGA chip on which one ppc405 processor is integrated. The FPGA contains 648 Kb on-chip two-port ram blocks (BRAM). The PPC405 processor is a 32-bit implementation of the PowerPC architecture targeting the embedded application. It is equipped of a 5-stage pipeline and 16KB instruction cache and 16KB data cache. It can work at a frequency as high as 450MHz. For the inter-card communication, we used the Ethernet switch from Netgear which supports 10/100Mbs connections. Figure 43 is a block diagram of the architecture of the ML403 board while Table 11 summarizes the features of the principal peripherals and ports.

Table 11 Principal components of the ML403 board

Class	Components	Description
Vitex-4 FPGA		
	XC4FX12	1
	Processor PPC405	1
	Slices	5,472
	Block RAM	648Kb
	Ethernet MACs	2
Memory		
	DDR SDRAM	64MB
	ZBT SRAM	1MB
	Compact Flash	512MB
	Linear Flash	8MB
	IIC EEPROM	4kb
Connectors and Interfaces		
	SMA connector (Differential Clocks)	4
	PS/2 Connectors (Keyboard/Mouse)	2
	Audio (In/Out, Microphone, Head Phone)	2
	RS-232 Serial Ports	1
	USB Ports (2 Peripherals/1 Host)	3
	PC4 JTAG	1
	DB15 VGA Display	1
	RJ-45 Ethernet Ports	1
	General-Purpose I/O (Buttons/LEDs)	Several

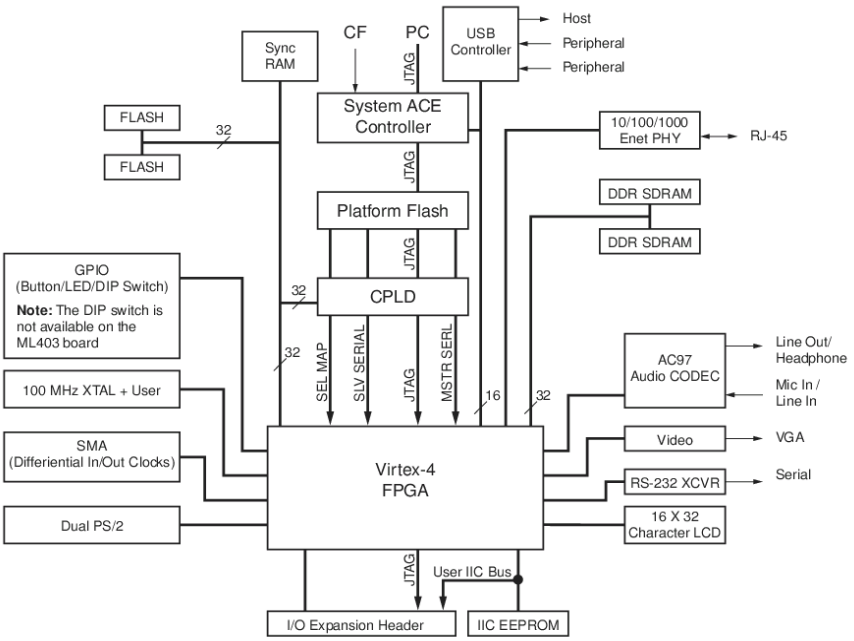

Figure 43 Block diagram of the ML403 board

4.3.1.2 Architecture of the distributed processing node based on a virtex4fx12 FPGA

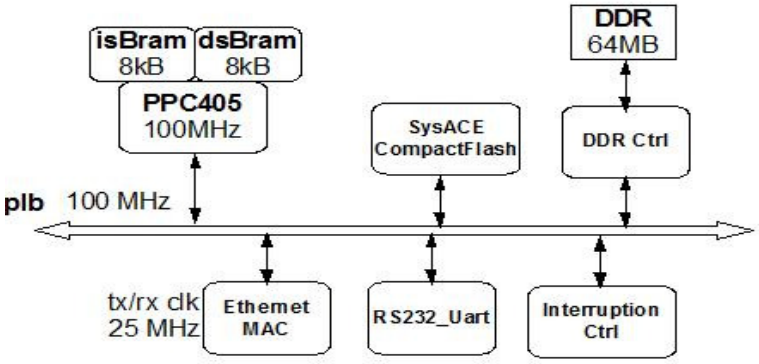

Figure 44 Block diagram of the architecture of the distributed node based on the virtex4fx12 FPGA

The FPGA sub-system is composed of the elements shown in Figure 44: one PPC405 processor that implements the Operating system, the stack TCP/IP; one SysACE CompactFlash controller that connects the processor to the CompactFlash card on which the operating system, communications and computation applications will be supported; one interruption controller; one Ethernet MAC and one RS232_uart. The processor Local Bus

(PLB) from IBM Coreconnect family is used as on chip high performance bus. The resource utilization of the design is summarized in Table 12.

Table 12 Resource utilization

Resource	Utilization	
Number of RAMB16s	23 out of 36	63%
Number of Slices	3641 out of 5472	66%
Number of SLICEMs	126 out of 2736	4%

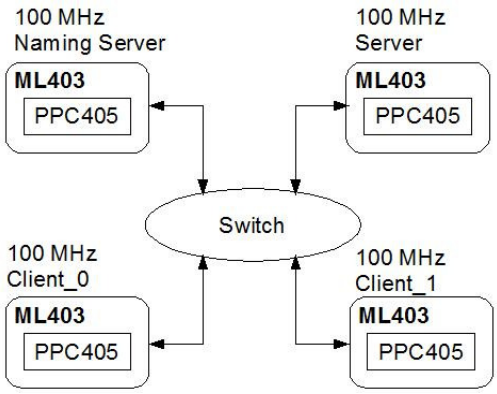

Figure 45 Embedded distributed system based on four FPGA node connected by an Ethernet switch

Figure 45 is a block diagram of the complete distributed platform consisting of four ML403 boards. The cards are connected with each other via an Ethernet switch. There is a great flexibility for the configuration of the cards. They can serve as a Client, Server or as a Common Object Server (COS). In our experiment, we will configure one card as the Naming service server, another as a server, the third and forth as clients for testing the case where there exist concurrent invocations.

4.3.2 Software architecture

As stated before, omniORB-4.1.3 was chosen [12] as the middleware implementation for the distributed application development. The default mechanism for GIOP transportation, IIOP, is used for the Client/Server message transfer. OmniORB is compiled and installed on the Linux 2.6.28 kernel. The Xilinx patched Linux Kernel source from [13] is utilized. It is one Linux kernel distribution equipped with the supplementary supports for Xilinx platforms. The Linux

kernel and the network applications are compiled in a cross environment. The host machine is an Intel CoreTM 2. The tool kit Buildroot is used for the creation of the cross-compilation (i386 – ppc) tool chain and the Linux file system. The file system is written on the ext2 partition of the SysACE CompactFlash of the ML403 platform. The Linux kernel is compiled with the help of Device Tree Generator [14] from Xilinx which is an integrated tool of the Xilinx EDK [15] kit for the automatic generation of the Board Support Package (BSP). It generates the device tree file containing information of the component of the system (memory-mapped address, interruption, driver compatibility, etc.) for the compilation of the Linux kernel. Figure 46 shows the software architecture of the system.

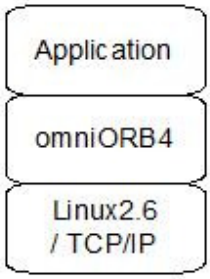

Figure 46 Software architecture of the embedded distributed system

Figure 47 illustrates the complete testing platform:

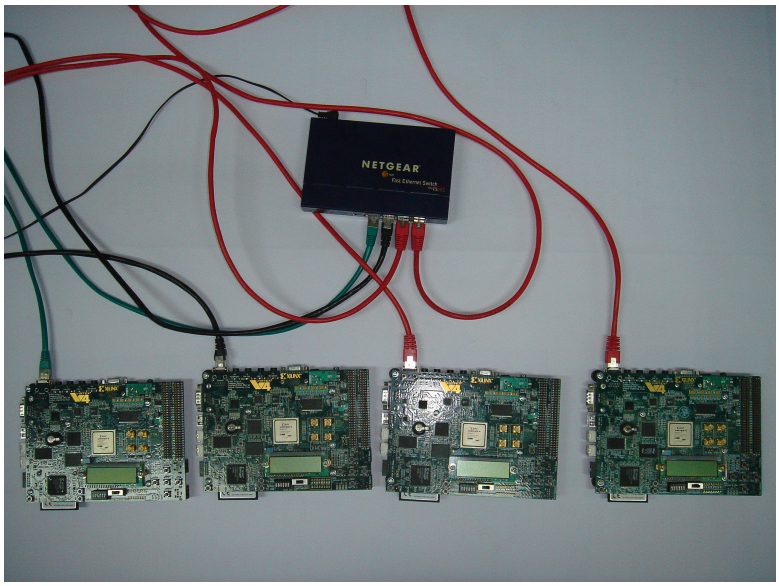

Figure 47 : Platform with four ML403 and a Switch

4.3.3 Performance Evaluation

4.3.3.1 Middleware benchmarking

Because of the diversity of ORB implementations, it is essential to have an open CORBA benchmark well understood and easy to measure. The Open Benchmarking Suite developed by Chares University represents an excellent effort towards the establishment of such a benchmark. [16][17][18]

Latency and throughput are two major factors for the performance evaluation of a communication system. The latency shows the added overhead of message marshalling/unmarshalling, the TCP/IP stack, the time spent on the network, etc, while the throughput presents the capacity of the system to process large quantities of data. The results of these two factors affect how the execution time of a transaction is felt by the user. The Open Benchmarking Suite developed by Charles University also proposes some precious remarks concerning the precision issues when benchmarking CORBA [19].

The experiments are carried out on the 4-FPGA-based platform with the PPC405 processors configured to 100MHz frequency, and equipped with 64MB main memory each. The platform configuration is: a Client/Server pair in which the client makes round trip calls to the server in sending n bytes of data. The C++ application, as well as the omniORB4.1.3, is compiled with the GNU g++ version 4.2.4 with the optimization flat –O2 activated. The execution time per call is measured on the client side by averaging 5, 000 consecutive calls. The IDL file is showed below:

```
interface Echo {
  string echoString(in string mesg);
};
```

The throughput is measured by sending various sized sequence of bytes in a single direction. The IDL source file in defined below:

```
interface BulkTransfer {
        oneway void transfer(in string data);
};
```

The throughput under the concurrent invocation condition is also tested by executing two clients sending requests to the same server at the same time.

4.3.2.2 Performance results

The measured results for the intra-machine communications as well as for the inter-machine communications are presented in Table 13.

Table 13 Time of round-trip Echo function without any message

Platform	Transport	Time per call (µs)
Linux PPC405 100MHz (Gcc-4.2.4-O2)	TCP/IP intra-machine loopback TCP/IP inter-machine	2121 2154

OmniORB4 takes 2121 µs for the intra-machine communication and 2154 µs for the inter-machine communication. For comparison, we cite the results from the work of [10] to show the round-trip echo call performance measured on other platforms. The results are showed in Table 14.

When comparing the two tables, we notice that the latency of the PPC405 platform is 112% (Pentium Pro 200 Mhz) ~ 69% (Pentium 166 Mhz) larger in inter-machine communication and 500% (Pentium Pro 200 Mhz) ~ 112% (Pentium 166 Mhz) larger in intra-machine communication. The comparison results come from the lower frequency and the more constrained resources (memory/thread) of our system compared to the other non-embedded environments. Table 15 (Figure 48) shows the throughput results in sending 1 MB of data by the client to the server in a one-way operation. The throughput with different packet sizes is measured for intra-machine communications as well as inter-machine communications when the client is sending 1MB of data.

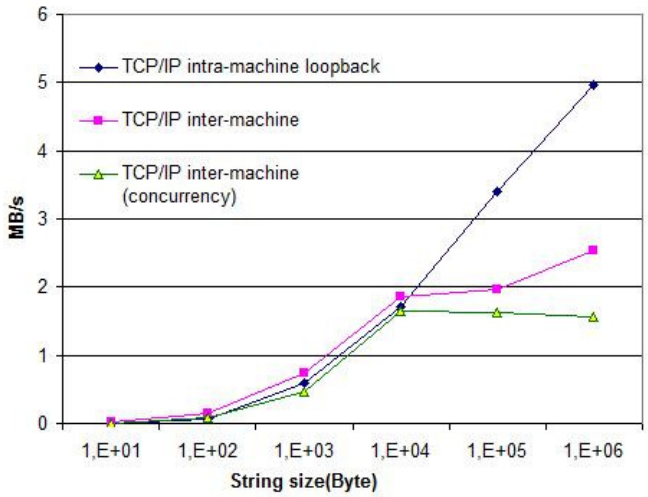

Figure 48 Throughput for 1 MB transfer in one-way invocation

Table 14 Latency measurements for other architectures

Platform	Transport	Time per call (µs)
Linux Pentium Pro 200 Mhz (gcc-2.7.2 no compiler optimization)	TCP/intra-machine	340
	TCP/ethernet (ISA card)	1000
	TCP/ATM	440
Windows NT 4.0 Pentium Pro (MS Visual C++ -- O2)	TCP/intra-machine	360
	TCP/ethernet (ISA card)	1000
Digital Unix 3.2 DEC 3000/600 (DEC C++ -- O2)	TCP/intra-machine	750
	TCP/ethernet	1050
Windows 96 Pentium 166 Mhz (MS Visual C++ -- O2)	TCP/intra-machine	1000
	TCP/ethernet (PCI card)	1250
Solaris 2.5.1 Ultra 1 167 Mhz (Sunpro C++ -- fast)	TCP/intra-machine	540
	TCP/ethernet	710

Table 15 Throughput for 1 MB transfer in one-way invocation

Packet size	Transport	Time per call (μs)
10	TCP/IP intra-machine loopback	0.007
	TCP/IP inter-machine	0.013
	TCP/IP inter-machine (concurrence)	0.008
100	TCP/IP intra-machine loopback	0.07
	TCP/IP inter-machine	0.15
	TCP/IP inter-machine (concurrence)	0.09
1,000	TCP/IP intra-machine loopback	0.59
	TCP/IP inter-machine	0.74
	TCP/IP inter-machine (concurrence)	0.47
10,000	TCP/IP intra-machine loopback	1.72
	TCP/IP inter-machine	1.86
	TCP/IP inter-machine (concurrence)	1.64
100,000	TCP/IP intra-machine loopback	3.40
	TCP/IP inter-machine	1.97
	TCP/IP inter-machine (concurrence)	1.63
1,000,000	TCP/IP intra-machine loopback	4.97
	TCP/IP inter-machine	2.53
	TCP/IP inter-machine (concurrence)	1.57

We noticed that the throughput of the system improves when the packet size increases. With larger packet sizes, the overhead introduced by each invocation, marshalling/unmarshalling for example, is reduced. We can also see that the best performance changes between the intra-machine communication and the inter-machine communication when the size exceeds 1KB bytes. We suppose that in the case of intra-machine invocation, the gain of performance on the communication is reduced by the time consumed in context switches between the client process and the server process. The frequency of context switches decreases when the packet size increases. Therefore at a certain point, the performance is dominated by the communication overhead and the throughput of intra-machine invocation exceeds that of the

inter-machine invocation. In Table 15, the third result of each line shows the throughput in the presence of request conflicts: two clients concurrently invoke the same server object. In that case, the performance deteriorates in average by 30% compared to the case where only one client exists.

4.3.4 Performance of the Server/Client distributed platform when increasing clock frequency

The results obtained in the former section are based on a homogeneous frequency configuration of the four FPGA systems, more precisely the PPC405 processor and the PLB bus. Though architecturally identical, the four FPGA sub-systems serve different roles in the distributed system. Therefore performance requirements are not necessarily the same for a sub-system configured as client compared to one configured as server. The effect of frequency scaling on performance enhancement also varies depending on the role the specific system takes in the distributed system.

A study of the frequency scaling effect on system performance is important to get satisfying performance under minimum energy budget. In our experiments, we defined three basic frequency configurations in terms of PPC405 and PLB bus and their combination in a distributed system is studied.

Basic frequency configuration choices:
PPC405 200MHz / PLB 100MHz (200/100)
PPC405 100MHz / PLB 100MHz (100/100)
PPC405 50MHz / PLB 50MHz (50/50)

Same as in the former section, the two performance metrics measured are latency and throughput. The benchmarking programs are defined in the same way as in the former experiments that we will remind below:

Latency:

```
interface Echo {
```

```
            string echoString(in string mesg);
};
```

Throughput:

```
interface BulkTransfer {
                  oneway void transfer(in string data);
};
```

1. Server side frequency scaling

Naming service configuration and Client configuration are both fixed at 50/50.

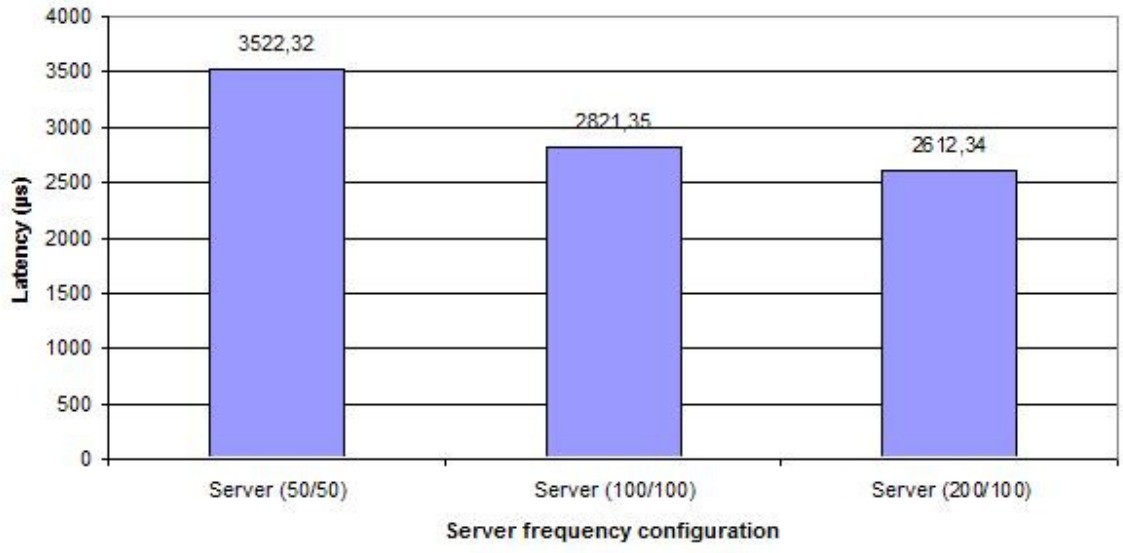

Figure 49 Influence of Server configurations on Latency

As the server configuration migrates from 50/50 to 100/100, the latency of Server/Client system reduces by 20%. When the server configuration moves to 200/100 from 100/100, the latency reduces by 7.4%.

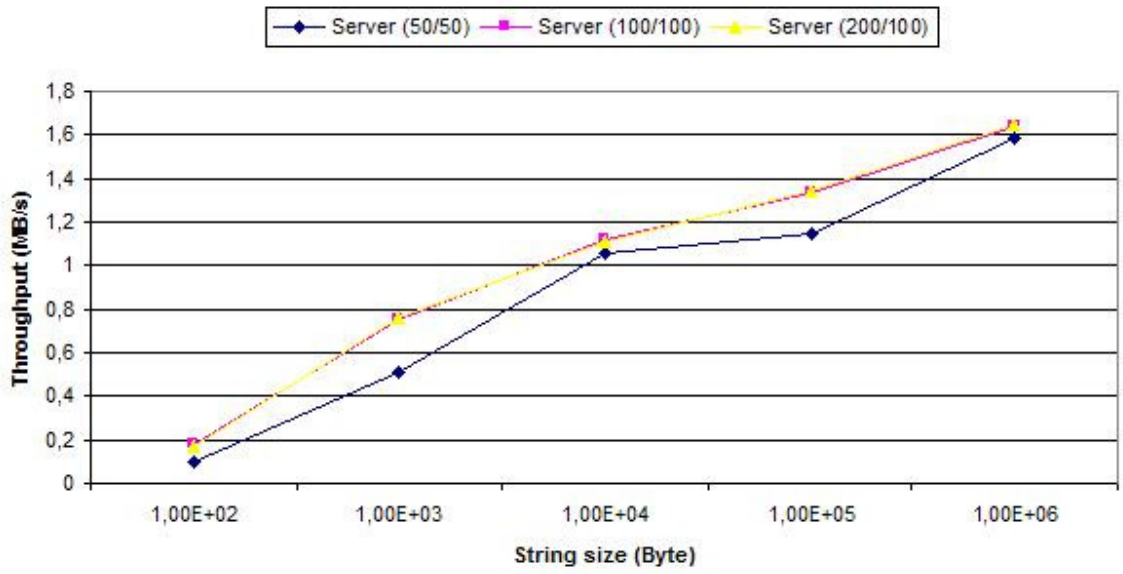

Figure 50 Influence of Server configurations on Throughput

According to the Figure 2, it is noted that when server configuration moves from 50/50 to 100/100, a significant throughput improvement is introduced with a maximum increase of 32% for a packet size of 1.00E+03 bytes, and 3.2% when the packet size is 1.00+06 bytes.
There is no visible throughput improvement when the server configuration migrates from 100/100 to 200/100.

2. Client side frequency scaling
The Naming service configuration is fixed at 100/100, while the Server configuration is fixed at 200/100.

As the client configuration migrates from 50/50 to 100/100, the latency of Server/Client system reduces by 20%. When the server configuration moves to 200/100 from 100/100, the latency reduces by 8.4%.

When the client configuration changes from 50/50 to 100/100, the throughput of the Server/Client model increases by 4%, 55%, 60%, 70%, 70% corresponding to the Packet size configuration of 1.00E+02, 1.00E+03, 1.00E+04, 1.00E+05, 1.00E+06 respectively.

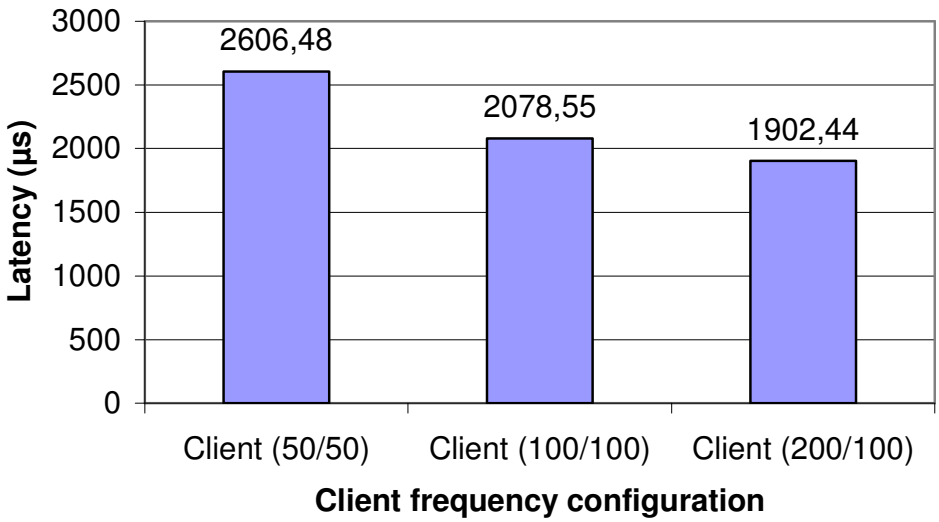

Figure 51 Influence of Client configurations on Latency

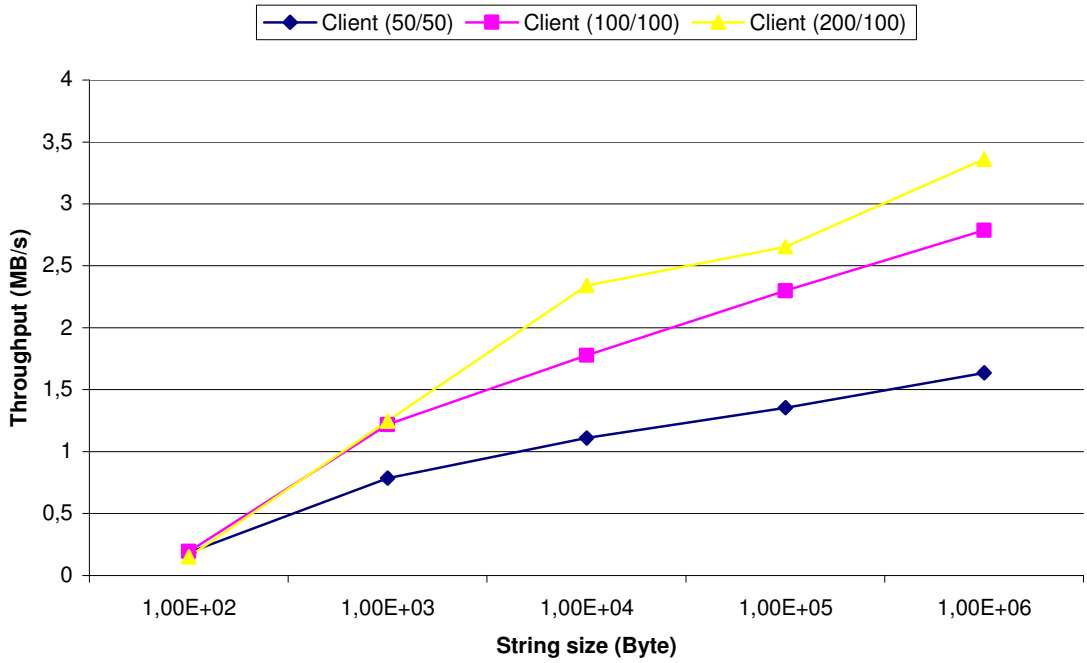

Figure 52 Influence of Client configurations on Throughput

When the client configuration changes from 100/100 to 200/100, the throughput of the Server/Client model increases by 2%, 24%, 15%, 21%, with a packet size of 1.00E+03, 1.00E+04, 1.00E+05, 1.00E+06 correspondingly. An exception arises when packet size equals

1.00E+02 bytes, which shows tiny throughput degradation despite the fact of frequency scaling.

3. Naming server side frequency scaling

Server configuration is fixed at 200/100, while Client configuration is fixed at 50/50.

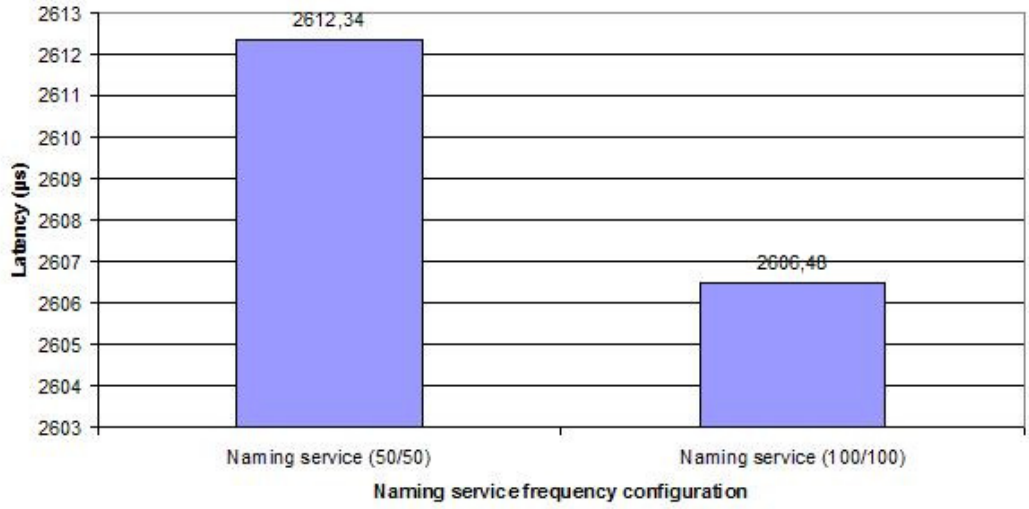

Figure 53 Influence of Naming service configurations on Latency

As the client configuration migrates from 50/50 to 100/100, the latency of Server/Client system reduces by 0.2%.

From Figure 54, it is noted that Naming service frequency scaling introduces negligible throughput improvement.

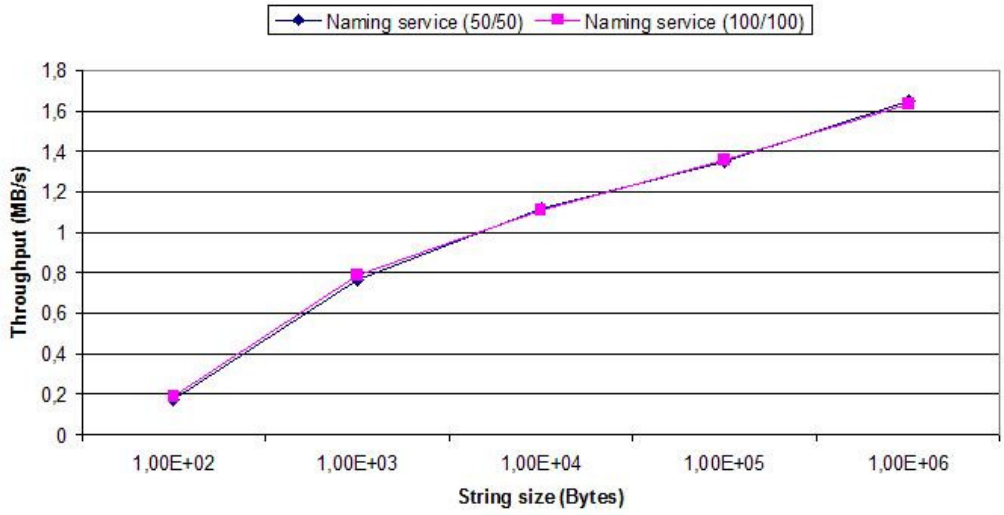

Figure 54 Influence of Naming service configurations on Throughput

4. Conclusion

The naming service omniNames is executed on a separate ml403 platform from either the client or the server. From the latency and throughput results we can see that frequency scaling on the machine on which Naming service is hosted doesn't introduce performance improvement. This is due to the fact that the performance metrics are obtained by averaging a large number of calls in the case of latency or by operating on a large problem size in the case of throughput. Therefore, the object reference query operation via the naming server that takes place once before all the object invocations is amortized by the following calculations. So in systems where there are only few object reference queries compared to the object invocation quantities on these references, which is the case of most distributed systems, the performance of the Naming service is not crucial.

We also noted that the Client side performance and the Server side performance have equally important roles in deciding the responsiveness of the system. The frequency scaling effects on system latency are 20%, 7.4% for the server, and 20%, 8.4% for the client. In the development of time critical distributed systems, it is very important to choose equivalent server / client configurations in order to achieve optimum system responsiveness.

When it comes to system throughput, the results show perfect scaling effects at the client side. The frequency increase from 50/50 to 100/100 results in more than 50% throughput increase for most of packet size configurations. While frequency increase from 100/100 to 200/100 introduces about 20% throughput increase for most of packet size configurations.

However the frequency scaling effects on the server side is not as clear. This can be explained in examining the definition of the IDL interface for throughput that is a one way invocation without the Client waiting for a response from the server.

4.3.5 Distributed Client-server with Multiprocessor Networked Embedded Latency and Bandwidth Analysis

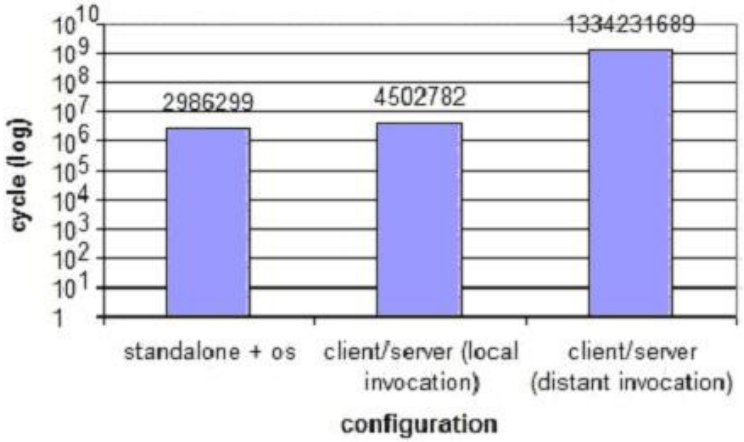

Figure 55 FFT distributed computing

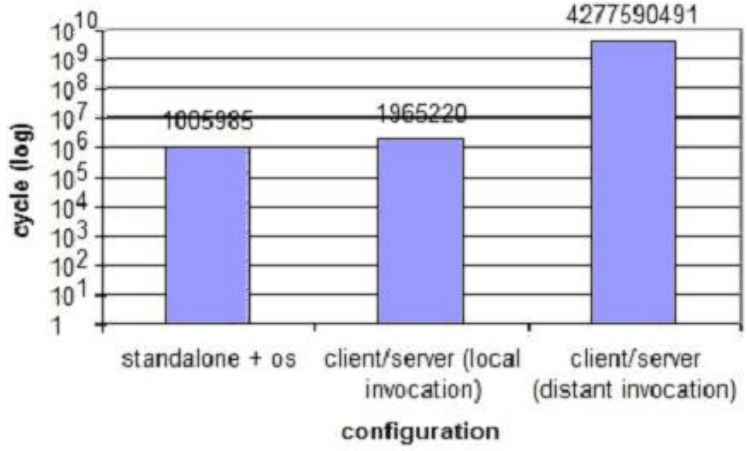

Figure 56 Matrix multiplication distributed computing

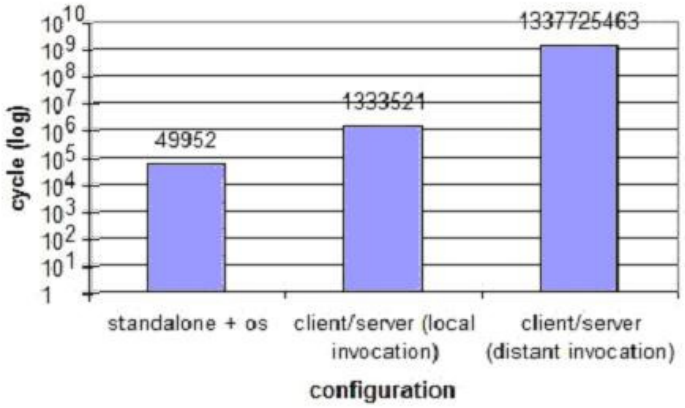

Figure 57 Qam-16 distributed computing

The Corba model will necessarily introduce overhead in terms of communication protocol stack. However the benefits of an easy deployment and integration should be considered.

4.4 Hybrid programming model

In this section, we will explore the hybrid programming model in introducing local parallel processing elements in the former distributed system. Figure 58 is a block diagram of the complete distributed platform consisting of four ML403 boards. The grey section in the center represents the globally distributed view in which four PPC based computing systems are interconnected via an Ethernet switch. This architecture leaves developers a great flexibility for the configuration of the cards. One can serve as a Client, a Server or as a Common Object Server (COS). In our experiment, we will configure one card as the Naming service server, another as the server, the third and forth as clients for testing the case in which there exists concurrent invocations. Table 16 summarizes the resource utilization of each FPGA sub-system.

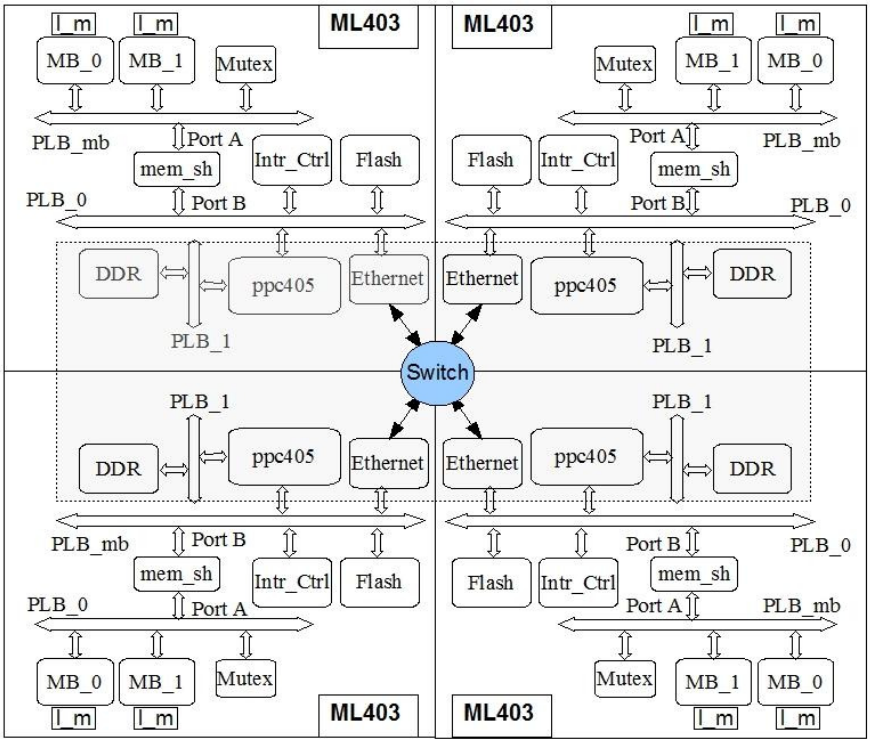

Figure 58 Block diagram of the embedded distributed system with parallel processing units (ml403 x 4)

Table 16 Resource utilization

Number of RAMB 16s	33 out of 36	91%
Number of Slices	5407 out of 5472	98%
Number of SLICEMs	579 out of 2736	21%

Some experiments have been done on this platform. The results are shown in the following figures for Lenth-64 FFT, Length-15 viterbi decoder, Qam-16 modulation respectively.

(A) 64-point FFT

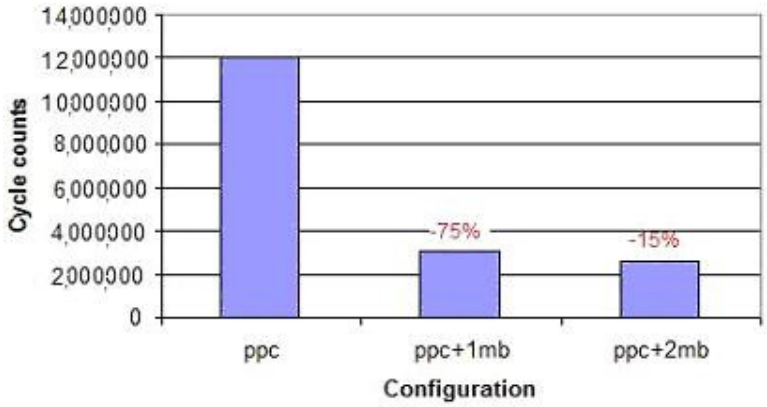

Figure 59 64-point single precision floating point FFT

The reason why ppc takes three times as much as that of single microblaze is due to the fact that ppc is configured at the same frequency as the microblaze and the access to the DDR memory is much slower than the bram access. The focus here, however, is on the parallel programming effects of the microblaze couple, and the execution result of powerpc is presented here just as a reference. This is true for the following applications. The ppc+1MB system has a reduction of 75% of execution time. Partly it is because of the addition of parallel processing and also because the microblaze benefits from its local memory. The addition of the second microblaze only adds more exploitation of parallelism which is closely related to the parallelism within the applications.

(B) Length-15 Viterbi decoder

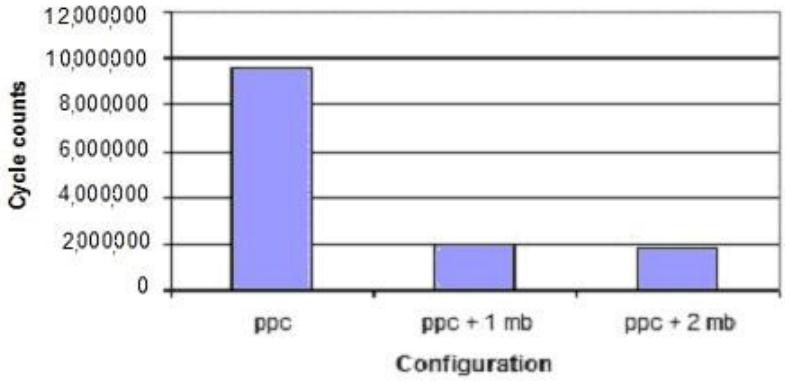

Figure 60 Length-15 viterbi decoder

As shown in Figure 60, moving from 1 microblaze to 2 reduces the execution cycle by 10%.

(C) Quadrature amplitude modulation (QAM)

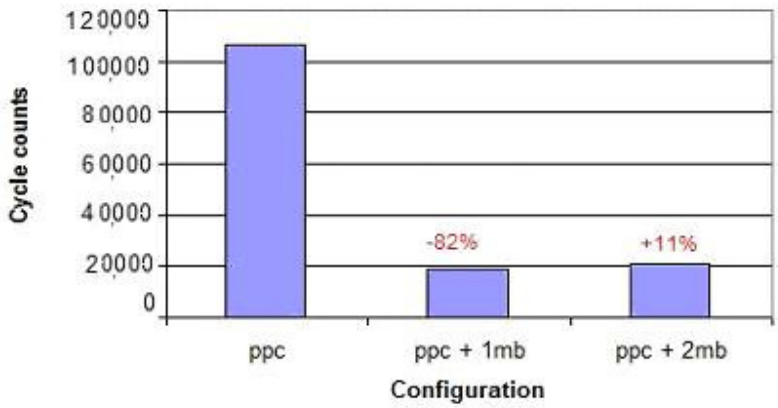

Figure 61 Qam-16 modulation (16-symbol outputs)

As shown in Figure 61, increasing the number of microblaze cores from one to two slightly deteriorates the performance by 11%. This is because the constellation alphabet is pre-calculated and offered to the microblaze core. The latter only calculates the gray code for the input data, and then indexes to the alphabet to find the corresponding symbol, which is a very trivial calculation.

If resources permit, the above system can be readily extended to a many-PEs hybrid architecture as shown in Figure 62.

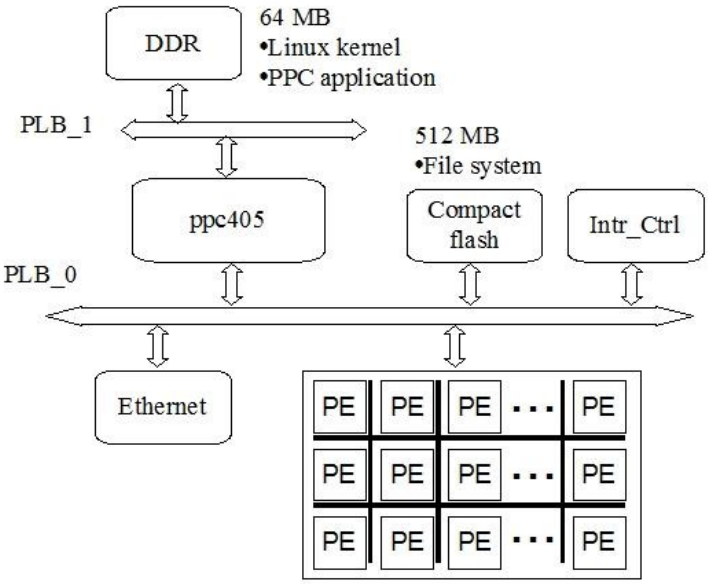

Figure 62 Hybrid architecture with mesh-like parallel processing elements

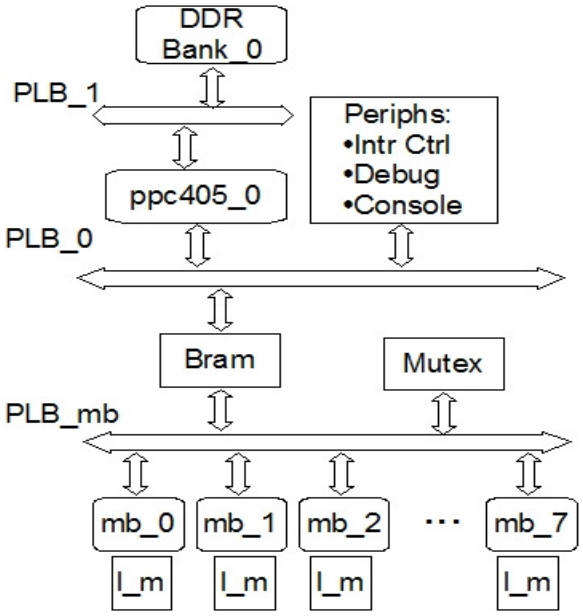

Figure 63 PPC405 + microblaze x 8

We implemented a ppc405 computing system with 8 microblaze processing elements as illustrated in Figure 63. Due to resources constraints, we synthesized the design with the Xilinx virtex4 FX140 FPGA. The results of resource utilization are summarized in Table 17.

Table 17 Resource utilization

Number of RAMB 16s	140 out of 552	25%
Number of Slices	6417 out of 63168	10%
Number of SLICEMs	824 out of 31584	2%

The hybrid programming model combines the parallel programming (parallel programming) with distributed programming. In this chapter, we detailed the construction of a parallel node for a distributed system. The performance evaluation results are done under a single application environment. In a real system, the PPC processor would be in charge of other applications as well as communication and control tasks. The benefits of adding more parallel processing elements to which the PPC can distribute calculation loads would be more important.

4.5 Multiobjective Optimization Based Automatic Design flow for CORBA based Distributed Networked Embedded Systems

The design flow based on multi-FPGA distributed embedded system is presented in Figure 64. The level-1 distributed system generator takes as input an application abstracted with Kahn Process Network (KPN). Processes are mapped to separated processors without considering the underlying SoC architecture. A function profiling is then carried out to determine the most time consuming process. The profiling results are used to filter out the system performance critical function that will then be accelerated by parallel programming while the other non-critical functions are bypassed to the final stage. The performance critical functions are parallelized by an automatic parallelizer. A default parameter set including information as memory hierarchy, number of processors, is first provided and is finely tuned during the local optimization loop, represented by the arrow connecting the “Platform gen” block and the “Parallelization” block. The network on chip topology is synthesized. Then the platform is generated by the platform generation engine. The parallelization and the bypass branch are then combined and deployed on the multi-FPGA based distributed/parallel platform. If the system requirements like performance, resources are satisfied while keeping a minimum frequency configuration to reduce the energy consumption, the flow ends. If one of

the above requirements is not met, we will loop back to the first level distributed system generation block through a multi-objective optimization engine that is responsible for resource or frequency optimization, parameter tuning, etc.

The multi-FPGA based designed flow serves as an important concept that will be extended for the single-chip based design flow that is discussed in the following chapters. However due to the resources constraints of the "ML403" test boards, we could not test an extended version of the hybrid architecture concept beyond one server/client plus two PE accelerators. It was anyway an important preliminary step for the validation of the concept. The natural next design phase is to move to a single large-scale FPGA chip.

4.6 Conclusion

This chapter has presented the OMG CORBA specifications including eCORBA. The omniORB has been used as middleware for the distributed application development due to its various characteristics like light-weight, high performance, and the GPL license policy.

A distributed embedded system based on CORBA and implemented on multiple FPGA was developed as a first step towards building a single-chip SCA compliant Software Defined Radio. We used the IIOP, the default mechanism for GIOP transportation, for the Client/Server invocation communication. OmniORB was compiled and installed on the Linux 2.6.28 kernel.

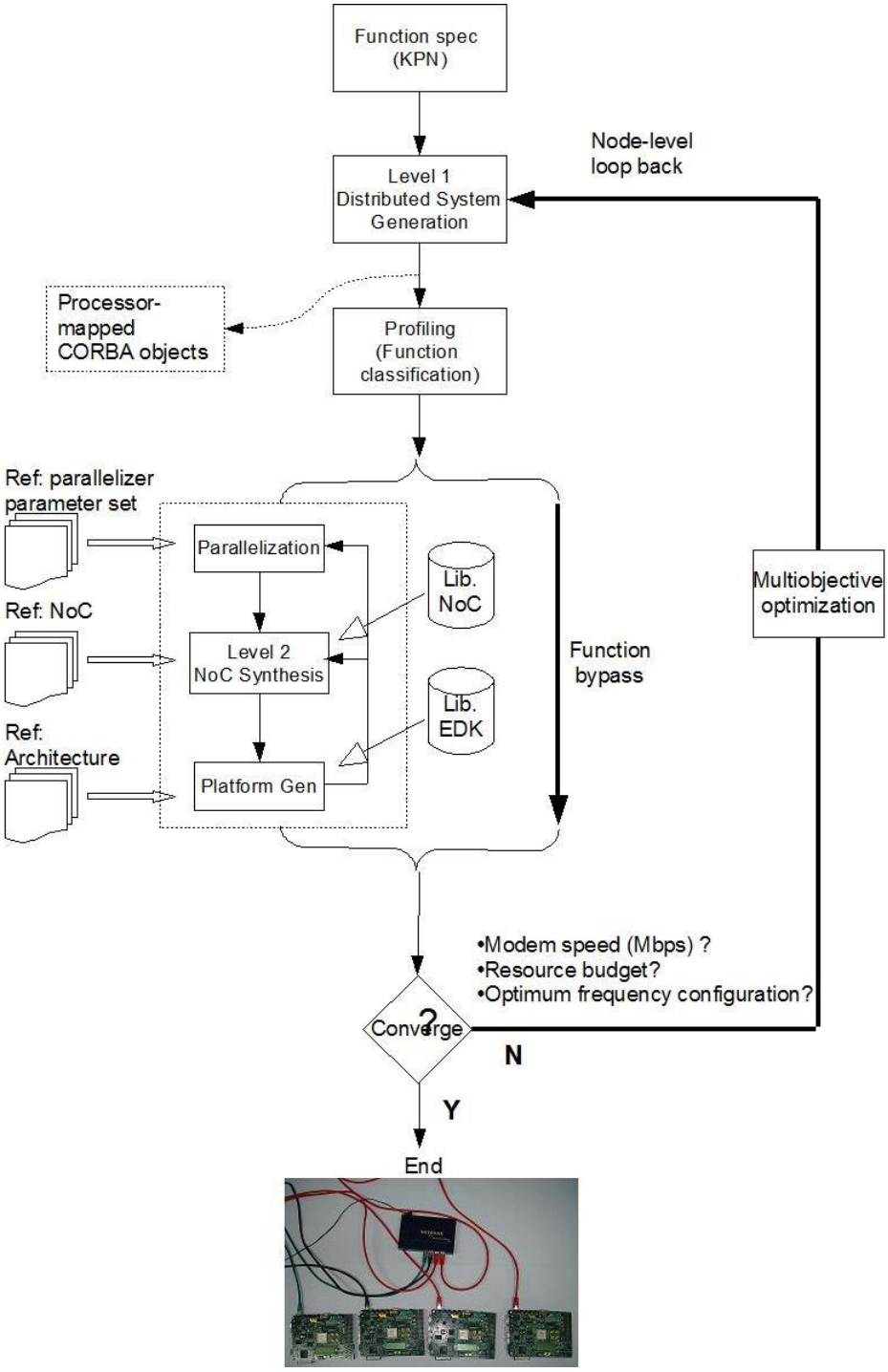

Figure 64 Design flow based on the hybrid programming model with multi-objective optimization

CORBA performance is evaluated on this platform. Also the frequency scaling impact on system performance was studied. The results serve as a good indication for system architects when building such distributed systems under stringent energy budget.

Chapter 5

Middleware mapping on Single Chip Multiprocessors

5.1 State of the art of Middleware Mapping on Multi-processor Platforms.

Heterogeneous multiprocessing is the future of chip design with the potential for tens to hundreds of programmable elements on single chips. Middleware that was traditionally used in the internet domain must be adapted to be applied on the single chip in order to mask the underlying architecture and OS heterogeneity, thus enabling application development to be carried in a portable and uniform way.

Authors in [139] talks about the trend of domain specific software programmable, heterogeneous SoCs in reducing nonrecurring expenses by providing a flexible platform. Better application programming tools are needed for effective utilization of such platforms by end-users. The article provides the MultiFlex programming model that inspired by mainstream approaches for large system development while adapted and constrained for the SoC domain. The Distributed System Object Component (DSOC) model, resembling CORBA enhanced with hardware object request broker, is provided to support the heterogeneous distributed computing. The SIDL interface defines a language-neutral representation of object call to enable interoperability between object implementations. The DSOC objects, combined with the SIDL interface compiler, allow an easy mapping of tasks to the platform hardware or software. The StepNP SoC platform is developed as a simulation environment for the

programming model whose architecture combined with an example of network domain application mapping is illustrated in Figure 65:

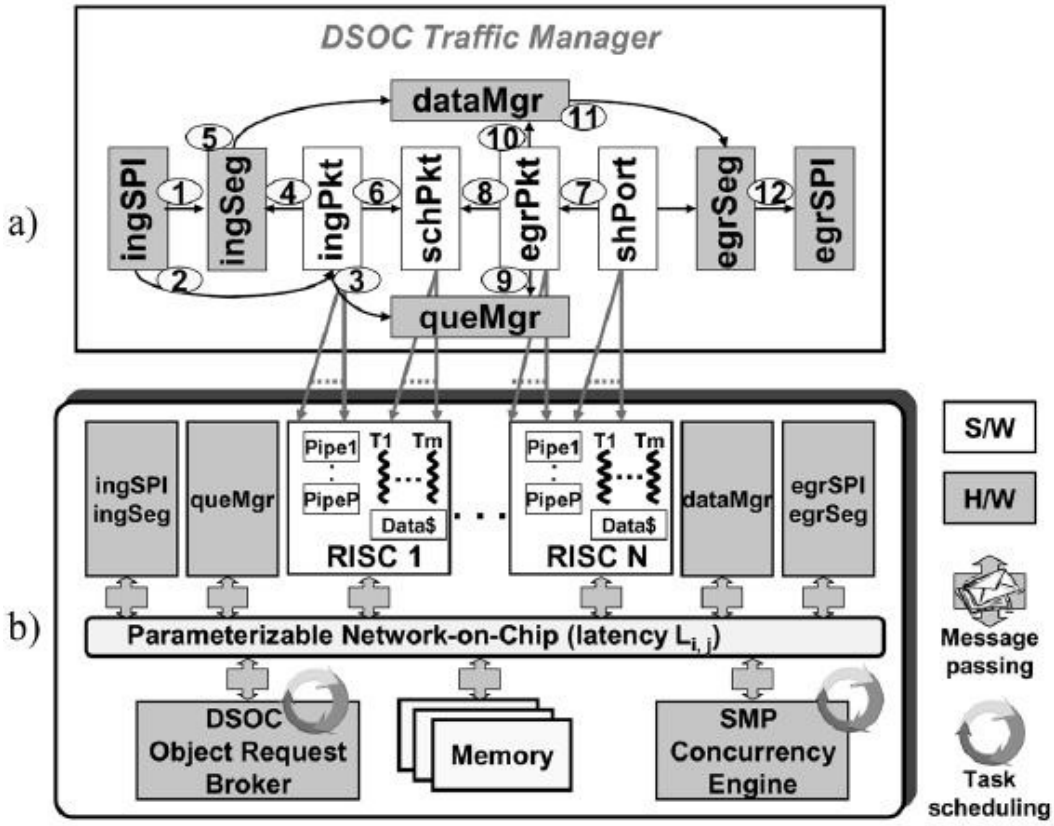

Figure 65 StepNP platform

Authors in [130] presents the design of MC-ORB, which is the first real-time object request broker (ORB) designed to address the nuances of multicore platforms with a novel core-aware middleware thread architecture and allocation service for soft real-time tasks. The work evaluated the cost of various thread management function calls in a multicore system, like load balance checks and thread migrations. The results show that the most costly function is thread migration among cores attending as much as 20 µs per migration. The major design goal is thus explicitly managing task allocation at the middleware level and minimizing thread migration. The MC-ORB is implemented using the ACE 5.2.7 framework and on the Linux 2.6.17 kernel. Empirical evaluations show that MC-ORB is highly efficient and effective on a multicore Linux platform, especially in comparison to a real-time ORB designed for single processor platforms.

The work in [130] mainly deals with core-level real time scheduling issues of a multi-core system with OS SMP support. It doesn't address the migration of middleware into a multiprocessor system on chip where each processor has its own instance of OS. In that case, the communication layer adaptation of the middleware should be considered according to the communication mechanisms provided by the embedded system.

The Multi-Writers-Multi-Readers (MWMR) communication middleware developed at SoClib implements a generic inter-task communication mechanism for shared memory multi-processors architectures. This protocol has been designed to support both communication between software tasks and hardware tasks, implemented as a dedicated hardware coprocessors. The MWMR protocol is implemented on top of POSIX threads API. [131][132] The MWMR defines a generic communication channel as a software buffer located in on-chip shared memory.

5.2 Network on Chip technology

Advances in semiconductor technology enable the integration of increasing numbers of IP blocks in a single System-on-chip (SoC). Network on chip (NoC) is a new approach for the design of the communication sub-system of SOC compared to the traditional bus based approaches. NoC brings networking theories to on-chip communication. Compared to traditional bus based architectures, NoC offers several advantages:

- Bandwidth scalability
- Process scalability
- Energy efficiency
- Easy IP integration with standard interface
- Reduced time-to-market [46][48]

We will briefly introduce the recent year NoC implementations in both academic and commercial community.

5.2.1 SPIN

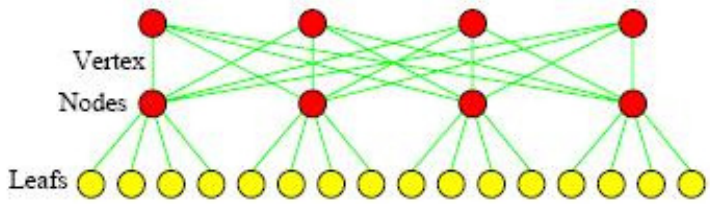

Figure 66 Flat tree topology

The SPIN network (Scalable Programmable Integrated Network) is one of the first published NoC [140]. It was developed by the University of Pierre and Marie Curie. It implements a fat-tree topology with two one-way 32-bit data paths at the link layer as shown in Figure 66. The fat tree is the most cost-efficient topology for VLSI realizations and 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 limiting the packet size. There are three types of flits: first, data and last flits. 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, the fat tree doubles the bandwidth at each level of the hierarchy up to the root but at a higher area cost.

5.2.2 AEthereal

The Aethereal NoC was developed by Philips Research Laboratories and offers both guaranteed service (GS) and best effort (BE) traffic. [141] [142] The guaranteed performance of GS connections results from wire and buffer reservations in the NoC. To give 100% guarantees, these reservations must be for the worst case, wasting any unused bandwidth. To increase the resource usage, the BE connections are introduced that use all unused bandwidth with a lower priority. It also put emphasis on the programming model and a design flow. AEthereal provides a combined distributed and centralized model.

5.2.3 Nostrum

Nostrum is a NoC developed by the LECS (Laboratory of Electronics and Computer Science) at the Royal Institute of Technology in Sweden. [143] Nostrum implemented a service of Guaranteed Bandwidth (GB) and latency in addition to the already existing service of Best-effort (BE) packet delivery. The guaranteed bandwidth is accessed via Virtual Circuits (VC) that are implemented using a combination of two concepts that are 'Looped Containers' and 'Temporally Disjoint Networks'. The Looped Containers are used to guarantee access to the network independently of the current network load without dropping packets; and the TDNs are used in order to achieve several VCs plus ordinary BE traffic.

5.2.4 MANGO

The MANGO (Message-passing Asynchronous Network-on-chip providing Guaranteed services over OCP interfaces) architecture [145], developed at the Technical University of Denmark, is an asynchronous NoC, targeted for coarse-grained GALS-type SoC. MANGO provides connection-less Best Effort (BE) routing as well as connection-oriented Guaranteed Services (GS). Guaranteed service connections are established by allocating a sequence of Virtual Channels through the network. The routers implement virtual channels as separate physical buffers. A scheduling scheme called the ALG (Asynchronous Latency Guarantees), schedules the access to the links, allowing guaranteeing the latency.

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

The BE router implements a source routing scheme. The three MSBs of the packet header indicate one of the 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.

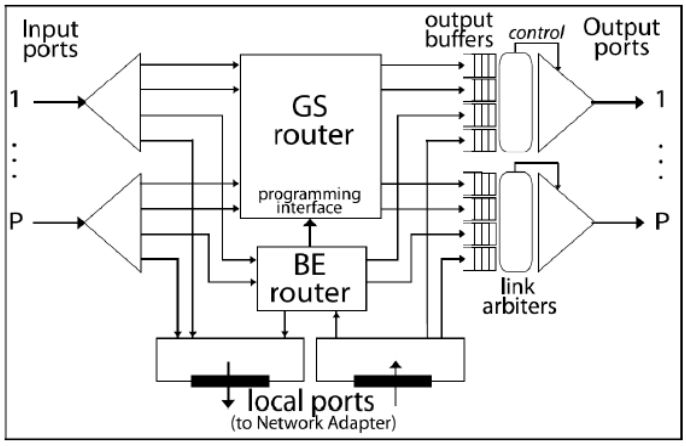

Figure 67 MANGO router

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.

5.2.5 Arteris NoC technology

In the design of our Network on chip, we used the Arteris NoC technology. Arteris Company was founded in 2003 in Paris and the company focuses on challenges associated with the next-generation System-on-chip (SoC) design: the on-chip communications. In 2005, Arteris introduced the first commercial implementation of NoCs delivered in form of IP library, the Danube 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 shown in Figure 68.

Arteris NoC technology provides a flexible and scalable solution that allows each designer to make the right trade-offs and achieve the specific design goals for their particular design. It is composed of two networks: a request network and a response network.

The Danube Intellectual Property Library that contains a set of configurable building blocks managing all on-chip communications between IP cores in SoC designs. The Danube IP

library comprises three types of units: Network Interface Units providing interfaces to the IP cores, Packet Transport Units and physical links building up the switch fabric user-defined topology. These units can be configured based on the system objectives and topology requirements. Figure 69 shows the mains components of the Danube Library while Table 18 lists the characteristics of the main IP components.

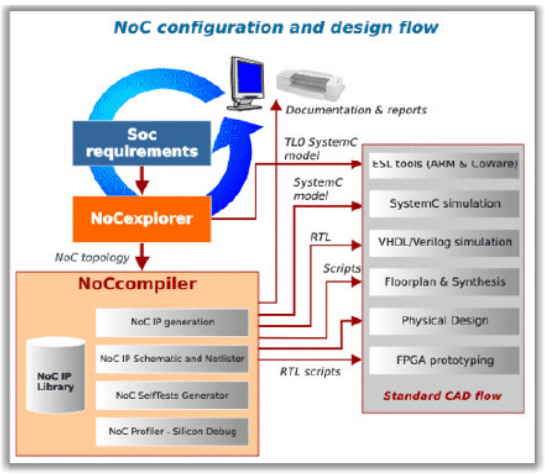

Figure 68 NoC design flow by Arteris

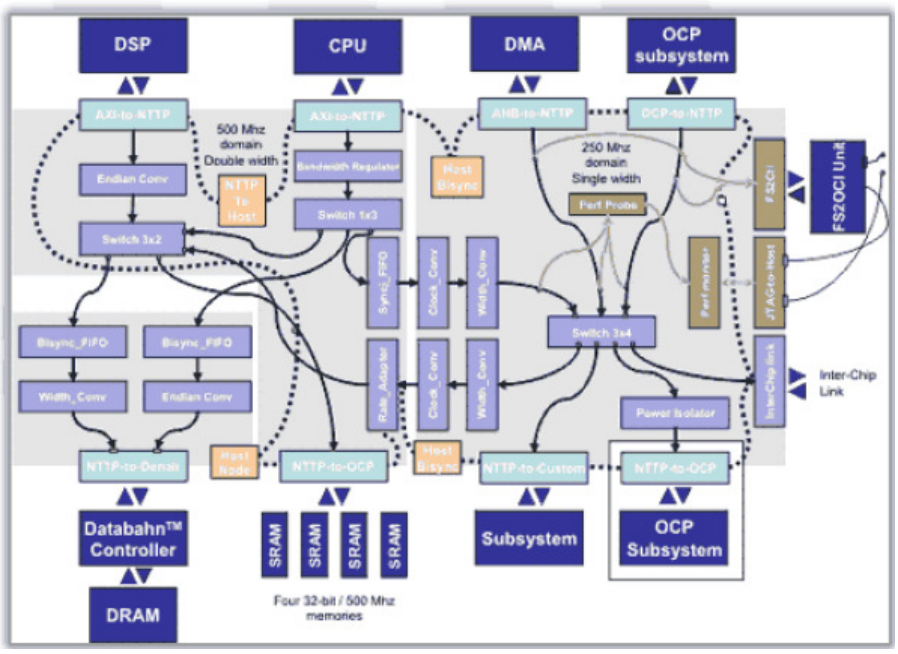

Figure 69 example of Danube IPs

Table 18 Arteris Danube transport units IPs

	Description
Switch	Accept packets from input ports and forward each packet to specific output port unchanged
Sync-Fifo	Stores packets at high rates of speed and moves packets to slower units
Bisync-Fifo	Provides resynchronization for units from asynchronous clock domain
Clock-Conv	Connects units from distinct clock domain
Width-Conv	Connects links of different width
Endian-Conv	Enables the choice of little or bit endian units
Rate-Adapter	Removes WAIT cycles
Bandwidth-Limiter	Prevents initiators from consuming too much bandwidth
Bandwidth-Regulator	Guarantee an average target bandwidth to an initiator that is subject to fluctuating throughput requirements
Meso link	Long distance transport in NoC
InterChip-Link	Connects one chip to another along the same wires

The Switch generator is an essential building block of the NoC interconnect system. Figure 70 illustrates an N input ports M output ports of the Danube Switch unit. It accepts NTTP packets carried by input ports, and forwards each packet to a specific output port unchanged. The Switch unit supports synchronous operation, full crossbar with up to one data word transfer per MINI port per cycle. It uses wormhole routing for achieving reduced latency. The unit can be software-controlled at run-time through the service network.

The statistic collector IP, shown in Figure 71, provides performance monitoring capability by probing NTTP or OCP links, recording events, and transmitting results to a debug unit through a dedicated NTTP link. It provides up to 8 probes that provide metrics such as throughput and latency on certain dataflows. The statistics collector monitors activity by connecting probes to NTTP or OCP signals, without introducing any flow control in the system. An NTTP port is used to export results as frames, for processing by a dedicated target.

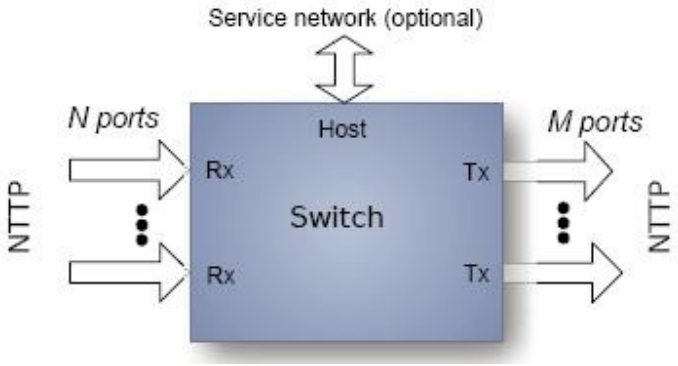

Figure 70 Arteris Danube Switch

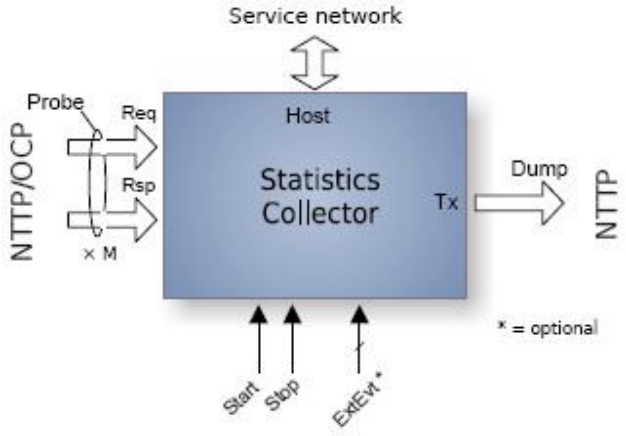

Figure 71 Statistic collector

Arteris provides two design tools for NoC exploration and implementation: the NoCexplorer and NoCcompiler. The NoCexplorer exploration tool provides an environment to capture the dataflow requirements of the IP blocks to be serviced by the NoC and allows the designer to rapidly utilize a very fast dataflow simulation engine and parameterizable dataflow sources and sinks to model the system behavior. The NoCcompiler tool creates a database of the specific instance of the NOC. It exports in a variety of languages the NoC design, for example, Verilog, VHDL, SystemC that is to be passed to other synthesis / place & route tools for the final system implementation.

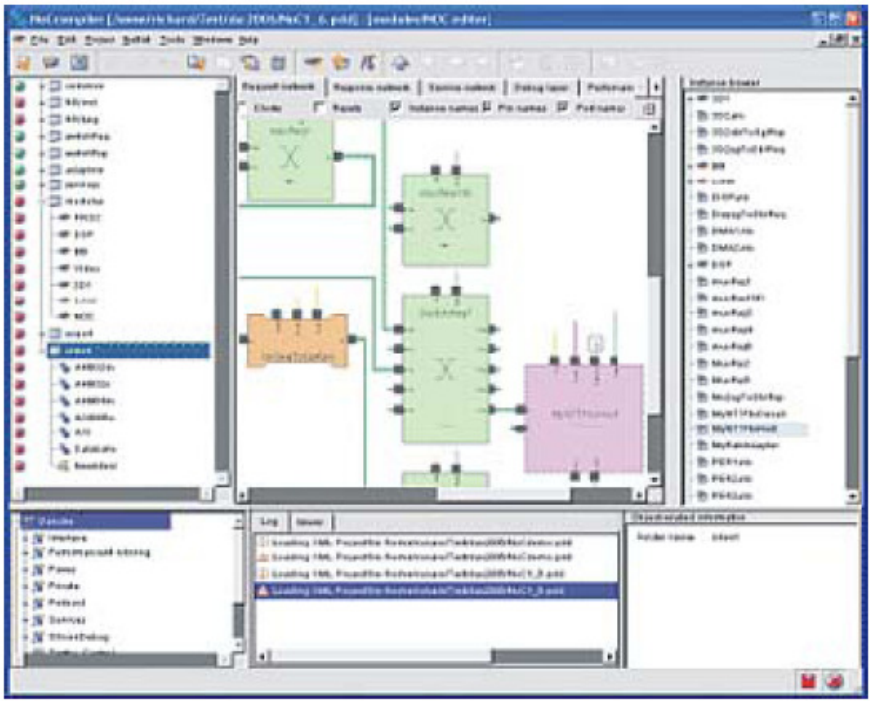

Figure 72 NoCcompiler GUI

5.3 Synchronization Issues with CORBA Based designs

The traditional CORBA synchronization mechanism is realized by the synchronous two-way client/server invocation. The client thread that invokes a server operation blocks until response is returned. Although simple in programming, this method lacks the support for exploiting the intrinsic parallelism in distributed systems including asynchronous invocation where one client can continue invoking another object existing on another server before the first invocation is done or group invocations as supported in MPI by multicast/select, multicast/gather, scatter/gather, etc. [135]

There are several approaches for achieving the asynchronous invocation. First, the two-way synchronous invocation can be used with multiple threads. However this solution is accompanied by the drawbacks of error-prone multi-threaded programming, scalability issue with thread creation overhead, or it is not even applicable in a single-threaded client. Second, we can use one way invocation. This solution has no guarantee of reliable delivery due to the best-effort semantic. The third solution is to use the Dynamic Invocation Interface (DII)

deferred synchronization. But using the DII interface means cumbersome programming and increased the program size. More over, type-safety is left to be guaranteed at the developer level rather at the compiler level. The most promising solution is to use the Asynchronous Method Invocation (AMI) (CORBA messaging specification). Unfortunately, it is an optional service and is not always supported by academic or commercial ORBs.

Works have been done to address the inefficiencies in the CORBA based server/client programming in exploiting the natural parallelism in a distributed system. [135] tackles these issues by providing a multilayered architecture, and API implemented in C++ classes to provide the necessary invocation semantics for parallel programming.

In [136], the authors propose a parallel programming model over CORBA, the P-CORBA, which addresses the issues concerning the parallel programming over a Network of Workstations. The model enriches CORBA with the notion of concurrency in introducing a metaobject regrouping a set of different objects of the same class that must be dispatched to different machines depending on the machines' load condition. The model also provides methods for dynamic load balancing and object migration. Experiments show a higher message sending overhead than MPI send calls but the overall performance on a clustered platform is better due to the dynamic load balancing feature. The CORBA implementation utilized in this work is MICO.

The paper [137] presents three agent based parallel interaction architecture that improves the performance of CORBA based on the traditional synchronized object invocation and serial server execution. It analyzes the three client-agent-server interaction architectures, parallel interaction architecture. The performance of these architectures is compared with the traditional sequential architecture. The execution results showed substantial performance benefit gain from using parallel interaction architecture especially at low to medium load. Multithreading can efficiently solve the bottleneck problem at the agent level that risks large queuing delays when the load is high.

5.4 OCP-IP Protocol and CORBA

Because the Network Interface Unit of the NoC in our MPSoC conforms to the OCP protocol, we discuss the CORBA middleware transport layer adaptation issues in this section.

The default transport mechanism that is requested to be supported by the CORBA specification is the IIOP protocol (routing GIOP packet on internet). According to the CORBA specification, the GIOP definition makes the following assumptions regarding to the transport behavior: [7]

1. The transport is connection-oriented. GIOP uses connections to define the scope and extent of request IDs.
2. The transport is reliable. Specifically, the transport guarantees that bytes are delivered in the order they are sent, at most once, and that some positive acknowledgment of delivery is available.
3. The transport can be viewed as a byte stream. No arbitrary message size limitations, fragmentation or alignments are enforced.
4. The transport provides some reasonable notification of disorderly connection loss. If the peer process aborts, the peer host crashes, or network connectivity is lost, a connection owner should receive some notification of this condition.
5. The transport's model for initiating connections can be mapped onto the general connection model of TCP/IP. Specifically, an agent (described herein as a server) publishes a known network address in an IOR, which is used by the client when initiating a connection.

Compared to the macro-world setting of work-stations cluster consisting of computers interconnected by internet via the network adapter, our MPSoC system is constructed by interconnecting multiple embedded processors via a packet switched on-chip network (NTTP) and the network interface corresponds to the OCP-IP Protocol.

In order to port the CORBA transport layer to the single chip environment, we must first study the feasibility by examining article-by-article the fulfillment of the above assumptions in the MPSoC communications fabric.

For the purpose of simplicity, we do not want to touch the entire transport and internet protocol layer of the protocol stack. We studied the three layers of the communication stack below the application layer, transport layer, network layer, and MAC, in terms of their respective necessity of modification and complexity of adaptation for making ORB communicate in our NoC based MPSoC. Interestingly, the necessity and complexity correspond in an inverse order, which means the layer that lies nearer to hardware has a stronger modification need but requires less efforts in terms of Linux kernel programming. On the other hand, the layers that approach the application level have less requirement of modification but require more kernel programming efforts to realize such a modification. The Table 19 summarizes the three choices discussed above from the less-complex to the more-complex ones.

Table 19 Communication layer adaptation choices

	Necessity of modification	Programming efforts
MAC layer only	Yes	Minimum
IP and MAC layers	No	Medium
TCP, IP and MAC layers	No	Maximum

Since the network interface unit is replaced by the OCP/NTTP NIU, the MAC layer protocol needs to be adapted to process the specific MAC layer headers. The IP layer and TCP layer remain untouched and the middleware communication architecture can be adapted with trivial efforts. The proposed inter-layer calls and packet migrations are illustrated in Figure 73. With this approach, the NTTP packet switch layer is hidden at the NIU level, and the device driver of the OCP NIU should deal with a custom OCP MAC address for higher level protocols to identify the correct network device. This is the approach we took in our design. However the two other approaches will also be discussed.

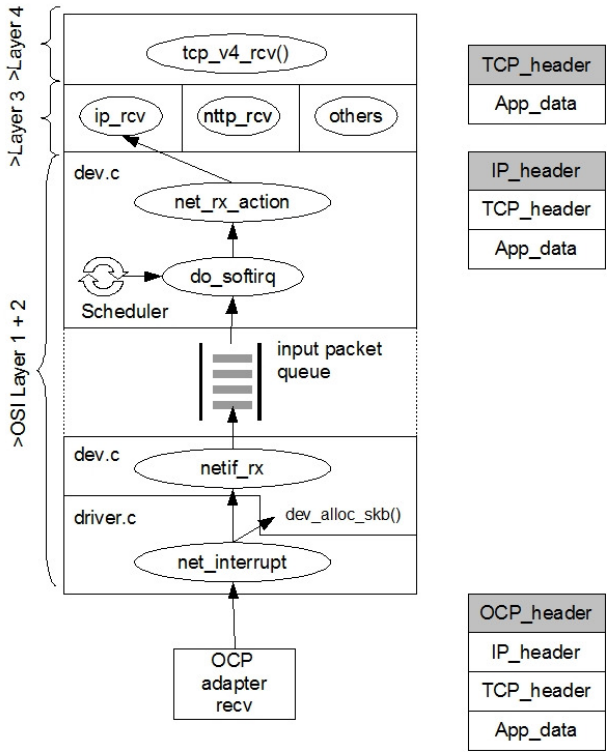

Figure 73 TCP/IP/OCP receive sequence

A second approach is to modify the IP layer protocol to reflect the specific properties of the NTTP packet. This approach is illustrated in Figure 74. With this approach, a new network work layer protocol should be registered with the Linux network kernel. Same as the first approach, this one requires trivial efforts for COBRA communication layer adaptation. But there exists one problem: the NTTP packet is generated at the NIU level, which is different from the traditional TCP/IP packet. This issue should be considered when taking this choice.

With the first two choices, the 5 assumptions of CORBA GIOP over the transportation layer are largely resolved by the TCP/IP layer and the OCP protocol is masked from the middleware point of view. However although inspired by the off-chip networks, NoCs offer more preferable characteristics than their off-chip counterparts. For example, NoC can avoid dropping data, assuming that a SoC operates reliably (that is, its routers do not fail, misrouting does not occur, and so forth). Moreover, the Arteris NoCs are composed of a request network and a response network. Every transaction should be acknowledged at the NoC level, which is treated in the transport layer in the case of off-chip network. All these features are not

exploited in the first two approaches. This is where the third solution, the TCP level adaptation becomes useful. In taking direct services provided by the NoC layer, the communications stack should become more efficient. However, since the CORBA communication layer function deals socket calls that deal directly the transport layer. Any modification in this layer should be done with considerations in mind no to break the standardized socket like calls.

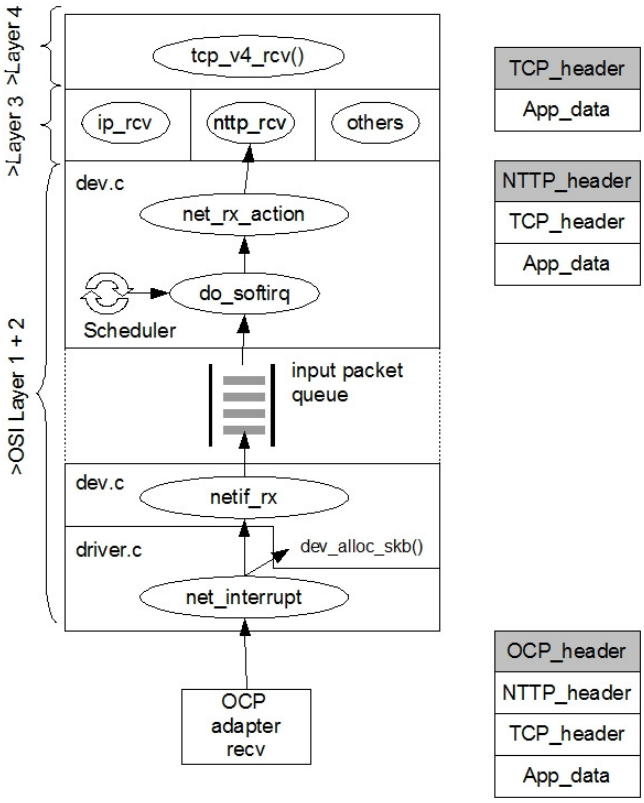

Figure 74 TCP/NTTP/OCP receive sequence

5.5 Network Interface design and low level APIs

Figure 75 shows the architecture of the PLB-OCP network interface. It is composed of two parts, a receiver and a transmitter. There are two packet buffers for stocking transmitted and received packets. The buffer size is set to 2k bytes.

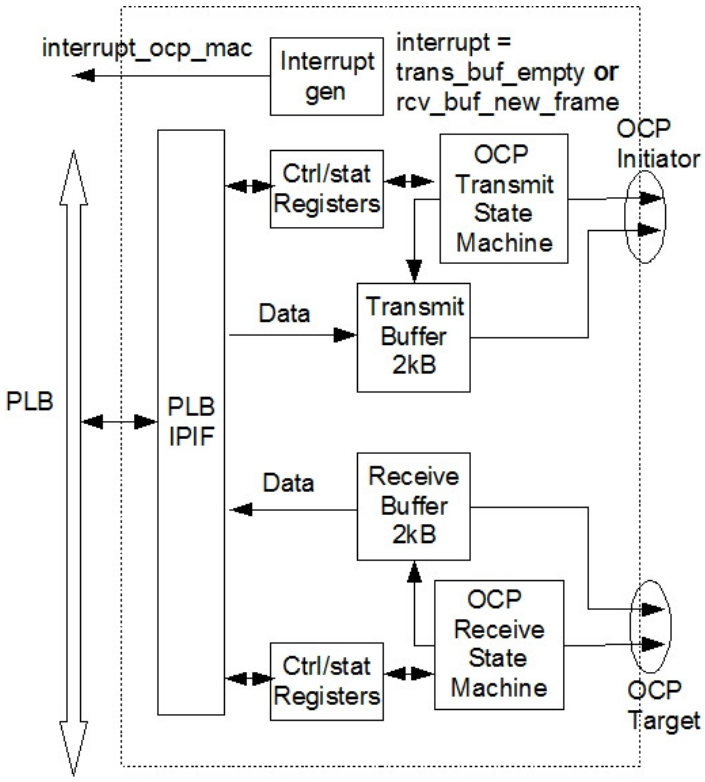

Figure 75 Architecture of the PLB-OCP network interface

When transmitting, the host writes a whole packet represented by the Linux *struct sk_buff* into the transmit buffer. The Maximum Transfer Unit is defined so that the size of packet represented by *len* field of the *sk_buff* structure doesn't surpass the capacity of the transmit buffer. The host processor then sets the frame ready bit of the transmit control register to indicate a valid frame. The interface then sends out the packet in a burst write mode to the corresponding destination processor.

When a burst write command arrives at the input of the network interface indicating the sent of a packet from a remote host, the interface checks the availability of its receiving buffer. If available, it accepts the command and stores the successive data load in the receiving buffer. It then sets the frame_busy bit of the receive control register to indicate the occupancy of the buffer by a valid packet.

A transmitter interruption is generated when a packet is sent out of the OCP interface and the transmit buffer becomes available to accept new frame. This serves to retry of a former failed

and delayed packet send attempt due to non availability of the transmit buffer. A receiver interruption is generated when a new packet is coming in and stored in the receiver buffer. The host processor then comes to store this packet in its main memory and clears the frame_ready bit.

Low level APIs are developed to facilitate the data operation, device control and interruption handling from the upper layer code, for example the network device driver. These APIs include:

```
int OCP_MAC_Send(OCP_MAC_t *OCP_MAC, u8 *FramePtr, unsigned ByteCount) ;
int OCP_MAC_Recv(OCP_MAC_t *OCP_MAC, u8 *data);
void OCP_MAC_DisableInterrupts(OCP_MAC_t *OCP_MAC) ;
void OCP_MAC_EnableInterrupts(OCP_MAC_t *OCP_MAC) ;
void OCPMAC_InterruptHandler(void *InstancePtr);
```

5.6 Network-on-chip design

5.6.1 Protocol definition

The NTTP protocol configuration considers the master address space and the slave address space and the data payload length burst transactions.

mstAddr: 2 bits (4 masters)
slvAddr: 2 bits (4 slaves)
len: 8 (512 payload cells)

The OCP protocol configuration has concerns with burst length that should be set according to NTTP settings, and transaction phase semantics.

burstType: TIE_OFF_INCR
Burstlength_wdth: 8
reqlast: true
resplast: false
writenonpost, writeresp_enable: false (precise burst write doesn’t need response)

5.6.2 NoC connection

Each plb_ocp network interface requires two interfaces, one for transmitting (ocp master) and the other for receiving (ocp slave). Figure 76 illustrates the NoC design of a two-node system consisting of only a 2x2 switch. The initiator interface of master_00 is connected to switch input port 0 and the initiator interface of master_01 is connected to switch input port 1. According to the route table definition that will be explained later, data targeted at master_00 are routed to the switch output port 0, and data sent to master_01 are routed to switch output port 1. As a result, the path input_0 -> output_0, confined in the red frame, forms the loopback path of master_00, while the path input_0 -> output_1, confined in the blue frame, forms the inter-node path between master_00 and master_01. The situation is similar for master_01.

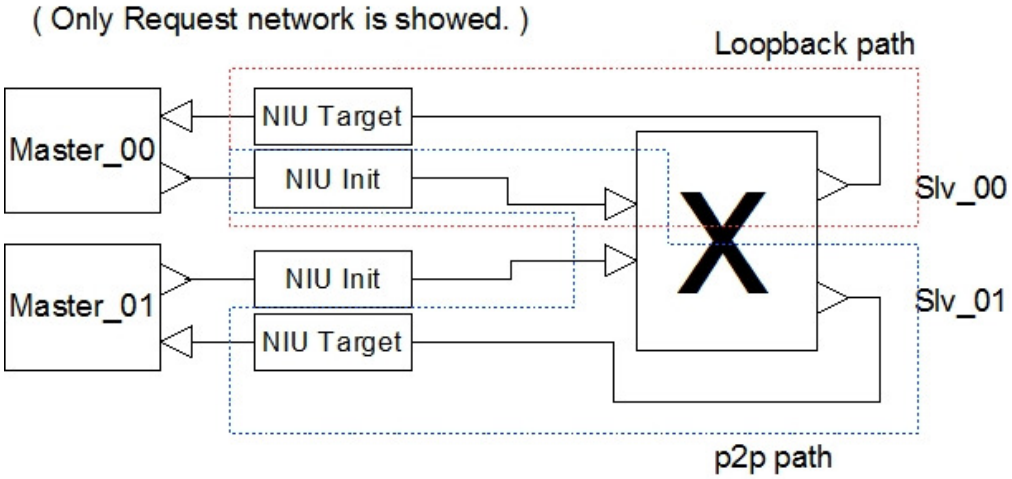

Figure 76 A two-node point-to-point connection

5.6.3 Memory mapping

The ocp MAC address is defined following the Ethernet mac address format that consists of six bytes, “\0ocpxx”. The first byte is ‘\0’ to avoid being a multicast address (the first of multicast is odd). The last two bytes indicates the id of the processor. Each master interface response port is identified by the ocp master address. The interface inserts its own address into the master address field of the request packets and is later copied to the response packets by the ocp slave in order that the response packets are routed back to the same master. In this

example, the master addresses are coded in two bits supporting up to 4 masters. The nttp global address base represents the starting address of the receive buffer in each plb-ocp interface in a system view. The respective size of the receiver buffer is represented by nttp address size. Because plb_ocp network interface can only initiate write operations, which send ocp frame to the corresponding receiver buffer, the memory map considers only the receiver buffer. The global memory map is summarized in Table 20.

Table 20 NTTP Global memory map

ocp_mac address	**ocp master address**	**nttp global address base**	**nttp address size**
"\0ocp00"	"00"	x"00000000"	x"800" (2kB)
"\0ocp01"	"01"	x"00000800"	x"800" (2kB)
"\0ocp02"	"10"	x"00001000"	x"800" (2kB)
"\0ocp03"	"11"	x"00001800"	x"800" (2kB)

Figure 77 illustrates the address translation from the ocp address domain to the nttp (NoC) address domain. The example aims to access the x"2bc" memory address of the receiver buffer of plb-ocp interface_01.

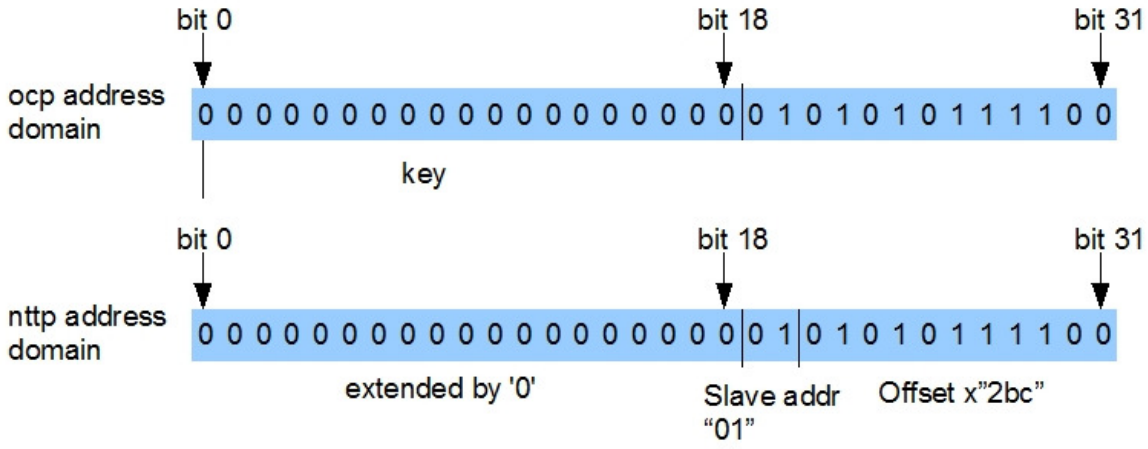

Figure 77 Address translation from ocp domain to nttp domain

5.7 Single FPGA Chip NOC Based Multiprocessor Design

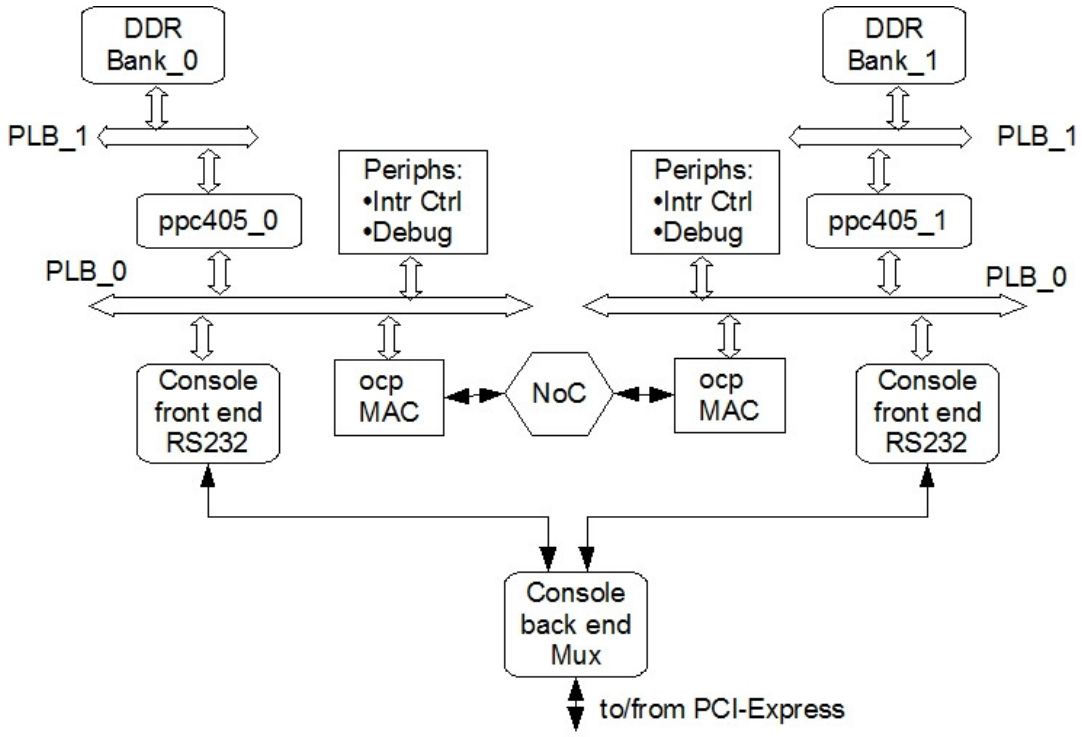

Figure 78 Two ppc405 connected with NoC with multiplexed output

A single chip platform is developed for testing of the on-chip CORBA based communication system. As illustrated in Figure 78, the system is composed of two separate computing systems, each having a ppc405 microprocessor and running Linux operating system. The two ppc405 communicates via a NoC fabric through the ocp MAC.

For Linux console output and debugging, we choose to use the PCI-express connection to communicate with a terminal on a host pc. Each ppc405 processor connects its console output to a console front end conforming to the RS232 protocol. Then the multiple front end outputs are tagged and multiplexed by the console backend component which in turn connects to the PCI-express bus.

On the host side, a monitor program checks the PCI-express input and distributed input data to the corresponding processor terminal according to data tag. The structure of the host side monitor and terminal program is illustrated in Figure 79.

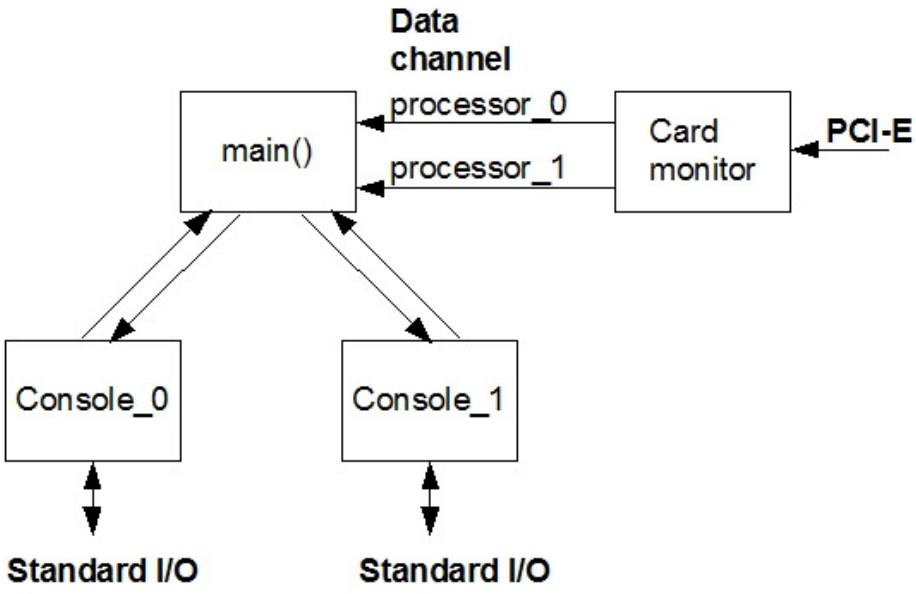

Figure 79 Host machine console system

5.8 Performance results

The measured results for the inter-processor communication are presented in Table 21.

Table 21 Time of round-trip Echo function with zero message body

Platform	Transport	Time per call (us)
Linux 2.6.28 PPC405 120MHz (Gcc-4.2.4-O2)	TCP/IP inter-processor	3412

5.9 Design Flow for Client-server with Automatic Parallelization Paradigm MPSOC

The entire proposed design flow is presented in Figure 80.

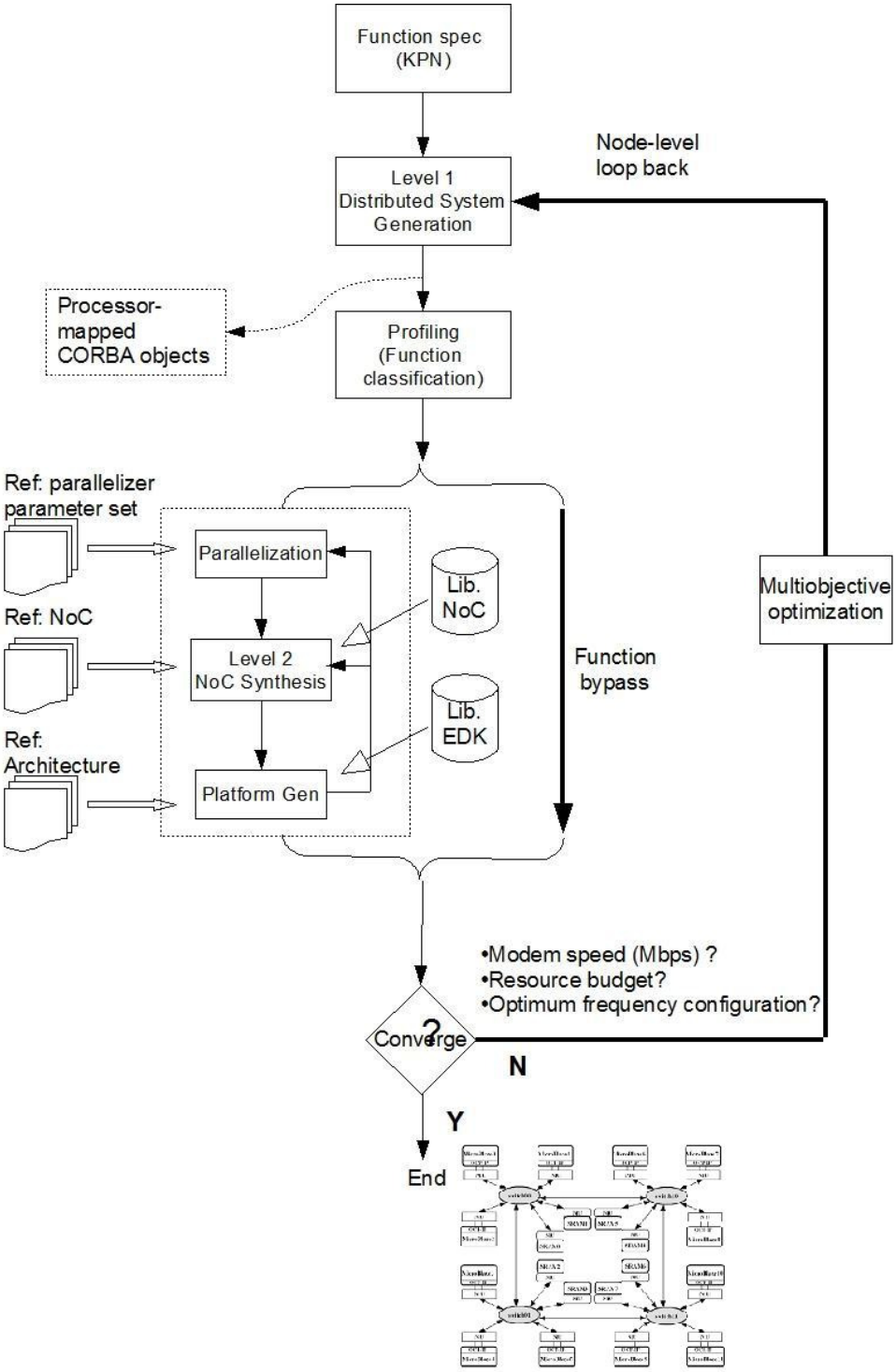

Figure 80 Design flow based on the hybrid programming model single chip

The level-1 distributed system generator takes as input an application abstracted in Kahn Process Network (KPN). Processes are mapped to separated processors without considering

the underlying SoC architecture. A function profiling is then carried out to determine the most time consuming process. The profiling results are used to filter out the system performance critical functions that will then be accelerated by parallel programming while the other non-critical functions are bypassed to the final stage. The performance critical functions are parallelized by an automatic parallelizer. A default parameter set including information as memory hierarchy, processor number, is first provided and is finely tuned during the local optimization loop, represented by the arrow connecting the "Platform gen" block and the "Parallelization" block. The network on chip topology is synthesized. The platform is then synthesized by the platform generation engine. The parallelization and the bypass branch are then combined and deployed on the SSM IP based distributed/parallel platform. If the system requirements like performance, resources are satisfied while keeping a minimum frequency configuration reducing energy consumption, the flow ends. If one of the above requirements is not met, we will loop back to the first level distributed system generation block through a multi-objective optimization engine which is responsible for resource or frequency optimization, parameter tuning, etc.

5.9.1 Network-on-chip synthesis

Authors in [41] focus on the synthesis of a bus matrix based communication architecture for the high bandwidth MPSoC design. They propose an automated approach, named bus matrix synthesis (BMSYN), for synthesizing a bus matrix communication architecture, which satisfies all performance constraints in the design and minimizes wire congestion in the matrix.

Figure 17 shows the automated BMSYN flow. The inputs to the flow include a common through graph (CTG) representing the performance constraints of the system, a library of IP models, a target bus matrix template, and a communication parameter constraint set. First of all, a fast transaction-level model (TLM) simulation of the system is carried out to determine the application-specific data traffic statistics. The information is then passed to the global optimization phase to reduce the full bus matrix architecture by removing unused busses and local slave components from the matrix. The resulting matrix is called a maximally connected reduced matrix. In the next step, an optimization engine based on a static branch and bound

algorithm is used to cluster the slave components, which further reduces the number of busses in the matrix. The resulting architecture is then passed to a fast bus cycle accurate simulation engine to validate and select the best solution that meets all the performance constraints, determine slave arbitration schemes, optimize the design to minimize bus speeds and OO buffer sizes and then finally output the optimal synthesized bus matrix architecture. The results from the synthesis of an AMBA3, AXI-based bus matrix for four MPSoC applications from the networking domain show a significant reduction in bus count in the synthesized matrix when compared with a full bus matrix (up to 9 x) and a maximally connected reduced matrix (up to 3.2x).

5.9.1.1 Definition of problem in terms of graph

The scenario we consider here corresponds to the shared memory model where each processor has equal access to the shared memory and the communication between processors is done via the share memory. The input to the NoC synthesis engine is a core graph that models the connection and bandwidth requirements of the system. Suppose H=(C, K) is an oriented graph, with |K|=p, and $w \in R^K$ is an indexed vector on the arcs of H. Each arc (u, v) of K corresponds to one demand of information transmission from component u to component v. The value of $w^{(u,v)}$ corresponds to the quantity of information to be transmitted from u to v. The pair (H, w) represent the above mentioned core graph.

The topology synthesis of a network on chip consists in determining one topology of NoC that satisfies the information transportation defined by the core graph while keeping minimum surface. This network is composed of routers and links. Several types of routers can be installed. The number of input ports, output ports, the surface consumption, and the bandwidth per port depends on the type of installed router. We suppose that there exists k types of different routers indexed from 1 to k. The number of input ports (resp. output ports) of router i, i = 1,…,k, is noted e_i (resp. s_i) and its maximum bandwidth per port is noted by Ω_i. Without loss of generality, we suppose that router 1 is a dummy router in that it doesn't consume any silicon surface, and has neither input port nor output port. We suppose that q, representing the number of possible routers constituting the NoC, is given. We use R to

indicate {1,…,q}. Let D = (V, A) be the graph representing the possible connections between the elements of the NoC, where V = R ∪ C. In addition, we suppose that no direct connection exists between any two core graph vertexes. Therefore A corresponds to the collection of arcs connecting a component to a router, a router to a component or two distinct routers. We note by m the number of arcs in D. The topology synthesis of network on chip then consists of, given (H, w) and q, determining a sub-graph D' of D, one type of router for each element in R, and the path from u to v for each demand (u, v) of H, so that:

- the number of entering arcs (resp. exiting arcs) of each vertex r among R is inferior or equal to the number of input port (resp. output port) of the router installed in vertex r,
- the number of entering or exiting arc of each vertex v of C is respectively inferior or equal to k,
- the path of demand utilize uniquely the arcs of D',
- the constraints of bandwidth are satisfied,
- the surface of all the installed routers is minimum.

5.9.1.2 Integer linear programming

The communication infrastructure plays a more and more critical role in the modern MPSoC design. The NoC based communication is more scalable and exploits better the parallelism of the architecture. An optimized NoC topology is important to get better performance under stringent on-chip resources constraints. The NoC topology synthesis can be modeled in form of integer linear programming. We will define the variables for the modeling of the demand path.

Let $x \in \{0,1\}^{mp}$ be the vector such that

$$x_a^k = \begin{cases} 1 \text{ if connection k is transported on arc a,} \\ 0 \text{ if not,} \end{cases} \qquad \forall k \in K, \forall a \in A$$

We define the second set of variables in order to model the sub-graph D' that represents the resulting NoC. We define $\in \{0,1\}^m$ such that

$$y_a = \begin{cases} 1 \text{ if arc a belongs to sub-graph D'} \\ 0 \text{ if not,} \end{cases} \qquad \forall a \in A$$

Finally in order to know which type of router is installed on each site r of R, we define z $\in \{0,1\}^{R*l}$ such that

$$z_r^i = \begin{cases} 1 \text{ if switch of type i is installed on site r,} \\ 0 \text{ if not,} \end{cases} \qquad \forall r \in R, \forall i = 0,1,..l.$$

The NoC topology synthesis problem corresponds then to the following linear programming:

$$\sum_{i=1}^{l} z_r^i = 1 \quad \forall r \in R$$

$$\sum_{a \in \delta^{out}(v)} x_a^k - \sum_{a \in \delta^{in}(v)} x_a^k = b_v^k \quad \forall k \in K, \forall v \in V$$

$$b_v^k = \begin{cases} 1 & \text{if v = } o_k, \\ -1 & \text{if v = } d_k, \\ 0 & \text{if others} \end{cases}$$

$$x_a^k \le y_a \qquad \forall a \in A, \forall k \in K$$

$$\sum_{k \in K} q^k x_a^k \le \sum_{i=1}^{l} \Omega_i z_v^i \qquad \forall v \in V, \forall a \in \delta(v)$$

$$\sum_{a \in \delta^{in}(r)} y_a \le \sum_{i=1}^{l} e_i z_r^i \qquad \forall r \in R$$

$$\sum_{a \in \delta^{out}(r)} y_a \le \sum_{i=1}^{l} s_i z_r^i \qquad \forall r \in R$$

5.9.1.3 Case study

In this section, we will present a case study in which a MPEG4 core graph, as illustrated in Figure 81, is used as input to the NoC topology synthesis engine.

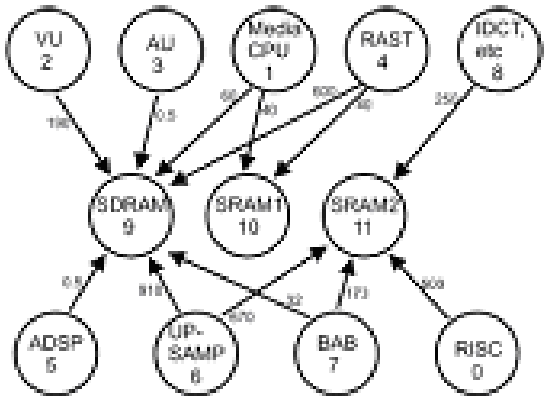

Figure 81 MPEG4 Core graph

The generated NoC topology is shown in Figure 82.

With this ILP model, the interconnection infrastructure, Network on Chip, of MPSoC can be tailored on a link-by-link basis, which optimizes the allocation of on-chip resources.

5.10 Conclusion

The CORBA middleware was originally utilized in large scale distributed system software developments. With the advance in the semiconductor process technology, more and more resources are now integrated on a single chip, large on-chip memories, embedded processors, DSPs, configurable IP accelerators, etc. CORBA formerly served as a software bus by abstracting the underlying architecture and operating system heterogeneity and by providing a uniformed function-call like interface to the programmer. It is now usable as well as necessary for the embedded domain to provide an efficient programming model to embedded application developers.

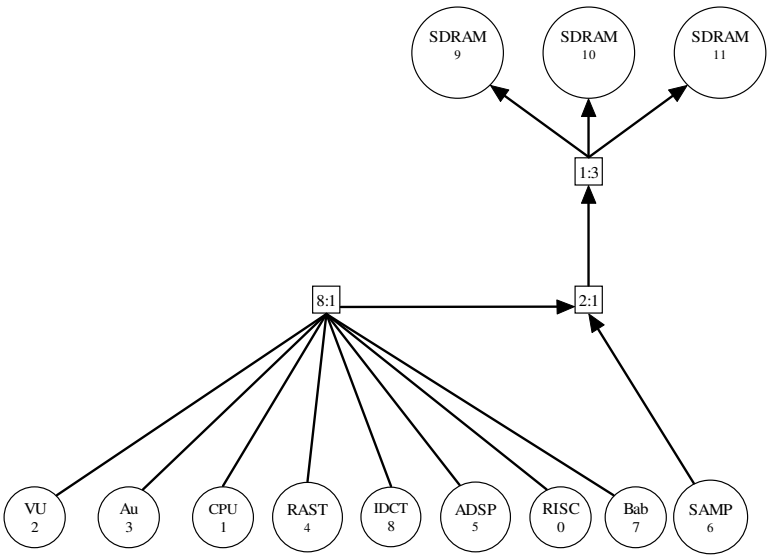

Figure 82 MPEG4 NoC topology

We implemented the CORBA middleware on a network-on-chip (NoC) based multi-processor single chip. The NoC draws analogy with the macro-world network: packets are routed to and from nodes that are connected to the NoC. We developed a network interface, the plb_ocp_mac, which provides services like a network adapter. It encodes the hardware addresses to the packets provided by the kernel through the TCP/IP stack, and calls the low-level driver to hand the packet to the NoC. During receiving, the network interface stores the packet sent to it, remove the hardware address, and then hands the packet to upper network stacks.

While inspired by the mainstream approaches for large system developments, the CORBA adaptation should take into consideration the characteristics and constraints specific to the SoC domain. We discussed in this chapter several optimization potentials which mainly consider the relatively reliable network transmission, and the resource constraints of the embedded system.

A part from the distributed programming model, modern MPSoC architectures expose also the SMP based parallel programming model. These two models should be combined according to different application calculation and traffic characteristics to attain maximum performance. In this chapter, we focused on the NoC topology synthesis. We developed the

mathematical model for a share-memory multiprocessor architecture, and use the Integer Linear programming tool to get the optimum solution of the NoC topology.

A SDR design flow is proposed with systematic architecture exploration and optimization based on the hybrid programming model (distributed client/server + parallel). A NoC topology synthesis engine was developed with linear integer programming. A complete SDR application has not yet been tested, but the tools and the design flow have been tested with all the features that are needed for implementing the SDR.

Chapter 6

Conclusion

This thesis proposes a design methodology and programming model for the efficient development and deployment of complex communications systems, specifically, the Software Defined Radio. Our contributions can be decomposed according to the following categories:

1. Design flow based on hybrid programming model

We are interested in the Software Defined Radio that is conforming to the Software Communication Architecture, which provides interoperability and reusability to radio waveforms. The SCA specification defines an operation environment that in which waveform applications are executed. It requires the use of CORBA middleware that provides abstraction of the underlying architecture and operating system for distributed objects. On the other hand, for some computation intensives functions, the signal processor based architecture doesn't fulfill the performance requirements under stringent energy budget and we resort to multiprocessor and parallel programming for function acceleration.

Based on the above hybrid programming model, the design flow proposes a two-state system generation engine with the first state generating distributed nodes and the second generating parallel processing elements with the help of an automatic parallelizer and network on chip (NoC) synthesizer.

2. Parallel programming and performance evaluation with automatic parallelizer

In this part, we use an automatic parallelizer, Pluto, for the source-to-source transformation of serial source codes to parallelized versions. Pluto is an automatic polyhedral source-to-source

transformation framework that can optimize regular programs for parallelism and locality simultaneously. Our main objective was to evaluate the efficiency of an automatic parallelizer within an “automatic” design flow.

We evaluated the Pluto parallelized codes on our NoC connected 16 PEs MPSoC platform. We noted several key elements that influence the effectiveness of parallelization. A comprehensive understanding of the characteristics of both the application and the architecture accompanied by an optimum combination of the two is necessary for a satisfying performance. Beyond this straightforward remark, we have shown that an automatic parallelizer can be used in our design flow.

The synchronization mechanisms play a fundamental role in efficient parallel programming and careful attention is necessary for the hardware implementation of these synchronization mechanisms. We have conducted performance experiments on the above single chip embedded multiprocessor. The experiments show that automatic parallelization can hardly exploit more than 8 processors despite the network on chip allowing communication concurrency.

3. NoC topology synthesis

Depending on the connection requirements between master and slave components and the profiling results regarding traffic, we used the ILP tool to automatically synthesize the topology of the network on chip in search of minimum chip surface utilization. Again, we have shown that an automatic synthesis tool to synthesize the NoC topology can be used in our design flow.

4. Adaptation of CORBA middleware on single chips with NoC communication architecture

The final goal is to integrate a macro world network based distributed SDR system on a single chip. For this purpose we first developed a multi-FPGA based distributed embedded system as a proof-of-concept. The multiple FPGA card are connected via an internet switch, each acting as a separate distributed node with a PPC405 processor. We tested the performance of

CORBA middleware and the potential of hybrid programming model by integrating in each PPC405 system a local parallel processing array for local parallel calculation while keeping a global distributed view.

Then we have worked on the adaptation of the TCP/IP stack to the on chip NoC communications. We have developed an OCP MAC adapter for accessing the NoC and tuned the TCP/IP stack parameters to fit in the on chip resources constraints. A first test has validated the CORBA execution on the single chip multiprocessor prototype consisting of two PPC405 processor connected by a NoC. There remains a large space for performance improvement of the communication based on this architecture. We have proposed several solutions including both a software stack and hardware optimizations. The tests with complete SDR baseband chains are currently being developed.

References

[1] S. Kyo et al., “A Low-Cost Mixed-Mode Parallel Processor Architecture for Embedded systems”, Proceedings of the 21st annual international conference on Supercomputing, SESSION: Architecture – multiprocessor systems, Pages: 253-262, 2007.

[2] Sankaralingam, K. et al., “Distributed Microarchitectural Protocols in the TRIPS Prototype Processor”, 39th Annual IEEE/ACM International Symposium on Microarchitecture, Page: 480-491, 2006.

[3] D. Burke, et al., “RAMP Blue: Implementation of a Manycore 1008 Processor FPGA System”, Proceedings of the Reconfigurable Systems Summer Institute, RSSI 2008, July 2008.

[4] Y. Hoskote, et al., “A 5-GHz Mesh Interconnect for a Teraflops Processor”, IEEE Micro, Volume 27, Issue 5, Page 51-61, 2007.

[5] X.Li and O.Hammami, An Automatic Design Flow for Data Parallel and Pipelined Signal Processing Applications on Embedded Multiprocessor with NoC: Application to Cryptography , International Journal on Reconfigurable Computing, Hindawi, 2009.

[6] www.omg.org

[7] “Common Object Request Broker Architecture (CORBA) Specification, Version 3.1” Part 2: CORBA Interoperability, OMG

[8] “Common Object Request Broker Architecture (CORBA) for embedded Specification, Version 1.0”, OMG Document Number: formal/2008-11-06, OMG

[9] D.C. Schmidt, D.L. Levine, and C. Cleeland, “Architectures and Patterns for High-Performance, Real-Time CORBA Object Request Brokers,” Advances in Computers, M. Zelkowitz, ed., Academic Press, 1998

[10] S. Lo, S. Pope, "The Implementation of a High Performance ORB over Multiple Network Transports", MIDDLEWARE'98

[11] G. Coulson, S. Baichoo, "Implementing the CORBA GIOP in a high-performance object request broker environment", Distributed Computing, Spinger, April 2001

[12] The omniORB version 4.1 User's Guide, D. Grisby, Apasphere Ltd. Sai-Lai Lo, David Riddoch, AT&T laboratories Cambridge, July 2007,. URL: http://omniorb.sourceforge.net/index.html

[13] URL: http://git.xilinx.com/cgi-bin/gitweb.cgi

[14] http://xilinx.wikidot.com/device-tree-generator

[15] ISE Design Suite 10.1 Release Notes and Installation Guide, Xilinx

[16] P. Tuma, A. Buble, "Overview of the CORBA Performance", Proceedings of the 2002 EurOpen.CZ Conference, Znojmo, Czech Republic, Sep 2002

[17] P. Brebner, et al.,"Middleware Benchmarking: Approaches, Results, Experiences, Concurrency and Computation: Practice and Experience", Vol. 17, No. 15, pp. 1799-1805, Wiley, Dec 2005

[18] T. Kalibera et al., "Automated Benchmarking and Analysis Tool", proceedings of First International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2006), Pisa, Italy, Copyright (C) ACM, ISBN 1-59593-504-5, Oct 2006

[19] A. Buble, L. Bulej, P. Tuma, "CORBA Benchmarking: A Course with Hidden Obstacles", proceedings of the 2003 International Parallel & Distributed Processing Symposium (IPDPS 2003), Nice, France, Copyright (C) 2003 IEEE, Piscataway, New Jersey, USA, ISBN 0-7695-1926-1, ISSN 1530-2075, pp. 279, CDROM DATA/W18_PMEO_11.PDF, Apr 2003

[20] M. Joseph. Cognitive Radio – An integrated Agent Architecture for Software Defined Radio. Thesis, Department of Teleinformatics, Royal Institute of Technology, Stockholm (Sweden), 2000.

[21] F. K. Jondral, “Parameter Controlled Software Defined Radio”. Software Defined Radio Technical Conference and Product Exposition, 2002.

[22] M. Joseph. The Software Radio Architecture. IEEE Communications Magazine, 33, 5, p.26-38, 1995

[23] M. Joseph. The Software Defined Radio. IEEE National Telesystems Conference, 1992

[24] Software Communications Architecture Specification, Version 2.2.2, Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS), 15 May 2006

[25] N. Hayes, “Software Communications Architecture”, July 2003

[26] Common Object Request Broker Architecture (CORBA) Specification, Version 3.1, OMG, January 2008 www.omg.org

[27] C. R. Aguayo Gonzalez, et al., “Open-Source SCA-Based Core Framework and Rapid Development Tools Enable Software-Defined Radio Education and Research”, IEEE Communications Magazine, Vol. 47, no. 10, October 2009

[28] http://www.crc.gc.ca/en/html/crc/home/research/satcom/rars/sdr/products/scari_open/scari_open 2010

[29] http://www.spectrumsignal.com/ 2010

[30] D. Nussbaum, et al., “Open Platform for Prototyping of Advanced Software Defined Radio and Cognitive Radio Techniques”, 12th Euromicro Conference on Digital System Design / Architecture, Methods and Tools, 2009

[31] Z. Miljanic et al., “The WINLAB Network Centric Cognitive Radio Hardware Platform – WiNC2R”, Mobile Netw Appl (2008) 13:533 – 541

[32] Q. Zhang, A.B.J. Kokkeler, G.J.M. Smit, K.H.G. Walters, “Cognitive Radio baseband processing on a reconfigurable platform”, Physical Communication (2009) doi:10.1016/j.phycom.2009.02.008

[33] M. Joseph III. Cognitive Radio Architecture: The Engineering Foundation of Radio XML. New Jersey: John Wiley & Sons, Inc.

[34] http://ossie.wireless.vt.edu

[35] Z. Miljanic, I. Seskar, K. Le and D. Raychaudhuri, “The WINLAB Network Centric Cognitive Radio Hardware Platform – WiNC2R”, Mobile Netw Appl (2008) 13:533 – 541

[36] D. Nussbaum, K. Kalfallah, R. Knopp, C. Moy, A. Nafkha, P. Leray, J. Delorme, J. Palicot, J. Martin, F. Clermidy, B. Mercier, R. Pacalet, “Open Platform for Prototyping of Advanced Software Defined Radio and Cognitive Radio Techniques”, 12th Euromicro Conference on Digital System Design / Architecture, Methods and Tools, 2009

[37] Q. Zhang, A.B.J. Kokkeler, G.J.M. Smit, K.H.G. Walters, “Cognitive Radio baseband processing on a reconfigurable platform”, Physical Communication (2009) doi:10.1016/j.phycom.2009.02.008

[38] A. Viebmann, et al., “FALCON, a software defined radio transceiver concept”, IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, no. 1, September 2006, pp. 1414-1418

[39] M. Horowitz, W. Dally, “How scaling will change processor architecture”, Proceedings of the IEEE Solid-State Circuits Conference, 2004, Digest of Technical Papers, ISSCC, pp. 132-133 (February 2004)

[40] T. Limberg, B. Ristau, and G. Fettweis, “A Real-Time Programming Model for Heterogeneous MPSoCs”, Proceedings of the 8th international workshop on Embedded Computer Systems: Architecture, Modeling and Simulation, Pages: 75 – 84, Samos, Greece, 2008.

[41] S. Pasricha, N. D. Dutt, M. Ben-Romdhane, “BMSYN: Bus Matrix Communications Architecture Synthesis for MPSoC”, IEEE Transaction on computer-aided design of integrated circuits and systems, VOL. 26, NO. 8, August 2007

[42] V. Dumitriu, G. N. Khan, “Throughput-oriented NoC Topology Generation and Analysis for High Performance SoCs”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, VOL. 17, NO. 10, October 2009

[43] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu, E. Deprettere, “Daedalus: Toward Composable Multimedia MP-SoC Design”, Proceedings of the 45th annual Design Automation Conference, Anaheim, California 2008

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

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

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

[47] W. Wolf, A. A. Jerraya, G. Martin, “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

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

[49] K. Sankaralingam et al., “Distributed Microarchitectural Protocols in the TRIPS Prototype Processor”, 39th Annual IEEE/ACM International Symposium on Microarchitecture, Page: 480-491, 2006.

[50] 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.

[51] M. Ito et al., “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

[52] D.N. Truong et 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.

[53] Z. Wang, O. Hammami, “A Twenty-four Processors System on Chip FPGA Design with On-Chip Network Connection”, IP SOC 2008, France.

[54] 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.

[55] Z.Wang, Design and Multi-Technology Multi-objective Comparative Analysis of Families of MPSOC PhD thesis, INPG, Nov. 2009.

[56] R.Allen and K.Kennedy, Optimizing Compilers for Modern Architectures: A Dependence-based Approach, Morgan Kaufmann, 2001.

[57] A. Yunheung Paek Navarro et al., “An advanced compiler framework for non-cache-coherent multiprocessors”, Parallel and Distributed Systems, IEEE Transactions on Volume 13, Issue 3, March 2002 Page(s):241 - 259

[58] U.Bondhugula, A.Hartono, J.Ramanujam, P.Sadayappan, “A practical automatic polyhedral parallelizer and locality optimizer”, PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation, June 2008.

[59] A.Kejariwal, A.V.Veidenbaum, A.Nicolau, M.Girkar, X.Tian, H.Saito On the exploitation of loop-level parallelism in embedded applications Transactions on Embedded Computing Systems (TECS) , Volume 8 Issue 2 January 2009

[60] 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.

[61] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: RAW machines", IEEE computer, 30(9):86-93, September 1997.

[62] Mechael Bedford Taylor, et al., "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams", Proceedings of the 31st annual international symposium on Computer architecture, ISCA'04, June 2004.

[63] The Message Passing Interface Forum, http://www.mpi-forum.org

[64] http://openmp.org/wp/

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

[66] OCP International Partnership http://www.ocpip.org

[67] Open Core Protocol Specification, Release 2.0, OCP-IP

[68] D.Culler , J.P. Singh , Anoop Gupta Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kauffman, 1998.

[69] R. Gupta, "Synchronization and communication costs of loop partitioning on shared-memory multiprocessor systems", IEEE Transactions on Parallel and Distributed Systems, Volume 3, Issue 4, July 1992 Page(s):505 – 512.

[70] S. Sriram and S. S. Bhattacharyya, Embedded Multiprocessors Scheduling and Synchronization, Marcel Dekker , 2000.

[71] Xilinx http://www.xilinx.com

[72] P. Gai, G. Lipari, M. Di Natale, M. Duranti, A. Ferrari, "Support for multiprocessor synchronization and resource sharing in system-on-programmable chips with softcores SOC", Conference, 2005. Proceedings. IEEE International 25-28 Sept. 2005 Page(s):109 - 110

[73] M. Monchiero, G. Palermo, C. Silvano, O. Villa, “Efficient Synchronization for Embedded On-Chip Multiprocessors”, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Volume 14, Issue 10, Oct. 2006 Page(s):1049 - 1062

[74] S. Liu, J-L.Gaudiot, “Synchronization Mechanisms on Modern Multi-core Architectures”, Asia-Pacific Computer Systems Architecture Conference 2007: 290-303

[75] B. B. Brandenburg, J. M. Calandrino, A. Block, H. Leontyev, J. H. Anderson, “Real-Time Synchronization on Multiprocessors: To Block or Not to Block, to Suspend or Spin?”, Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS '08. IEEE, 22-24 April 2008 Page(s):342 - 353

[76] S. Fide, S. Jenks, “Architecture optimizations for synchronization and communication on chip multiprocessors”, Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on 14-18 April 2008 Page(s):1 - 8

[77] T. Ono, M. Greenstreet, “A modular synchronizing FIFO for NoCs Networks-on-Chip”, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on 10-13 May 2009 Page(s): 224 – 233

[78] A.Nicolau, G.Li, A.Kejariwal, “Techniques for Efficient Placement of Synchronization Primitives”, Proceedings of the 14th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP’09, Pages 199-208, 2009.

[79] CLooG: The Chunky Loop Generator. http://www.cloog.org.

[80] http://www.piplib.org/

[81] “The Polyhedral Loop Parallelizer: LooPo”, University of Passau, http://www.infosun.fim.uni-passau.de/cl

[82] J.Cavacoz, G.Fursin, F.Agakov, E.Bonilla, M.F.P. O’Boyle, O.Temam, “Rapidly Selecting Good Compiler Optimizations using Performance Counters”, Proceedings of the International Symposium on Code Generation and Optimization”, Pages 185-197, 2007.

[83] K.D.Cooper, D.Subramanian, and L; Torczon, “Adaptive optimizing compilers for the 21st century”, Journal of Supercomputing, 23(1):7-22, August 2002.

[84] D.Parello, O.Temam, A.Cohen, J.Verdun, “Toward a Systematic, Pragmatic and Architecture-Aware Program Optimization Process for Complex Processors”, Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Page 15, 2004.

[85] D.Barthou, S.Donadio, P.Carribault, A.Duchateau, W.Jalby, “Loop Optimization using Compilation and Kernel Decomposition”, International Symposium on Code Generation and Optimization”, 2007.

[86] N.Vasilache, A.Cohen, L.Pouchet, “Automatic Correction of Loop Transformations”, Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques”, Pages 292-304, 2007.

[87] A.Hartono, M.Manikandan, C.Bastoul, A.Cohen, S. Krishnamoorthy, B.Norris, J.Ramanujam, P.Sadayapppan, “Parametric Multi-Level Tiling of Imperfectly Nested Loops”, Proceedings of the 23rd international conference on Supercomputing, Pages 147-157, 2009.

[88] C.Ancourt, F.Irigoin, Scanning polyhedra with DO loops, Proceedings of the third ACM SIGPLAN symposium on Principles and practice of parallel programming, p.39-50, April 21-24, 1991, Williamsburg, Virginia, United States

[89] C.Bastoul. Efficient code generation for automatic parallelization and optimization. In ISPDC, page 23, 2003.

[90] C.Bastoul, Code Generation in the Polyhedral Model Is Easier Than You Think, Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, p.7-16, September 29-October 03, 2004.

[91] F. Irigoin, R. Triolet, Supernode partitioning, Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, p.319-329, January 10-13, 1988, San Diego, California, United States.

[92] F.Quilleré , S.Rajopadhye , D.Wilde, Generation of Efficient Nested Loops from Polyhedra, International Journal of Parallel Programming, v.28 n.5, p.469-498, Oct. 2000.

[93] TLoG: A Parameterized Tiled Loop Generator. Available at http://www.cs.colostate.edu/MMAlpha/tiling/.

[94] E.S.Chung and al, ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs, ACM Tr

[95] R.Benmouhoub, O.Hammami, “MOCSOC: Multiprocessor on chip synthesis from OCCAM”, SASIMI, 3-4 April, 2006.

[96] F.Sun, Ravi, S., Raghunathan, A., Jha, N.K., “Application-specific heterogeneous multiprocessor synthesis using extensible processors”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 25, Issue 9 Sept. 2006 Page(s):1589 – 1602

[97] I. Auge, F. Petrot, F. Donnet, P. Gomez, “Platform-based design from parallel C specifications”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 24, Issue 12, Dec. 2005 Page(s):1811 – 1826

[98] H. Nikolov, T. Stefanov, E. Deprettere, “Multi-processor system design with ESPAM “, Proceedings of the 4th international conference Hardware/software codesign and system synthesis, 2006. 22-25 Oct. 2006 Page(s):211 – 216

[99] H. Nikolov, T. Stefanov, E. Deprettere, “Systematic and Automated Multiprocessor System Design, Programming, and Implementation”, Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE, Volume 27, Issue 3, March 2008.

[100] X.Li and O.Hammami, An Automatic Design Flow for Data Parallel and Pipelined Signal Processing Applications on Embedded Multiprocessor with NoC: Application to Cryptography , International Journal on Reconfigurable Computing, Hindawi, 2009.

[101] M. D. Hill, M. R. Marty, “Amdahl's Law in the Multicore Era”, ComputerVolume 41, Issue 7, July 2008 Page(s):33 - 38

[102] X. Sun, Y. Chen, “Reevaluating Amdahl’s law in the multicore era”, Journal of Parallel and Distributed Computing, May. 2009.

[103] S.Schneider, J.Yeom and D.S.Nikolopoulos, “Programming Multiprocessors with Explicitly Managed Memory Hierarchies”, IEEE Computer, pp.28-34, Dec.2009.

[104] L.Pouchet, U.Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam and P. Sadayappan, “Hybrid Iterative and Model-Driven Optimization in the Polyhedral Model”, INRIA Technical Report No. 6962, June 2009.

[105] 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.

[106] J. Kim, S. Hyeon, and S. Choi, “Implementation of an SDR system using graphics processing unit”, IEEE Communication Magazine, Vol. 48, no. 3, March 2010.

[107] M. Palkovic, et al., “Future Software-Defined Radio Platforms and Mapping Flows ", Signal Processing Magazine, IEEE Volume: 27 , Issue: 2 Digital Object Identifier: 10.1109/MSP.2009.935386, Publication Year: 2010 , Page(s): 22 - 33

[108] F. Clermidy et al., “A 477mW NoC-based digital baseband for MIMO 4G SDR “, Digital Object Identifier: 10.1109/ISSCC.2010.5433920, Publication Year: 2010 , Page(s): 278 - 279

[109] L. Alaus et al., “Promising Technique of Parameterization For Reconfigurable Radio”, the Common Operators Technique: Fundamentals and Examples, march 2009, Revue Journal of Signal Processing Systems, Editor Springer New York ISSN 1939-8018 (Print) 1939-8115 (Online), DOI 10.1007/s11265-009-0353-4

[110] C. Moy, L. Doyle, Y. Sanada, “Cognitive Radio: From Equipment to Networks”, Annals of Telecommunications, Editorial article of the Special issue on Cognitive Radio, vol. 64, number 7-8, Aug. 2009 ; DOI : 10.1007/s12243-009-0125-y

[111] F. Harris, R. W. Lowdermilk, “Software defined radio: Part 22 in a series of tutorials on instrumentation and measurement”, Instrumentation & Measurement Magazine, IEEE, Volume: 13 , Issue: 1, Digital Object Identifier: 10.1109/MIM.2010.5399214, Publication Year: 2010 , Page(s): 23 - 32

[112] M. Hasan et al., “A middleware based network hot swapping solution for SCA compliant radio”, Consumer Electronics, IEEE Transactions on, Volume: 55 , Issue: 3, Digital Object Identifier: 10.1109/TCE.2009.5277994, Publication Year: 2009 , Page(s): 1315 - 1321

[113] J. Guan, X. Ye, J. Gao, B. Wang, “The Flow of Software Defined Radio Waveform Development Based on SCARI”, Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on Digital Object, Identifier: 10.1109/WICOM.2009.5302559, Publication Year: 2009 , Page(s): 1 - 4

[114] J. L. Shanton, “A software defined radio transformation”, Military Communications Conference, 2009. MILCOM 2009. IEEE Digital Object Identifier: 10.1109/MILCOM.2009.5379729 Publication Year: 2009 , Page(s): 1 - 5

[115] T. Kempf, S. Wallentowitz, G. Ascheid, R. Leupers, H. Meyr, “A Workbench for Analytical and Simulation Based Design Space Exploration of Software Defined Radios”, VLSI Design, 2009 22nd International Conference on Digital Object Identifier: 10.1109/VLSI.Design.2009.24 Publication Year: 2009 , Page(s): 281 - 286

[116] H.Wang, Thèse "Architectures reconfigurables à base d’opérateur CORDIC pour le traitement du signal: Applications aux récepteurs MIMO", SCEE, Supelec, France, April 28, 2009.

[117] S.T.Gul, Thèse "Optimization of Multi-standards Software Defined Radio Equipments: A Common Operators Approach ", SCEE,
Supelec, France, April 28, 2009.

[118] A.Ykhlef, Thèse "Séparation aveugle de sources dans les systèmes de communication MIMO", SCEE, Supelec, France, 2008

[119] L.Godard, Thèse "Modèle de Gestion Hiérarchique Distribuée pour la Reconfiguration et la Prise de Décision dans les Équipements de Radio Cognitive", SCEE, Supelec, France, 2008

[120] A.Al ghouwayel, "Intérêt des techniques de paramétrisation pour des architectures radio logicielle reconfigurables", SCEE, Supelec, France, 2008

[121] B.Riwahi, Thèse "Analyse et réduction du Power Ratio des systèmes de radiocommunications multi-antennes", SCEE, Supelec, France, 2008

[122] B. Le Guen, Thèse "Adaptation du contenu spatio-temporel des images pour un codage par ondelettes", SCEE, Supelec, France, 2008

[123] M. Ghozzi, Thèse "Des terminaux multi modes aux terminaux intelligents sensibles à leur environnement. Architecture d'un terminal Radio Cognitive", SCEE, Supelec, France, 2008

[124] A. Kumar, S. Fernando, Y. Ha, B. Mesman, H. Corporaal, "Multi-Processor System-Level Synthesis for Multiple Applications on Platform FPGA" Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on Digital Object Identifier: 10.1109/FPL.2007.4380631 Publication Year: 2007, Page(s): 92 - 97

[125] K. Kokkinen, V. Turunen, M. Kosunen, S. Chaudhari, V. Koivunen, J. Ryynanen, "FPGA implementation of autocorrelation-based feature detector for cognitive radio", NORCHIP 2009

[126] D. Cabric, “Cognitive radios: System design perspective,” Ph.D dissertation, University of California, Berkley, 2007

[127] S. Shantaraskul, K. Moessner, “Implementation of a genetic algorithm-based decision making framework for opportunistic radio”, Communications, IET, March 26 2010, Page: 495-506, ISSN: 1751-8626

[128] L. Biard, D. Noguet, T. Gernandt, P. Marques, A. Gameiro, “A hardware demonstrator of a cognitive radio system using temporal opportunities”, Cognitive Radio Oriented Wireless Networks and Communications, 2009. CROWNCOM’09. 4th International Conference on

[129] J. M. Paul, D. E. Thomas, and A. S. Cassidy, “High-level modeling and simulation of single-chip programmable heterogeneous multiprocessors”, ACM Transactions on Design Automation of Electronic Systems, Volume 10, Issue 3, Page: 431 - 461, July 2005

[130] Y. Zhang, C. Gill, C. Lu, “Real-Time Performance and Middleware for Multiprocessor and Multicore Linux Platforms”, Embedded and Real-Time Computing Systems and Applications, 2009. RTCSA’09. 15th IEEE International Conference on

[131] A. Greiner, E. Faure, N. Pouillon, D. Genius, “A generic hardware / software communication middleware for streaming applications on shared memory multiprocessor system-on-chip”, Specification & Design Languages, 2009. FDL 2009. Forum on, ISSN: 1636-9874

[132] https://www.soclib.fr/trac/dev/wiki/Tools/Mwmr, 2010

[133] K. Goossens, J. Dielissen, A. Radulescu, “AEthereal network on chip: concepts, architectures, and implementations”, Design & Test of Computers, Volume: 22, Issue: 5, IEEE, 2005

[134] C. Lee, S. Kim, and S. Ha, “A systematic design space exploration of MPSoC based on synchronous data flow specification”, Journal of Signal Processing Systems, Volume 58, Issue 2, February 2010.

[135] H.D. Kim, C.S. Jeong, “Object Clustering for High Performance Programming”, Journal of Supercomputing, 19, 267-283, 2001

[136] D. Janadi Ram, A. Vijay Srinivas, P.Manjula Rani, "A model for Parallel programming over CORBA", Journal of Parallel and Distributed Computing, Volume 64, Issue 11, Pages 1256-1269, 2004

[137] M. Huo, S. Majumdar, "Performance of parallel architectures for CORBA-based systems", Proceedings of the 4th international workshop on Software and performance, Pages: 249-253, 2004.

[138] J. Keinert et al., "SystemCoDesigner - an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications", ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 14, Issue 1, Jan. 2009

[139] P. G. Paulin, C. Pilkington, M. Langevin, E. Bensoudane, D. Lyonnard, O. Benny, B. Lavigueur, D. Lo, G. Beltrame, V. Gagne and G. Nicolescu, "Parallel Programming Models for a Multiprocessor SoC Platform Applied to Networking and Multimedia", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, VOL. 14, NO. 7, July 2006

[140] A. Andriahantenaina, A. Greiner, "Micro-Network for SoC: Implementation of a 32-port SPIN network", Design, Automation and Test in Europe (DATE'03), Munich, Germany March 03-March 07, ISBN: 0-7695-1870-2

[141] 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

[142] K. Goossens, J. Dielissen, and A. Radulescu, " AEthereal Network on Chip: Concepts, Architectures, and Implementations", IEEE Design & Test, Volume 22, Issue 5, Pages: 414 - 421, 2005, ISSN: 0740-7454

[143] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip", Proceedings of the Conference on Design automation and test in Europe, Volume 2, 2004

[144] N. Hoven, R. Tandra, and A. Sahai, "Some Fundamental Limits on Cognitive Radio", University of California at Berkeley, February 11, 2005

[145] 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, 2005

List of publications

International conferences:

1. G.Tian and O.Hammami, “Performance Evaluation of Synchronization Mechanisms on 16 PE NoC Based MPSOC”, IEEE ICECSI09
2. O.Hammami and G.Tian, “Performance Evaluation of Parallel Applications On Multiprocessor Systems On chip”, the 14th IEEE Mediterranean Electrotechnical Conference, May, 2008, Ajaccio

Posters

3. G.Tian and O.Hammami, “Automatic Parallelization Experiments on 16 PE NoC Based MPSOC”, IEEE ASICON 2009 (Prize of Excellent Poster)

French conferences

4. O.Hammami, G.Tian, “Conception de système embarqué distribué à base de multiprocesseur MPSOC paramétrable”, Innovation Technologique et Systèmes de Transport (ITT’09), Paris, 2009
5. O.Hammami, G.Tian, “Analyse des performances des communications embarquées dans un environnement embarqué distribué à base de FPGA”, Innovation Technologique et Systèmes de Transport (ITT’09), Paris, 2009

Journal

1. G.Tian, O.Hammami and D. Etiemble, “Performance Evaluation of Automatic Parallelization and Multiprogramming on a NoC-based 16-PE Multi-core System on Chip”, Journal of Systems Architecture (Submitted)


ENSTA
ParisTech