Khalid Latif

Design Space Exploration for MPSoC Architectures

Turku Centre *for* Computer Science

TUCS Dissertations
No 166, December 2013

Design Space Exploration for MPSoC Architectures

Khalid Latif

To be presented with the permission of the Faculty of Mathematics and Natural Sciences of the University of Turku, for public criticism in Auditorium Beta on December 20, 2013, at 9:00 am.

University of Turku
Department of Information Technology
20014 Turun yliopisto

2013

Supervisors

Docent Tiberiu Seceleanu
ABB Corporate Research, Västerås
Västerås, Sweden

Professor Hannu Tenhunen
D.Sc. (Tech.) Ethiopia Nigussie
Department of Information Technology
University of Turku
Turku, Finland

Reviewers

Professor Olli Vainio
Department of Computer Systems
Tampere University of Technology
Tampere, Finland

Associate Professor Marisa López-Vallejo
Departamento de Ingeniería Electrónica E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid
Madrid, Spain

Opponent

Professor Koen Bertels
Computer Engineering, EEMCS
Delft University of Technology
Delft, The Netherlands

ISBN 978-952-12-2976-3
ISSN 1239-1883

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

Abstract

Multiprocessor system-on-chip (MPSoC) designs utilize the available technology and communication architectures to meet the requirements of the upcoming applications. In MPSoC, the communication platform is both the key enabler, as well as the key differentiator for realizing efficient MPSoCs. It provides product differentiation to meet a diverse, multi-dimensional set of design constraints, including performance, power, energy, reconfigurability, scalability, cost, reliability and time-to-market. The communication resources of a single interconnection platform cannot be fully utilized by all kind of applications, such as the availability of higher communication bandwidth for computation but not data intensive applications is often unfeasible in the practical implementation.

This thesis aims to perform the architecture-level design space exploration towards efficient and scalable resource utilization for MPSoC communication architecture. In order to meet the performance requirements within the design constraints, careful selection of MPSoC communication platform, resource aware partitioning and mapping of the application play important role. To enhance the utilization of communication resources, variety of techniques such as resource sharing, multicast to avoid re-transmission of identical data, and adaptive routing can be used. For implementation, these techniques should be customized according the platform architecture. To address the resource utilization of MPSoC communication platforms, variety of architectures with different design parameters and performance levels, namely Segmented bus (*SegBus*), Network-on-Chip (NoC) and Three-Dimensional NoC (3D-NoC), are selected. Average packet latency and power consumption are the evaluation parameters for the proposed techniques.

In conventional computing architectures, fault on a component makes the connected fault-free components inoperative. Resource sharing approach can utilize the fault-free components to retain the system performance by reducing the impact of faults. Design space exploration also guides to narrow down the selection of MPSoC architecture, which can meet the performance requirements with design constraints.

Dedicated to my late parents
Who believed in the richness of learning

Acknowledgements

It gives me great pleasure to be able to express my gratitude to the people and institutions that have helped me to accomplish this research work. First and foremost, I would like to thank my supervisors Docent Tiberiu Seceleanu, D.Sc. (Tech.) Ethiopia Nigussie, and Prof. Hannu Tenhunen for their inspiration, guidance and support.

I owe a huge debt of thanks to Docent Tiberiu Seceleanu for his invaluable and continuous guidance. His comments and criticism had a significant impact on the research presented here. I am very grateful to D.Sc. (Tech.) Ethiopia Nigussie for her guidance and encouragement throughout the period of my research. I am also indebted to Prof. Hannu Tenhunen for inspiring me to pursue my research in on-chip interconnection platforms. I wish to thank Prof. Olli Vainio from the Tampere University of Technology, Finland and Associate Professor Marisa López-Vallejo from Universidad Politécnica de Madrid, Spain for reviewing the thesis.

Turku Centre for Computer Science (TUCS) is gratefully acknowledged for funding my doctoral studies. This research work was also financially supported by the Academy of Finland, the Nokia Foundation, the Finnish Foundation for Technology Promotion (Tekniikan edistämissäätiö (TES)), the HPY:n Tutkimussäätiö and the Ulla Tuominen Foundation.

I would like to acknowledge all my colleagues at the IT department. I am grateful to everyone who have co-authored papers with me which are included in this thesis. I am thankful to D.Sc. (Tech) Ethiopia Nigussie, D.Sc. (Tech) Liang Guang and D.Sc. (Tech) Thomas Canhao Xu for proof-reading part of this thesis. Associate Professor Pasi Liljeberg deserves special thanks for his guidance in last phase of the thesis in spite of his busy schedules. I am grateful to my colleagues Kameswar Rao Vaddina, Rajeev Kumar Kanth, and Amir-Mohammad Rahmani for their time to have valueable discussions. I am very thankful TUCS and IT department management personnels especially Irmeli Laine, Tomi Mäntylä, Maria Prusila, and Late Maarit Pöyhönen.

I would like to express my deepest gratitude to my siblings for their constant support, encouragement and most importantly taking care of my responsibilities at home. I am also indebted to my brothers Tariq Mahmood Wains and Daoud Ahmad Wains, who always encouraged me to follow my

ambitions, irrespective of how much they missed me. I am grateful to my friends Ahmad Tariq and Abdul Samad from Tampere, Nauman Khan and Ali Shuja for their nice company in Turku, and Adnan Ahmed, Sulman Mahmood and Ganguly Debashish for spending nice time with them during my stay in Stockholm.

Turku, November 2013
Khalid Latif

Contents

List of Figures

List of Tables

List of Abbreviations

APL	Average Packet Latency
APAR	Average Packet Arrival Rate
BiNoC	Bidirectional NoC
BIST	Built In Self Test
BU	Border Unit
CA	Central Arbitration Unit
CMOS	Complementary Metal Oxide Semiconductor
CRC	Cyclic Redundancy Check
DCS	Dual Connected mesh Structure
DSB	Distributed Shared Buffer
ECC	Error Correction Code
ERAVC	Enhanced Reliability Aware Virtual Channel architecture
FLOPS	FLoating-point Operations Per Second
FU	Functional Unit
FVS	Full Virtual channel Sharing
GALS	Globally Asynchronous Locally Synchronous
GDS	Graphic Database System
IC	Integrated Circuit
IP	Intellectual Property
NI	Network Interface

MPSoC	MultiProcessor System-on-Chip
NoC	Network-on-Chip
OFDM	Orthogonal Frequency-Division Multiplexing
PE	Processing Element
PSDF	Packet SDF
PVS	Partial Virtual channel Sharing
SA	Segment Arbitration unit
SDF	Synchronous Data Flow
SegBus	Segmented Bus
SoC	System-on-Chip
TSV	Through-Silicon Via
UML	Unified Modeling Language
VC	Virtual Channel

Chapter 1

Introduction

Ever-increasing performance and reliability requirements on electronic systems are the key driving factors for evolution of the integrated circuit technology. These applications can be categorized as computation intensive and data intensive [19, 20]. Computation intensive applications require the computational elements with high processing power. On the other hand, data intensive applications are designed for working with large data-sets which require high bandwidth for inter-component data communications. In short, each type of application has its own requirements on processing and inter component communication hardware. Thus the conventional high performance computing platforms are not enough to fulfill the requirements of modern applications.

Buses and point to point connections are attractive solutions for on-chip communications with limited bandwidth requirements and smaller number of connected components. The bandwidth provided by bus can be effectively utilized because it is shared by several communication partners. However as the constant scaling of CMOS technology enabled integration of more computation components onto a single die, the bandwidth provided by on-chip buses was not scalable to higher number of integrated computation components [76]. Other than bandwidth limitations, power consumption because of high capacitive load due to long interconnection wires makes it inefficient to use the on-chip buses for larger number of components. As a consequence, segmented bus (*SegBus*) [63] and hierarchical bus [2] were proposed to deal with the mentioned issues. Once the number of connected components exceeded the certain limit, bandwidth limitations became a major bottleneck again. As a result, Network-on-Chip (NoC) [122], and Three dimensional NoC (3D-NoC) [3] platforms were proposed which have the potential to meet the scalability requirements.

Though the main task of efficient computing systems is to minimize the task completion time but power consumption, cost, and design time are also

the key parameters, which are leading the integrated circuit technology road map from single-core to multi- and many-core systems. Enormously parallel processing has become a mainstream and promising computing platform for high performance computing solutions with reduced power consumption [1]. To get full benefit of parallel processing, a multiprocessors system needs an efficient on-chip communication architecture [76, 108]. This thesis presents the scalability issues for on-chip interconnection platforms using novel system level optimization techniques for fault tolerance, power consumption and throughput performance.

1.1 Driving Forces for Parallel Computing

Almost a decade ago, Intel abandoned the plan of high clock rates and started putting multiple processing cores on a single chip because of severe heat problems [4]. Until that time, increasing the clock rate showed direct and positive impact on software performance without any change in software code. But this technological paradigm shift changed the basic principle of software programming to evaluate the application's performance. Only the applications, which can be redesigned in an efficient way to run parallel tasks on different cores can be benefited form this fundamental change.

The very first challenge after switching to multi-core systems was to provide an efficient interconnection platform for on-chip communications among different cores. Different on-chip communication platforms have been proposed, such as bus, hierarchical bus, bus matrix (crossbar), networks-on-chip (NoC) or customized application-specific architectures. Each platform has its own pros and cons. The evolution of on-chip communication architectures with time and when different manufacturers commercially manufactured them is shown in Figure 1.1. There are different driving forces which motivated the designers to switch from sequential to parallel computing. Few of those have been discussed here.

1.1.1 Computation

To meet the computation requirements of modern applications like multimedia signal processing, cloud based servers, satellites generating massive amounts of data or analyzing other larger scientific data-sets, parallel and distributed computing became the major focus of research community in computer science and engineering. Accordingly, high performance computing systems were commercially manufactured in recent years as can be seen in Figure 1.1. For instance, H.265 video codec recently manufactured by Texas Instruments [5], an 8-core digital signal processor at 1.25 GHz achieves 160 GFLOPS (160×10^9 floating-point operations per second). TILE64 processor manufactured by Tilera in 2007 [7], a 64-core processor is able to

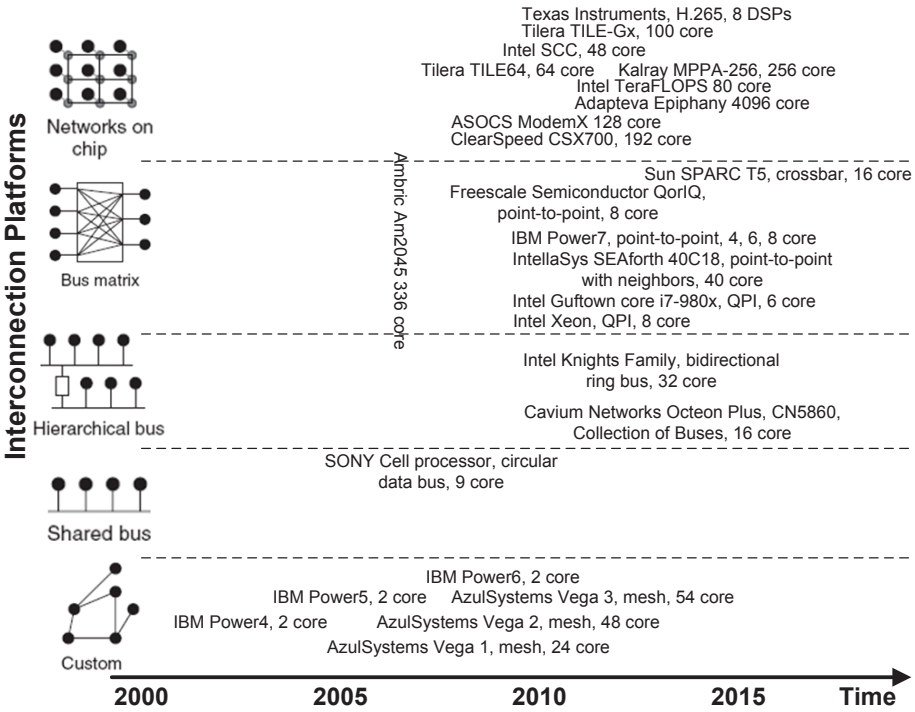


Figure 1.1: Evolution of on-chip communication architectures as a commercial product.

execute 166 billion 32-bit operations per second at 866 *MHz*. The Gulftown score i7-980X processor manufactured by Intel in 2010 [6], a 6-core processor at 3.33 *GHz* can achieve 79.992 GFLOPS. The Knights Ferry processor manufactured by Intel in 2010 [6], is a 32-core processors, which can exceed 750 GFLOPS at 1.2 *GHz*. The TeraFLOPS manufactured by Intel in 2011 [6], an 80-core processor at 3.16 *GHz* can achieve over 1.0 TFLOPS. Thus there is a pool of multi-core systems, where each system contains massive computing power. Now, it is up to the designer to select the appropriate hardware according the application requirements.

1.1.2 Communication

After having many cores with massive computation capability, the next issue is the inter core communication. Multi-core systems use different platforms for on-chip communications, which manifest various features and provide different levels of bandwidth. It is the task of the application designer to select the platforms which provide high bandwidth for the applications requiring high rates of data transfers between different cores.

The H.265 codec, TILE64 and TeraFLOPS processors use NoC platform

for on chip communications and provide 2 *Tbps*, 2660 *Gbps* and 1.62 *Tbps* bisection bandwidth respectively. Gulftown processor uses QuickPath Interconnect (QPI) which is a point-to-point processor interconnect developed by Intel and provides 409.6 *Gbps* bandwidth for on-chip communications. The Knights Ferry processor uses the bi-directional ring bus for on-chip communications which can achieve the bandwidth of 307.2 *Gbps*). All these bandwidth values are reported for the same operating frequencies, mentioned in section 1.1.1.

1.1.3 Power Consumption

Integrating a lot of computation and communication resources can make the system capable to meet the modern application requirements. However an efficient system should meet these requirements with minimum power consumption. The purpose of minimizing the power consumption of a chip is not only to save power but to minimize noise, timing contention, clock skew and chip temperature. This subsequently decreases the cost of cooling and packaging design, maximizes system reliability, simplifies power supply circuitry design, and increases battery lifetime [8]. At the same frequencies, mentioned in section 1.1.1, TeraFLOPS, H.265 codec, Gulftown and Knights Ferry processors consume 62 W, 7.5 W (at 75 ^{o}C, 9.5 W at 105 ^{o}C), 130 W TDP and 300 W TDP respectively. The thermal design power (TDP) refers to the maximum amount of power the cooling system in a computer is required to dissipate to prevent overheating. TILE64 processor consumes 60.8 W at 700 MHz operating frequency.

After having the detailed analysis and comparison of different multi-core systems from computation capability, communication bandwidth and power consumption perspectives, the target hardware should be selected according to the application requirements.

1.2 Design Challenges of Parallel Embedded Systems

On-chip communication architectures can be evaluated in terms of overheads and communication bandwidth availability [115]. The overheads are measured as silicon area, energy and power consumption. In order to meet high performance requirements, the communication architecture needs massive number of wires and complex control logic which cost in terms of a much larger power consumption and area overhead. The designer should ensure while selecting the communication architecture that the performance requirements are satisfied without violating the area and power consumption constraints. The second and major parameter to evaluate performance

of communication architecture is the value of supported bandwidth. For this purpose, design space exploration of communication architectures is necessary. In order to narrow down the design solution from a large set of design choices (topologies, control logics, application mapping techniques, traffic routing algorithms, and inter-core communication protocols are possible within the same architecture model), efficient design space exploration is critical to the shortening of time-to-market.

To evaluate the overheads and bandwidth of a distributed system, scalability is a primary metric. According to [9], scalability of distributed systems has three dimensions: numerical, geographical, and administrative. The numerical scalability represents the ability of the system to integrate an increasing number of components. The geographical scalability means the distance, over which the system components are scattered. The administrative dimension of scalability represents the administrative convenience with sizable system. The geographical scalability is extraneous for on-chip parallel systems and thus beyond the scope of this thesis. On the basis of design constraints and scalability dimensions, [10] has defined three aspects of scalability of on-chip parallel systems: Performance, Overhead and Design effort.

The performance metrics for any computing system are task completion time, on-chip communication latency, system throughput, and resource utilization. While switching to a parallel processing system from a single processor system, the basic parameter to evaluate the performance scalability is the speedup [11]. The speedup factor of multiprocessor systems is limited by application characteristics [12]. With advent of various on-chip communication platforms for parallel systems, the communication architecture has great impact on the speedup as well [115]. Thus collectively, there are several factors which limit the speedup factor like inter-core communication time, additional computations in parallel programmes and idle time slots for processors due to data dependencies on other processing elements.

All the mentioned issues for performance scalability can be dealt with at the expense of different overheads like silicon area and power consumption. Thus achieving speedup beyond the requirement without considering overheads is not an economical solution. An ideal solution should have better performance with minimum overheads. The silicon area has been the primary concern for chip designers. With continuous scaling of transistor size, silicon area became an exponentially cheaper resource relative to power and energy consumption [13]. Thus, the power consumption has replaced area as the primary constraint for SoC design. For smaller number of cores and lower bandwidth requirements, bus based communication is an efficient solution due to minimal communication overheads compared to NoC. On the other hand, bus based system does not scale well with more number of PEs because of the power consumption due to higher capacitive loads.

The last few decades were characterized by revolutions in information and communication technology. To cope with this ever changing revolution in our daily lives, never-ending update is required [14]. In this case, accelerating the time to market plays a vital role in product success. Generally, scalability in terms of design effort becomes a bottleneck to achieve this objective [15]. The design effort normally scales with number of integrated components. This design effort can be minimized by using techniques like reusability, early error detection and orthogonalization [16]. This minimizes the time for design and development process.

Another dimension, which affects system scalability is reliability. In conventional computing architectures, fault on a component makes the connected fault-free components inoperative. In such situations, the only drawback is not only the performance degradation. These affected fault-free components consume power as well. In modern systems, due to aggressive technology scaling, the probability of transient and permanent faults is increasing because of increased process variation and reduced noise-margins. Thus fault occurrence can severely and unexpectedly degrade the system performance. To deal with system reliability issues, fault detection should migrate to fault prediction [17]. It will reduce the number of unscheduled system stalls and cycle wastes. The main problem to switch from detection to prediction is to build precise, robust, power efficient, real-time updateable, and generic models for prediction. Another approach to retain the system performance in case of fault occurrence is to utilize the fault-free components which cannot be used due to the faults on the other resources.

Unfortunately, a system that is scalable in one or more scalability dimensions may cost in other scalability dimensions [18]. For the selection of on-chip communication platform, the application requirements should be considered with scalability issues.

1.3 Thesis Objectives and Contributions

A shift from computation centric design methodologies to communication centric design methodologies became essential in modern parallel systems [21]. Consequently, the design space exploration for on on-chip communication architectures is crucial. This thesis addresses the selection of on-chip communication platforms with an optimal tradeoff between system performance, reliability and overheads according to the application requirements and design constraints. To achieve this goal, different on-chip communication architectures and communication services have been proposed and evaluated regarding the computation performance, power consumption, silicon area and fault tolerance. The main contributions of this thesis are:

- The design flow for segmented bus (*SegBus*) platform including the

communication services is proposed. Different communication services like multicast, interrupt and split are introduced in *SegBus* platform. H.264 video encoder is used to demonstrate the proposed services. The designer can select the services according to the application and performance requirements.

- Application mapping technique for Multiprocessor System-on-Chip (MPSoC) platforms with homogeneous processing nodes is proposed. For this purpose, task prioritization criteria is proposed, which is used to map the application on MPSoC platform. NoC is used as a case study to compare the application mapping performance with existing task allocation techniques. The proposed technique shows significant improvement in system performance and reduction in power consumption.

- A novel Partial Virtual channel Sharing (PVS) NoC architecture is proposed which improves the utilization of resources to enhance the performance with minimal overheads. The PVS-NoC architecture also reduces the impacts of faults on performance and tolerates the faults in routing logic. The runtime allocation of the buffers depends on incoming load and fault occurrence. Furthermore, an efficient and reliable Network Interface (NI) assisted routing strategy for NoC using PVS architecture is proposed. For this purpose, NoC system is divided into clusters. Each Processing Elements (PE) can inject data to the network through a router in the cluster, which is closer to the destination node. The proposed architecture can also recover the PE disconnected from the network due to network level faults by allowing the PE to transmit and receive the packets through other routers in the cluster. Simulation results reveal that PVS architecture improves the performance significantly in presence of faults, compared to other virtual channel (VC) based NoC architectures.

- An efficient architecture to optimize system performance, reduce power consumption, and enhance reliability of stacked mesh 3D NoC is proposed. For this purpose, an inter-layer communication mechanism is developed which addresses the load balancing and system fault-tolerance issues by enhancing the buffer utilization. Overall, the performance is enhanced by an adaptive inter-layer routing scheme called AdaptiveZ. For the sake of simplicity and to observe the contribution of AdaptiveZ in performance improvement, intra-layer communication uses static XY routing algorithm. Our extensive experiments show significant reduction in power consumption and average packet latency as compared to the typical stacked mesh 3D NoC architectures.

1.4 Overview of the Thesis

The rest of the thesis is organized as follows. Chapter 2 presents the concept of on-chip communications for multi-core architectures and its main design choices like bus and network based architectures have been addressed. Chapter 3 describes the unified design methodology for application and MPSoC platform. *SegBus*platform has been used as an example MPSoC architecture. To improve the system performance and resource utilization, communication services like multicast, interrupt communication and SPLIT transactions for *SegBus*platform have been introduced in Chapter 4. To enhance the resource utilization in NoC based architectures, an application mapping technique has been presented in Chapter 5. Furthermore, the resource utilization analysis has been presented as well. On the basis of resource utilization analysis, Partial Virtual channel Sharing NoC (PVS-NoC) architecture has been proposed to address the utilization of virtual channel buffers in NoC router. Different techniques to reduce the impact of faults on system performance have been discussed in Chapter 6. Chapter 7 presents the adaptive inter layer communication mechanism for 3D NoC-Bus hybrid architectures to enhance the system performance and reduce the overall power consumption. Chapter 8 concludes the dissertation.

1.5 Research Publications

The work discussed in this thesis is based on and extended from the international journals and proceedings of conferences listed below:

1. Khalid Latif, Ethiopia Nigussie, Tiberiu Seceleanu. Unified Prioritization Criteria for Application Mapping on MPSoC Platforms. Submitted to ACM Transactions on Embedded Computing Systems (TECS).
   Author's contributions: The author contributed with the problem formulation, conducted experiments and wrote the manuscript.

2. Khalid Latif, Amir-Mohammad Rahmani, Ethiopia Nigussie, Tiberiu Seceleanu, Martin Radetzki, Hannu Tenhunen. Partial Virtual Channel Sharing: A Generic Methodology to Enhance Resource Management and Fault Tolerance in Networks-on-Chip. Journal of Electronic Testing: Theory and Applications (Springer-JETT), 29(3), pp 431-452, June 2013. DOI 10.1007s10836-013-5389-5
   Author's contributions: The author formulated the resource sharing technique to enhance the system performance and address the fault tolerance issues. The author also wrote the manuscript. Amir-Mohammad Rahmani performed the simulations.

3. Khalid Latif, Amir-Mohammad Rahmani , Tiberiu Seceleanu, Hannu Tenhunen. Cluster based Networks-on-Chip: An Efficient and Fault-Tolerant Architecture using Network Interface Assisted Routing. Accepted for publication in IGI-Global International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), 2013.
   Author's contributions: The author formulated the network interface assisted routing technique to reduce the network load and wrote the manuscript.

4. Khalid Latif, Tiberiu Seceleanu, Cristina Seceleanu, Hannu Tenhunen. Service based communication for MPSoC platform-SegBus. Microprocessors and Microsystems, Elsevier, 35(7), pp. 643-655, October 2011. DOI 10.1016j.micpro.2011.06.006
   Author's contributions: The author contributed with the problem formulation, conducted experiments and wrote the manuscript.

5. Amir-Mohammad Rahmani, Khalid Latif, Pasi Liljeberg, Juha Plosila, Hannu Tenhunen. A Stacked Mesh 3D NoC Architecture Enabling Congestion-Aware and Reliable Inter-Layer Communication. In proceedings of IEEE/Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP) in Special Session on On-Chip Parallel and Network-Based Systems, pp. 423-430, Ayia Napa, Cyprus, February 2011.
   Author's contributions: The author proposed the idea, suggested experimentation methods, and wrote the manuscript.

6. Khalid Latif, Amir-Mohammad Rahmani, Tiberiu Seceleanu and Hannu Tenhunen, "An Autonomic NoC Architecture using Heuristic Technique for Virtual Channel Sharing," in Phan Cong-Vinh (Eds.), Autonomic Networking-on-Chip: Bio-inspired Specification, Development, and Verification. Part of the Embedded Multi-core Systems (EMS) Book Series, CRC Press, December 2011. 1st Edition, pp. 47-68, ISBN: 143982911X, 9781439829110
   Author's contributions: The author formulated the problem and wrote the manuscript.

7. Khalid Latif, Amir-Mohammad Rahmani, Tiberiu Seceleanu, Hannu Tenhunen. Power- and Performance-Aware IP Mapping for NoC-Based MPSoC Platforms. In proceedings of IEEE International Conference on Electronics Circuits and Systems (ICECS), pp. 760-763, Athens, Greece, December 2010.
   Author's contributions: The author developed the mapping technique, conducted experiments and wrote the manuscript.

8. Khalid Latif, Moazzam Niazi, Hannu Tenhunen, Tiberiu Seceleanu, Sakir Sezer. Application development flow for on-chip distributed architectures. In proceedings of 21st IEEE International SoC Conference (SOCC), pp. 163-168, Newport Beach, CA, USA, September 2008.
Author's contributions: The author proposed the application design flow for MPSoC, conducted experiments and wrote the manuscript.

Chapter 2

Parallel MPSoC Architectures

Multiprocessor System-on-Chip (MPSoC) addresses the never ending complexity of applications. Like a traditional SoC design flow, MPSoC design starts with application specifications and requirements as shown in Figure 2.1. After the application specification, a functional model can be formulated using system-level design tools like Simulink or high level language such as C/C++ to verify and optimize the application algorithm using different configurations and random as well as normal inputs. In next step, the implementation approach for each application task is decided. It is a decisive step for power, performance and area optimizations of computation units. The computation tasks implemented as a circuit are called hard components. These hard components can be implemented using different logic architectures like Programmable Logic Devices (PLD), General-Purpose Logic (GPL) devices, Field Programmable Gate Arrays (FPGA) or Application-Specific Integrated Circuit (ASIC). While rest of the tasks are implemented as a sequential set of instructions on a microprocessor which are called soft components. At this stage, application architecture model is available. All inter-node communication in architecture model use point-to-point (P2P) links.

For applications, where subset of cores communicates with most of the nodes in application, P2P based communication architecture can provide the best communication performance at the expense of silicon area and high power consumption [21]. At this stage, MPSoC design flow differs from traditional SoC design flow. An efficient architecture for inter-core communications with minimum overheads is required for MPSoC designs. MPSoC's performance is highly dependent on the selection of communication architectures [56]. The focus of this thesis is to facilitate the designer for communication platform selection in MPSOC designs. In this chapter, different on-chip communication platforms will be presented which provide the foundation for the rest of the thesis.

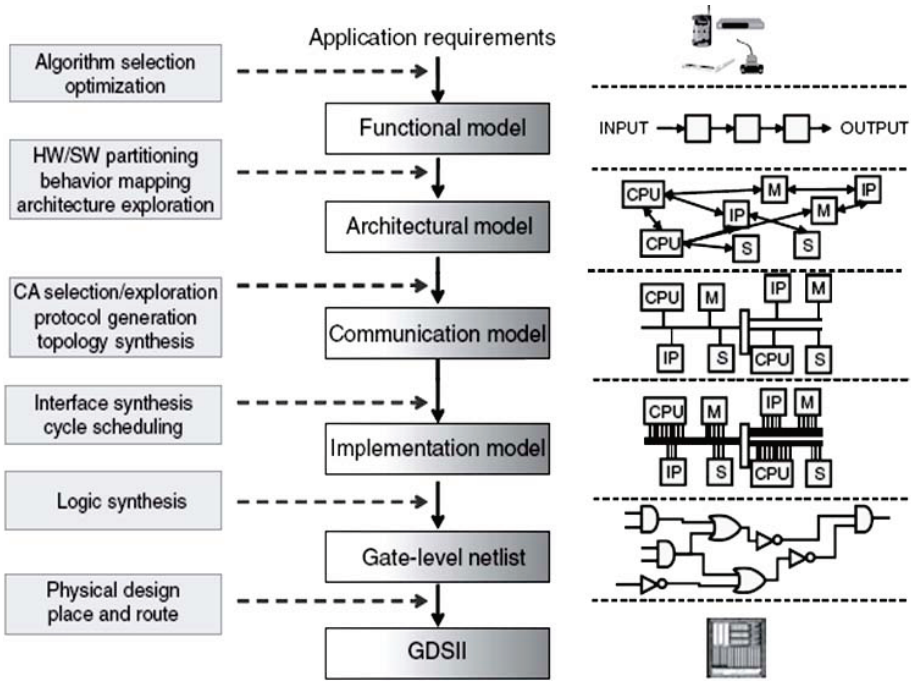


Figure 2.1: Design flow for Parallel MPSoCs [115].

2.1 Segmented Bus Architecture

A segmented bus (*SegBus*) is a bus which is partitioned into two or more segments. Each segment acts as a normal bus between modules that are connected to it and operates in parallel with other segments. Neighboring segments can be dynamically connected to each other in order to establish a connection between modules located in different segments. Due to the segmentation of the bus, parallel transactions can take place, thus increasing the performance. A high level block diagram of the segmented bus system which we consider in the following sections is illustrated in Figure 2.2.

The *SegBus* platform [63] is thought as having a single central arbitration unit (**CA**) and several local segment arbitration units (**SA**), one for each segment. The **SA** of each bus segment decides which device, generically referred as *functional unit* (**FU**), within the segment will get access to the bus in the following transfer burst.

Platform communication. Within a segment, data transfers follow a "traditional" bus-based protocol, with **SA**s arbitrating the access to local resources. The inter-segment communication is a packet based, circuit switched approach, with the **CA** having the central role. The interface components between adjacent segments, the *border units* - **BU**s, are basically FIFO ele-

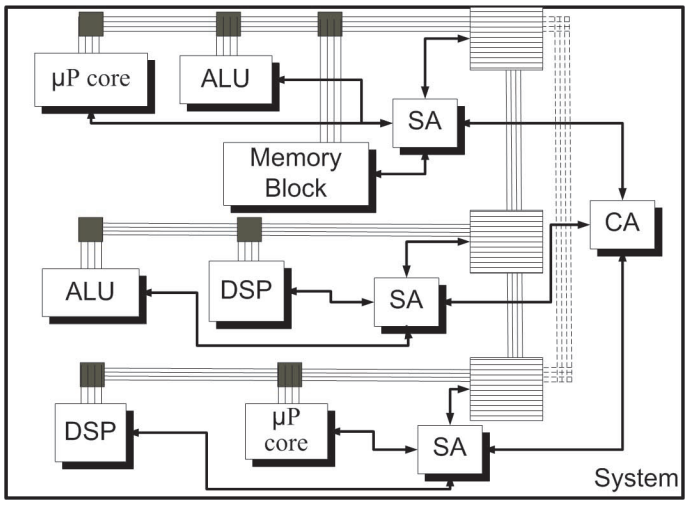


Figure 2.2: Segmented bus structure.

ments with some additional logic, controlled by the **CA**. A brief description of the communication is given as follows.

Whenever one **SA** recognizes that a request for data transfer targets a module outside its own segment, it forwards the request to the **CA**. This one identifies the target segment address and decides which segments need to be dynamically connected in order to establish a link between the initiating and targeted devices. When this connection is ready, the initiating device is granted the bus access. This one starts filling the buffer of the appropriate bridge with the packet data. The latter is taken into account by the corresponding next segment **SA** which forwards it further, until it reaches the destination. At this point, the **SA** of the targeted segment routes the packet to the own segment lines, from here it is collected by the targeted device.

A transfer from the initiating segment k to the target segment n is represented in Figure 2.3. The packet structure shown in figure contains different fields. Destination ID field is the **FU**'s ID as packet destination. Similarly Source ID is the **FU**'s ID as packet source. Data payload is the actual data to be transferred. The segments from k to n are released for possible other inter-segment operations in a cascaded manner, from the source k to the destination, n as specified by the packet header. However, the figure stresses the relatively long duration of an inter-segment transfer: whenever the data has arrived in the **BU** FIFOs, such a transaction collides with ongoing local activities. A solution in this sense, that is, speeding up the global communication, comes in the form of interrupts [61]: when a data packet arrives at one **BU**, the local operations of the next segment to be traversed is interrupted, to make way for the inter-segment packet.

The arbitration at **CA** level, that is, for global transfers, implements

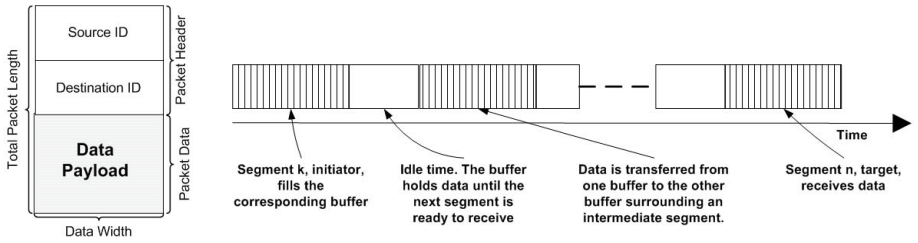


Figure 2.3: The data packet structure and the flow of an inter-segment packet transfer.

the application dataflow, with respect to these transfers. Hence, one has to implement accurate control procedures for inter-segment transfers, as possible conflicting requests must be appropriately satisfied, in order to reach performance requirements and to correctly implement applications.

The bus snooping mechanism is shown in Figure 2.4. Wrapper (**W**) provides the abstraction between processing elements (**PE**) and communication platform to make the system plug-and-play. Here, the task of the wrapper is requesting the bus, reading data from bus and other control signals communication. In packet based communication, packetization and depacketization are additional tasks of the wrapper.

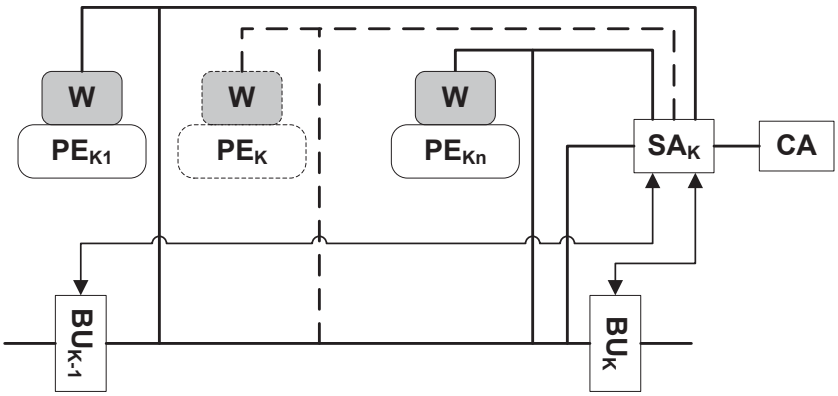


Figure 2.4: Abstraction between computation and communication.

The *SegBus* platform specifics consist in a set of global parameters that have a great impact on the implementation [63]: (i) topology - a linear or circular geometry; (ii) number of segments; (iii) size of the data packet.

2.2 Network-on-Chip

Network-on-Chip (NoC) is a general purpose on-chip communication concept which answers to the problems induced by wire density in SoCs and offers

high throughput, which are the basic requirements to deal with complexity of modern systems [56,76,122]. It is an attractive solution to replace the conventional communication architectures such as shared buses or point-to-point dedicated links. NoC provides better scalability than on-chip buses because as more resources are introduced to a system, also more routers and links are introduced to connect them to the network. The additional links and routers fulfill the communication capacity requirements for new resources. A typical NoC based system consists of processing elements (PE), network interfaces (NI), routers (R) and inter-router communication channels as shown in Figure 2.5.

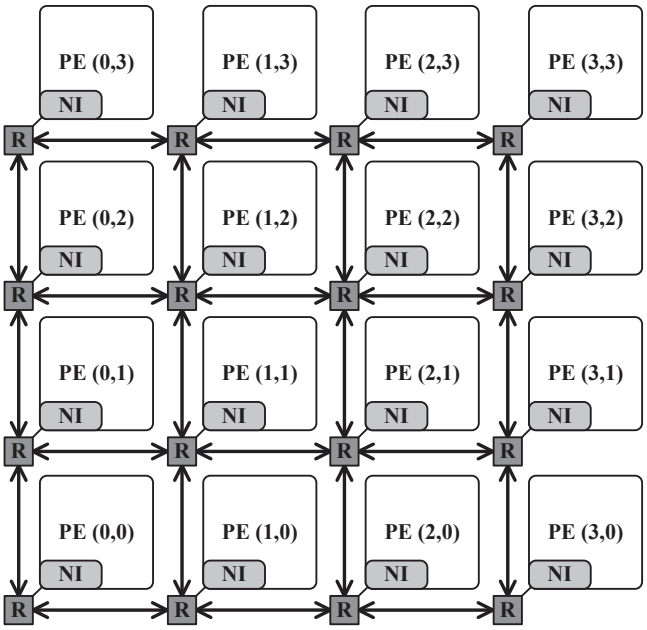


Figure 2.5: Homogeneous 4×4 NoC with Mesh Topology.

The NoCs can be categorized as homogeneous and heterogeneous. Homogeneous NoCs are based on the replication of identical PE units, while heterogeneous NoCs are based on variety of PE cores. The PEs can be a general purpose processor core, a cache bank, a memory controller, an FPGA block or even a custom logic component. Generally, heterogeneous NoCs are more efficient than homogeneous ones. However, homogeneous NoCs provide better flexibility, fault tolerance and scalability [22]. PEs act as the source and sink for the information/data packets, inject to the network. In rest of the thesis, homogeneous NoCs are used to provide the better comparison of communication performances. Thus, all possible software procedures are mapped within the general purpose hardware devices.

In NoC based system, PEs communicate with each other by breaking

up a message into small packets for transmission. At destination node, the received packets are re-ordered to extract the real message. The network interface (NI) is the component which packetizes the data before injecting it to the network and de-packetizes the packet after it leaves the network but before delivering it to the destination PE [122]. The packetization/de-packetization provides abstraction between computation and communication and makes the system plug-and-play for heterogeneous PEs [108]. A well designed NI enables the PE to utilize the full bandwidth with minimum latency offered by the network.

The task of the router (R) is to compute the route of the received packets and direct them towards the destination accordingly. These packets are received either from neighboring routers or the local PE. The router can be buffered or bufferless. For buffered routers, packets are stored in router buffers. The routing information is extracted from packet header flit and route is computed according to the implemented routing strategy. If the destination address of the packet matches with the address of local PE attached to the router, the packet is delivered the attached PE core, otherwise the packet is routed to the neighboring nodes according to the computed route. The concept of virtual channels has been introduced to address the utilization of communication channels and avoid deadlock. The VC based router contains a routing logic, a VC allocator, a switch allocator, a VC identifier, a crossbar switch, and several VC buffers and as shown in Figure 2.6. The number of IO channels for the router depends on topology. Different NoC router architectures like BiNoC [103], ViChar [114] and DSB-NoC [118] have been proposed which address different issues.

All inter-router communication channels in NoC can be simultaneously used for data transmission, which provides a high level of parallelism. Different interconnection design techniques and wire models can be used to design these channels. Similarly, variety of data encoding/decoding techniques and on-chip signaling schemes are available for data transmission on these links [23].

Other than mentioned hardware (HW) components, there are three other concepts which define NoC: Topology, Routing techniques and Flow control [122]. All these elements play an important role in determining the NoC performance. Network topology describes the the arrangements of routers and communication channels. Like the NoC concept, most of the proposed NoC topologies have been borrowed from computer networks or from network of multiple computation units in supercomputers. The main topology metrics are degree, hop count and path diversity. The well known NoC topologies are ring, mesh, torus, folded torus and tree structures (binary tree and fat tree).

The routing process decides, which path should be followed by a packet to reach its destination. A router can be defined as an ordered set of inter-router

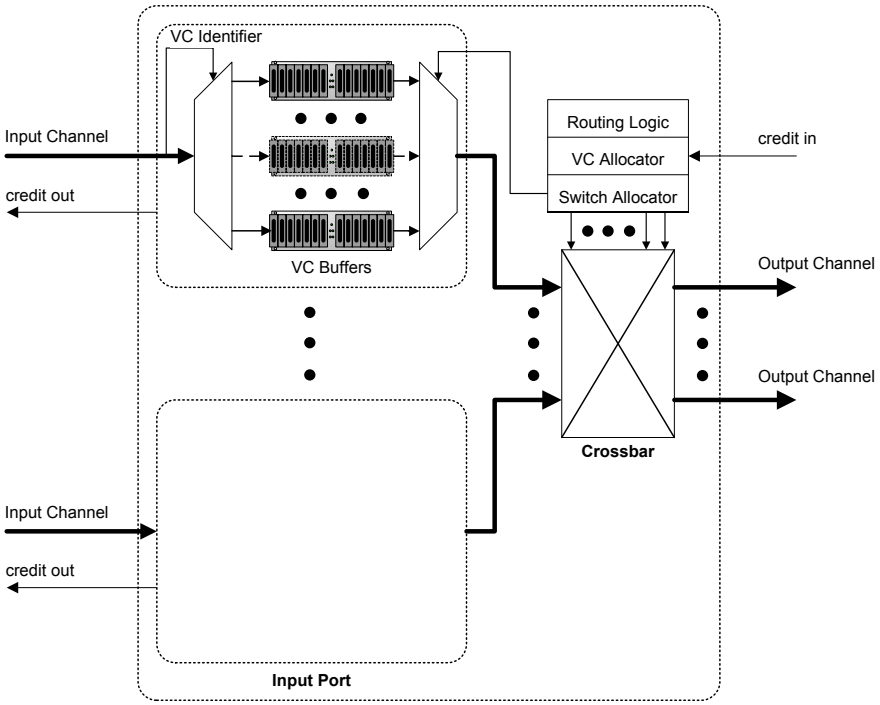


Figure 2.6: Typical Virtual Channel Router Architecture [109].

communication channels. The routing information is inserted in header flit of the packet by NI during packetization. There are two basic types of routing: source routing and node-table routing [122]. In case of source routing, NI contains the routing table and complete route is defined and saved in packet header at the source node. On other hand, node-table routing uses distributed approach where each router is assigned an address and also contains the routing table. The route is computed at each router on the way of packet. In this case, only the destination address is included in packet header. Normally, deterministic and oblivious routing algorithms use source routing, while adaptive routing uses node-table routing to take advantage of network state information at intermediate hops along the route.

Flow control mechanisms in NoC manages the resource allocation for packets as they are traversed through the network. It also defines the granularity level of packet transfer. In a situation, when two packets are competing for the same resource, flow control mechanism resolves the conflict. For bufferless routers, flow control schemes like hot-potato have been defined. For routers with buffering capacity, flow control schemes like store-and-forward, virtual cut-through and wormhole routing have been defined.

2.3 Three Dimensional NoC (3D-NoC)

A three-dimensional integrated circuit (3D-IC) is a major paradigm which can continue the Moore's law of integration [24]. 3D ICs offer a considerable reduction in the number and length of global interconnects as compared to traditional 2D ICs, which reduces wire delay and power consumption and enhances the system performance [25]. Due to all these benefits of 3D-ICs, the paradigm shift from 2D-NoC towards 3D-NoC was a very necessary step.

Different 3D-NoC architectures have been proposed [26], like Symmetric 3D Mesh, Ciliated 3D Mesh, Hybrid 3D NoC-Bus, True 3D NoC, Tree-Based 3D NoC and De-Bruijn Graph-Based 3D NoC. The simplest one for understanding is Symmetric 3D Mesh NoC. This approach is an extension of 2D mesh NoC, where in addition to existing links, each node has two additional links (up and down) for inter-layer communications. A 3×3×3 3D Symmetric Mesh NoC structure with 27 nodes is shown in Figure 2.7. For 3D Symmetric Mesh NoC structure, inter- or intra-layer hop bear the same characteristics at system level.

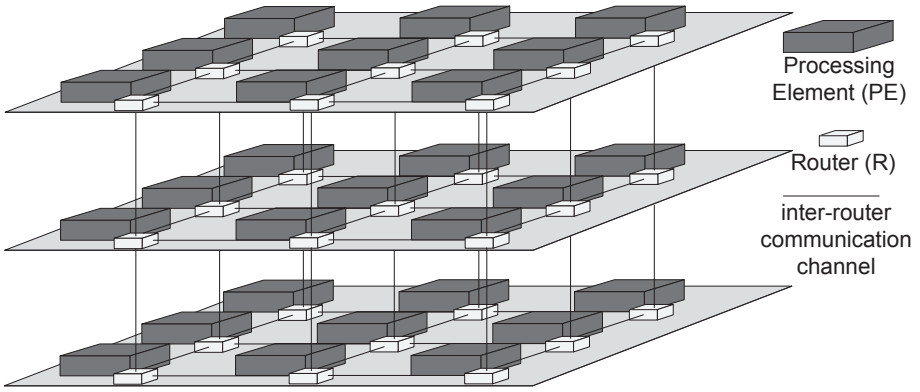


Figure 2.7: 3D Symmetric Mesh NoC structure.

The system performance of 3D Symmetric Mesh NoC structure is better than 2D Symmetric Mesh NoC because of two reasons: more communication resources (two extra ports for each router of 3D Symmetric Mesh NoC) and better bandwidth support by Through-Silicon-Vias (TSV) for inter-layer communications. TSV pads distributed across planar layers show relatively high area overheads. This may impose constraints on number of TSVs to avoid wire routing congestion [27]. Another drawback of 3D Symmetric Mesh NoC is the crossbar size. It requires a 7×7 router crossbar because of two extra links for inter-layer communications. It can be observed at a later stage from the simulation results that crossbar scales inefficiently due to increase of critical path length and extra gate count. To address all these issues and further enhance the performance of 3D-NoC structure, an efficient hybrid

3D NoC-Bus architecture and adaptive inter-layer routing scheme will be presented in Chapter 7.

With all the performance benefits, 3D technology introduces the on-chip thermal issue which increases the packaging and cooling costs. All these issues are faced by 3D-NoC architectures as well but it is beyond the scope of this thesis.

2.4 Chapter Summary

In this chapter, different parallel on-chip communication architectures have been presented which are the foundation for the next chapters. Different aspects and design issues of each platform like scalability, communication mechanism, bandwidth limitations and power consumption were discussed. The problems mentioned here have been addressed during the research work for this thesis. It is the task of the designer to select an appropriate interconnection platform according to the performance requirements and other design constraints such as power consumption for the application.

Chapter 3

Design Flow for *SegBus* Platform

One of the reasons behind the difficulties in MPSOC development is the lack of design methodologies [29]. Due to environmental and application requirements, the operation and communication characteristics of the employed devices and architectural instances may vary greatly from system to system. Performance measures are intrinsically related to the specifics of the underlying hardware platform. The lack of information availability at the higher abstraction (application) layers affects how specification requirements are reflected in the final system realization. Another important issue is the control of data transfers between different devices, as concurrent communication will certainly create conflicting situations.

At the same time, there is a growing demand for performance of multimedia applications. In order to address such issue, both performant platforms, but as well efficient design methodologies need to be developed. Employment of Intellectual Property (IP) designs is one of the high requirements in order to allow a fast deployment of new design solutions. Alternatively, hardware design languages might prove at times to be too restrictive, as only a small part of the design community has good respective knowledge. The tendency is therefore to replace, or to make transparent, whenever possible, VHDL (for instance) based design with higher level constructs, for instance C-like languages. The new challenges reside now in having a good platform representation at these higher levels, such that early evaluations are possible to perform.

The present work delves into aspects related to design methodologies for MPSOC. We describe the principles of a stepwise design methodology that targets a distributed on-chip architecture, Segmented Bus (*SegBus*) platform [63]. We continue the work of previous research results in the direction of raising the levels of abstraction at which such methodology is beneficial.

We also take a step further in the direction of automation, by providing platform models in the framework offered by Matlab [79]. We are interested in Matlab/Simulink as a high level design environment which allows the exploration of allocation results and offers the possibility for early assessment of application - platform mapping.

The most common current methods to deal with concurrency are *threads, semaphores, mutual exclusion locks*, etc. However, these approaches are intended to build *virtual* parallel environments, most often not well suited for current *heterogeneous* multi processor systems. For instance, threads are defined as sequential processes, exchanging information through shared memory resources, and several synchronization methods must be implemented in order the ensure the security and reliability of the shared data. This is because threads are highly non-deterministic, and a immense effort is dedicated to establishing an order of execution.

Our approach here is based on the existence of segment and central arbiters that contain the schedule for data exchanges between devices within the same segment, or in different ones. Out of a possible group of "enabled" transfers, these devices select the appropriate one with a built-in policy of granting. The present study builds on the work of Truscan et. al [39], and it provides an improved tooling support for the development of applications.

3.1 Existing Design Methodologies for On-Chip Communication Architectures

In recent years, research started to address on-chip solutions. Different on-chip communication architectures can use different design methodologies. Here we discuss the alternative design approaches for bus based on-chip communication architectures.

Lahiri et al. [33] address design optimality for a segmented bus platform similar to the *SegBus*. The segmented bus architecture [33] is, however, *memoryless*, different to our case, where the segments are separated by storage devices. Moreover, the protocols are fit to one application, and contentions can be extracted following a higher level simulation. The approach introduces a valuable simulation-based trace extraction, to indicate the communication patterns, considered consistent, after which an algorithmic solution is found to the allocation problem. Arbitration issues are not specifically addressed, and hence, possible contention problems and precedence relations are not analyzed. The intermediate arbitration tables, in our case, solve both the contention and the precedence issues.

Srinivasan et al. [37] introduce an AMBA-like hierarchy of a segmented bus. The authors employ genetic algorithms for finding optimal segmented bus allocations, but the methodology is not continued to other levels of

abstraction. There is a similarity with [33], in the sense that no control procedures, either for local or inter-segment activities, is presented. The arbitration is possibly organized following AMBA protocols, but this may affect both allocation optimality and solving the conflicting task execution.

De Jong [32] elaborates a system design flow based on UML and SDL, mainly for the purpose of control, communication and synchronization refinement of both hardware and software components. As it pertains more to the area of software-hardware co-design, this study is viewed as a complementary research to the present work.

Dekeyser et al. [31] propose a "Y-chart" methodological approach to multiple SOC system design with UML. While the results are applicable to our specific platform-based approach, in general, several design steps, such as application and platform refinement, granularity, communication restrictions, are not captured in [31].

The approach we illustrate here does not impose restrictions towards other MPSOC platforms. For example, the existing network-on-chip [76] models like [40, 41] can be used in order to enlarge the basis of the solution. Considered together with earlier results [39] on high level design methodologies, we approach the realization of a complete framework for the design of multiprocessor systems which provides the unified representation of both interconnection platform and application.

3.2 Unified Design Methodology for Application and MPSoC platform

The design methodology for multiprocessor platforms with focus on unified representation of interconnection platform and application will be presented in this section. The proposed design flow is illustrated in Figure 3.2. It integrates Matlab/Simulink, HDL design and co-simulation (DSP builder), in order to raise the design abstraction level which takes a step closer to design automation. We are interested in Matlab/Simulink since it allows the exploration of task allocation results and offers the possibility for early assessment of application - platform mapping. As a running example we employ a H.264 encoder [35] as application (Figure 3.1) and *SegBus* as the interconnection platform.

3.2.1 Tool Environment

Matlab / Simulink. Matlab Simulink Environment [79] is a tool commonly used for modeling, simulation, analysis and profiling of multi domain systems. These systems range from a simple adder to complex application like Video coding, transceiver synchronization in communication systems or

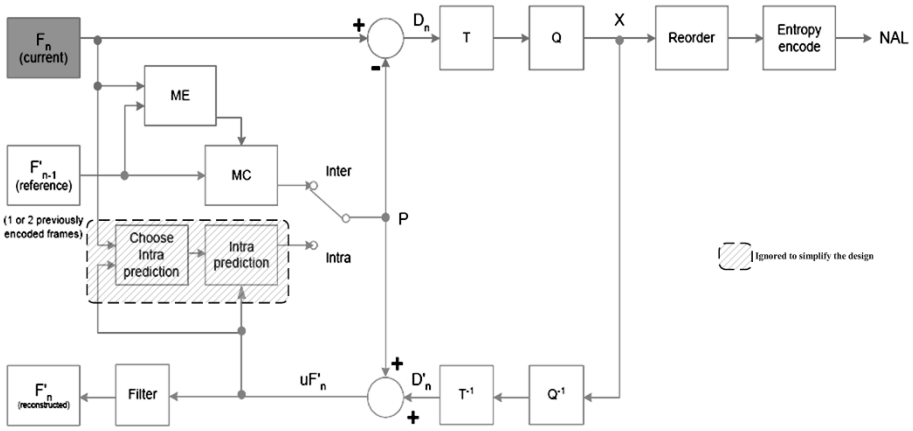


Figure 3.1: H.264 Encoder

control system design. It comprises of different block sets, libraries and programming functionalities. After the application specification, a working Simulink model can be modeled and application algorithm can be verified using different configurations and random as well as normal inputs.

Here, we use the "Video and image processing" blockset from Simulink to model the H.264 Encoder application. This blockset provides a variety of functions that can be used for modeling of Image and Video applications. The Matlab Simulink environment also supports obtaining the communication matrix necessary to compute the optimal allocation scheme for the interconnection platform.

Altera. At the time, the implementation technology for the *SegBus* platform is offered by Altera [42] devices. Hence, after application modeling and platform customization the flow is taken into the Quartus design environment, where previously defined functional units are mapped on actual devices. Following compilation, a simulation is performed within a Modelsim [43] framework.

Application development. We start by analyzing the targeted application by splitting it in processes. The interaction between these is observed in terms of input-output data-flows. In subsequent steps the top-level process is decomposed hierarchically into less complex processes and the corresponding data-flows between these processes.

The decomposition process is based on designer's experience and ends when the granularity level of the identified processes maps to existent library elements or devices that can be developed by the design team. The H.264 encoder application shown in Figure 3.1 was modeled in Simulink. The application tasks were further partitioned into number of sub-tasks in order to have enough granularity and communication complexity between tasks

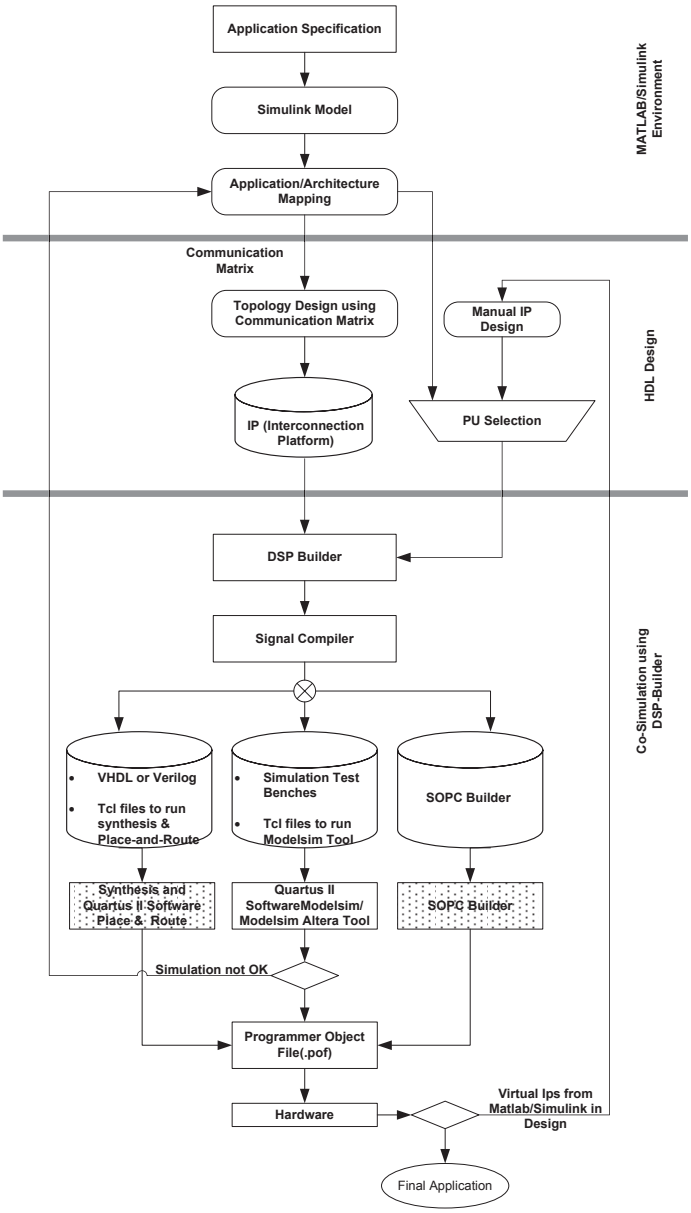


Figure 3.2: Unified Design Process for Application and MPSoC platform

to analyze the MPSoC platform under consideration. The corresponding model is shown in Figure 3.3. The values on the edges represent the number of transaction packets for processing of one video frame, where each data packet is of same size (66 bytes).

The communication between processes is organized as a *Packet SDF* diagram [62]. The PSDF representation will be used to extract the programs

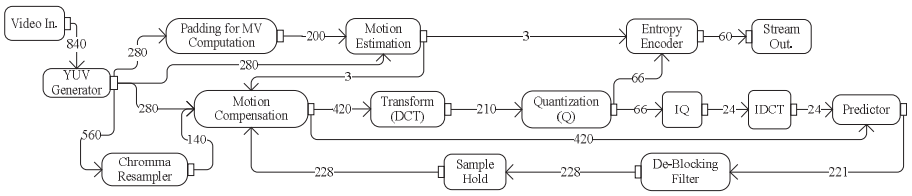


Figure 3.3: The H.264 Video encoder application model

controlling the activity (grant distribution schedule) of the **SA**s and of the **CA** [65].

The *Packet SDF*. A PSDF comprises mainly two elements: *processes* and *data flows*; data is, however, organized in packets. Processes transform input data packets into output ones, whereas packet flows carry data from one process to another. A *transaction* represents the sending of one data packet by one source process to another, target process, or towards the system output. In [62], a *packet flow* is a tuple of two values, P and T. P represents the number of successive, same size transactions emitted by the same source, towards the same destination; T is a relative ordering number among the (packet) flows in one given system. Thus, a flow is understood as the number of packets issued by the same process, targeting the same destination, and having the same ordering number.

A third element of the PSDF tuple characterizes the *kind* of the packet. The *kind* - a natural number I - identifies if a packet is to be routed to multiple destinations, thus establishing the modeling basis for multicasted or broadcasted transmissions as will be discussed in Chapter 4. Packet flows having same I value carry the same content of data, from the same source towards multiple destinations.

The PSDF of a certain system becomes hence a sequence of packet flows, $< (P_1, T_1, I_1), \ldots, (P_n, T_n, I_n) >$, $P_i \neq P_j$ where $\forall i, j \in \{1, \ldots, n\}$ and $T_1 \leq T_2 \leq \ldots \leq T_n$. Flows sourcing in the same node and with identical Is, will also have identical Ts, identifying a multicast packet.

The non-strictness of the relation between T values of the above definition models the possibility of several flows to coexist at moments in the execution of the system. In the case of the *SegBus* platform, this most often will describe *local* flows, that is flows where the source and the destination are situated in the same segment. However, considering a segment number larger than 3, *global* flows, where the source and the destination are in different segments, are also possible to be characterized by the same ordering number. In this case, it means that the **CA**, if possible, allows a simultaneous execution of transactions from all the "same T" global flows.

The corresponding PSDF diagram for application model in Figure 3.3 is shown in Figure 3.5. For the moment, the reader should ignore the parti-

From / To	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
P0	0	35840	17920	17920	17920	0	0	0	0	0	0	0	0
P1	0	0	0	0	8960	0	0	0	0	0	0	0	0
P2	0	0	0	12780	0	0	0	0	0	0	0	0	0
P3	0	0	0	0	176	0	0	176	0	0	0	0	0
P4	0	0	0	0	0	26880	0	0	0	0	26880	0	0
P5	0	0	0	0	0	0	13440	0	0	0	0	0	0
P6	0	0	0	0	0	0	0	4200	4200	0	0	0	0
P7	0	0	0	0	0	0	0	0	0	0	0	0	0
P8	0	0	0	0	0	0	0	0	0	1536	0	0	0
P9	0	0	0	0	0	0	0	0	0	0	1536	0	0
P10	0	0	0	0	0	0	0	0	0	0	0	0	14136
P11	0	0	0	0	14539	0	0	0	0	0	0	0	0
P12	0	0	0	0	0	0	0	0	0	0	0	14539	0

Figure 3.4: The communication matrix for the example

tion in segments, which is based on developments in the next sections. The processing elements ($P0, P1, \ldots, P12$) correspond respectively to YUV generator, Chroma resampler, Motion vector estimator units, etc. Figure 3.4 shows the the number of bytes to be transferred between two **FU**'s for the processing of one video frame.

3.2.2 Application Partitioning

We consider that the application is already partitioned and mapped on the available devices as described in Figure 3.5. In general,this means also that all possible software procedures are already mapped within the hardware devices. However, this is not the case in Figure 3.5, where all the devices are hardware elements.

At this moment, we can extract the communication features, that is, the frequency with which the various devices communicate with each other. We group these frequencies in the so-called "communication matrix". For the application at hand, this matrix is illustrated in Figure 3.4. The matrix was obtained by using the signal dimension option in Simulink.

The matrix is fed into the *PlaceTool* programme which delivers the allocation costs for various scenarios [34]. The output results of the *PlaceTool* are shown in Table 3.1, where || represents the segment borders. It can be observed that performance may go down by increasing the number of segments due to increase in communication overhead. In this case, a two segment platform delivers the best performance; however, we decide to select a three segment platform, in order to analyze a more complex structure as explained in [53]. The resulting segmented application model is obtained as in Figure 3.5.

Table 3.1: The task allocation and associated cost results.

Nr. Segs	Cost	Allocation	Improvement(%)
1	233000	0 1 2 3 4 5 6 7 8 9 10 11 12	100 (Reference)
2	132000	4 5 6 7 8 9 10 11 12 || 0 1 2 3	-43
3	137400	9 || 4 5 6 7 8 10 11 12 || 0 1 2 3	-41
4	143100	9 || 8 || 4 5 6 7 10 11 12 || 0 1 2 3	-39

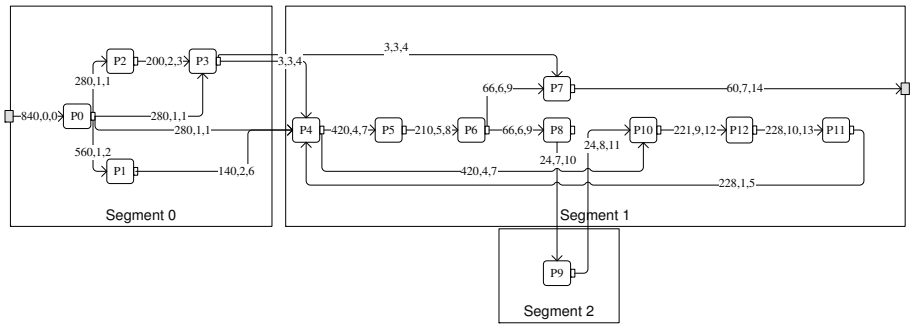


Figure 3.5: PSDF application specification

3.2.3 Code Generation

The segmentation process, while providing premises for a more performant execution, it raises the complexity related to finding a (good) schedule for both the processing tasks, but, mostly for the data transfers. The communication matrix is just a means to obtain the best possible allocation of resources with respect to global (inter-segment) transfers, but in order to implement the application functionality, both local (intra-segment) and global transfers must be appropriately scheduled.

The *PSDF* representation helps in creating such a communication & processing schedule. This is applied in two turns, once at the segment level and once at the platform level, in order to obtain the *programme* that will coordinate the activity of the segment and central arbiters, respectively.

Arbitration programmes. The programme for both the **SA**s and the **CA** is a grouped collection of VHDL statements placed in the controlling process of the arbiter's specification. Through specific mechanisms (described in the further paragraphs) the sequential execution of VHDL statements within a process is improved with a non-deterministic interleaved execution model. This gives the possibility for several lines to be perceived as executed in parallel, whenever appropriate.

The segment level arbitration is similar to any single segment bus situation. Activities in the segment are sequential, the **SA** deciding which device can access the bus lines. Any attached **BU** behaves like a local master, but the respective requests will have the highest priority. A master willing to transfer data on the bus raises the request line, while it also specifies the segment to which it wants to communicate. The **SA** identifies the target and, if it is outside the own segment, it forwards the request to the **CA**. If the request target is within the own segment, it proceeds to granting it.

These activities are collected in the *application control code* (ACC) which will drive the *SegBus* communication strategy at runtime [65]. The ACC is basically a binary matrix where each line controls the granting algorithm

such that the “right" master obtains the access to the bus. The code is parsed at every arbitration execution, and it contains *nrLines* lines of code - a parameter of the arbiter module. One line of code, assimilated to a *program line* (an array) has the following field structure (see also Figure 3.6),

1. *PC*. This is the Programme Counter, providing reference to the lines of instructions possible to be accessed from other instructions. It ranges from 0 to *nrLines*-1.

2. *Guard*. The *Guard* signals if the respective line is possible to be selected for execution. This is to enforce the necessary order of data transfers. Devices not part of a granted transfer may, meanwhile, proceed with their processing tasks. When $guard = 0$, the respective line is *enabled*, that is, the arbiter may consider it for selection. When $guard > 0$, the line is *disabled*, that is, it cannot be considered in the arbitration. The arbiter marks a line as *executed* whenever the respective *count* value reaches 0, by establishing $guard = nrLines$. Several lines with guards evaluated to 0 are potentially selectable for granting operation. However, only one of the instructions can be actually "executed".

3. *Source*. This field contains the address of the requesting master - the initiator of a transfer request. Devices on the platform (masters, slaves) are identified by an unique number.

4. *Destination*. This field contains the address of the targeted device - the slave.

5. *Dest_Seg*. This field contains the address of the segment where the *Destination* is located.

6. *toGrant*. This is the instruction for the arbiter to grant the requesting master. At this moment the specification is obsolete, but the field is preserved for future developments.

7. *Count*. Identifies the number of packets the master has to send to the specified targets. It corresponds to the first number in the PSDF description. Every time the master is granted and performs the transfer, this number is decreased. When it reaches 0, the line cannot be anymore selected for execution, even if the *Grant* field is also 0.

8. *enables*. The purpose of *enables* field is to address the data dependency between different tasks. *Disabled* lines will become *enabled* during the execution of the programme. The *enables* field (one per instruction) specifies which line can be moved towards enabledness at the end of the current transfer. This is achieved by subtracting 1 from the present value of the *Guard* field of the respective line.

	PC	Guard	Source	Destination	Dest_Seg	toGrant	count	enables
	5	0	2	5	0	2	200	6
Example:	6	1	6	3	0	6	120	9

Figure 3.6: The structure of the programme line, with two examples.

The programme construction considers also requests coming from **BU**s as events to be part of schedules. In this case, as an actual example, it may be interesting to observe the whole code describing the operation of the **SA** for segment 2 of the H.264 application, given as follows.

```
program(0) <= (guard => 0, source => RFL, dest => 9,
        dest_seg => 2,  togrant => RFL, count => 24,
        enables => 1);
program(1) <= (guard => 1, source => 17, dest => 5,
        dest_seg => 1, togrant => 17, count => 24,
        enables => 0);
```

In the above, the "RFL" term stands for "request from left". In brief, and in correlation with the flow described in Figure 3.5, the **SA** of segment 2 waits first that a transfer is received from left (segment 1), after which a transfer from the local device ($P9$ - Figure 3.5) is able to be executed, targeting a device in segment 1.

The application execution ends when all the lines are marked *executed*. That is, we have $PC = nrLines - 1$ and, for all lines, $guard = nrLines$. This triggers the arbiter to restore the initial values of the ACC content.

A similar approach is taken at the level of the **CA** for request-grant activities (containing only info about segment requests).

3.3 Experimental Results

We have applied the illustrated techniques for the implementation of a H.264 model on a traditional single bus platform and on a 3 segment *SegBus* platform, both on the same Stratix III device. The *SegBus* solution is characterized by a linear topology (as in Figure 3.5) and 66 words packet - similar for the single bus. The first one run at a clock frequency of 100 MHz, while the *SegBus* solution utilizes four clock domains (one for each segment - 100 MHz, 60 MHz, 50 MHz and one for the **CA** - 30 MHz).

The performance (throughput) results came close (within 1%) to the ones anticipated by the Table 3.1 for the respective solution. Intuitively, this also means an approximated 40% reduction in power. But the results from the Altera's *PowerPlay Power Analyzer* tool show that there is slightly higher then 5% reduction in power consumption of *SegBus*solution compared to the single bus solution, both in a vectorless and toggle-rate based approach. The

power consumption is not reduced with the same proportion as the throughput performance enhancement, this is due to power overhead of the cross border communications. These results are application dependent. While the core dynamic power dissipation was in the favor of the single bus solution (due to the additional switching activity of the **BU**s), the I / O and the total power dissipation go in the favor of the *SegBus* platform.

3.4 Chapter Summary

The construction of design methodologies for on-chip multiprocessor platforms, with the focus on segmented bus (*SegBus*) platform has been presented. It was studied how applications can be mapped on such distributed architectures and how to build the concrete level software procedures that will coordinate the control flow on the platform. The methodological chain used in this study (Matlab-DSP Builder-Quartus-Modelsim) proved to offer a suitable framework for the application development on the *SegBus* platform. We have described the employment of arbiter programmes for scheduling with a mostly static characteristic, but with a certain degree of (useful) nondeterminism in practice. The approach employs models developed in the Matlab-Simulink environment considering a unified representation of both platform and application. The running example is represented by the H.264 encoder. Allocation of processing elements on the platform, structure and functionality and the eventual control code for arbiters are the main topics described here.

Chapter 4

Communication Services in *SegBus* Platform

A computation mechanism comprises multiple functions and tasks generally expressed at different levels of abstraction. Individually, such functions or tasks can be considered as a service. We define here the two types of services for SoC: *computation* and *communication* services as shown in Figure 4.1. A computation service means that the chip offers service(s) which will (together) complete a certain task, or execute some application. A communication service contains functions that facilitate or support the data transfer from source to destination, such as monitoring, scheduling, arbitration, multicast and SPLIT transfers. These communication services can be customized and included in MPSoC communication platforms according to the application requirements. The implementation of communication services for on-chip communications is completely different from off-chip or computer system communications, though the basic concept is exactly same. For on-chip communications, challenges are deep submicron effect, cross talk, thermal noise and many other issues, which are not the key problems for typical communication systems. Apart from that, for systems based on MPSoC platforms, the customer considerations - power consumption, area and latency, are different than for communication systems. There are few services, required for on-chip communications but not used in telecommunication systems like cache-coherence or interrupt communication. Similarly, there are services for communication systems, which are not very important for on-chip communications like security. Some services cannot be completely defined as computational or communication, for instance the thermal monitoring. Such services are found then on the border between computation and communication services as shown in Figure 4.1.

Two well known architectures for implementation of communication fabrics for SoCs are transaction-based - *buses* and packet based - *NoCs* [56].

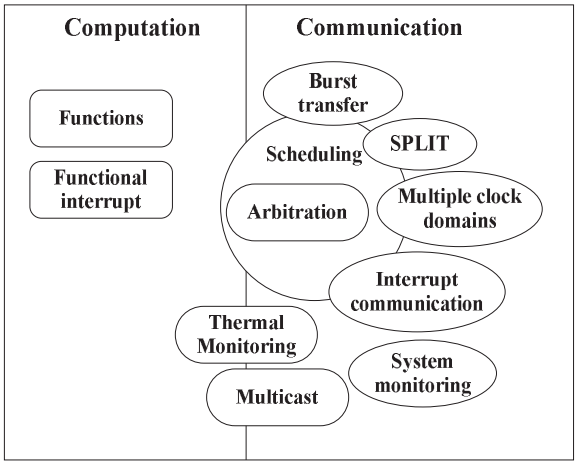


Figure 4.1: Classification of on-chip services.

Implementing the communication services for transaction based data transfers is not economical because of the service overhead for each transaction. Communication services built on packet based transactions can show a significant improvement in system performance. Communication services have already been implemented for NoC but for bus based systems (e.g., *SegBus*), only the obvious services like scheduling and arbitration are most commonly considered.

Congestion control, cache coherence and power consumption are the key problems for on-chip distributed architectures. A first technique to address congestion control is the avoidance of identical data re-transmission, which leads to the idea of a multicast approach in many streaming applications like H.264 codec, MP3 codec or video conference application. Multicast techniques are simple and easy to implement in bus based systems as compared to the implementations of networks on chip (NoC). Considering the later, apart from implementation, multicast introduces a serious communication overhead. For segmented bus approaches, multicast approaches bring only a negligible overhead. This makes multicast power and latency efficient for bus based systems.

We describe here the realization of a multicast protocol for the segmented bus platform *SegBus*. We perform this in the context of a perspective change, where activities related to the design and execution of applications on the *SegBus* platform are viewed now as *services*. While scheduling can be seen as a service at design time, arbitration, and various kinds of communication features (SPLIT, interrupt, multicast, etc) are treated as run-time platform services. Multicast features of communication protocols are not a recent development. This comes as a performance improvement to repetitious transactions containing the same data. Lately, multicast procedures have been

analyzed in the context of on-chip multi processor architectures. Similarly, interrupt communications, scheduling, task allocation mechanism and other services are discussed.

4.1 Existing Communication Services for MPSoC Architectures

Different MPSoC design models have already been proposed [46, 47, 57, 59, 68]. We approach the problem by considering the communication cost and throughput, which are dependent on each other and make the design process automated. By introducing different communication services one-by-one or in a group, communication cost can be reduced. Then, a service or the group of services with minimum communication cost and overhead is selected for final implementation.

A bus system and its variants do not scale well with the system size in bandwidth and clocking frequency [54]. However a bus platform is very efficient when considering broadcasting, since all clients are directly connected to it. A unicast transaction is in fact broadcasted to all clients in the bus segment, but read only by the destination device.

Bus snarfing is proposed as well for performance improvement of multiprocessor systems [49, 50]. Broadcasting technique can be used to reduce memory latency for bus-based multiprocessor systems [45]. In the case of the *SegBus* platform, segment level or selected number of segments level broadcasting is possible without using the whole platform.

Tranberg-Hansen et al. [67] present a unified model for application and platform. Authors present the service model in an abstract model of a hardware component implementing the behavior of the component by offering the services. For this purpose, the service model has been proposed. We complement here by introducing the services block in our design methodology instead of having a service model. The authors also pointed out that Artemis [47] and the subproject Sesame [70] present the application and architecture model separately. We approach similarly the problem, by separating the computation and communication models.

Cornelius et al. [71] introduce the service oriented approach for NoC based communications. A useful comparison of centralized and distributed service oriented architectures is presented. In case of centralized approach, extra communication from all the nodes will be needed to coordinate with Central Coordination Node (CCN). In this case, CCN will be overloaded but it simplifies the monitoring of services. On other hand, the distributed nature of the system control promises to circumvent the hot spot around a single resource (like the CCN). Similar approaches can be adopted for any MPSoC platform but here, we adopt a hybrid approach. In the *SegBus*

platform, a hierarchical solution is used, as **CA** works only if some service needs cross border communication. **CA**deals with multicast service only when a multicasted packet is to be delivered across the border.

Zhang et al. explain the use of snoopy protocol for bus-based MPSoCs in [69]. It makes use of the broadcasting and the serialization properties of buses, resulting in an economical solution.

Faizal et al. [60] presents the architecture of a multicast parallel pipeline router for NoC. The routing engine computes the direction from each header flit and writes it in the register of the routing table. After that, according to direction register entries, payload flits are broadcasted in multiple directions. In our case, just one or two (for both directions) packet copies are generated. This provides power efficiency and simplicity of the implementation, compared to [60]. [52] propose the dynamic multicast routing protocol for traffic distribution in NoC. In this approach, packet prioritization service is not considered, which is a rising requirement for most of the upcoming applications. One may also infer a power performance overhead at the source node, as all the destination addresses need to be sorted. There is no need of sorting and prioritization in our approach, where also prioritization is considered during the placement of processing elements.

4.2 Development of Communication Services

The design methodology for *SegBus* platform has already been proposed by [62]. In this methodology, optimization of communication cost is considered according to the application and platform model. Communication cost can be further optimized by introducing different communication services like multicast, cache coherence or proper scheduling approach according to the application requirements. Figure 4.2. shows the updated design methodology. We approach the problem by introducing the services block in the existing methodology. Services block offers different kind of services with a variety of architectures. The services are introduced one by one and performance improvement regarding communication cost is observed by simulation. The group of services, which provides the best performance can be selected for final implementation. The criteria to select some service for final implementation is discussed individually for each service in following subsections.

4.2.1 Scheduling

Scheduling is the basic and mandatory service to use the interconnection platform. Scheduling can be divided into two steps: task allocation and arbitration as shown in Figure 4.3.

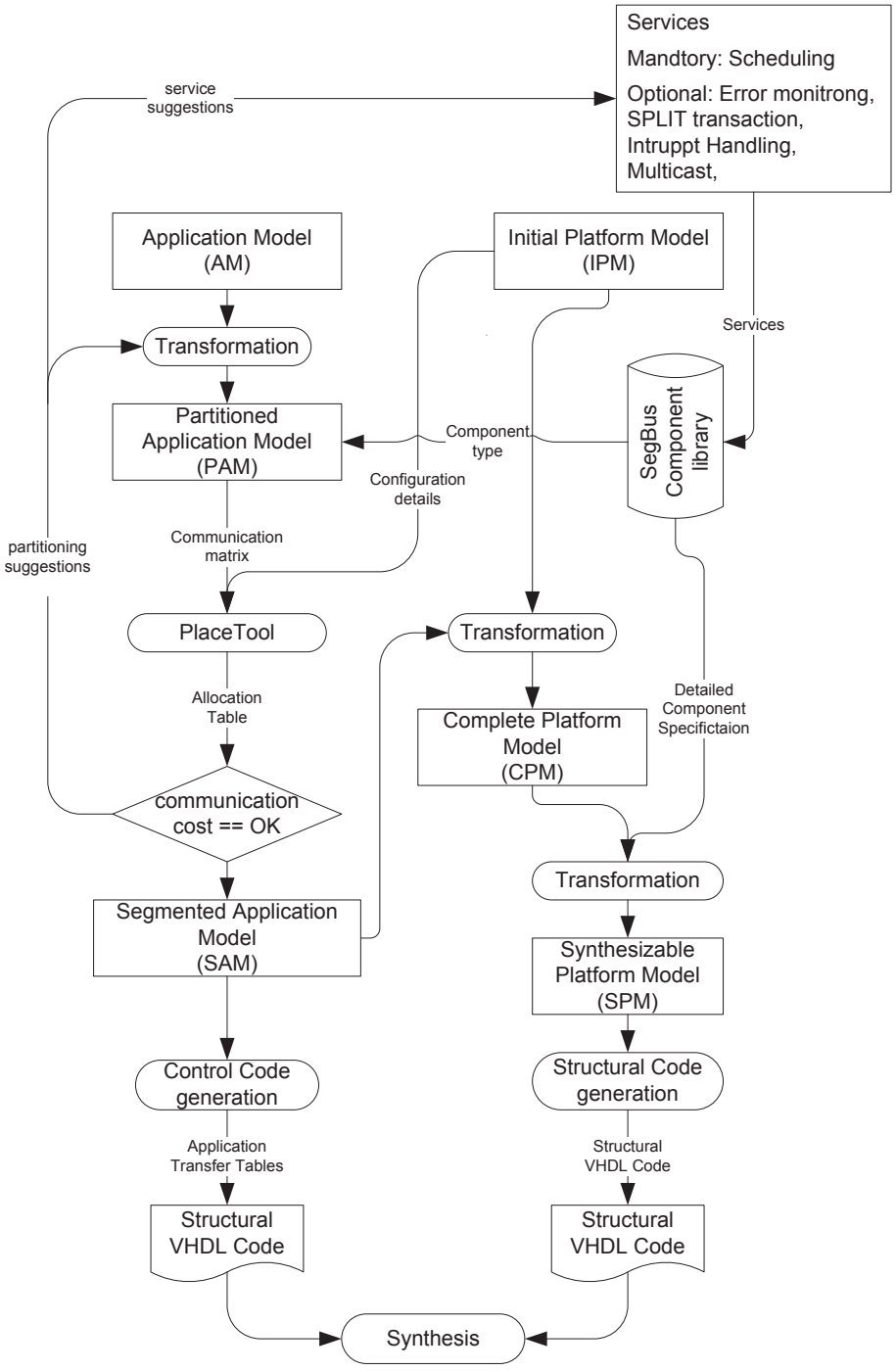


Figure 4.2: Segbus Design Methodology.

Communication features of the running application are needed for the proper placement of IPs. Here, we are interested in the transaction frequency between processing units, their relative sequencing and scheduling. System performance will depend on the utilization of throughput and the balanced traffic load. With all these considerations, the *PlaceTool* [64] has been developed, to deliver the allocation cost for various scenarios.

For *SegBus*, the *PlaceTool* works as the task allocator. The communication matrix is extracted from PSDF diagram and fed to the *PlaceTool* to approximate the effect of segmentation on the performance of the application (tasks) / platform mapping as discussed in Section 3.2.3. After having the placement of tasks and processes, next step is the scheduling on bus, which is controlled by arbitration. For *SegBus* platform, arbitration mechanism can be further divided into three steps as depicted in Figure 4.3.

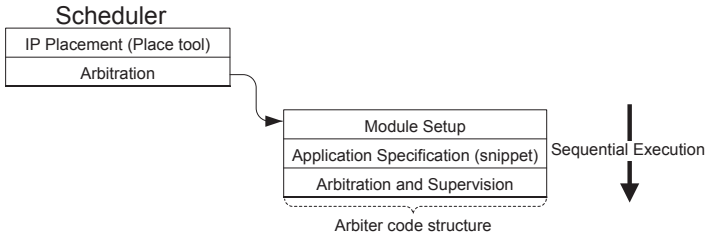


Figure 4.3: Segbus scheduler structure.

For an example, consider the arbitrary task graph shown in Figure 4.4. t_A represents the processing time for task A and similarly, the processing time for other tasks is mentioned. The values on the edges represent the number of transaction packets between two nodes, connected by the corresponding edge. *PlaceTool* allocates the tasks A, F, G, H, I on segment '0' and the tasks B, C, D, E on segment '1'. The scheduling on a single bus and also for the segmented bus with two segments is presented in Figures 4.5 and 4.6 respectively. The context switching time in scheduling is considered zero. The total execution time is reduced by 21% but this value is application dependent. The detailed description of *PlaceTool* and arbitration mechanism is presented in following subsections.

Arbitration

The **SA**s and the **CA** are VHDL defined modules, with a similar structure. The code runs with multiple parameters as required by the platform specification. We see the application as a set of correlated transactions that must be ordered in their execution by the arbiters. The specification of the schedule - as supplied by the PSDF representation, is provided by a snippet introduced in the **SA** or the **CA** codes, representing the projection of the application flow at the respective level and location [65].

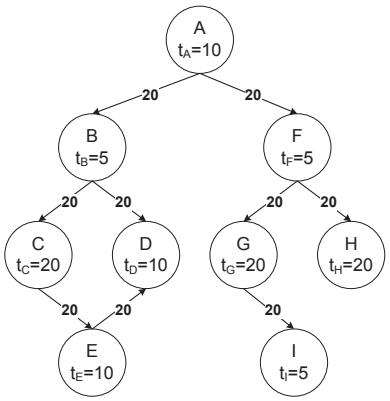


Figure 4.4: Task graph.

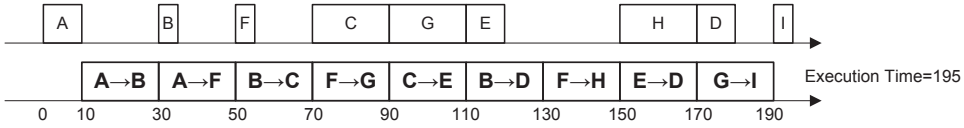


Figure 4.5: Single bus scheduling.

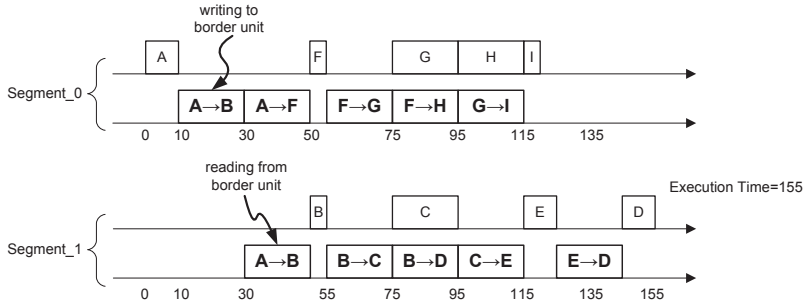


Figure 4.6: Two segment *SegBus* scheduling.

The structure of the arbiters is depicted in Figure 4.3. The "Module SetUP" and the "Arbitration & Supervision" blocks are concerned with application-independent procedures, such as reading the input signals, selecting the granted master, counting the number of transactions performed in a granted activity, etc. The middle block, "Arbitration specification", brings in the application specific requirements for scheduling grant decisions.

The application snippet is part of the actual arbiter VHDL code, and, as such, will be executed. The addressed variables will be read or written by the other arbitration code blocks.

Enhancements in arbitration programmes. The arbitration programme structure has already been discussed in Chapter 3. Here, we discuss only the

extensions to the existing programme structure fields, which will be used for enhancement of communication services. The modified programme line structure can be observed in Figure 4.7.
• *dest.* Identifies the target slaves. The number of maximum targets for one transmission is a parameter of the arbiter module (*max_ dest*). If one of the *dest* sub-fields equals the ID of the *source*, the content is ignored.

• *dest_ seg.* Identifies the target slave's segments. The number of maximum targets for one transmission is a parameter of the arbiter module (*max_ segs*). This in compliance with the allocation results and the *dest* field content. The sub-fields ignored for the *dest* specification will also be ignored here.

• *enables.* Whenever a line is marked *executed*, the **SA** will *enable* the line specified by this field, by subtracting 1 from it's current *guard* value. In order to become enabled, a line with an initial *guard* > 1 will require that several previous operations (execution lines) to have finished. If, for a given line, *enables* $=$ *nrLines*, then the arbiter does not try to enable any other line, when the current one is marked *executed*. One line may enable multiple downstream lines. The number of maximum enable targets for one line is a parameter of the arbiter module (*max_ enable*). If one of the sub-fields equals the current line number, the information is ignored by the arbiter.

Example:

PC	Guard	Source	Destination			Dest_Seg	toGrant	count	enables		
0	0	0	1	X	X	0	0	560	2	X	X
1	1	0	2	3	4	0	1	280	3	4	5
---	---	---	---			---	---	---	---		
5	1	4	10	5	X	1	4	420	7	11	X
---	---	---	---			---	---	---	---		

Figure 4.7: Program line example, with parameters: *max_ dest*=3, *max_ segs*=3, *max_ enable*=4.

The VHDL code corresponding to the table in Fig 4.7 is:

```
-- SA segment 0 snippet
program(0) <= (guard => 0, source => 0,
    dest1 => 1, dest2 => 0, dest3 => 0,
    dest_seg => 0, count => 560,
    enables1 => 1, enables2 => 0, enables3 => 0);
program(1) <= (guard => 1, source => 0,
    dest1 => 2, dest2 => 3, dest3 => 4,
    dest_seg => 0, count => 280,
    enables1 => 3, enables2 => 4, enables3 => 5);
...
```

4.2.2 SPLIT Transactions

Either the transfer is local or across the border, it is possible that the slave is not able to to receive the data or cannot respond the master with required data immediately. In this situation, master will hold the bus and other masters cannot utilize the bus, even when it is free. This reduces the degree of bus utilization. A *SPLIT transfer* improves the overall bus utilization by splitting the operation of the master providing the address to a slave from the operation of the slave responding with the appropriate data [44]. Thus by using the *SPLIT transfers* idle bus cycles can be used for other transactions.

The basic criteria to have SPLIT transaction service depends on bandwidth requirements of application and the packet size. If packet size is very small, SPLIT transaction overhead to arrange the SPLIT mechanism will not be economical and even might reduce the utilization. According to the design methodology discussed in section 4.2, SPLIT mechanism can be introduced and improvement in communication cost can decide to either use or do not use the SPLIT service.

The *SegBus* communication situations can be divided mainly into two categories: Local situation and Cross border situation. Cross border situation can be further divided into three situations [63]: Local-External, External-External and External-Local. In the following sections, we detail the *SPLIT transaction* approach for mentioned communication situations.

Local SPLIT Transactions

In this situation, both the communicating modules are placed in same bus segment. The initiating master requests the bus ownership by raising the *req* signal to the corresponding local **SA**. There are two reasons, when bus ownership cannot be granted to the requesting master for local transaction. First, if another transaction is under completion on the bus. Second, the target slave is busy and cannot respond the request. If the bus is busy, **SA** assigns the bus ownership to the requesting master later at some time. If the target slave is not able to serve the request, **SA** assigns the bus ownership to another master until the slave is ready to serve, which is exactly the same mechanism as employed in AMBA busses [44].

Cross border SPLIT transactions

SPLIT transaction for cross border communication is the enhancement of existing SPLIT mechanism. Consider the situation that a master module from segment0 requests to send the data packet to a slave located in segment2. In this transaction, segment1 will be used as well because it is on the way of transaction. It will not be an economical option to allocate all the bus segments for this transaction for the time span of generating the packet at

source and delivering it to the destination. To enhance the bus utilization in this scenario, we split the transaction into the number of steps equal to the number of segments including the source and destination segment on its way. Neighboring **BU**s are considered as local modules for the corresponding **SA**. Thus the mechanism for local SPLIT transactions will be followed for each segment traversal in cross border SPLIT transactions.

The bus request come along with target slave address. **SA** checks, if the request can be served in current segment or not. If the request cannot be served in current segment (inter-segment request), request is forwarded to the **CA** with target address. In the meanwhile, no other external request is served by the **SA**, until the pending operation is completed. Here, the bus can stay idle for a number of cycles. To deal with this situation, interrupt communication service was introduced as discussed in section 4.2.3. In this section, we address another issue of bus utilization.

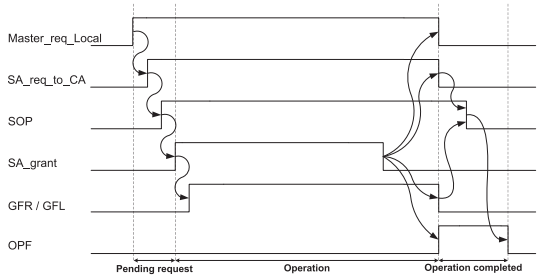


Figure 4.8: Inter Segment transfer control.

Whenever the **CA** is able to serve the request, it informs the **SA**s, from initiator to target of the imminent transfer (signal SOP - operate). As soon as the current operation finishes in the initiator segment, that **SA** grants the requesting master to access either the left, or the right **BU**. In a circular set-up, the **CA** selects the shortest possible distance from the initiator segment to the final one.

Upon filling up the FIFO, the **BU** informs further the next segment that data is waiting to be transferred. The corresponding **SA** allows for the current operation to end, after which it will grant the transfer from one segment border to the other (by setting the granting lines (GFL or GFR, respectively). Hence, the packet waits in the FIFO the period of time required to end the current local transfer in the next segment. When this operation completed, the **CA** receives the OPF (operation finished) signal from the corresponding **SA**, and answers by lowering the respective SOP line. When OPF is also reset, the segment is ready for a new inter-segment transfer. The whole mechanism is shown in Figure 4.8. In a cascaded manner, the above scenario repeats all the way to the target segment (as also illustrated in Figure 2.3).

4.2.3 Interrupt Communication

As mentioned in section 4.2.2, a packet may have to wait in **BU** for number of clock cycles during the cross border transactions. By using an *interrupt service*, the delay in **BU** can be significantly reduced [61]: an inter-segment transfer, when reaching one of the **BU**s on the road from source to destination, will preempt the local activities of the next segment to be crossed.

The local **SA** is the controller that supervises any activity within the segment. The moment of interruption, with respect to the completion of the running local transfer, while the data packet is waiting in the intermediate **BU** FIFO, is of prime importance with highest criticality value. Hence, the decision to interrupt, or continue the current local activity will fall into the attributions of the **SA**. The interrupt transaction will be non-preemptive from start to the completion of execution.

In every clock cycle during the execution of a local activity, the corresponding **SA** monitors if an external request for inter-segment data transfer is raised. When such request is detected, the local grant is put down in the subsequent clock cycles. The whole process from detection of interrupt to the resetting of local grant takes four clock cycles. The ID of the master that has just been interrupted is saved by the **SA** and it will be granted again access, immediately when the inter-segment transaction completes. The respective master then continues to send the information remaining from the interrupted operation.

To illustrate the interrupt mechanism, consider the task graph shown in Figure 4.4. Suppose that it is the graph of a streaming application like audio/video codec and all these tasks execute repeatedly. As shown in Figure 4.9, after completing the local transaction G→I on segment 0, next transaction will be again cross the border (A→B). After filling the **BU**, an interrupt will be generated by border control unit to the **SA** of segment 1, which will preempt the current transaction and will read the buffered data from **BU** for transaction A→B. The context switching time is supposed to be zero for to make the explanation simple. After reading the complete data packet from **BU**, the interrupted transaction (E→D) will be resumed. Now consider the situation that there is no interrupt service available. In that situation, not only the transaction A→B will be delayed but the rest of the processing and transactions on segment 1 will be delayed as well and this delay value will go on increasing because of previous delay. Thus after few application cycles, segment 1 will be lagging too much behind segment 0. In this way, interrupt communication enables the pipelining of tasks on *SegBus*.

The selection criteria of interrupt service to use for final implementation is data dependency and urgency. It depends on the application the how urgent, the data packet stored in **BU** is needed by the destination node. Another issue is data dependency: how many nodes will have to wait directly or

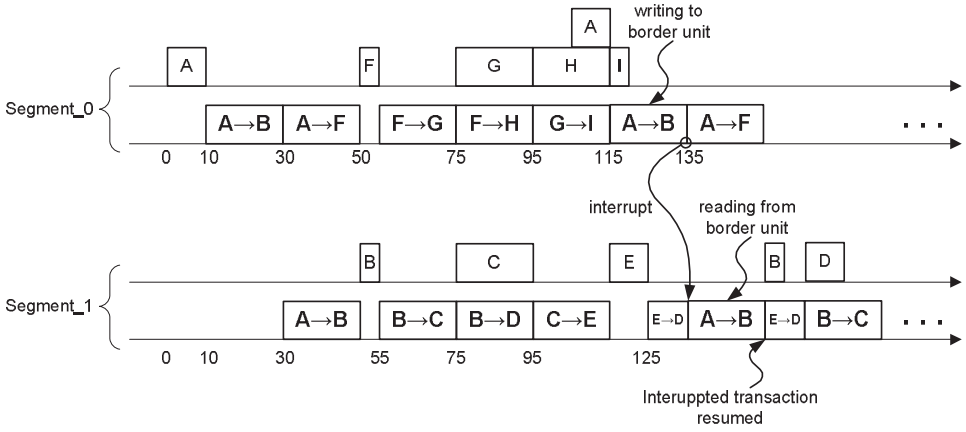


Figure 4.9: Interrupt scheduling.

indirectly due to the delay of the packet in **BU**. In Figure 4.9, the data packet for transaction A→B will not delay only the processing on node B but all of the processing elements on segment1. Thus, improvement in communication cost will be the parameter to favor interrupt service to be used for final implementation.

4.2.4 Multicast Transactions

We introduce here an additional communication feature, namely *multicast transactions*, meant to further improve performance aspects of the platform, in the situations when a single device must send the same data packet to multiple destinations. Such a situation can be observed for the application at hand, in Figure 3.5: $P0$ has to send the same packet no less than three times, to $P2, P3$ and $P4$, respectively.

Without the multicast feature, $P0$ has to execute three requests and send, in some sequence, the data to the necessary destinations. It is natural that a single transaction, if possible, would dramatically reduce thus the communication load, at least in this context.The multicast service will show improvement in communication cost, only if, there is a big fraction of identical data in the application.

In the following sections we illustrate the impact of providing such multicast feature on the activities performed by the local and central arbiters.

Implementation

For multicast transactions, the source device request to the corresponding **SA** is accompanied by the destination IDs without having any information of their relative placement. It is the task of the **SA** to identify the respective

destination segment. A local table is available to all **SA**s, indicating the placement of resources as from the *PlaceTool* selection. **SA** will read the requested slave IDs from the program line to compute the direction and destination segment or segments (for both directions). The table is modeled by the assignment:

```
--(Processing Unit) => (Segment ID)
Segmentation <= (0 =>0, 1 =>0, 2 =>0, 3 =>0, 4 =>1, 5 =>1, 6 =>1,
                7 =>1, 8 =>1, 9 =>2, 10=>1, 11=>1, 12=>1);
```

In case of a broadcast transaction, the requested slave ID (the destination) is a universal code "11...1" (which must not be assigned to any processing element). The width of this requesting code will be equal to the data bus width. Broadcast is a special case of multicast transaction. Multicast is used more often in today's on-chip applications. So, for implementation details, we focus on multicast transactions.

Multicast transaction. Figure 3.5 shows that the third value of the tuple is the same for communication links from source P0 to destinations $P2, P3$ and $P4$. The payload of these packets is the same. In this case, a single packet can be transmitted instead of sending three different copies of identical packets. So, P0 will request the processing element with ID "1111", the broadcast code. **SA** will read *dest* from the respective programme line after receiving the request from P0. *dest* will provide the destination slave IDs 2, 3 and 4. **SA** then obtains the corresponding destination segment IDs from *Segmentation* (0, 0 and 1 respectively).

```
program(0) <= (guard => 0, source => 0, dest => 2,3,4, togrant => 0,
               count => 280, enables => 4,5,6,7);
```

Using the requested segment IDs, only one or two segments will be selected for transmitting packets either in left, right or in both directions. The selection of segments and direction for current transaction is made as illustrated in Figure 4.10.

This example code corresponds to the multicast transaction initiated in segment '0'. After computing the destination segment IDs, the **SA** will decide if the transaction is local or across the border. If the transaction is local, **SA** will check the status of segment and make proper signaling to initiate the transaction. If the transaction is across the border, **SA** will forward the request to **CA** with destination segment IDs. After receiving the signal *InS* from **CA**, **SA** will allow the requester to start the transaction.

Then packets are transmitted according the new packet format, shown in Figure 4.11. The packet header contains the operational code with destination address because some processing elements may offer more than one operations. Source opCode is the operation done by source element for current packet generation.

```
DestSegmentIDs :=(0, 0, 1);
extremeRight = max(DestinationSegs);
extremeLeft = min(DestinationSegs);
if(extremeRight = CurrentSegment and
  extremeLeft = CurrentSegment) then
  null; --Local Transaction;
else
   if(extremeRight < CurrentSegment) then
     Dir <= left;
     SegToCA <= extremeLeft;
   else
     if(extremeLeft > CurrentSegment) then
       Dir <= right;
       SegToCA <= extremeRight;
     else
       Dir <= both;
       SegToCA <= (extremeLeft, extremeRight);
     end if;
   end if;
end if;
```

Where:

- **DestSegmentIDs** is ...
- **extremeRight** is the ID of the rightmost segment
- **extremeLeft** is the ID of the leftmost segment
- **DestinationSegs** is...
- **CurrentSegment** is...
- **Dir** is the direction of the transaction to be granted
- **SegToCA** is the targeted transaction destination

Figure 4.10: Packet read mechanism.

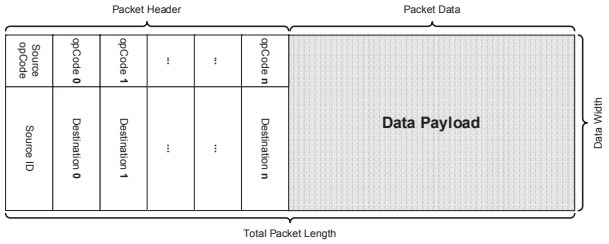


Figure 4.11: Packet Format

Once the communication link is established and the packet is injected into the platform, the slave will read the packet according to the mechanism shown in Figure 4.12. When the packet header arrives and *granted* is inserted by **SA**, all slaves start snooping the data bus. If destination ID is matched in packet header to the current slave, *Slave_ Acq* is raised high. Slaves snoops the bus, even after this event because more than one operational code may be assigned to one slave for a single packet. *Slave_ Acq* enables *Slave_ Data_ Read* at the end of packet header.

In NoC, packet prioritization requires extra processing for broadcast or multicast communication. For the *SegBus* platform, no prioritization is required because processing elements are placed with the consideration of communication requirements and packets are transmitted towards the extreme destination segments. By inspecting the bus lines, each destination will receive the requested traffic. In the same way, a very economic cache coherence snoopy protocol can be implemented.

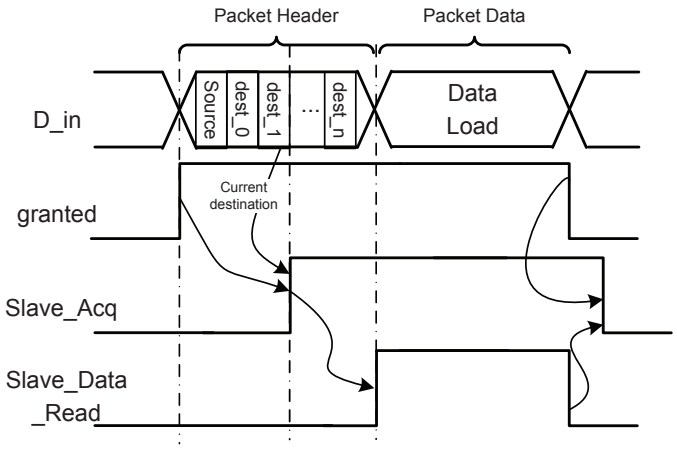


Figure 4.12: Packet read mechanism.

4.3 Experimental Results

To demonstrate the feasibility of proposed design methodology, the multicast service was implemented. As a running example, the H.264 video encoder application was mapped to *SegBus*platform with three segments. Results of around 24% reduction in traffic load show a significant reduction in latency and communication overhead. This comes in comparison to the original, "not multicast enabled" *SegBus* platform. These results are application dependent because multicast service will show significant improvement when a big fraction of identical traffic is injected to the platform. The best performance is achieved when all multicast destinations are placed in one direction with reference to source processing element. However, this is a difficult arrangement to reach.

The interrupt approach brings further improvements to the platform performance. The communication load with and without multicast service for each platform component is shown in Table 4.1. It can be observed that the individual segment load values (S0, S1) show significant reduction in the presence of the multicast service. But there is no significant improvement on border unit loads (BU0, BU1), due to an efficient placement by the tool - as indicated, the *Place Tool* already tries to minimize the cross border transactions. Reduction in the total system power is proportional to the reduction in traffic load. However during broadcast, all destination elements read the bus, thus increasing the capacitive load on the bus. Still, the reduction in power due to the reduction in traffic load dominates the power consumption overhead due to capacitive load. Power consumption results for *SegBus* platform with and without multicast service are presented in Table 4.2. The results have been extracted by Altera *PowerPlay* tool [42].

Table 4.1: Communication cost with and without multicast service for H.264 application with three segments.

SegBus Element	S0	S1	S2	BU0	BU1
Transactions without multicast	1746	2333	48	426	48
Transactions with multicast	1183	1844	48	423	48
Reduction in muber of transactions	32%	21%	0%	0.7%	0%

Table 4.2: Platform power consumption with and without Multicast service.

SegBus Platform	without multicast service	with multicast service	Improvement
Static Power Consumption	755.64 mW	755.72 mW	-0.01%
Dynamic Power Consumption	137.15 mW	134.20 mW	2.15%

4.4 Chapter Summary

We have illustrated here a new perspective on the *SegBus* platform, based on services. The design methodology for *SegBus* platform including the communication services is presented. Each service is considered individually, but relations between them can be also noted. The designer can select the services according to the requirements. We have introduced a solution that provides multicast services on the *SegBus* platform, and shown the impact on arbitration and scheduling, down to the VHDL code. Our intuition about the performance enhancement was proved correct by the implementation results as exercised on the H.264 encoder application, where a further improvement in performance has been observed. The broadcasting feature is implemented with a minimal overhead in terms of arbitration computation, and in terms of data packet size. While the former impact is overcome by the parallel activities of the arbiter and of the functional modules, the latter can be seen as a small price for a possibly very large improvement in performance, when multiple destinations are required.

Chapter 5

Improving Resource Utilization in NoCs

By maximizing the utilization of available resources, network performance can be improved with a minimum overhead [90]. To address the utilization, first we need to consider the packet traversal mechanism in the network. For buffered NoC architectures, the packet traversal mechanism is shown in Figure 5.1. When a packet is injected from a source processing element (PE) to the network for delivery to the destination PE, the packet is forwarded hop by hop on the network according to the decision made by each router on the way. For any buffered NoC router, each packet is stored in the buffer upon arrival. In the next step, control logic reads the header flit of the packet to make the routing and channel arbitration decisions. In the last phase, the packet is traversed through the crossbar and delivered to the next router via an inter-router link. This process is repeated until the packet is delivered to the destination core. Each packet injected to the network passes through three resources repeatedly before it is delivered to the destination core: Input buffer, Inter-router link and Crossbar as shown in Figure 5.1. Different techniques have been proposed for performance enhancement and utilization of these resources, like VC based NoC architecture [109], which addresses the channel utilization, while [73] addresses the crossbar performance enhancement by decomposing a larger 5×5 crossbar into two smaller 2×2 crossbars.

The communication frequency between the cores is decided by the running application and the location of these cores is decided by an application mapping technique. During the execution of the application, a routing algorithm decides the route of data packets between the source and destination nodes. This whole mechanism decides the utilization of NoC resources for the running application at a system level. However, resource utilization techniques, like the time division multiplexing of packets on inter-router channels

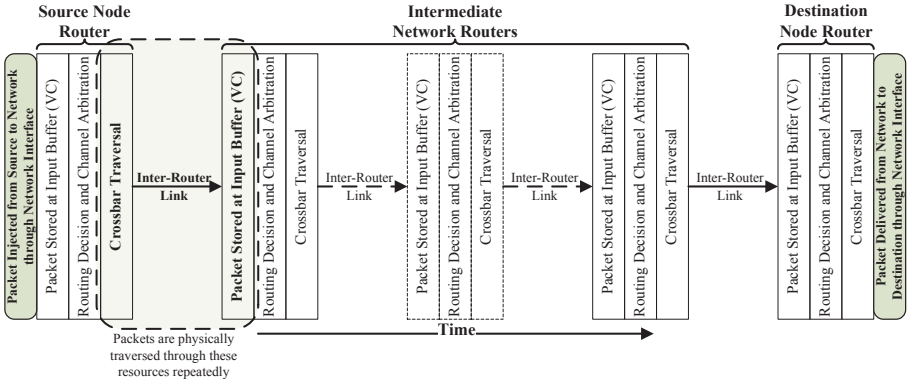


Figure 5.1: Packet Traversal Mechanism for Buffered NoC Architectures.

for channel utilization, is done at a micro-architecture level. Thus, the NoC resource utilization can be addressed at different abstraction levels. In this chapter, the resource utilization is addressed by application mapping at a system level and then input buffer utilization at a micro-architecture level.

5.1 Existing Resource Utilization Techniques

As already mentioned, this chapter addresses the resource management to improve performance in two phases: application mapping and buffer management. The discussion on existing techniques focuses on both approaches separately.

5.1.1 Application Mapping Techniques

The number of task allocation and scheduling techniques for the communication and resource management of MPSoC systems with homogenous and heterogenous cores have already been proposed. An application specific algorithm for task allocation and IP core placement is not a recently addressed topic but one which still needs attention in order to deal with the upcoming requirements of modern applications.

Srinivasan et al. [81] and Hu et al. [82] presents a heuristic approach for the bandwidth constrained mapping of cores onto NoC architecture. In their approach, initially, the core with maximum communication demand is placed onto one of the mesh nodes with a maximum number of neighbors. Then, for each core yet to be mapped, the core communicating more with the already mapped cores is selected and placed onto the node, this minimizes the communication cost with the mapped cores. After mapping all the cores, iterative pair-wise swapping is used to improve the mapping till the system delivers the best performance. The issue with these approaches is that they

do not consider the traffic distribution of each core. A core may have high communication requirements with many nodes but a large fraction of the traffic might be only for one core. In this situation, this core should not be the first to be mapped.

Walter et al. [84] propose to divide the cores into a number of classes according to the core functionality. Initially, the random mapping is generated and then, repeatedly, cores are swapped to reduce the application execution time unless further improvement is not obtained for a predefined number of iterations. The problem with this approach is that, because of random nature, the optimal solution cannot be consistently achieved.

The approach we illustrate here does not impose restrictions towards other MPSOC platforms. In our approach, we propose an algorithm for placement of IPs according to the application requirements. As a running example, two multimedia applications, a video conference encoder (VCE) and an MPEG-4 video encoder, are mapped on a 2D-mesh NoC platform.

5.1.2 Buffer Management Techniques for Bandwidth Utilization

Maximizing the buffer utilization in Network-on-Chip (NoC) has been the subject of prior research. Buffer utilization can be enhanced by sharing buffers among ports. Router architectures with buffer sharing among all the input ports can deliver high throughput, but this comes at the expense of area and power consumption. Thus, a thoughtful tradeoff among performance, power consumption, and area should be advised by an efficient design approach. Lan et al. [103] address buffer utilization by making the channels bidirectional, which shows significant improvement in system performance. In this case, each channel controller has two additional tasks: dynamically configuring the channel direction and allocating the channel to one of the routers. These additional tasks make the controller circuit complex. There is a 40% area overhead over the typical NoC router architecture due to double crossbar design and control logic which also results in an additional power consumption overhead.

In order to improve buffer utilization, a Dynamic *Virtual Channel Regulator* (ViChaR) for NoC routers has been proposed by Nicopoulos et al. [114]. To this end, the authors use a unified buffered structure (UBS) instead of individual, statically partitioned FIFO buffers. A UBS provides each router port with a variable number of VCs, depending on the traffic load. The architecture achieves around 25% improvement in system performance at a small cost of power consumption. However, the architecture only enhances buffer utilization when a port is under heavy traffic load. In the case of no traffic load, the free buffer resources cannot be used by neighboring overloaded ports in the router.

A distributed shared buffer (DSB) NoC router architecture has been presented by Ramanujam *et al.* [118]. Because of the extra crossbar and complex arbitration scheme imposed by this architecture, it shows a significant improvement in throughput at the expense of area and power consumption.

We address the mentioned problems by proposing a NoC architecture which offers low communication latency through utilization of VC-buffers with minimal overheads. The proposed architecture does not impose restrictions on network level system design.

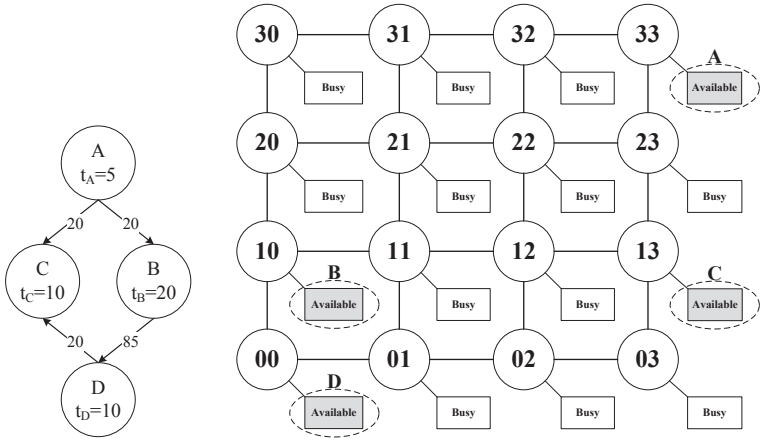
5.2 Application Mapping for Minimal Routing

In this section, we present the application mapping technique onto a mesh NoC architecture for minimal routing algorithms. Average Packet Latency (APL) and power consumption are the key evaluation parameters for applying application mapping techniques onto NoC architectures [88]. Ideally, both of these performance indexes should be optimized. We approach the problem by optimizing the communication cost which has direct impact on APL and power consumption. The value of the communication cost is the total number of hops for each packet multiplied by the number of packets for one application cycle. Thus, we need to optimize the number of hops for each packet. The minimum possible value of the communication cost is equal to the number of packets injected into the network (average hop count = 1). Due to the limitations of interconnection platforms, like a node in a 2D-mesh NoC not being able to communicate directly with more than four nodes, the value of average hop count is always more than '1'.

The basic approach for any mapping technique is to map the cores on neighboring nodes, which communicate with each other. Due to platform limitations and unavailability of cores at the time of mapping, it is not an easy task. Mapping is possible when number of cores to be mapped are less than or equal to the number of available cores on the platform. Otherwise, application partitioning should be redefined and the requirement for the number of cores should be reduced.

For example, consider the application graph shown in Figure 5.2(a), which needs to be mapped on a 4x4 2D-mesh NoC of homogeneous cores. Now suppose that other applications are already running on the system and four cores are available to serve the new application. Thus the mapping can be defined. The available cores are marked and shown in Figure 5.2(b). The available cores are far from each other except two neighboring cores. In the example graph, cores 'B' and 'D' require more bandwidth to communicate with each other as shown by the weighted edges in Figure 5.2(a). Therefore, cores 'B' and 'D' will be mapped first as neighboring nodes at '10' and '00', respectively. Cores 'A' and 'C' are mapped after the mapping of 'B' and

'D' in the two other available cores with a minimum communication cost. IPs can be prioritized to first map on the basis of communication load and the number of neighboring nodes that communicate directly with that node, such as node 'B' communicating directly with node 'D'.



(a) Example application graph

(b) Mapping on 2D-mesh NoC

Figure 5.2: Example application mapping with limited availability of cores.

Here we use the VCE application to test the proposed mapping technique. For VCE, the PSDF diagram is shown in Figure 5.3. The PSDF diagram has already been explained in section 3.2.1. There are two values on each graph edge. The first value represents the number of successive same-size transactions during one application cycle and the second value represents the relative ordering among the data flows in the given application. The application is a group of sub-applications, an H.264 video encoder, MP3 audio encoder and OFDM transmitter.

The mapping technique can be divided into two steps: Prioritization and Placement. The prioritization of IP cores is platform independent while the placement phase is platform and topology dependent.

5.2.1 Prioritization of IP Cores

After having the inter-core communication data for the given application, next step is to prioritize the IP cores for placement. The prioritization of IP cores is based on following parameters:

Total number of packets to be communicated to, or by, the core(N_{Pi}). The total number of packets transmitted and received by core 'i' can be computed by the expression: $N_{Pi} = \sum_j (C_{i,j} + C_{j,i})$. Where $C_{i,j}$ represents the

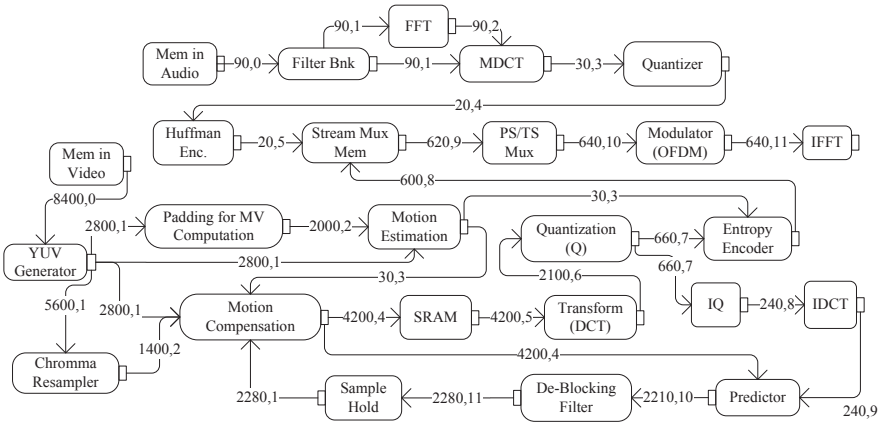


Figure 5.3: Video conference encoder (VCE) application.

number of packets generated by node i for node j. Similarly, $C_{j,i}$ represents the number of packets transmitted by node j for node i.

Number of neighboring cores to be communicated (N_i).
The number of cores which communicate with core 'i' can be computed by the expression: $N_i = \sum_j (nnz(C_{i,j}) + nnz(C_{j,i}))$. Where nnz is a function to count the number of communication edges with non-zero load. To count the number of neighbors for a core, a communication edge is considered between each pair of cores. The edges with non-zero values represent the communication edge between the two cores in application graph.

Traffic distribution (σ^2_{xi}).
Traffic distribution is the statistical load variance for a core. Suppose an IP communicates with five different cores, but 95% of its traffic is for a single IP core. With the rest of the cores, it communicates for very few packets. Thus, only one core is strongly connected and should be mapped on the neighboring node of the current node. The remaining four cores can be mapped on nodes far from the current node, if enough nodes are not available in the neighborhood. Thus, the current node does not require a central node for mapping even if it communicates with many cores. To deal with such situations, statistical variance in communication can be used to decide whether the communication load is distributed for an IP core among it neighbors or not. Statistical variance can be computed by Eq.5.1.

$$\sigma^2_{xi} = \frac{1}{N_i} \sum_{n=1}^{N_i} (x_{ni} - \overline{X}_{Ni})^2 \tag{5.1}$$

Where, $\overline{X}_{Ni}$ = Average link load for IP_i.
$= N_{Pi}$ / N_i
x_{ni} = Number of packets to be communicated between IP_n and IP_i.

After extracting the communication parameters, as discussed above, the next step is to define the prioritization expression. Priority is directly proportional to N_P and N_i and is inversely proportional to the σ_x^2. Now, we have an expression to compute the priority of the *ith* processing element (P_i) shown in Eq.5.2.

$$P_i = \frac{N_{Pi} * N_i}{\sigma_{xi}^2} \tag{5.2}$$

The pseudocode to compute and sort the priority of 'N' application cores to be mapped on the basis of mentioned parameters is shown in Algorithm 1.

ALGORITHM 1: PRIORITIZATION OF CORES

Input: $N, N_{Pi}, N_i, \sigma_{xi}^2$;
Output: **P**; //Prioritization sequence
$\mathbf{P_{TEMP}}$; //Prioritization sequence before sorting

1: $\mathbf{P_{TEMP}} = 0$;
2: **for** $i = 1$ **to** N **do**
3: $P_{TEMP_i} = N_{Pi} * N_i / \sigma_{xi}^2$;
4: **end for**
5: $\mathbf{P} \leftarrow DESCENDING\_SORT(\mathbf{P_{TEMP}})$;
6: **return P**;

The priority sequence for the application shown in Figure 5.3 is Motion compensation, YUV Generator, Motion estimation, and SRAM (etc.), in descending order.

5.2.2 Placement (Platform Dependent)

After obtaining the prioritized and sorted sequence of IP cores to be mapped, the next step is to map the application cores onto the MPSoC platform. The placement algorithm is platform and topology dependent. The task of the placement algorithm is to optimize the APL value and reduce the power consumption. With a higher value of average hop count, the values of APL and power consumption are increased. The optimization of the average hop count optimizes the power and performance characteristics. The mapping algorithm for a 2D-mesh NoC is shown in Algorithm 2.

In Algorithm 2, *Neighbors* is a function to check which of the mapped cores in communication with the current core 'P' is to be mapped. *Cores*

ALGORITHM 2: APPLICATION MAPPING

Input: **P**; //Prioritization sequence generated by Algorithm 1
Cores_Availability; //Free cores on MPSoC platform
PSDF; //Application PSDF stored as a matrix
Output: *MPSoC_Platform*; //2D Mesh NoC

1: **if** ($|Cores\_Availability| > |PSDF|$ **then**
2: **Mapping can be defined;**
3: **else**
4: **Redefine application partitioning and reduce number of core requirements;**
5: **exit;**
6: **end if**
7: $MPSoC\_Platform(Cores\_Availability$ (Node with maximum neighbors)) $\leftarrow$ **P** (1);
8: **for** $j = 2$ **to** $|PSDF|$ **do**
9: //Following loop is for already mapped cores.
10: **for** $k = 1$ **to** $j - 1$ **do**
11: **if** Neighbors(**P** (j), **P** (k), PSDF) **then**
12: $Neighbor\_Cores$ (j) $\leftarrow$ **P** (k);
13: **end if**
14: **end for**
15: **if** ($|Neighbor\_Cores(j)| \geq 1$) **then**
16: MAP ($\mathbf{P}(j), Neighbor\_Cores(j), Cores\_Availability, MPSoC\_Platform, \mathbf{P}(1 : j-1)$);
17: //**P** (j-1) represents already mapped cores.
18: **else**
19: MAP_AWAY ($\mathbf{P}(j), Cores\_Availability, MPSoC\_Platform$);
20: **end if**
21: **end for**
22: **return** $MPSoC\_Platform$;

Availability keeps a record of the available cores on the platform for mapping and for already mapped IPs. Using *Neighbor Cores*, *MPSoC_ Platform* and *Cores_ Availability*, the current IP is placed by using the function *MAP*. *MAP* optimizes the communication cost for mapping a single core. It has been presented in Algorithm 3. If the next IP core to be mapped does not communicate with already placed IPs, it is placed away from the already placed IP cores using the function *Map_ Away*. By using the proposed mapping algorithms, the overall application mapping has complexity of $O(N^2)$.

The placement of IPs for the VCE application presented in Figure 5.3 is shown in Figure 5.4. The values at the edges represent the number of packets, exchanged over the corresponding link in one application cycle.

5.2.3 Simulation Results

To demonstrate the better power and performance characteristics of the proposed mapping algorithm, a cycle-accurate NoC simulation environment was implemented, running two applications mapped with four different mapping algorithms (Proposed, NMAP [81], Walter et al. [84], and PBB [82]) in HDL. The simulations were performed for 4x4 and 5x5 mesh NoCs with MPEG4 and VCE applications, respectively. The comparison is performed in terms

ALGORITHM 3: MAP: SINGLE CORE MAPPING

Input: $Core\_To\_Be\_Mapped$; //**P**(j) in MAP function call in Algorithm 2
$Neighbor\_Cores$; //$Neighbor\_Cores(j)$ in MAP function call in Algorithm 2
$Cores\_Availability$;//Free cores on MPSoC platform
$MPSoC\_Platform$;//2D Mesh NoC
$Already\_Mapped\_Cores$; //**P** (1 : j-1) in MAP function call in Algorithm 2
Output: $MPSoC\_Platform$; //2D Mesh NoC, updated and returned

1: Map $Core\_To\_Be\_Mapped$ on nearest node of $Already\_Mapped\_Cores(1)$ according to the $Cores\_Availability$ with minimum communication cost;
2: $Mapping\_Node$ = Node position, to which $Core\_To\_Be\_Mapped$ has been mapped.
3: C = Communication Cost;
4: Remove the mapping of $Core\_To\_Be\_Mapped$.
5: **for** $i = 2$ **to** $|Already\_Mapped\_Cores|$ **do**
6:
7: **if** $Core\_To\_Be\_Mapped$ and $Already\_Mapped\_Cores$ (i) are Neighbor **then**
8:
9: Map $Core\_To\_Be\_Mapped$ on nearest node of $Already\_Mapped\_Cores$(i) according to the $Cores\_Availability$ with minimum communication cost;
10: **if** $Current\_Communication\_Cost < C$ **then**
11: C = $Current\_Communication\_Cost$;
12: $Mapping\_Node$ = Node position, to which $Core\_To\_Be\_Mapped$ has been mapped.
13: **end if**
14: Remove the mapping of $Core\_To\_Be\_Mapped$.
15: **end if**
16: **end for**
17: Map $Core\_To\_Be\_Mapped$ on $Mapping\_Node$ of $MPSoC\_Platform$;
18: **return** $MPSoC\_Platform$;

of power consumption and APL. The packet latency was defined as the time duration from when the header flit is created at the source node to when it is delivered to the destination node. For each simulation, the packet latencies were averaged over 50,000 packets. Latencies were not collected for the first 5,000 cycles to allow the network to stabilize. It was assumed that the packets had a fixed length of 66 flits, the buffer size of each synchronous buffer was eight flits and the data width was set to 32 bits. The NoC switches exploit two virtual channels for each input port. To perform the simulations, we used an XY wormhole routing algorithm. A 50 MHz clock frequency is applied to the NoC, resulting in a maximum transmission rate per link, equal to 400 Mbps. To estimate the switch power consumptions, the high level NoC power simulator presented in [80] was used.

The NoC system performance and power consumption for four different mapping algorithms are given in Figure 5.5. It can be observed from this figure that the proposed approach shows more than a 20% reduction in APL compared to the PBB mapping technique. The values for other techniques show similar results. The difference in performance is greater, as the number of cores in the application are increased and can be observed in Figures 5.6 and 5.5. The mapping technique shows more improvement for the 5x5 NoC compared to the application mapped to the 4x4 mesh NoC.

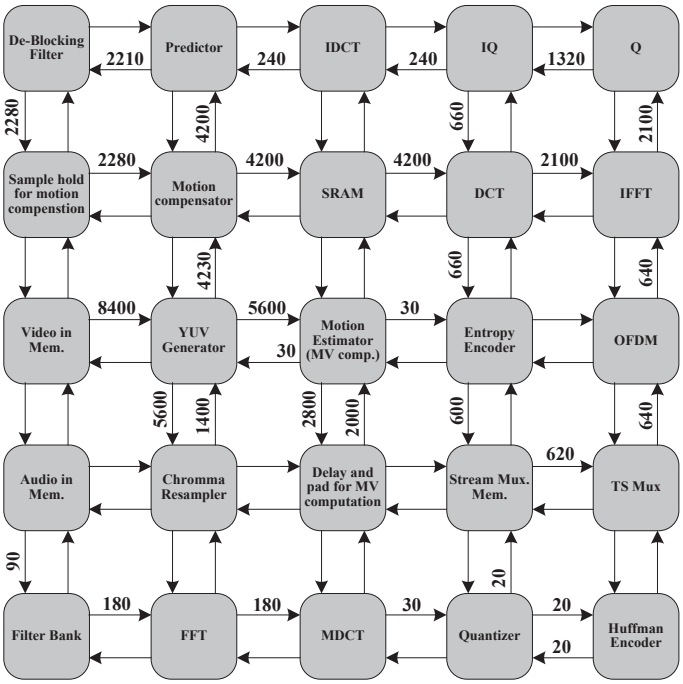


Figure 5.4: Mapping generated for the VCE application.

The power consumptions of the interconnection network, which are based on 35nm standard CMOS technology, are presented in Figure 5.6. The pattern of power comparison is similar to APL, but the reduction factor is different because reducing the number of hops directly affects APL. However, power consumption is not reduced by the same factor because packet generation and static power are the major power sinks, which are independent of the mapping technique.

5.3 Channel Utilization

Efficient resource utilization is necessary to execute a given application with minimized overhead. It can be observed from Figure 5.4 that even after efficient mapping, more than half of the interconnection links are not used at all by the application. The first step towards enhancing the utilization of interconnection resources is examining the purpose of each network resource individually. If multiple applications execute on an MPSoC simultaneously, the traffic pattern is unpredictable and makes it difficult to analyze the utilization of individual resources. The routing algorithm controls the utilization of communication channels. The system then utilizes the router resources according to the load on the incoming channels. We employ chan-

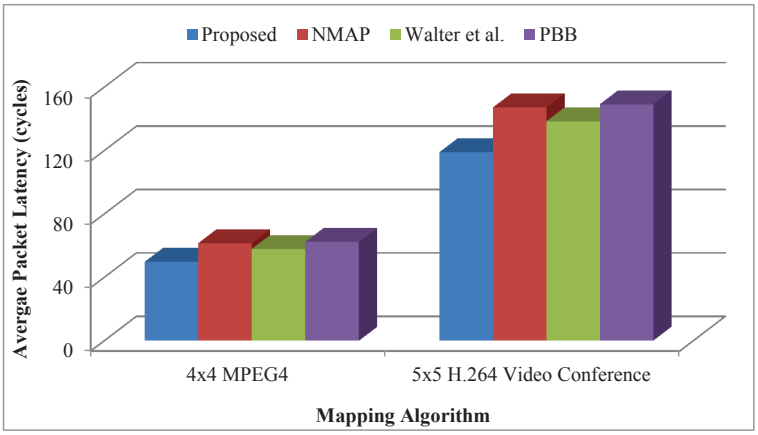


Figure 5.5: Average Packet Latency with XY routing algorithm.

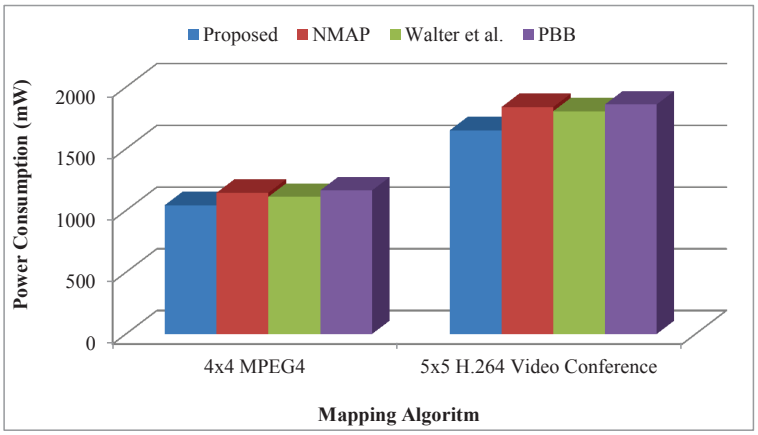


Figure 5.6: Power consumption with XY routing algorithm.

nel load analysis to provide the basis for determining the utilization of these router resources in the following sections for synthetic and application specific benchmarks.

5.3.1 Synthetic Traffic Analysis

In synthetic traffic analysis, the average load for each link is determined for a variety of traffic patterns. In our case, uniform, transpose, bit complement and negative exponential distribution (NED) traffic is analyzed with XY routing. In the uniform traffic pattern, a node sends a packet to any other node with an equal probability while in the transpose traffic pattern, each node (i,j) only communicates with node (j,i). For the bit complement traffic load, each node (i,j) communicates only with node $(M\text{-}1\text{-}i,N\text{-}1\text{-}j)$, if the

mesh size is MxN. The NED is a synthetic traffic model based on Negative Exponential Distribution, in which the likelihood that a node sends a packet to another node exponentially decreases with the hop distance between the two cores. This synthetic traffic profile accurately captures the key statistical behavior of realistic traces of communication among the nodes [117]. Figure 5.4 shows the percentage load for each link on the network for different traffic patterns, measured by eq. 5.3.

$$L_{(i,j)\rightarrow(k,l)} = \frac{\text{TLL: } (i,j) \rightarrow (k,l)}{\text{TNL}} \tag{5.3}$$

where,

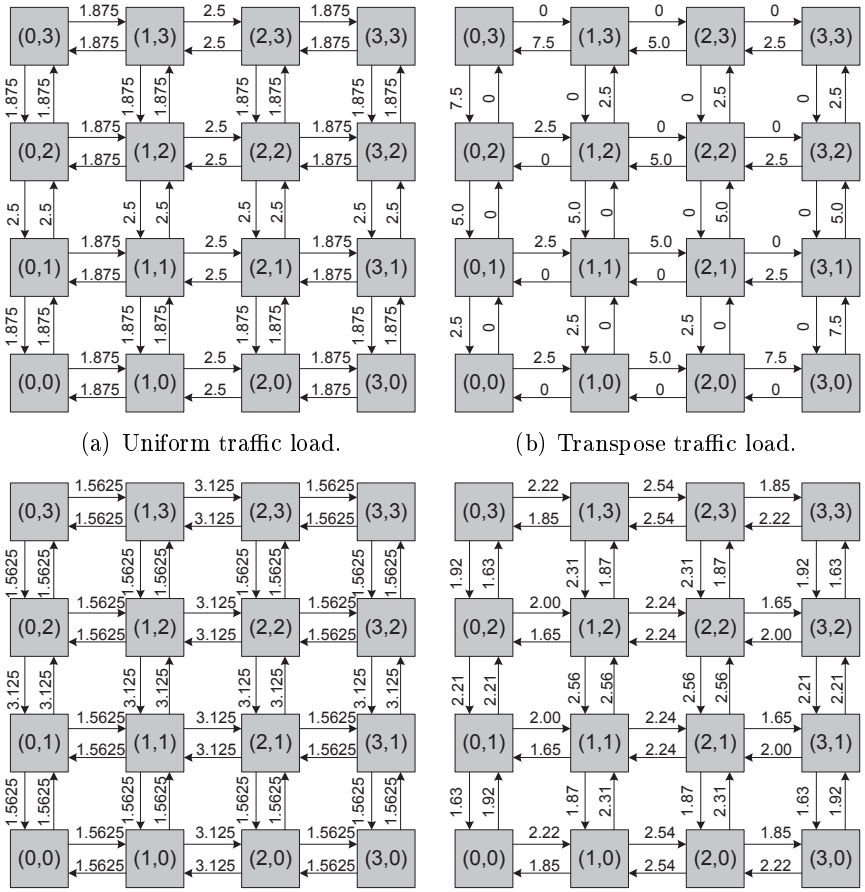
$$\text{TLL: } (i,j) \rightarrow (k,l) = \sum_{\substack{0<x<(M-1)\\0<x'<(M-1)}} \sum_{\substack{0<y<(N-1)\\0<y'<(N-1)}} \begin{cases} (S(x,y), D(x',y') \mid \\ via(i,j)) \text{ Then } via(k,l) \end{cases}$$

and

$$\text{TNL} = \sum_{\substack{0<m<(M-1)\\0<n<(N-1)}} \sum_{\substack{0<o<(M-1)\\0<p<(N-1)}} L_{(m,n)\rightarrow(o,p)}$$

To measure the total link load (TLL) on a specific link directed from node (i, j) towards node (k, l), the traffic load from the source nodes represented by S(x, y) routed via node (i, j) and then via node (k, l) towards the destination nodes, represented by D(x', y'), is considered. The destination node D(x', y') could be the node (k, l) because these packets will contribute to the link load for the link directed from node (i, j) towards node (k, l). For the total network link load (TNL), the link load of all the interconnection links is summed up. The expression is topology independent and can be extended to any number of dimensions.

The normalized link load percentage computed using eq. 5.3 for uniform, transpose, bit complement and NED traffic loads are shown in Figure 5.7(a), 5.7(b), 5.7(c) and 5.7(d) respectively, with an XY routing logic. For example, consider the node '12' in Figure 5.7(b). The input ports to receive data from nodes '11' and '13' are not used at all during the entire simulation, independent of the total simulation time. But, the input ports from left and right receive the traffic load. The traffic load from node '22' is twice that of the load from node '02'. The link from node '22' towards '12' is overloaded but cannot utilize the available resources of other ports. Similar behavior can be observed for odd-even routing.



(a) Uniform traffic load.

(b) Transpose traffic load.

(c) Bit complement traffic load.

(d) NED traffic load

Figure 5.7: Traffic load analysis for XY-routing.

In all of the investigated cases, some input ports were overloaded, compared to other ports. In order to balance the load and to enhance the resource utilization, resources can be shared among over- and underutilized input ports. The threshold for distinguishing between over- and underutilization is selected in such a way that half of the ports have a higher load value and the other half has a lower load value.

The resources could be shared among all the input ports, but this would require large crossbar switches, which increases power consumption, area and switching delay. The other option is sharing the resources among multiple ports (but not among all ports) so that loads are balanced, resource utilization is improved, and throughput is close to an architecture with full VC buffer sharing.

In higher-dimensional NoCs (e.g. 3D NoCs), the partial virtual-channel sharing (PVS) approach benefits from the increased number of router ports, which opens more grouping options as listed below:

$$\begin{aligned}
\text{Typical 2D-mesh:}\mathtt{R}(5) &:< (5), (4,1), (2,2,1), \ldots, (1,1,1,1,1) > \\
\text{Stacked 3D-mesh [127]:}\mathtt{R}(6) &:< (6), (5,1), (2,2,2), \ldots, (1,1,1,1,1,1) > \\
\text{Typical 3D-mesh:}\mathtt{R}(7) &:< (7), (6,1), (3,3,1), \ldots, (1,1,1,1,1,1) >
\end{aligned}$$

Where R(n) represents the router with n ports, <(p, q, ...)... (f, g, h, ...)> represents the set of different grouping options. Each grouping option is denoted as a tuple of group sizes; for example, $(3, 2, 2)$ represents one group of 3 ports and two groups of 2 ports. Ports in the same group can share their resources. In the examples provided for the grouping options, the stacked 3D-mesh NoC is different from a typical 3D mesh NoC in terms of inter-layer interconnects. The stacked 3D-mesh NoC is a NoC-Bus hybrid architecture which requires a 6-port router, since the bus adds a single additional port to the generic 2D 5-port router for inter-layer communications in both directions (up/down) [116]. On the other hand, a typical 3D mesh NoC requires a 7-port router with two extra ports for upward and downward communication, compared to the generic 2D 5-port router.

5.3.2 Application Traffic Analysis

The MPEG4 application introduced in [102] has been selected for resource utilization analysis. The NoC-mapped application and its bandwidth requirements are shown in Figure 5.8. Consider, for example, the link loads of the DR-SDRAM node. If its East and South ports share their resources, the heavy load value of 942 MB/s from the East port can utilize the resources of the South port, which receives a smaller load value of 60.5 MB/s. Thus, sharing communication resources among multiple ports can balance the input load on all ports without increasing the crossbar size too much. In this case, the average load on input ports is comparable to the average load per port which can be achieved by sharing the resources of all four ports.

The ports are grouped so that the load sums of the different groups are balanced. The port with the maximum load should be grouped together with the port with the minimum load. Grouping it with an average load port would not make sense because such a port does not have free resources and does not require extra resources. For load balancing, the selection of ports to share the resources should be made during the design phase, according to Algorithm 4, as described and analyzed below.

The input parameters of the algorithm are the number of router ports, P, the number of VCs per port, V, the vector representing the input bandwidth

requirements for each router port, L, and the number of partitions (groupings) of ports, d. In the algorithm, the average bandwidth requirements per VC are represented by l and the number of ports which share the VC buffers in a group is represented by vector S. Similarly, the total bandwidth requirements for each group of ports sharing the VC buffers are represented by vector W. The output of the grouping algorithm are group combinations represented by C.

The total number of VC buffers in the router is $P \times V$. Similarly, the total number of VC buffers in group i are $S_i \times V$. The value of l can be computed by a summation of the bandwidth requirements of all the router input ports divided by P×V. The input ports in group i are grouped to share the VC buffers in such a way that the total incoming load for the group ($\sum_i \mathbf{L}(\mathbf{C_i})$) approaches the value of $l \times S_i \times V$. The sum of all the S values is equal to P. Each value of W is equal to l multiplied by the corresponding value of S. Different combinations of ports are tested by using a *for* loop such that the sum of the bandwidth requirements of the ports in the combination is close to the corresponding value of W. The whole process is repeated until the best combination is achieved and the difference between W and the sum of communication bandwidth requirements of the ports in the group is minimal. Finally, the grouping combinations, C, are returned.

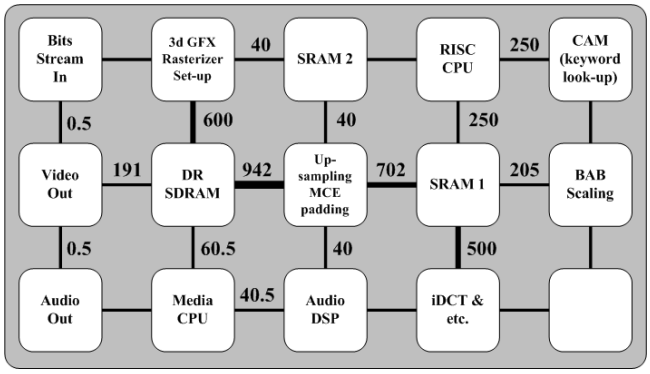


Figure 5.8: MPEG4 application [102].

The proposed algorithm can support any topology including irregular topologies with any number of ports, P. For example, Murali et al. [110] propose an application specific power efficient topology which requires an eleven port router. If Algorithm I is used to generate the grouping combinations according to the input load requirements, further system performance enhancement can be achieved by selecting an optimal combination from the following set of potential groupings:

ALGORITHM 4: Grouping

Input: P: Number of router ports, V: Number of VCs per port, d: Number of partitions,
$\mathbf{L} = [L_1, L_2, \ldots, L_P]$: input bandwidth requirements
Output: $\mathbb{C} = [\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_P}]$;
$//\mathbf{C_i}$ contains the port IDs in segment i
$\mathbf{S} = [S_1, S_2, \ldots, S_d]$;
$//S_i$ is the number of ports sharing the VCs at segment i
l = Average bandwidth requirement per VC
$\mathbf{W} = [W_1, W_2, \ldots, W_d]$;
$//W_i$ is total bandwidth requirement for partition i

1: $L_{ACC} = 0$; //**L** Accumulator
2: **for** $i = 1$ to P **do**
3: $L_{ACC} \leftarrow L_{ACC} + L_i$;
4: **end for**
5: $l \leftarrow L_{ACC}/(P \times V)$;
6: **loop**
7: Random $\mathbf{S} \mid \sum_{i=0}^{d} S_i = P$;
8: $\mathbf{W} \leftarrow l \times \mathbf{S}$;
9: **for** $j = 1$ to d **do**
10: $\mathbf{C_j} \leftarrow$ Combination of ports $\mid \sum_i \mathbf{L}(\mathbf{C_i}) \sim W_j$;
11: **end for**
12: **if** $\forall(l \times sizeOf(\mathbf{C_i})) \sim \sum_i \mathbf{L}(\mathbf{C_i})$ **then**
13: **exit;**
14: **end if**
15: **end loop**
16: **return** $\mathbb{C}$;

$$\mathtt{R}(11) :< (11), (10,1), (6,5), (5,5,1), (4,4,3), (3,3,3,2), \ldots, (1,1,1,1,1,1,1,1,1,1,1) >$$

5.4 Partial Virtual Channel Sharing (PVS) Router Architecture

To address the resource utilization issues discussed in section 5.3, we propose the partial virtual-channel sharing NoC (PVS-NoC) architecture. Due to sharing, the proposed approach enhances VC utilization because free buffers can be utilized by other channels. Maximum VC utilization could be achieved by sharing among all the input ports. However, full sharing increases the control logic complexity and power consumption. Thus, a tradeoff between resource utilization and power consumption is needed.

This tradeoff can be achieved with the PVS approach by forming groups with a limited number of input ports that share resources according to the communication requirements. With this technique, the buffer utilization is

increased and comes close to the utilization level of the fully shared architecture without its significant silicon area and power consumption overhead suffering.

Data is injected into the network in the form of packets produced by the Network Interface (NI). While receiving a packet, the NI de-packetizes it and delivers the payload to the PE. The packet format is shown in Figure 5.9. The header flit carries the operational code (OP), the source address (SA), and the destination address (DA). The beginning of packet (BOP) and end of packet (EOP) are the indicators of header and tail flits, respectively. SA and DA are composed of two parts: horizontal (X) and vertical (Y) coordinates. The number of bits for X and Y are determined by the number of rows and columns in a 2D Mesh. The extra bit (0/1) in DA is used for PE recovery and will be explained in Chapter 6.

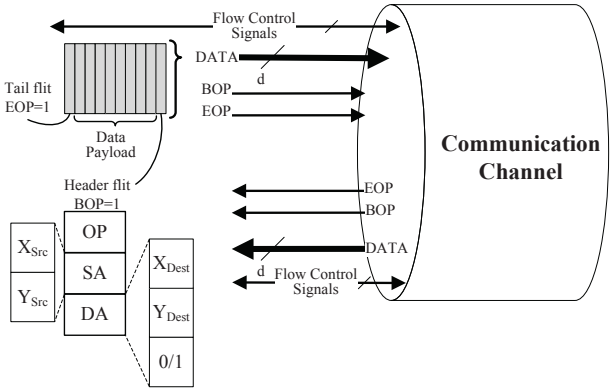


Figure 5.9: Data transmission format.

5.4.1 Virtual Channel Sharing Logic

The PVS approach is implemented on the input ports of the router. The buffer utilization is enhanced by dynamically allocating free buffers to overloaded ports. The definition of which buffers are shared among which ports is parameterized and can be adjusted to match any number of input ports according to the topology requirements. Only the processing element uses dedicated buffers for packet injection, which are not shared with any other router ports.

In the PVS approach, the input control logic is responsible for buffer allocation and receiving the data packets. An example of the PVS architecture, with two groups of two channels sharing VC buffers, is shown in Figure 5.10. Within each group, each port has its own (distributed) routing logic whereas VC allocation is centralized. Both the VC allocator and routing logic operate independently, without communicating with the control logic

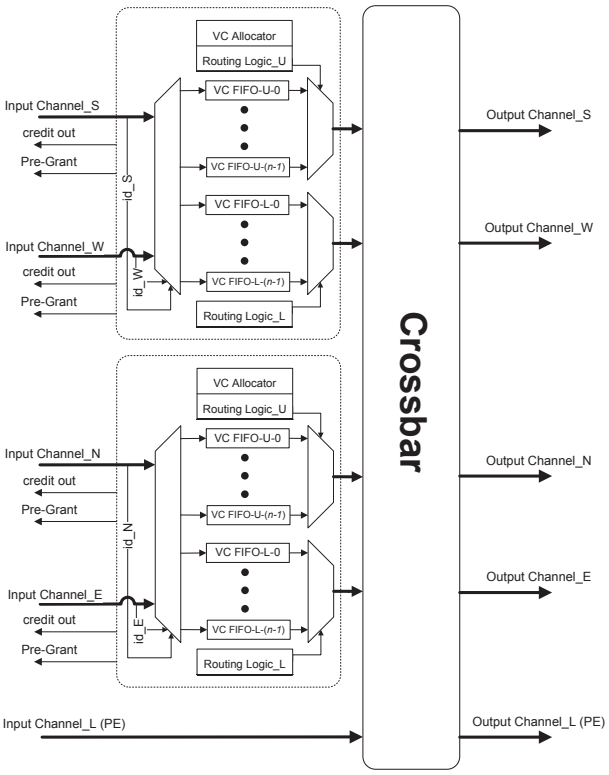


Figure 5.10: Proposed PVS approach for conventional Virtual Channel Architecture.

of other groups. The task of the VC allocator is to keep track of free buffers and to allocate them to the incoming traffic. After allocation, the routing logic computes the route for the packet and controls the crossbar for packet switching. In Figure 5.10, 'Routing Logic_U' refers to the control logic for the upper group of virtual channel buffers and 'Routing Logic_L' refers to the control logic for the lower group.

Selection Criteria

For load balancing, the selection of ports to share their VC buffers should be made on the basis of the number of router ports, the number of VCs per port, input bandwidth requirements for each input port and the number of groups (sharing VCs). The input ports are grouped to share the buffers in such a way that the total incoming load for the whole group approaches the average value of the bandwidth requirements per VC multiplied by the number of VC buffers in the current group, as described in Algorithm 4.

The MPEG4 application presented by [102] was discussed in section 5.3.2 for link load analysis. In the scenario described, the East and South ports of

the DR-SDRAM node should share their VC buffers, and the West and North ports should form a second sharing group. The heavy traffic, amounting to 942 MB/s at the East port, can utilize the resources of the South port which receives less traffic, only 60.5 MB/s only. When one of the incoming links of the DR-SDRAM node fails, its VC buffers can be utilized by the other port in the same group when needed. Moreover, under control logic faults, a port can utilize the VC resources of other ports in its group.

Allocation Policy

The PVS approach uses wormhole switching with partial sharing of virtual channel buffers. Due to sharing, race conditions may occur. This happens, for example, if only one buffer is available and multiple channels in the sharing group request ownership of the same buffer. To avoid such situations, a 'Pre-Grant' signal is used, as shown in Figure 5.10. 'Pre-Grant' is a single bit signal for each physical channel and set to high by default.

When two VCs are left for allocation and a VC request comes from any of the physical channel, one VC will be granted and the 'Pre-Grant' signal will be made '0' for that physical channel. After that, only the one remaining physical channel can access the one remaining VC. This avoids the race condition. However, a starvation situation can happen to the channel which does not get the access to VC. To address this problem, whenever a VC becomes free, the 'Pre-Grant' signal is raised for the channel which did not get the access to VC in previous case or has been waiting for the longest period. Another option is to assign the priority value for each channel in a sharing group. This approach is more useful for a handshake protocol. For application specific NoCs, the priority is proportional to the channel bandwidth requirements. The channel with the higher priority value is allowed to use the buffer.

Routing Algorithm

Different routing algorithms can be used with the PVS technique. However, there is the possibility of deadlock when more than one input ports share their VCs. For instance, if the North and East input ports share their resources and all the VCs are occupied by the flits coming from the East and going to the West, then the flits traversing from the North to the South direction in the upstream router (i.e., North router) have to wait. If the scenario results in a cyclic dependency, then a deadlock will occur.

In order to avoid a deadlock in static routing algorithms such, as static XY, at least one VC should be dedicated for each input port. Therefore, for a PVS unit which has w number of VCs being shared among u number of input ports, u dedicated VCs are needed (one VC for each input port) and

the rest of the (*w-u*) VCs can be shared. Similarly, since for the dynamic routing algorithms at least two input VCs are required for each input port to avoid deadlock [113], *w-2u* VCs can be shared.

5.4.2 Crossbar Switch

The *Output* part consists of a typical N×N crossbar switch with central control logic, where N is the total number of ports including the local PE port. The crossbar size can be customized according to the topology requirements. Wormhole switching is used for packet transmission, which makes efficient use of buffer space as the number of flit buffers per VC can be less than the packet size [122].

5.4.3 Comparison with Existing Architectures

A 5-port router with two unidirectional links per port and 10 internal buffers has been investigated and has been compared with other NoC architectures. Table 5.1 shows the result of this comparison.

The typical VC NoC represents a conventional virtual channel NoC architecture, with 2 VCs per port, which uses unidirectional channels to communicate with neighboring routers. Thus, two channels are required between two neighboring routers for two way communication. In the case of heavy traffic load on a certain port, the typical virtual channel architecture can provide only 2 VCs to receive the packets on that port. The PVS-NoC can provide 4 VCs to the same port under heavy traffic load. Thus, the VC availability has been doubled with only a small overhead of crossbar size.

A BiNoC has two bidirectional channels per port, for which the direction can be switched at run-time to meet communicate requirements [103]. Compared to the BiNoC architecture with 10 in-out channels, the PVS-NoC approach provides 5 input and 5 output physical channels. A PVS-NoC can provide 4 input and 4 output VCs per physical port whereas a BiNoC has only two physical channels per port, without VCs. The option of direction selection is provided at the cost of a large crossbar switch. Another issue to be addressed here is scalability. The number of VC buffers can be selected according to the application and topology requirements for our proposed architecture. To insert a new VC, the buffer and a controller are needed without any modification of the existing logic, and at the cost of only a slight increase in crossbar resources. To insert a new buffer in the BiNoC architecture, a separate buffer allocator is required and the crossbar is significantly larger.

The DSB-175 and DSB-300 router architectures have been described in [118]. To make an exact comparison, we define a DSB-160 architecture in accordance with [118]. The DSB-160 is a router with 160 flits of aggregate

Table 5.1: Comparison with existing NoC router architectures

Architecture⇒ Resource⇓	Typical VC NoC	BiNoC	DSB-160	FVS-NoC	PVS-NoC
Number of Buffers	10	10	10	10	10
Channels/Direction	1-in 1-out	2-inout	1-in 1-out	1-in 1-out	1-in 1-out
Max. VCs/Channel	2	1	2	10	4
Buffer Size	16 flits	16 flits	8 flits	16 flits	16 flits
Total Buffer Size	160 flits	160 flits	160 flits	160 flits	160 flits
Crossbar Size	4(1×2) + 5×5	10×10	2(5×5)	5×10 + 10×5	2(2×4) + 5×5

buffering. The buffers are divided between 5 middle memory banks with 16-flit buffers per bank and an aggregate of 80-flit input buffers comprising two 8-flit buffers (VCs) at each input port. The five memory banks are not considered in the comparison in Table 5.1 because only one flit can be written into and read from a middle memory in the DSB architecture, which reduces the utilization of memory banks. Thus, the static power consumption, without an increase in the system performance, is the major overhead of DSB architecture as compared to a PVS-NoC.

For further comparison, the Fully-Virtual-channel shared NoC (FVS-NoC) architecture has been investigated. In this architecture, any of the 10 VC buffers can be allocated to any input port. FVS-NoC channels are unidirectional. The architecture provides the maximum utilization of VC buffers at the cost of significantly larger crossbars, which makes the solution area and power expensive, compared to the proposed architecture.

5.4.4 Simulation Results

To demonstrate the performance characteristics of the proposed architecture (PVS-NoC), a cycle-accurate NoC simulation environment has been implemented in VHDL. The packets have a fixed length of seven flits, the buffer size is eight flits, and the data width is set to 32 bits. The 5×5 2D mesh topology is used for interconnection. Each input port has 4 VCs. With the same parameters, the typical virtual channel and FVS-NoC architectures are analyzed. The static XY wormhole routing algorithm is used.

The PVS approach with a grouping combination of (2, 2, 1) is used in the simulation, where '1' represents the buffer dedicated to the local PE. The critical path limits the operating frequency of the PVS router, which is 3.5% less than the operating frequency of the baseline virtual channel router. For the grouping combination of (3, 1, 1), the maximum operating frequency is around 9% less than the maximum operating frequency of the baseline virtual channel router.

Synthetic Traffic

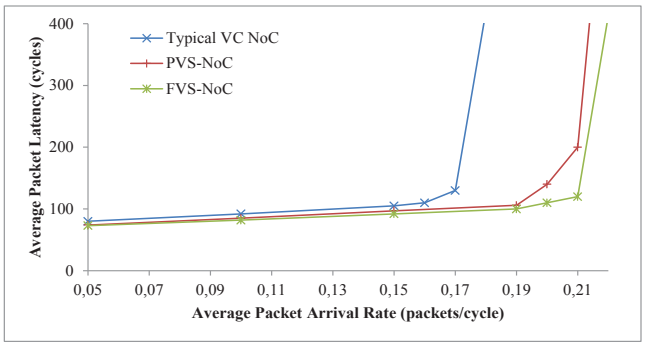
We compare the simulation results in terms of APL and saturation points using typical, PVS, and FVS virtual channel management policies. In the traffic analysis, we have evaluated the performance of the network using latency curves as a function of the packet injection rate. The packet latency is defined as the time duration between the generation of the first flit at the source node and the delivery of the last flit to the destination node. For each simulation, the packet latencies are averaged over 50,000 packets. Latencies are not recorded for the first 5,000 cycles to allow the network to stabilize. In the simulations, uniform, transpose and NED [117] traffic patterns are used.

The latency curves for uniform, transpose and NED traffic patterns are shown in Figure 5.11. It can be observed for all the traffic patterns that the PVS-NoC architecture saturates at higher injection rates when compared to the typical VC architecture, but at slightly lower rates than the FVS-NoC architecture. The proposed architecture manages bandwidth limitations by proper resource utilization and by making the load more balanced, without increasing the communication resources. The saturation point of PVS-NoC is just before that of the FVS-NoC because an FVS-NoC provides more buffer utilization by sharing the VC buffers among all the input ports. However, an FVS-NoC is not a power efficient solution, as verified below with application traffic.

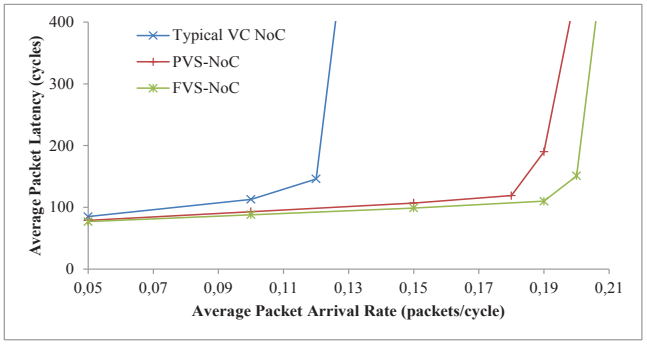
Application Benchmark

For application benchmark analysis, the encoding part of a video conference application with sub-applications of an H.264 encoder, MP3 encoder and OFDM transmitter is used. The video stream used for simulation purposes contains frames of 300×225 pixels, with each pixel consisting of 24 bits. Thus, each video frame amounts to 2025 KBytes and can be broken into 8400 data packets of 7 flits, including the header flit, where each flit is 32 bits wide. The *Mem. In Video* component generates the 8400 packets for one application cycle, equivalent to one video frame. The frame rate for the video stream is 30 frames/second and the data rate for the encoded video stream is 6167 kbps.

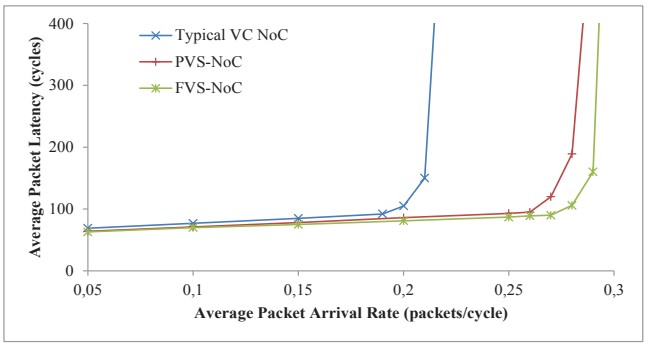
The application graph with 26 nodes is shown in Figure 5.3. It consists of processes and data flows; data is, however, organized in packets. Processes transform input data packets into output packets, whereas packet flows carry data from one process to another. A transaction represents the sending of one data packet by one source process to another, target process, or towards the system output. A packet flow is a tuple of two values (P, T). The first value, P, represents the number of successive, same size transactions emitted



(a) Uniform traffic load.



(b) Transpose traffic load.



(c) NED traffic load.

Figure 5.11: Average Packet Latency vs. Packet injection rate for 5×5 Mesh 2D NoC with (2, 2, 1) combination of PVS approach.

by the same source, towards the same destination. The second value, T, is a relative ordering number among the (packet) flows in the given system. For simulation purposes, all the possible software procedures are already mapped to hardware devices. The application is mapped to a 3×3 3D-mesh NoC. The details of this mapped application model are presented in [104].

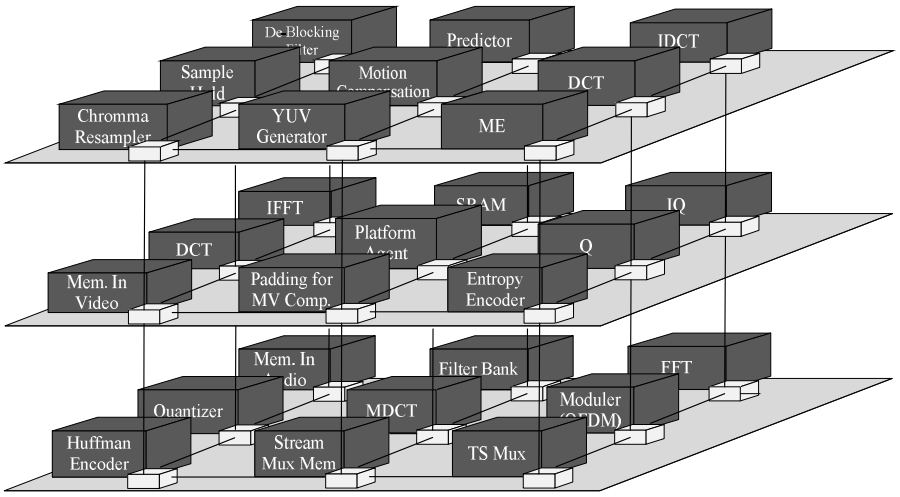


Figure 5.12: VCE application mapped to 3×3×3 3D-Mesh NoC.

Algorithm 4 was used to determine the individual grouping combination for each node according to the bandwidth requirements.

Meeting the application requirements, the application was mapped to a 3×3 3D-mesh NoC using Algorithm 2, as shown in Figure 5.12. The central node (1, 1, 1) is used as a platform agent for monitoring purposes. By applying Algorithm 4, an individual grouping combination has been determined for each node according to the bandwidth requirements. To estimate the power consumption, the high level NoC power simulator presented by [101] has been extended to support 3D-NoC architectures. The power consumption of the interconnection network (NoC switches, bus arbiters, intermediate buffers, and interconnects) is based on 35nm standard CMOS technology. The simulation results for APL, power consumption and the average router silicon area for the VCE application are shown in Table. 5.2.

The area of the 3D-symmetric-mesh-based routers (with 7×7 crossbars) is computed after synthesizing it with CMOS 65nm *LPLVT STMicroelectronics* standard cell library using Synopsys Design Compiler. The results for the average router silicon area of a 3×3×3 3D NoC are shown in Table 5.2. Each input port has 4 VC buffers. The size of the buffers is 8 flits and the data width is set to 32 bits. Here, the average silicon area is reported because different sharing combinations for PVS according to Algorithm I have different crossbar sizes and thus different silicon areas. The figures given in the table demonstrate that the area overhead of the proposed PVS technique is more reasonable compared to a fully shared virtual channel technique.

The PVS-NoC shows around 21% reduction in power consumption and around 7% reduction in silicon area but around 6% higher APL over the

Table 5.2: Experimental results for VCE application, mapped to 3×3×3 3D-Mesh NoC.

3D NoC Architecture	Power Consumption (W)	Average Packet Latency (cycles)	Average Silicon Area (μm^2)
Typical Symmetric 3D NoC (7x7)	1.587	186	195154
PVS-3D-NoC	1.713	144	203596
FVS-3D-NoC	2.182	136	220282

Table 5.3: Experimental results for VCE application, mapped to 5×5 2D-Mesh NoC, shown in Figure 5.4.

2D NoC Architecture	Power Consumption (mW)	Average Packet Latency (cycles)	Average Silicon Area (μm^2)
Typical VC	66.4	112	145712
PVS-NoC	72.1	87	151412
FVS-NoC	87.9	82	163742

FVS-NoC architecture. On other hand, the PVS-NoC shows approximately 22% reduction in APL value but around 8% higher power consumption and around 4% larger silicon area over the symmetric 3D-NoC architecture. Thus, the proposed PVS-NoC architecture provides a superior tradeoff between APL, power consumption and silicon area.

2D mesh mapped VCE application (Figure 5.4) was simulated using the same parameters and results are shown in Table 5.3. The power consumption results of the interconnection network (NoC switches, bus arbiters, intermediate buffers, and interconnects) are based on 35nm standard CMOS technology. The PVS-NoC showed 18% reduction in power consumption but 6% more APL over the FVS-NoC architecture. On other hand, the PVS-NoC showed 22.32% reduction in APL value but 7.9% more power consumption over the typical VC architecture. Thus, the proposed PVS-NoC architecture provides a better tradeoff between APL and power consumption.

Trade-offs

As already discussed in previous sections, the proposed PVS approach provides a tradeoff between system performance, area and power consumption. As shown with the synthetic traffic analysis for a 2D mesh NoC, the saturation point of the PVS latency curves comes close to the fully shared virtual channel architecture. On the other hand, PVS saturates at a significantly higher packet arrival rate as compared to the typical virtual channel architecture. For a uniform traffic load, the PVS architecture does not show significant improvement in the saturation point for the packet injection rate compared to typical VC architectures. The reason for this is that all the resources are equally loaded. The PVS based network has a lower average

Table 5.4: PVS-NoC router Silicon Area for different grouping combinations.

Grouping Combination	Silicon Area (μm^2)
Typical VC router (1,1,1,1,1)	145712
PVS router (2,2,1)	151412
PVS router (3,1,1)	153338
PVS router (3,2)	156180
PVS router (4,1)	159208
Full VC Shared router (5)	163742

packet latency for transpose and NED traffic loads than the typical VC architecture because it is better able to balance the link load, as discussed in section 5.3.1. PVS with (2,2,1) sharing groups requires 7% less area than FVS. Compared to the typical VC architecture, PVS has only a 3% area overhead. Silicon areas with different PVS sharing combinations, again synthesized on CMOS 65nm *LPLVT STMicroelectronics* standard cells using Synopsys Design Compiler, are presented in Table 5.4. It can be observed that the PVS area significantly increases as the sharing group size increases.

In the case of the video conference encoder application, the PVS architecture shows a significant reduction in the average packet latency with minor overhead of silicon area and power consumption. Moreover, the proposed architecture shows a significant reduction in area and power consumption over the FVS architecture with minor overhead of average packet latency.

5.5 Summary

In this chapter, resource-aware task allocation, resource utilization analysis and, on the basis of that, VC sharing technique were presented. For task allocation, IP cores with more communication requirements which directly communicate with a larger number of cores are given higher priority over less demanding IP cores. Afterwards, IP core mapping is performed according to a priority order. After defining an efficient mapping technique, resource utilization analysis was conducted. It was observed that many resources were not used at all or the level of utilization was very low, while at the same time some resources were overloaded. By maximizing the resource utilization of available resources, network performance can be improved with minimum overhead. Buffers consume the largest proportion of the dynamic and leakage power of an NoC router. To enhance the utilization of VC buffers and reduce the load of overloaded buffers in a router, a novel NoC architecture with a better tradeoff between resource utilization, system performance and power consumption than conventional VC based architectures was proposed.

Chapter 6

Network Level Fault Tolerance in PVS-NoC Architecture

In nanometer technologies, devices are exposed to a large number of noise sources such as capacitive and inductive crosstalk, power supply noise, leakage noise, thermal noise, charge sharing, and soft errors. This impacts the reliability of the manufactured devices [115] [106] and thereby the overall reliability of the system. For NoC based multi-core systems, there are a number of fault tolerant solutions at different abstraction levels of the system, for example routing algorithms [107] [125], architectures [112] [123], and error control coding schemes [99] [119]. Some of the proposed fault tolerant NoC architectures use intelligent routing algorithms [97] [95]. The major drawback of this approach is that the fault-free resources which are interconnected with the faulty resource cannot be used. This leads to a reduction in system performance. For instance, if there is a link failure in a VC based NoC, the VC buffers connected to the failed link cannot be used. To mitigate the effect of faults on system performance, such unused resources should be utilized by the system. A well designed network exploits all available resources to sustain performance.

In this chapter, technique to maximize the utilization of resources in the presence of faults, which in turn helps to retain the performance of PVS-NoC architecture (presented in 5) is presented. In addition, PVS-NoC architecture is further extended to provide the PE protection in case of network level faults. The architecture for PE protection provides the option for network interface (NI) assisted routing which also reduces the network load.

6.1 Existing Fault Tolerance Techniques

Faults can be categorized as permanent, intermittent, and transient [94]. Different techniques are required to deal with different kinds of faults. Neishabouri

et al. [111] proposed the *Enhanced Reliability Aware Virtual Channel* (ERAVC) architecture for NoC. ERAVC enables dynamic VC allocation and reliability aware sharing among input channels. More memory is allocated to the busy channels and less to the idle channels. In addition, ERAVC uses fault-tolerant flow control which allows packet retransmission without requiring the extra buffers. ERAVC shows significant reduction in Average Packet Latency (APL) for typical system operation at the expense of complex memory control logic. If a router node is marked faulty, the approach balances the traffic load well. However, the approach cannot utilize the intact resources of a partially-faulty router.

Fick et al. [98] devise a strategy to utilize the inherent redundancy at network and router level to maintain correct operation. In the proposed Vicis architecture, the network layer is reconfigured by swapping ports so that defective ports come together as pairs on the same link, thereby increasing the number of usable links with two intact ports. Router level reconfiguration is used to tolerate internal faults of a router which are not visible at the network level. A crossbar bypass bus is used to tolerate crossbar failure. Error correction coding (ECC) is used to protect data path elements. Each router uses built-in-self-test (BIST) to diagnose the exact locations of hard faults, intended for better utilization of ECC, port swapper and crossbar bypass bus. To minimize the overhead, the port swappers does not need to be fully connected, i.e., not every port can be connected to every physical link. The link to the local network adapter is able to connect to three different input ports and other links are able to connect to two input ports. For swapping to be still effective, the pair of failed input and output ports must belong to the same swapping group.

Concatto et al. [93] present a highly reconfigurable fault tolerant NoC router architecture. The architecture can dynamically stop using a faulty flit buffer unit and instead borrow the flit buffer units from the neighboring channels to sustain performance. However, the fine-granular bypassing and borrowing of flit buffer units makes the control logic very complex. Thus, the proposed solution is not area and power efficient.

In typical NoC architectures, a fault in a router or in a network interface (NI) results in an unconnected resource. Lehtonen et al. [105] achieve fault tolerance in such situations by introducing multiple-NI architectures. This approach improves the system fault tolerance on topology level. The throughput performance can also be enhanced by utilizing multiple routes and reducing the number of communication hops. However, this approach has significant area overhead, and it is not power efficient for synchronous systems unless power gating is introduced for all NIs. Zonouz et al. [126] propose a dual connected mesh structure (DCS) like [105] and has similar problems.

A lightweight fault tolerant mechanism for NoCs based on default backup

paths has been proposed by Koibuchi et al. [96]. These backup paths are able to maintain the connectivity of healthy routers and processing cores in presence of the faults. However, the critical path for packet transmission increases with the number of faulty routers on the transmission path of the packet. Moreover, the packet is transferred via both, the intermediate routers and their local PEs. To avoid the transmission overhead via PEs, additional logic is required which increases the area and power overhead. Another issue with the proposed architecture is its scalability. The authors claim that all the PEs can still be connected even in case of failure of all the routers. In this case, the toplogy becomes a ring which cannot meet the bandwidth requirements of hundreds of connected cores.

Lotfi-Kamran et al. [107] presented a decision making routing algorithm to avoid congestion in 2D NoC architectures. In addition, the proposed dynamic routing approach can tolerate a single link failure. However, the resources connected to a faulty link, e.g. VC buffers and control logic, cannot be utilized by the proposed technique. A similar issue occurs with other fault tolerant routing techniques. Hence, we propose a NoC architecture which addresses the utilization of the available communication resources in the presence of faults.

The main motivation of this work is to formulate a NoC architecture which offers low communication latency and high network throughput and minimizes design overheads. The tradeoffs among performance, power consumption and area are balanced using the proposed PVS approach. It also reduces the impact of different fault scenarios on system performance without requiring additional redundant hardware. Unlike the conventional architectures, the proposed architecture maintains system performance by utilizing the fault-free subcomponents which are part of a faulty component. In addition, the proposed architecture works reliably even if routing logic is faulty and also makes the processing element accessible to the network if its router fails without affecting the critical path length of proposed architecture.

6.2 Fault Scenarios

In this section, effects of different faults in router and communication links are described. Fault tolerance techniques for PEs are out of the scope of this thesis. Approaches like [105] [126] can be used to create fault tolerant NIs.
CASE-1: Faulty Links. Consider that a fault occurrs on a network inter-router-link. In typical architectures, the input/output buffers connected to this link cannot be used anymore. Assume that there are X faulty links in a NoC based system, each input port contains V virtual channels with buffer depth d, and each flit size or buffer width is f bits. If there is a VC controller for each virtual channel, the resources which cannot be used by the system

amount to $X \cdot V \cdot d \cdot f$ fault free memory cells, $X \cdot V \cdot f$ connecting wires and X control logic units.

These resources could be switched off using power gating technique to avoid at least their unnecessary power consumption. However, this would leave their chip area wasted. Instead, it is more beneficial to utilize those resources to improve the system performance or to reduce performance degradation in case of faults. For example, consider the NoC platform shown in Figure 6.1. A packet routed from node '001' to node '002' would take the vertical link upward in absence of faults. If the communication link between the nodes is broken as shown in Figure 6.1, the packet is rerouted via nodes '101' and '102'. At the same time, if node '111' is the source and node '000' is the destination, the packets for this source-destination pair will be routed via nodes '101' and '100'. In this situation, the resources on the new routes and especially the router at node '101' will be overloaded. The input ports from '111' and '001' are overloaded due to the fault, while other inputs might not be congested.

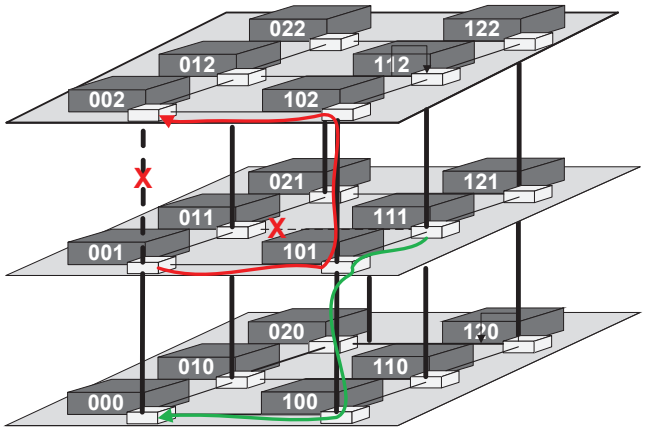


Figure 6.1: Routing in presence of faulty links.

CASE-2: Deadlock due to Faults. VCs are used to avoid deadlock [113]. Consider the situation that there are *N* VCs and out of them, *N-1* VCs have a fault. This fault can occur in any part of the FIFO, e.g. Rd./Wr. controller, content counter or memory flip-flops, see Figure 6.2. In this case, the architecture becomes equivalent to the typical non-VC architecture and deadlock can occur if the routing relies on the availability of virtual channels for deadlock avoidance.

CASE-3: Load Management. If a FIFO is faulty, the communication performance of the corresponding port is reduced because it is overloaded by the packets which are supposed to be transmitted via the port's VCs. Now consider that the neighboring port is free and thus the corresponding VC buffers are available. Without buffer sharing, the overloaded port cannot

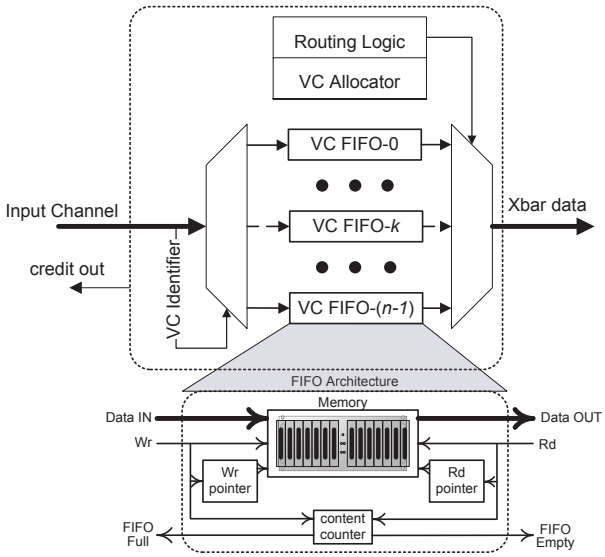


Figure 6.2: Typical Virtual Channel Input Port Architecture.

utilize these resources to manage the load. Such faults become a bottleneck for the overall system performance. Existing works do not consider load management mechanisms at micro-architecture level, although there are traffic routing algorithms dealing with load balancing on a higher level as discussed in section 6.1.

CASE-4: Faulty Routing Logic. If a fault occurs in the logic of a VC allocator, the corresponding physical link and VC buffers cannot be used anymore. Then the traffic needs to be re-routed using some fault-tolerant routing algorithm as discussed in section 6.1. The use of non-minimal routes may be necessary. In this case, not only the system throughput is considerably reduced but also there is unnecessary power consumption for VC buffers and control logic. Congestion on some nodes and power consumption due to non-minimal paths may raise thermal issues as there is a vicious circle between heat and power consumption [120]. This scenario is similar to CASE-1 but differs in the faulty resource (VC routing logic instead of inter-router link).

CASE-5: Resource Reclamation under Faults. Assume that a fault occurs in routing logic or VC allocator of one port and that another fault occurs on the physical link of a second port. Without sharing, the physical link of first port cannot utilize the VC allocator and buffers of the second port. So neither of the physical links can be used for packet transmission. This may cause the complete router to fail.

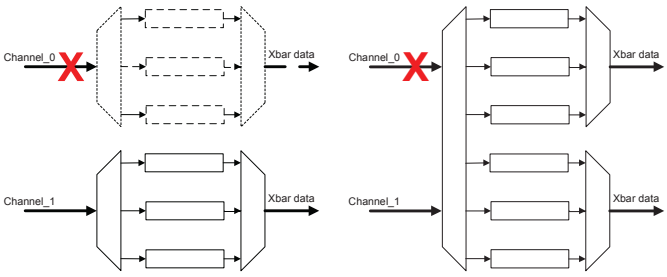
CASE-6: Processing Element Recovery. Once a router has been marked as faulty or its link to the local PE is broken, a well functional PE is isolated

from the rest of the system. Network level fault tolerance issues cannot address this problem. Multiple NI architecture and default backup paths have been proposed to maintain the accessibility of PE to the rest of the system as discussed in section 6.1. However, multiple NI architecture is inefficient due to its area and power overheads, and the default backup path approach increases packet latency.

6.3 Performance Sustainability under Faults

The main feature of the PVS-NoC architecture is to retain the system performance up to a certain level after the occurrence of faults. In NoC based interconnection platforms, a fault can occur in four types of components: physical link, buffer, controller or network interface. The faults on these resources can make the connected components non-functional. By our proposed sharing approach, the functionality of affected fault-free components can be restored to retain the system performance. Subsequently, the fault cases mentioned in section 6.2 are addressed. The dashed boundary (− − −) in figures ranging from Figure 6.3 to Figure 6.7 represents fault free resources which are not functional due to the faults on other resources.

CASE-1: Faulty Links. When a fault occurs on a physical link, buffers and control logic cannot be used by the NoC based system as can be observed in Figure 6.3(a). If a fault occurs in 'Channel_0', its VC buffers and routing logic cannot be used to route a packet. Now, consider this situation for the PVS approach as shown in Figure 6.3(b). If the fault occurs on 'Channel_0', 'Channel_1' can utilize the VC buffers and control logic to enhance the system throughput and avoid the unnecessary static power consumption by the VC buffers and control logic.



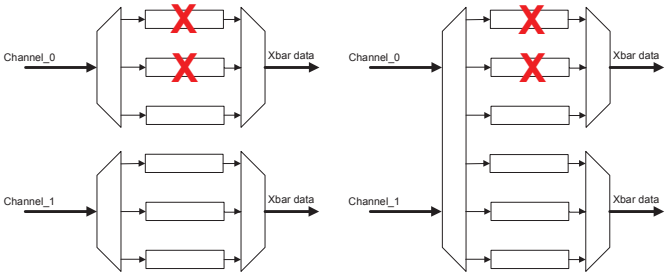
(a) Conventional architecture: fault affecting resource utilization.

(b) PVS architecture: fault in channel not affecting use of buffers.

Figure 6.3: Resource utilization under faults by PVS approach.

CASE-2: Deadlock due to Faults. If most of the VC buffers of a spe-

cific port become faulty, the number of available VCs may drop below the requirements of a deadlock-free routing algorithm. An example is depicted in Figure 6.4(a), for the input port of 'Channel_0', all but one of the VC buffers are faulty. The architecture becomes equivalent to a non-VC architecture and deadlock may occur. Now consider the PVS architecture shown in Figure 6.4(b). Only one VC buffer is left of the three normally allocated to 'Channel_0', but it can benefit from the shared VC buffers of 'Channel_1' and maintain enough VCs to operate without deadlock.



(a) Conventional architecture: Faulty VC buffers making the port overloaded and deadlock can occur.

(b) PVS Architecture: Impact of faulty buffers reduced and deadlock avoided.

Figure 6.4: Load management in PVS approach under faults on VC buffers.

CASE-3: Load Management. To balance the load on input buffers and to provide a relief to the loaded ports, the sharing of VC buffers by the input ports allows to balance the negative effect of a faulty FIFO. In a typical VC architecture, if one or multiple buffers become faulty as shown in Figure 6.4(a), the port ('Channel_0') becomes overloaded as packets are waiting to be routed but less functioning resources are available. In the PVS architecture, as shown in Figure 6.4(b), the available VC buffers of 'Channel_1' can be used to route the blocked packets. Thus, the fault impact will be distributed equally over 'Channel_0' and 'Channel_1'.

CASE-4: Faulty Routing Logic. Consider the 3×3 NoC mesh shown in Figure 6.5. If a fault occurs at node '12' in the routing logic for the input port from node '11', the packets for node '12' from node '11' is re-routed through the paths shown with blue color according to the assumed fault tolerant routing algorithm [121]. However, any other fault tolerant routing scheme can be used to observe the similar issue. All the resources on that input port cannot be used by the NoC system as shown in Figure 6.6(a). Without virtual channel sharing as in PVS approach, no packets can be routed through 'Channel_0' due to the fault in the 'Routing_Logic_0'.

With the PVS approach, the VC allocator marks the routing logic faulty

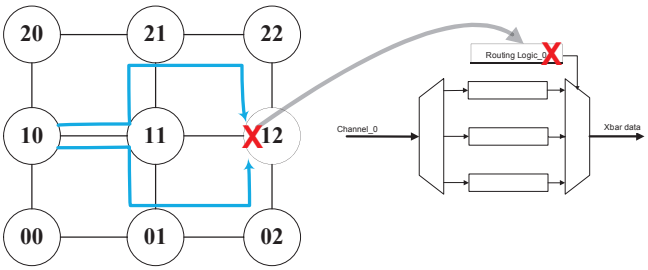
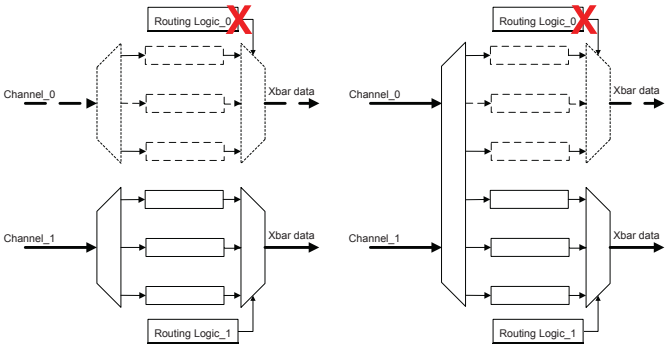


Figure 6.5: Impact of faulty routing logic on routing.

and the VC buffers controlled by that routing logic are not allocated to any packet for the purpose of transmission. Even if fault occurs on 'Routing_Logic_0', 'Channel_0' can be used as shown in Figure 6.6(b). Only the buffers shared by 'Channel_1' are available for both channels ('Channel_0' and 'Channel_1') which lowers the bandwidth. But the packets do not need to be rerouted and thus the fault can be tolerated with minimized overhead.



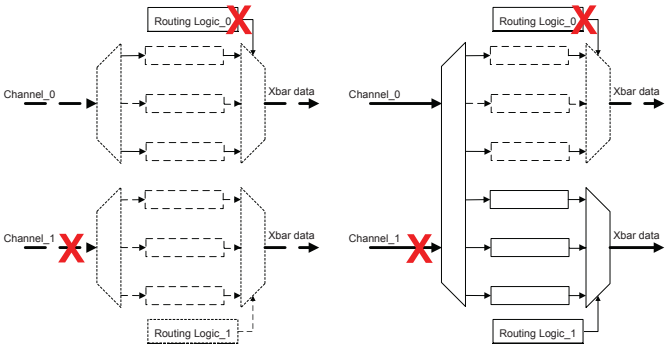
(a) Conventional architecture: Faulty routing logic requires re-routing of packets.

(b) PVS Architecture: Routing logic fault tolerated.

Figure 6.6: Routing logic fault tolerance by PVS approach.

CASE-5: Resource Reclamation under Faults. If a fault occurs in the routing logic of one port, the port's resources cannot be used for packet transmission even if its link and VC buffers are still functional. Similarly, if a fault occurs on a physical link, the connected port cannot receive packets from that link even if its routing logic, VC buffers and mux/demux are functional. Now consider that these faults occur on resource-sharing ports of the same switch as shown in Figure 6.7(a). That is, faults occur on 'Routing_Logic_0' and 'Channel_1'. Without PVS, neither port can receive packets. Also, many functional subcomponents cannot contribute towards system performance while consuming static power. By using the PVS ap-

proach, the functional resources can be recovered and 'Channel_0' can use 'Routing_Logic_1' to receive the packets as shown in Figure 6.7(b). This sustains the system performance and avoids waste of power.



(a) Conventional architecture: Faulty-free components cannot be used because of fault on other components.

(b) PVS Architecture: Fault free components from neighboring nonfunctional channels reclaimed to make one channel functional.

Figure 6.7: Resource reclamation by PVS approach under faults.

CASE-6: Processing Element Recovery. The critical path for packet transmission via shared VC buffers limits the maximum operating frequency. The length of the critical path increases with the number of input ports that share VC buffers due to the increased complexity of the VC controller and the input multiplexer. If the VC buffers dedicated to a local PE are shared with another input port as shown in Figure 6.8, the resulting path is not longer than all the other paths in the router. Hence, it will not affect the maximum operating frequency. This gives an opportunity to recover PEs in the presence of network level faults.

For PE recovery, each PE is connected to multiple routers by a multiplexer-demultiplexer pair. For demonstration, PE0 is connected to two routers, R0 and R1, as shown in Figure 6.8. Any existing on-line router fault detection technique, for example the one proposed in [89], can be be used in the 'Fault Detector' component. When the fault detector detects a fault of router R0, it changes the injection and reception paths of PE0 to backup paths using the multiplexer (Rx) and demultiplexer (Tx). After fault occurrence on router R0, PE0 transmits and receives the traffic via router R1. In this novel approach, each PE can be connected to multiple routers using only a single network interface (NI), keeping the overhead low and the system compatible with single NI cores. In addition, overhead is very low compared to the multiple NI architecture presented by [105]. The critical path length for packet transmission is slightly increased by a 2×1 multiplexer, which is significantly

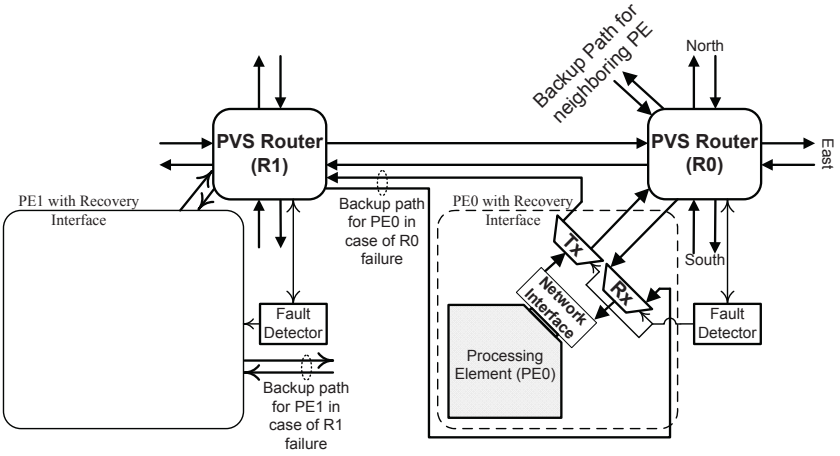


Figure 6.8: PE Recovery architecture.

lower overhead compared to the backup path approach presented in [96].

Fault tolerant routing algorithms, for example [125] and [107], can only deliver the traffic to the desired destination if the PEś router is functional. Using the proposed PE recovery approach, the traffic can reach at its destination even if the router is faulty but it needs a new addressing scheme.

Consider that router R0 in Figure 6.8 is marked as faulty. PE0 can still transmit and receive traffic through router R1. Besides the failure information of R0, the address of the router (R1) which replaces R0 has to be broadcasted in the network. Nodes that generate and transmit data originally destined at R0 have to update their local address information and address their data to destination R1 instead of R0. R1 must differentiate between the packets for the local PE of R1 and packets for PE0. This is achieved by an additional bit appended to the destination address (DA) as shown in Figure 5.9. Only one additional bit is needed if PEs are connected to two routers. If PE0 is connected to more than two routers, multiple bits are required. Whenever the appended bit is set to '1', the packet is delivered through the backup path to the PE of the neighboring faulty router node (PE0). In the current scenario, it is possible to use all the PEs in the system if 50% of routers are failed but if only one router is functional in each pair as shown in Figure 6.8.

The architecture of the PVS router with PE recovery is shown in Figure 6.9. It can be observed that each PVS block at the router input side has two inputs and two outputs whereas the PE recovery block has just one output. It is possible to design all the blocks with a single output, which would result in a 3×6 crossbar. However, this would halve the physical output bandwidth. This is acceptable only for the PE recovery block as the backup path for the PE is used only for packet injection and in case of fault occurrence.

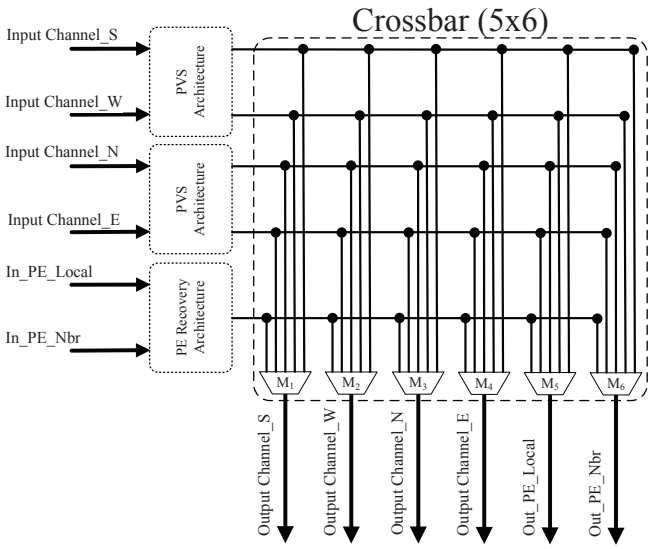


Figure 6.9: PE Recovery Router.

The PE recovery block with two inputs and a single output for packet injection to the network is shown in Figure 6.10. To receive the packets from the network, both PEs need the dedicated link from the crossbar. Thus, a 5×6 crossbar can be used for the PE recovery architecture. As compared to a 5×5 crossbar, a 5×6 crossbar requires one more 5×1 multiplexer (M_6) without increasing the critical path length whereas a 6×6 crossbar increases the critical path length in addition to extra gate count.

The insertion of a 2×1 multiplexer (Figure 6.9: PE Recovery Architecture) does not affect the critical path of the router if at least two other ports share the VC buffers. This is because the inserted 2×1 multiplexer for PE recovery operates in parallel with the other PVS blocks.

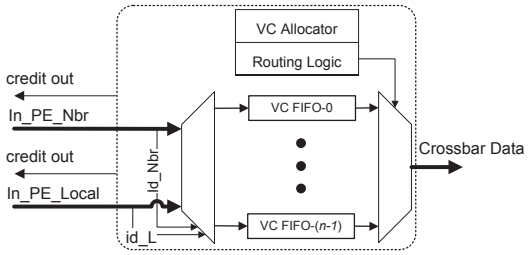


Figure 6.10: PE recovery architecture with dual inputs and single output.

6.4 Network Interface Assisted Pre-Routing

The processing element (PE) recovery architecture presented in CASE-6 of section 6.3 can be used for NI assisted routing of packets which helps to reduce the network load. For this purpose, we divide the network into clusters. For demonstration, each cluster is comprised of two nodes as shown in Figure 6.11. The approach is scalable to any number of routers in a cluster. In current scenario, there are three possibilities for routing of packets between two clusters. For demonstration, consider 'Cluster 15' and 'Cluster 11'.

- If nodes '05' and '53' communicate with each other. In this case, source and destination, both are at far ends of the clusters. In this case, pre-routing approach can save two hops.
- If nodes '05' and '43' or nodes '15' and '53' communicate with each other. In this case, one of the nodes is at inner end of the cluster while other node is at far end of the cluster. Using the proposed pre-routing approach, one hop can be saved.
- If nodes '15' and '43' communicate with each other. In this case, both nodes are at inner ends of the communicating clusters. In this case, normal routing technique can be used because pre-routing cannot contribute to save any hop.

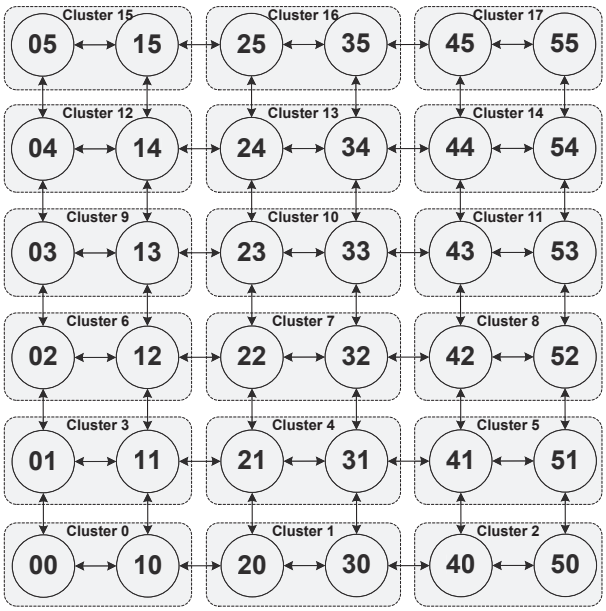


Figure 6.11: Clustering of 6×6 NoC for NI assisted pre-routing.

The next step is to propose an architecture to divide the network into clusters and implement the pre-routing approach without significant overheads. For this purpose, we analyze the PVS architecture. The critical path for packet transmission via shared VC buffers in PVS architecture limits the maximum operating frequency. The critical path length increases, if the number of input ports sharing VC buffers is increased. This is due to larger VC controller and demultiplexer. If the VC buffers dedicated to local PE are shared with neighboring PE (PE_Nbr) as shown in Figure 6.9, the critical path length of the router does not increase because the PVS architecture for PEs operates in parallel with other PVS architectures in the router. So, it will not affect the maximum operating frequency. This gives an opportunity to pre-route the packets by NI.

As each PE will inject the traffic to the network through one of the routers in cluster at a time, it is possible to apply the single port output PVS approach after NI assisted pre-routing. The designed router will route the packets by using any existing routing algorithm. The pre-routing of packets will be done by NI before injecting the packet to the network. The detailed NI architecture is shown in Figure 6.12.

For packet injection, the pre-routing logic computes the node identification bit on the basis of destination address (DA), topology table and the cluster table. Accordingly, the pre-routing logic selects the packet transmission node by T_x Router Selector bit. Once the the transmission node has been selected, other flits follow the route header flit. For packet reception, the control logic and FSM will be explained in section 6.4.2.

6.4.1 Addressing Scheme

For pre-routing, each PE is connected to two routers by a multiplexer and demultiplexer. For demonstration, PE0 is connected to two routers R0 and R1. During transmission, NI of PE0 will decide, which of two routers in transmission cluster is near to the destination cluster. According to that, NI will select the demultiplexer output using transmitter router selection signal 'Tx Router Signal'. PE0 can inject a packet to the network via router R1, if it is closer to the destination node and saves one hop of packet traversal in the network. In addition to selection of R0 or R1 for packet transmission, another task of NI is to put the destination address (DA) in packet header. NI will select the node in destination cluster, which is closest to the transmission cluster, to deliver the packet. The detailed cluster architecture for NI assisted pre-routing is shown in Figure 6.8.

Here, the problem is to differentiate between the packets for PE0 and PE1 at the time of reception. To resolve this problem, node identification bit (NIB) is appended with DA as discussed in Chapter 5 and shown in Figure 5.9. Whenever NIB is set to '1', the packet is delivered to other PE

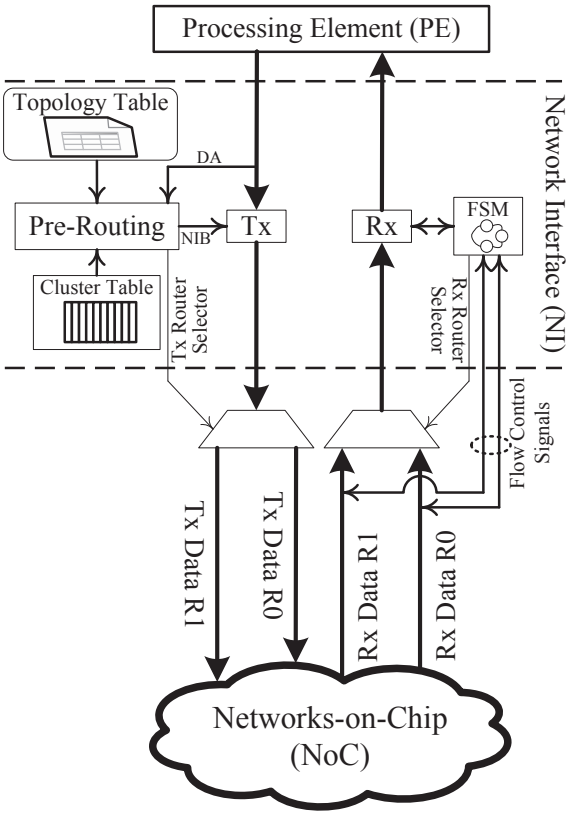


Figure 6.12: Network Interface architecture for pre-routing of packets.

in the cluster. Only one bit wide NIB is needed, when PE is connected to two routers. Multiple NIB bits are needed, if PE0 is connected to more than two routers.

6.4.2 Control Logic for Packet Reception

For data reception, control logic block in Figure 6.12 will decide, which router can deliver the traffic to PE0 using 'Rx Router Selector' signal. The state machine for control logic is quite simple and area efficient. To be fair, we choose the round-robin policy for arbitration. It should be noted that to maintain full compatibility with the existing NI transmission protocols, a packet based arbitration is used. This means that the arbitration is performed for each packet, not for each flit. The state machine contains four main states, which are R0-Turn, R0-Sending, R1-Turn and R1-Sending as shown in Figure 6.13. At the initialization time of the whole system, access to the output port of router R0 is granted to deliver the packet to PE0. In this state, if there is a request from router R0 (Req R0 = '1'), it will be

granted and the current state will be changed to R0-Sending. The current state remains in R0-Sending as long as the packet is being sent. End-of-packet signal (eop R0 = '1') indicates the transmission completion for router R0. At this time, the current state switches to R1-Turn indicating this is the R1 turn to send a packet (Req R0 = '0' and Req R1 = '0'). Alternatively, it will directly get the R1-Sending state, on condition that there is a request from router R1 (Req R1 = '1'). At the initial stage (R0-Turn), providing that there is no request from the router (Req R0 = '0'), the request signal from the router R1 will be checked. If there is a request from R1 (Req R1 = '1'), the current state will switch to R1-Sending and the packet transmission will be initiated. A similar procedure is followed when the current state is R1-Turn.

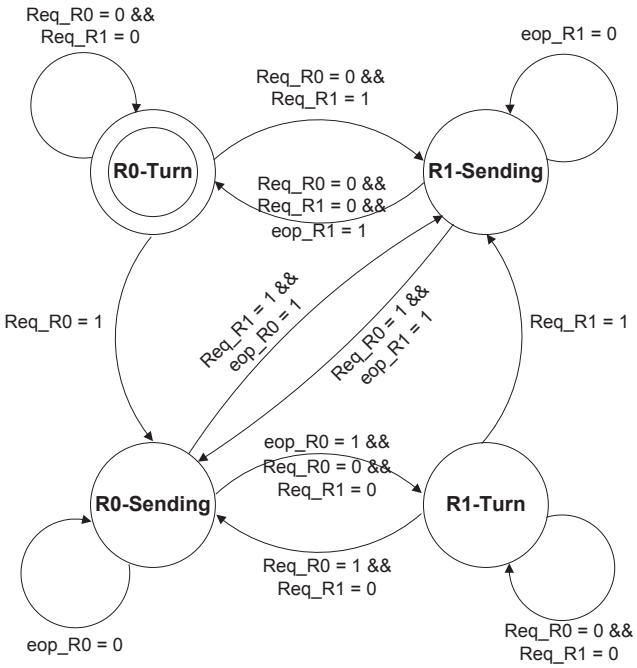


Figure 6.13: Finite state machine for the proposed control logic for packet reception of PE0 in Figure 6.8.

In this approach, each PE can be connected to multiple routers using only a single network interface (NI), keeping the overhead low and the system compatible with single NI cores. In addition, overhead is very low compared to the multiple NI architecture presented by [105]. Similarly, the critical path length for packet transmission is slightly increased by 2×1 multiplexer which is significantly lower compared to the backup path approach presented in [96].

6.5 Simulation Results

To demonstrate performance characteristics of the proposed architecture (PVS-NoC) under faults, a cycle-accurate NoC simulation environment has been implemented in VHDL. The packets have a fixed length of seven flits, the buffer size is eight flits, and the data width is set to 32 bits. The 5×5 2D mesh topology is used for interconnection. Each input port has 4 VCs. With the same parameters, typical virtual channel and FVS-NoC architectures are analyzed. The static XY wormhole routing algorithm is used for both non-faulty and faulty scenarios.

For the faulty scenario, it is assumed that an appropriate fault detection mechanism (test unit) similar to the one used in [100] detects the faulty links, and stores the fault information in the configuration registers of the routers connected to the faulty link. In this case, these routers will not send any traffic to the corresponding links and will reroute packets through one of the other adjacent routers by using the fault tolerant routing algorithm presented in [121]. The PVS approach with grouping combination of (2, 2, 1) is used in the simulation, where '1' represents no-sharing for PE buffers.

6.5.1 Performance Sustainability under Link Faults

We compare the simulation results in terms of APL and saturation points for two cases: a normal network with no fault using typical, PVS, and FVS virtual channel management policies, and an example of a faulty network with two faulty links using typical and PVS virtual channel management schemes as depicted in Figure 6.14. The system performability under link faults with PVS approach has been discussed in CASE-1 of Section 6.3.

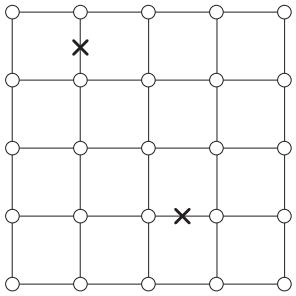
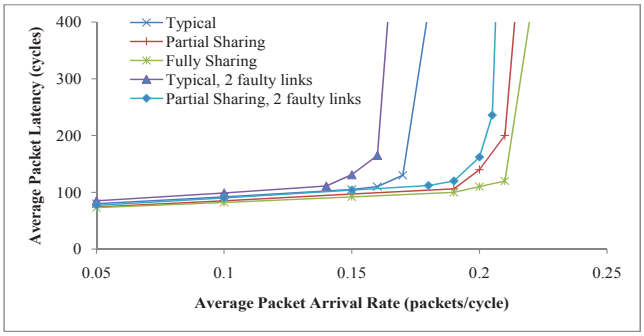


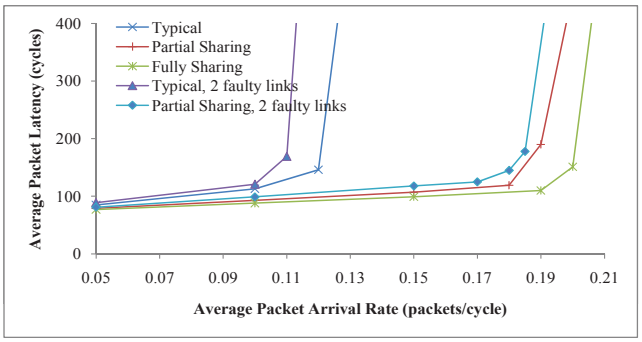
Figure 6.14: 5×5 2D-Mesh NoC with two faulty links

In traffic analysis, we have evaluated the performance of the network using latency curves as a function of the packet injection rate. The packet latency is defined as the time duration between the generation of the first flit at the source node and the delivery of the last flit to the destination node. For each simulation, the packet latencies are averaged over 50,000 packets.

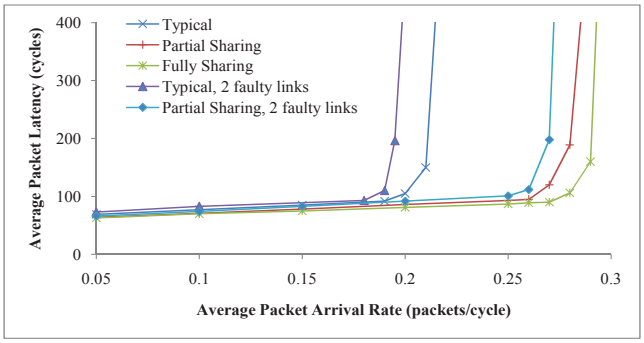
Latencies are not recorded for the first 5,000 cycles to allow the network to stabilize. In the simulations, uniform, transpose and NED [117] traffic patterns are used.



(a) Uniform traffic load.



(b) Transpose traffic load.



(c) NED traffic load.

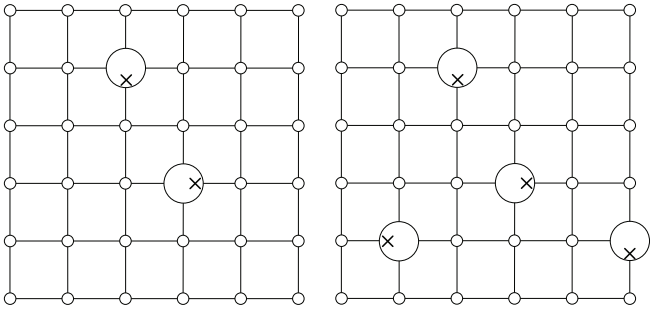
Figure 6.15: Average Packet Latency vs. Packet injection rate for 5×5 Mesh 2D NoC with (2, 2, 1) combination of PVS approach.

The latency curves for uniform, transpose and NED traffic patterns are shown in Figure 6.15. The curves for a normal network with no faults have

already been discussed in section 5.4.4 and presented here for the comparison purposes with faulty networks. For faulty networks, the curves reveal that for all the traffic patterns, the proposed architecture suffers less performance degradation compared to the typical architecture. The reason is that the VCs connected to the faulty links are utilized by the other channel which helps to reduce the average packet latency.

6.5.2 Fault Tolerance for Routing Logic

As discussed in CASE-4 of Section 6.3, if a fault occurs in the routing logic, the PVS architecture can tolerate the fault and packets do not need to be re-routed. To demonstrate that, the 6×6 2D mesh NoC with 2 VCs per port is simulated. The values of the other system parameters are the same as in Section 6.5. For demonstration, the faulty network with two and four faulty routing logic blocks is shown in Figure 6.16.

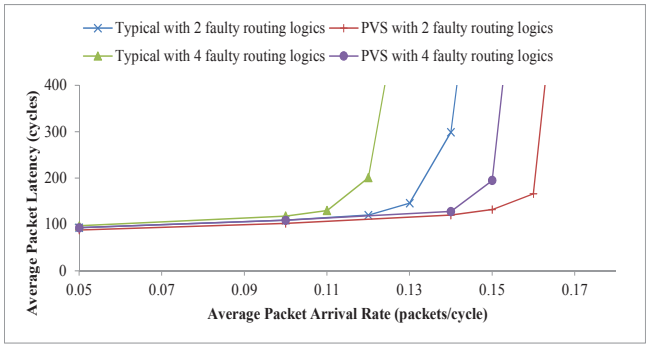


(a) Two faulty routing logics.

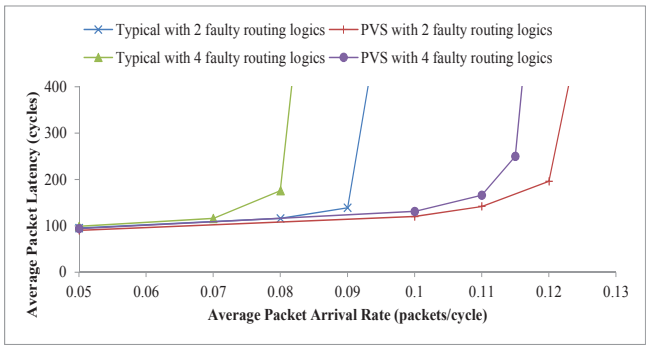
(b) Four faulty routing logics.

Figure 6.16: 6×6 2D-Mesh NoC with faulty routing logics.

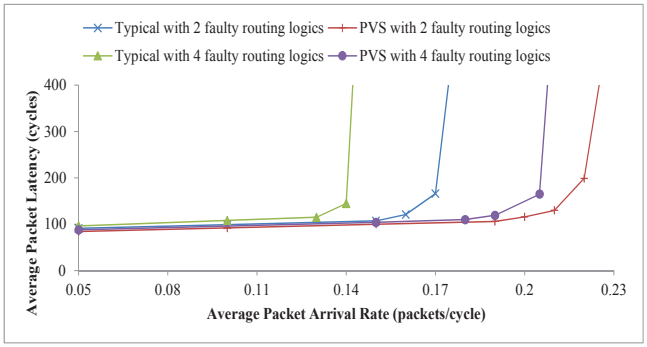
The latency curves with faulty routing logic for the network of Figure 6.16 are shown in Figure 6.17 for uniform, transpose and NED traffic patterns. It can be observed that for all traffic patterns the PVS-NoC architecture saturates at higher injection rates as compared to the typical VC architecture. The reason is that the PVS architecture does not re-route the packets in case of a routing logic fault. Instead, another routing logic block within the sharing group is used to route the packets through the same channel, and thus the average hop-count is not increased. For the PVS approach, it can be observed that the performance degrades just slightly when the number of faults increases. On other hand, in the typical virtual channel architecture, the system performance is severely affected as the number of faults increases.



(a) Uniform traffic load.



(b) Transpose traffic load.



(c) NED traffic load.

Figure 6.17: Average Packet Latency vs. Packet injection rate for 6×6 Mesh 2D NoC with fault on routing logics shown in Figure 6.16(a).

6.5.3 Reduction in Average Packet Latency by Network Interface Assisted Routing

To demonstrate performance characteristic of the proposed NI assisted routing approach, the same simulation parameters and routing technique was

used as discussed in section 6.5. The PVS approach with grouping combination of (2, 2, 1) was used for simulation purposes, where '1' represents the buffer dedicated to the local PE. The clustered PVS approach with grouping combination of (2, 2, 2) was used for simulation purposes, where last tuple '2' represents two PEs sharing the VC buffers.

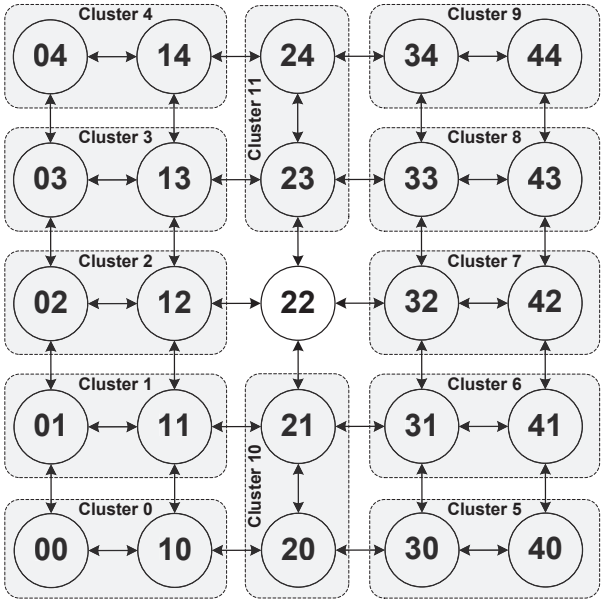
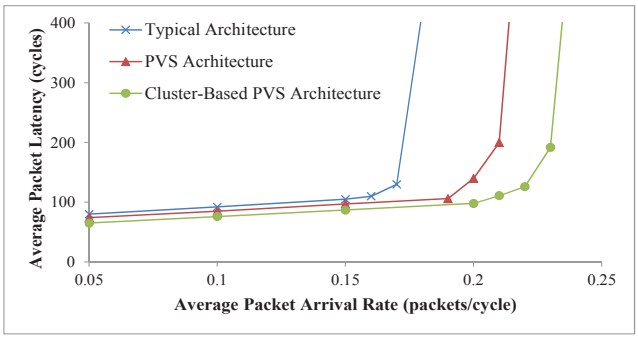


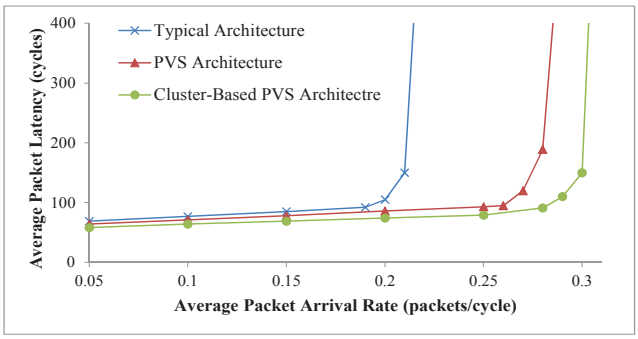
Figure 6.18: Clustering of 5×5 2D Mesh NoC for simulation of NI assisted pre-routing.

For simulation purposes, the 5×5 2D mesh PVS-NoC was clustered according to Figure 6.18. Due to odd number of nodes in NoC system and having two nodes per cluster, one of the nodes cannot be clustered. Here, node '22' was not clustered. There are options to cover all the nodes for clustering. One option is to have one cluster in the system with three nodes. In this case, node '22' can be included in one of the neighboring clusters like 'Cluster 2', 'Cluster 7', 'Cluster 10' or 'Cluster 11'. In this case, critical path length of the router will be increased due to sharing group of three PEs.

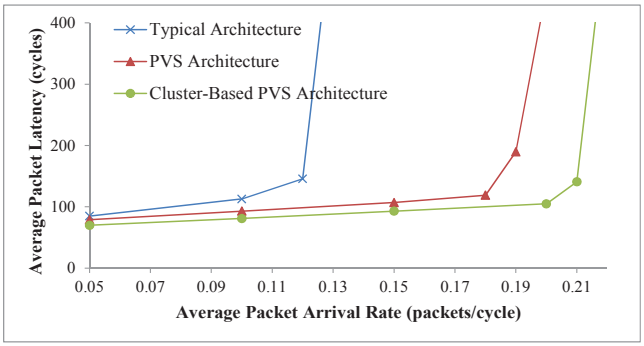
In traffic analysis, the performance of the network was evaluated using latency curves as a function of the packet injection rate. The packet latency was defined as the time duration when the first flit is created at the source node to when the last flit is delivered to the destination node. For each simulation, the packet latencies were averaged over 50,000 packets. Latencies were not collected for the first 5,000 cycles to allow the network to stabilize. Uniform, transpose and NED (Negative Exponential Distribution) [117] traffic patterns were used in the simulation. The NED is a synthetic traffic model based on Negative Exponential Distribution where the likelihood that



(a) Under uniform traffic load.



(b) Under NED traffic load.



(c) Under transpose traffic load.

Figure 6.19: Simulation curves for average packet latency (APL) vs. packet injection rate for 5×5 Mesh 2D NoC with (2, 2, 1) combination of PVS approach.

a node sends a packet to another node exponentially decreases with the hop distance between the two cores. This synthetic traffic profile is better in capturing key statistical behavior of realistic traces of communication among the nodes. The latency curves for uniform and NED traffic patterns are shown in

Router	Area (μm^2)	Overhead (%)
Typical VC based router	145712	Reference
PVS router: Single-PE (2,2,1)	151412	3.76
PVS router: Dual-PE (2,2,2)	152403	4.39

Table 6.1: Silicon area of the existing and proposed NoC router architectures.

Figure 6.19. It can be observed that for all the traffic patterns, the clustered PVS-NoC architecture saturates at higher injection rates as compared to the typical VC architecture and PVS architecture without clustering.

To demonstrate the overheads of the proposed technique, area of the typical VC based, PVS-NoC and clustered PVS routers was computed once synthesized on CMOS 65nm LPLVT STMicroelectronics standard cells using Synopsys Design Compiler. The simulation results for the router silicon area for a 2D-Mesh based NoC router are shown in Table 6.1. The sharing combinations used for PVS are also mentioned in Table.6.1 with router types. The figures given in the table demonstrate that the area overheads of the proposed cluster based technique is reasonable when compared to other techniques without clustering.

6.6 Summary

In this chapter, partial virtual channel sharing NoC (PVS-NoC) architecture has been used to reduce the impact of faults on system performance. In a conventional VC router, when a fault occurs on a channel, its corresponding VC buffers and routing logic cannot be used to route packets. However in PVS-NoC architecture, other channels can use the VC buffers and control logic of a faulty channel to enhance the system throughput and avoid unnecessary static power consumption by the VC buffers and control logic. In similar manner, if a fault occurs on a channel and also on a routing logic of another channel, PVS-NoC architecture can utilize the fault free components to make one channel active to reduce the impact of faults on system performance. The PVS-NoC architecture can also tolerate the routing logic faults without requiring additional logic. In addition, the PVS-NoC architecture is further enhanced to provide PE protection with minimal overheads and without affecting the critical path length of the PVS-NoC architecture. The PE protection architecture can also be used for network interface assisted pre-routing of packets to reduce the network load.

Chapter 7

AdaptiveZ Routing for 3D NoC-Bus Hybrid Architectures

Three-dimensional networks-on-chip (3D-NoC) is an extension of 2D-NoC, which utilizes the benefits of 3D IC technology such as reduction in length of global interconnects and decrease in power consumption while an increase in system performance [134]. A typical 3D-NoC switch has total of seven I/O channels. If bus architecture is used for inter-layer communications, the switch has total of six I/O channels including the connections to local PE. The stacked hybrid 3D NoC-Bus architecture has already been proposed by [127], and shown in Figure 7.1. In this architecture, routers connected to pillar nodes are different, as an interface between the dTDMA pillar (vertical link) and the NoC router must be provided to enable seamless integration of the vertical links with the 2D network within the layers. An extra physical channel is added to the router for the vertical communication. The extra channel has its own dedicated buffers, and is indistinguishable from the other channels. This hybrid system provides both performance and area benefits.

However, despite this encouraging result, there is an opposite side of the coin which paints a rather bleak picture, because the bus approach also suffers from a following drawback. Since the bus is a shared medium, it does not allow concurrent communication in the third dimension. Therefore, in high network loads, probability of contention and blocking critically increases. As result of this, there is a considerable degradation in inter-layer bandwidth despite single-hop vertical communication does improve performance in terms of overall latency. In this chapter, the buffer utilization and routing issues for stacked mesh architecture are addressed. The proposed approach also provides fault tolerance against single bus failure.

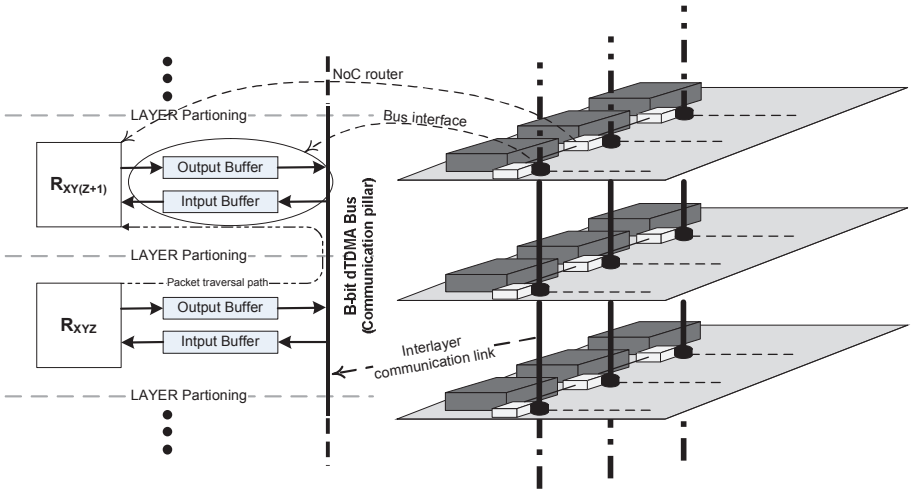


Figure 7.1: Side view of the 3D NoC with the dTDMA bus

7.1 Motivation and Contribution

In the static XYZ routing algorithm for stacked mesh architecture presented by [127], consider that R_{XYZ} is the source, which needs to send a data packet to the destination R_{XYZ+1} as shown in Figure 7.1. This particular architecture is neither power nor performance efficient because of inefficient bus utilization. This can be justified by considering different communication scenarios as follows.

Bus is busy but input buffer of R_{XYZ+1} is free. The packet will get stuck in R_{XYZ} output buffer as long as the bus is busy and arbiter does not give bus access to R_{XYZ}. If the neighboring bus becomes free at a later stage to route the packet, the routing cannot take place even with adaptive routing algorithms after the packet has been written to the output port buffer of R_{XYZ}.

Bus is free but input buffer of R_{XYZ+1} is full. The packet will be delayed for an undetermined length of time, because another transaction may start at the bus. In the meanwhile destination buffer may be available. Also, if another node with a high priority packet requests access to the bus or even worse the R_{XYZ+1} input buffer, the delivery time of packet becomes completely uncertain.

Bus is busy and also the input buffer of R_{XYZ+1} is full. For the stacked mesh architecture, if the destination node is not exactly above/below the current node, packets will wait unnecessarily even if the rest of the network resources are available.

Thus there are two major drawbacks in the stacked mesh architecture. First, each packet is traversed through two buffers, which increases the dy-

namic power consumption. Secondly, the output buffers are not beneficial in reducing the latency. They also hinder the on-chip network from implementing adaptive and congestion-aware routing algorithms. This happens because when the packet is written to an output buffer, it is not possible for the connected router to bring it back and re-route it. The presence of the output buffer also affects the static power consumption.

To address these problems, an efficient inter-layer communication scheme and routing algorithm which enable congestion-aware communication are proposed. The contribution of this chapter is to address the issues arising when two different communication media (NoC and Bus) are hybridized to form a 3D communication scheme. We will present in detail in Section 7.2 and Section 7.3 that by removing the output buffers, setting up a smart inter-layer communication, and exploiting an appropriate adaptive routing algorithm, the average packet latency (APL) and power consumption for stacked mesh architecture (presented in [127]) are reduced while fault tolerance has been enhanced.

7.2 Proposed Architecture

It is not an efficient approach to connect a NoC router to a vertical bus without considering the characteristics of both platforms. The extra output buffer in Figure 7.1 can hinder the routing adaptivity and load balancing. In this section, we approach the problem in different phases. Initially, the buffer on output port is removed to reduce power consumption and to enable implementing an adaptive routing algorithm. Then, an adaptive inter-layer routing algorithm called *AdaptiveZ* ($\widetilde{Z}$) is applied. For the sake of simplicity, it is assumed that for intra-layer communication a static XY routing algorithm is used. Note that the proposed inter-layer routing algorithm is not dependent to intra-layer routing policy. Therefore, in $\widetilde{Z}$XY routing algorithm, the XY routing is projected in different layers identically as shown in Figure 7.2 with the adaptive inter-layer communication. The *AdaptiveZ* routing algorithm is elaborated in Algorithm 5.

As can be seen from Figure 7.2, for inter-layer communication, the first bus pillar that is available on the way for the vertical communication will be used. For this purpose, the router sends a request signal *req* with the destination layer ID to the bus arbiter after each hop on its way until it reaches the target layer or being exactly below/above the destination router. Based on the status of the target buffer and bus, the arbiter decides to grant, reject, or suspend the request. Once the *grant* or *wait* signals are received, the packet waits there accordingly, to be delivered to the intermediate node in the destination layer. In order to determine if the destination is free or busy, we use the stress values of the bus which is calculated based on the

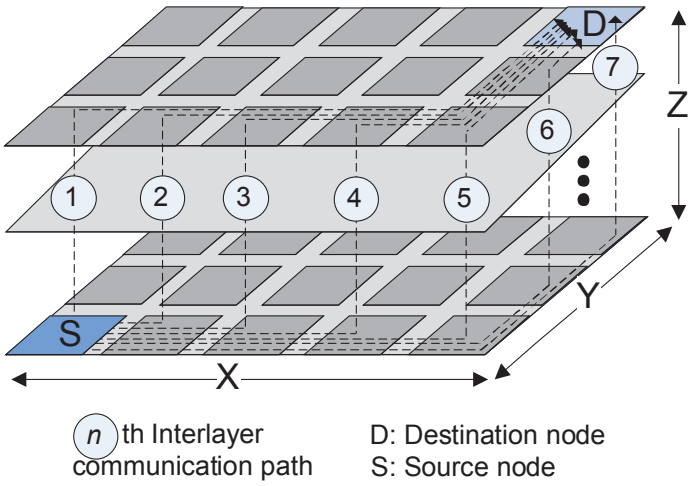


Figure 7.2: Example of *AdaptiveZ* routing

number of the requests pending on the bus. In this work, the wait signal is deasserted when the number of pending requests are more than $\lfloor n/2 \rfloor$. However, other criteria to calculate the stress value of the bus can be used. Once, the packet is in the destination layer ($Z_{diff} = 0$), normal XY (or any other) routing algorithm is used. The packet is delivered to the destination node, once $X_{diff} = Y_{diff} = Z_{diff} = 0$. Thus, in Figure 7.2 vertical link #1 has the highest probability to serve the packet under consideration and link #7 has the lowest probability to be used for routing of the current packet. Routing in the vertical dimension is done adaptively by considering the load on vertical links for load balancing purposes. This was done because the bus is a shared medium and adaptive vertical link communication can increase the bus utilization. Note that if the output buffer existed, it would not be possible to bring back the packet and re-route it.

This solution is not deadlock free due to adaptivity. To deal with deadlock, typical virtual channel architecture is used as shown in Figure 7.3. Consider that the input buffer of router R_{001} contains a packet, which needs to be transmitted to the neighboring node R_{101}. Router R_{101} is already waiting for the availability of dTDMA bus to deliver a packet at the node R_{100}. Also, consider that the node R_{000} and R_{100} have the same situations. In case of typical stacked mesh architecture with adaptive vertical routing and without virtual channel, there will be a deadlock. In the proposed architecture, with the same number of buffers as compared to stacked mesh with dTDMA bus architecture [127], we modify the inter-layer communication scheme to support the VC architecture. The output buffer from the router for bus communication was removed. An extra input buffer is used to receive the data packets and support the VC concept. There is very small

ALGORITHM 5: *AdaptiveZ* Routing Algorithm

Input: $(X_{current}, Y_{current}, Z_{current})$, $(X_{destination}, Y_{destination}, Z_{destination})$, *wait*
Output: Next Hop (E, W, N, S, L, U/D)

```
1: X_diff = X_current − X_destination;
2: Y_diff = Y_current − Y_destination;
3: Z_diff = Z_current − Z_destination;
4: if (X_diff = Y_diff = Z_diff = 0) then
5:     Deliver the packet to the local node and exit;
6: end if
7: if (Z_diff = 0) then
8:     Use a 2D intra-layer routing algorithm;
9: else if (X_diff = Y_diff = 0) then
10:    Send a request to the bus arbiter along with the destination layer ID;
11:    Wait until receiving the grant;
12: else if (X_diff ≠ 0 or Y_diff ≠ 0) then
13:    Send a request to the bus arbiter along with the destination layer ID;
14:    if (wait = '1') {the destination is free, but the bus is busy} then
15:        Wait until receiving the grant;
16:    else {the destination is busy}
17:        Withdraw the request;
18:        Use a 2D intra-layer routing algorithm;
19:    end if
20: end if
```

area overhead of one 2×1 multiplexer, one 1×2 demultiplexer and few signaling wires. In addition, the added VC not only avoids deadlock, but it also improves the throughput. Now the intermediate buffers are used in a more efficient way and enhance the system power and performance characteristics. The reduction in power consumption is achieved due to removal of the buffer from the traversal path of the packet and routing adaptivity.

The detailed stacked mesh architecture is shown in Figure 7.4. The bus arbiter receives the bus request from different nodes by *req* signal along with destination layer ID (*dest_layerID*). The bus arbiter deals with bus request by considering the different parameters like stress value (indicating availability of buffer slots) of VC buffers, ongoing transactions and the transactions in the queue waiting for the bus grant. We use local information which is the current queue length of the corresponding VCs connected to the target router to generate the stress value. Depending on the situation, the bus arbiter generates *grant* and *wait* signals. If the node receives a *grant* signal, then it will proceed with the transaction. Instead, if it receives a *wait* signal, the node will continue to wait for the *grant* signal and the bus arbiter will put the request in the queue. If no *grant* and no *wait* signals are received, the node will withdraw the request and move the packet within the layer using any of the 2D routing algorithms. On the next node, again same procedure will be repeated. On the same node, if the bus is granted for vertical communication, the packet will be delivered to the intermediate node in the destination layer. If the packet does not get any bus access on

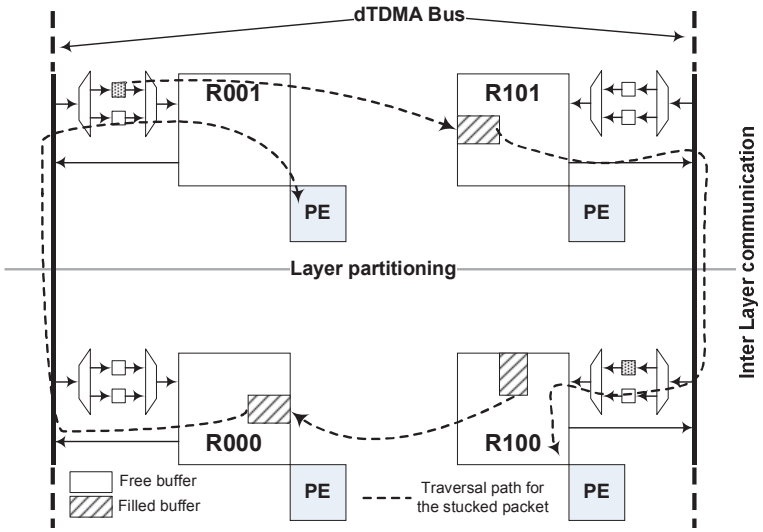


Figure 7.3: VC architecture for the stacked mesh architecture

its way for vertical communication until it reaches exactly below or above the destination node, it will wait there without any further routing to get the bus arbitration and will be delivered directly to the destination node. This is the situation for path #7 in Figure 7.2.

7.3 Inter-layer Fault Tolerant Routing

In 3D integration, the silicon vias provide communication links for dies in the vertical direction which is a critical design issue. Like other physical components, the fabrication and bonding of TSVs can fail. A failed TSV may cause a number of stacked known-good-dies to be discarded. As the number of dies to be stacked increases, the failed TSVs increase the manufacturing cost and decrease the yield [128]. A reliable inter-layer communication scheme can considerably cope with these issues. In this section, we explore how the available signals can enable the routing algorithm to avoid these paths (faulty buses) when there are other available paths between the source and destination pair. In this work, we assume that all intra-layer communication links are free from faults.

The proposed dynamic routing approach can tolerate a single link failure by utilizing the available communication resources. In this section, we enhance our proposed *AdaptiveZ* algorithm to deal with fault tolerance for inter-layer communication of stacked mesh architecture. Similar to the EDXY routing algorithm, the existing signaling used for adaptive inter-layer communication is reused to achieve fault tolerance without adding any extra

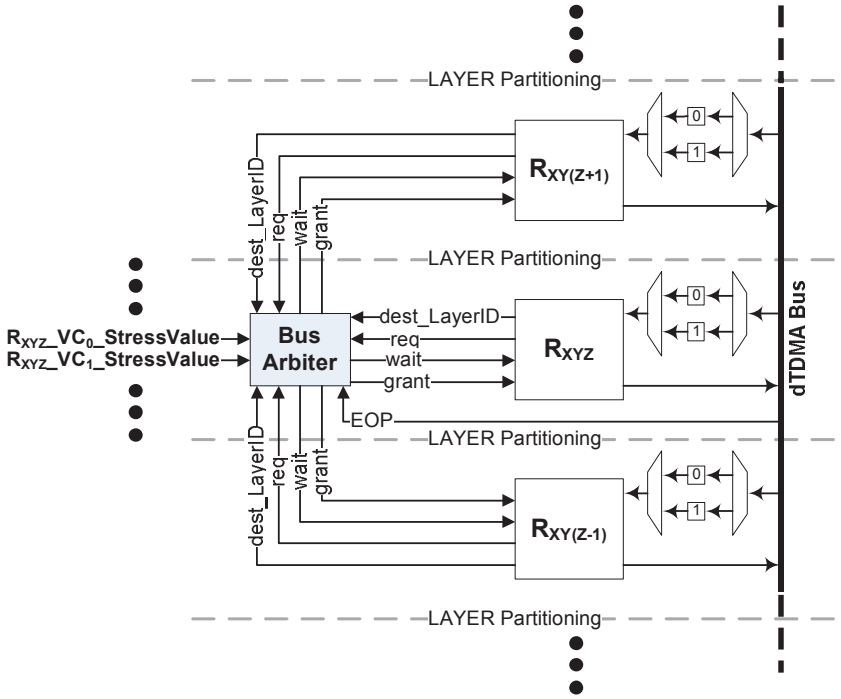


Figure 7.4: Side view of the proposed stacked mesh architecture

wires. As explained in Section 7.2, *wait* signal acts as a congestion flag in normal conditions. In case of any fault on the dTDMA bus, *wait* signal is permanently asserted to '0' and bus arbiter does not serve any request raised by *req* signal. Thus, routing adaptivity is not affected by introducing the fault tolerance in the existing architecture.

The proposed architecture employs Cyclic Redundancy Check (CRC) codes to detect possible faults in a received flit that may have been corrupted. Before traversing the bus towards the input of the downstream intermediate buffer, each flit is error encoded at the upstream router using payload bits to store a CRC checksum for error detection. The CRC characteristic polynomial that has been used in this work is $g(x) = x^8 + x^2 + x + 1$. The polynomial adds 8 check bits to flit data. The same CRC code has been used in the Header Error Control of ATM networks, and is able to correct single-bit errors (we do not make use of this correction capability) and detect many multiple bit errors [130]. If the error is detected in received flit, the bus arbiter asserts '0' on the *wait* signal. When router checks the *wait* signal, it will re-route the packets within the layer and in future, will not drive any traffic for inter-layer communication to the corresponding dTDMA bus. The minimal path routing will be used even in the case of faulty links without any modification in the proposed routing algorithm. However there are the

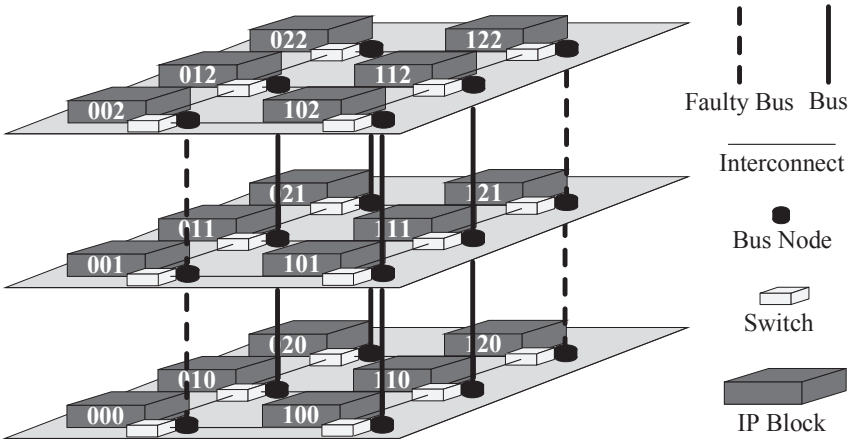


Figure 7.5: Example of modifications to the proposed routing algorithm to guarantee single bus-link failure tolerance

cases, when routing mechanism needs to be modified to follow a non-minimal path to reach the destination node.

Consider the node '000' in Figure 7.5 which needs to send a packet to node '112'. The vertical links on two paths are faulty as shown in Figure 7.5. According to the proposed routing algorithm in Section 7.2, the vertical dTDMA bus connecting the nodes '000', '001' and '002' should be tried first, but that bus link is faulty in the current situation. So, the packet will be routed to either of the nodes '010' or '100' according to the routing algorithm.

We route the packet via non-minimal path through one of the neighboring nodes of the current node, when both the current and the destination nodes are connected to the same faulty pillar. Consider the situation that the packet is exactly below or above the destination node. In this case, it cannot be delivered to the destination if the bus link is faulty. So, the packet will be rerouted to the neighboring node within the layer, where the packet is currently residing and then will be delivered to the destination. This is the situation, when the node '120' contains a packet for node '122'. For the proposed routing algorithm, the packet can only be routed through the vertical dTDMA bus connecting the nodes '120', '121' and '122', which is faulty. In this situation, the packet will be routed to the nodes '110' or '020' via a non-minimal path using one of the neighboring pillars.

This is especially important in future 3D integration in which the failure rate is high. If the interlayer channels are fabricated correctly, the added wires used to reduce network latency, can guarantee packet transmission in the vertical dimension in the case of a bus link failure.

7.4 Simulation Results

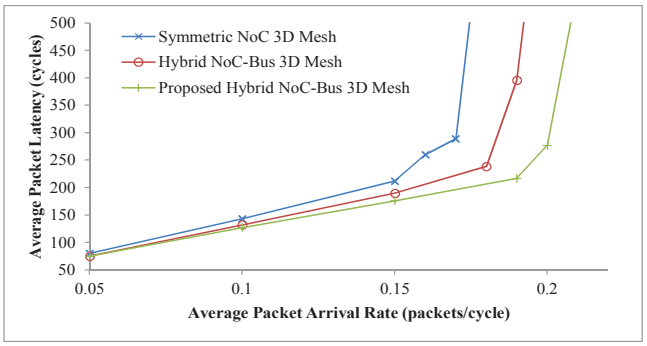
A cycle-accurate NoC simulation environment was implemented in VHDL to simulate the proposed techniques and compared its performance with existing 3D-NoC architectures. Symmetric 3D-mesh NoC [131], Hybrid 3D NoC-Bus mesh [127], and the proposed architecture using *AdaptiveZ* routing algorithm was analyzed for synthetic and realistic traffic patterns. Static ZXY routing was used for Symmetric and Hybrid 3D NoC-Bus mesh. For the proposed architecture, *AdaptiveZ*($\widetilde{Z}$XY) routing was used.

7.4.1 Synthetic Traffic Analysis

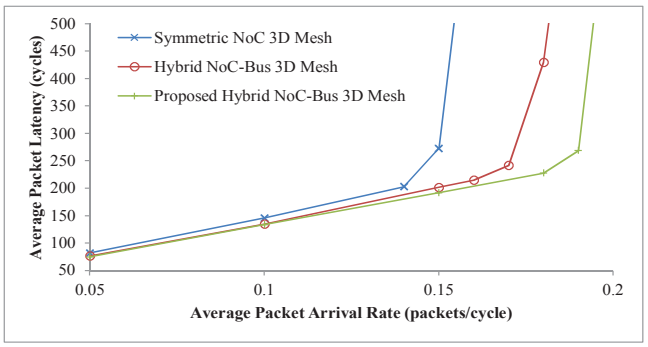
To perform the simulations under synthetic traffic profiles, we used uniform, hotspot 10%, and Negative Exponential Distribution (NED) [132] [133] traffic patterns. For each simulation, the packet latencies were averaged over 50,000 packets. Latencies were not collected for the first 5,000 cycles to allow the network to stabilize. It was assumed the buffer size of each FIFO was eight flits, and the data width was set to 64 bits.

In the first experiment set, 3×3×3 3D meshes and packets with a length of seven-flits were used. The average packet latency (APL) curves for uniform, hotspot 10% and NED traffic patterns with varying average packet arrival rates (APAR) are shown in Figure 7.6(a), 7.6(b), and 7.6(c), respectively. For the hotspot 10% traffic pattern, the node at (2, 2, 2) is destined as the hotspot node. It can be observed for all the traffic patterns, that the network with proposed architecture saturates at higher injection rates. The reason being that the *AdaptiveZ* routing for inter-layer communication increases the bus utilization and makes the load, balanced. In the case of static ZXY routing, the Hybrid 3D NoC-Bus architecture cannot deliver the desired throughput because of bandwidth limitations. For the proposed architecture, bandwidth limitations are managed by proper buffer utilization without increasing the communication resources. For NED traffic, the throughput curves show a significant difference in performance as shown in Figure 7.6(c).

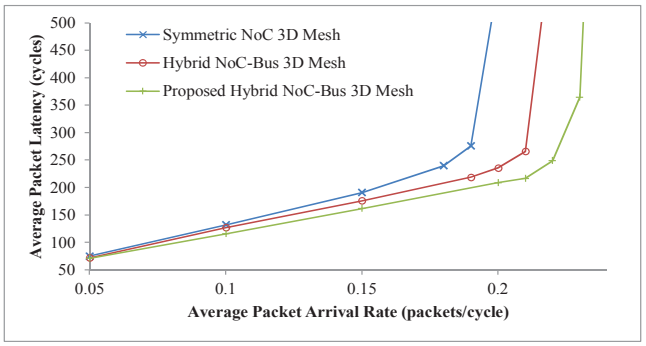
In the second experiment set, we changed some of the NoC parameters considered in the first set of experiments. For the simulations, 3×3×4 3D meshes and packets with a length of ten-flits were used. The nodes at (2, 2, 2) and (2, 2, 3) are chosen to receive more packets for the hotspot 10% traffic pattern. The packet latency for uniform, hotspot 10% and NED traffic patterns with varying APAR are shown in Figure 7.7(a), 7.7(b), and 7.7(c), respectively. For this experiment set, our results also reveal improved average packet latency for varying average packet arrival rates compared to the typical Hybrid 3D NoC-Bus mesh and the Symmetric 3D-mesh NoC.



(a) Under uniform traffic profile



(b) Under hotspot 10% traffic profile

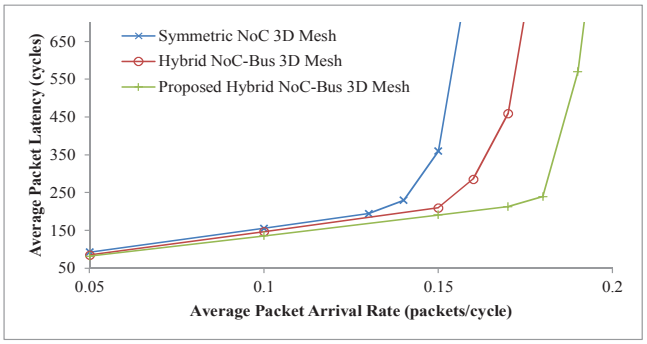


(c) Under NED traffic profile

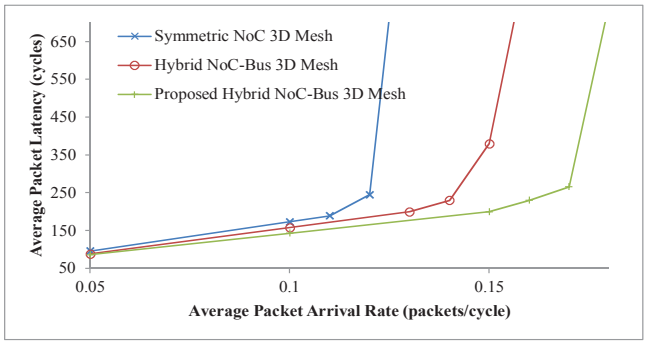
Figure 7.6: Latency versus average packet arrival rate for a 3×3×3 NoC

7.4.2 Videoconference Application

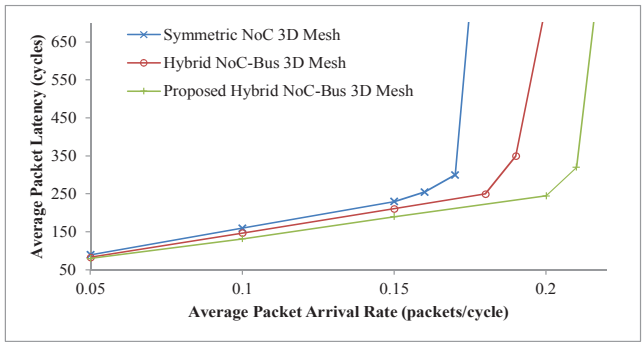
For realistic traffic analysis, we used the encoding part of videoconference application with sub-applications of H.264 encoder, MP3 encoder and OFDM transmitter presented in Chapter 5. The video stream used for simulation



(a) Under uniform traffic profile



(b) Under hotspot 10% traffic profile



(c) Under NED traffic profile

Figure 7.7: Latency versus average packet arrival rate for a 3×3×4 NoC

purposes was of size 300×225 pixels and each pixel consists of 24 bits. Thus each video frame is composed of 1.62 Mbits and can be broken into 8400 data packets each of size 7 flits including the header flit. The data width was set to 64 bits. We modeled the application graph, mapping strategy, frame rate, buffer size, number of nodes, layers and generated packets, supply voltage

and clock frequency used in Chapter 5 for the real application simulation. The application that is mapped to 3×3×3 3D-mesh NoC is shown in Figure 5.12.

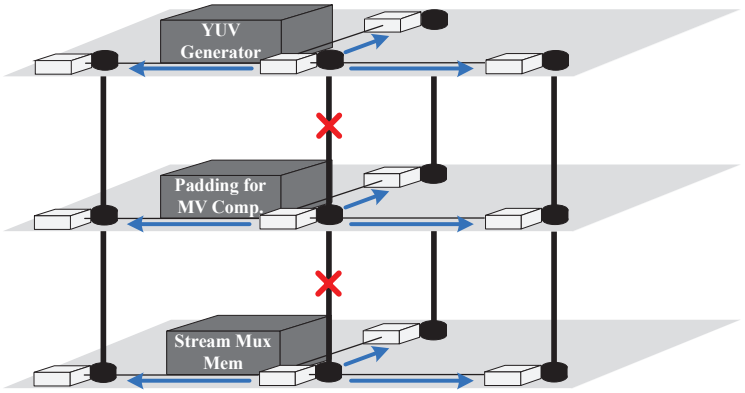


Figure 7.8: 3D NoC running the video conference encoding application with one faulty bus

Table 7.1: Power Consumption and Average Packet Latency

3D NoC Architecture	**Power Consumption (W)**	**Average Packet Latency (cycles)**
Symmetric NoC 3D Mesh	1.587	186
Hybrid 3D NoC-Bus Mesh	1.439	166
Proposed Hybrid 3D NoC-Bus Mesh (*AdaptiveZ* routing)	1.382	153
Proposed Hybrid 3D NoC-Bus Mesh with 1 faulty bus (*AdaptiveZ* routing)	1.491	179

To estimate the power consumption, we used the high level NoC power simulator presented in [80] to support the 3D NoC architectures. The simulation results for power and performance of the videoconference encoding application are shown in Table. 7.1. The proposed architecture using the *AdaptiveZ* routing algorithm showed 13% and 4% drop in power consumption over the Symmetric 3D-mesh NoC and Hybrid 3D NoC-Bus mesh architectures, respectively. Similarly, 18% and 8% reduction in APL over the Symmetric 3D-mesh NoC and Hybrid 3D NoC-Bus mesh architectures was also observed for our proposed architecture. Note that the proposed technique only deals with inter-layer communication, but these values are calculated for the whole system. It could be deduced that the proposed architecture has considerably optimized the inter-layer communication with negligible area overheads, presented in Table 7.2. Note that for all the routers, the data width and buffer depth were set to 32-bit and 8, respectively. The figures

given in the table reveal that the area overheads of the proposed routing units and the bus control module are negligible.

Table 7.2: Power Consumption and Average Packet Latency

Component	**Area** (μ**m**^{2}) for Typical Architecture	**Area** (μ**m**^{2}) for AdaptiveZ routing
6-port router	40127	41256
6-Port Router with 2 VCs	92194	93591
Bus arbiter for a three-layer NoC	267	694

As discussed in Section 7.3, if a fault occurs on a vertical bus, the proposed architecture can tolerate the fault and packets will be re-routed to one of neighboring nodes. To demonstrate that, the system running the video-conference (Figure 5.12) with one faulty vertical bus was simulated. The rest of the system parameters were kept unchanged. As shown in Figure 7.8, we assumed that the bus connecting *Stream Mux Mem*, *Padding for MV Computation*, and *YUV Generator* components is faulty because there is a high inter-layer communication via this bus. Thus, the arbiter of the faulty bus asserts '0' on the *wait* signal and the connected routers re-route the packets within the layer and do not drive any traffic for inter-layer communication to this bus. The power consumption and average packet latency of the proposed architecture using the *AdaptiveZ* routing with one faulty bus are shown in Table 7.1 as well. As mentioned, the proposed architecture can tolerate a single faulty bus using the congestion signal triggered by the bus arbiter with a negligible latency and power overhead.

7.5 Summary

This chapter presented a novel inter-layer communication scheme for Hybrid 3D NoC-Bus Mesh architecture. The proposed technique enhanced the system performance, reduced power consumption, and improved the system reliability. To this end, a congestion aware adaptive inter-layer communication algorithm was introduced. To deal with deadlock, an appropriate VC architecture was used with same number of buffers as compared to the existing stacked mesh architectures. In addition, the congestion signal triggered by the bus arbiter was used to deal with fault tolerance. The effectiveness of the proposed architecture regarding average packet latency and power consumption has been demonstrated by experimental results using the synthetic as well as the realistic traffic loads. At congested locations, the simulation results for proposed architecture and routing scheme showed lower latencies compared to other architectures.

Chapter 8

Conclusions

More and more computation power is a key demand from computer industry. To address the issue, there are two options: increase the system clock speed or increase the number of computation resources. Increasing the clock speed shows direct and positive impact on software performance without any change in software code but it increases the power consumption density which raises the thermal problems. Thus, industry opted for the second option, which recommended to integrate more number of computation units while operating at same frequencies. A sequential application which has been designed to run on a single core, cannot gain any performance improvement by running on a multi-core system unless an efficient application partitioning and mapping technique is used and an efficient inter-core communication platform is available. These two multi-core design steps are interlinked and face the scalability problems.

An efficient application mapping technique for MPSoC platforms with homogeneous computation nodes has been presented. The mapping approach has been divided into two steps. In the first step, platform independent task prioritization criteria is defined and the cores with higher communication demands are prioritized over less demanding cores. In the next step, platform specific mapping technique is presented and cores are mapped according to the defined priority order while minimizing the communication cost. The simulation results have shown that reducing the communication cost improves the system performance in terms of both the communication time and the power consumption.

The second dimension for MPSoC design is an efficient on-chip communication platform. Traditional bus-based architecture is an efficient solution for the applications, where most of the tasks generate identical copy of data for multiple tasks (multicast). On other hand, buses can support the limited bandwidth and cannot support the connectivity for larger number of cores because of high throughput requirements and power consumption over-

heads. Segmented bus (*SegBus*) has been proposed to address the problems of bus architectures. To further enhance the performance of *SegBus* based system in terms of the communication time and power consumption, different communication services are introduced including multicast and interrupt transactions. Unfortunately, scalability becomes an issue for *SegBus* because of significant increase in fan-out and critical path length of the logic with multiplexer size.

To address the scalability of on-chip communications, Network-on-Chip (NoC) has already been proposed. The increase in performance of NoC based system comes with an increased power-density and reliability issues. The proposed partial virtual channel sharing NoC (PVS-NoC) architecture mitigates these issues by enhancing the resource utilization of virtual channel buffers with minimal overheads. The PVS-NoC architecture is tolerant of routing logic faults. In addition, it reduces the impact of faults on system performance by utilizing the components which have been indirectly affected by faults on other components. The PVS-NoC is further enhanced to address the processing element protection and reduce the network load by pre-routing of packets. The H.264 video encoder was simulated for *SegBus* and NoC based communications platforms. The simulation results for inter-task communication showed that NoC can provide better throughput performance as compared to *SegBus* at the cost of extra power consumption.

In case of 2D-NoC, the value of average hop-count for data packets is higher, which directly affects the congestion, power consumption and latency. To address all these issues, 3D-NoC has been proposed. In typical 3D-NoC architecture, 7-port router is used which shows significant overheads in terms of silicon area and power consumption as compared to the 5-port 2D-NoC router. To address this issue, an efficient 3-D NoC-Bus hybrid architecture was proposed which uses bus based architecture for inter-layer communications. To address the bandwidth limitations of bus, AdaptiveZ routing is introduced which is a minimal routing with static intra-layer but adaptive inter-layer communication. The simulation results for video encoder application (comprising of four sub-applications: H.264 video encoder, MP3 encoder, Audio/Video multiplexer and OFDM transmitter) showed that the proposed NoC-Bus hybrid architecture is an optimal trade-off of average packet latency and power consumption as compared to the typical 2-D and 3-D NoC architectures.

The presented design space exploration for MPSoC design enables more appropriate design choices for given design constraints. It leads the system designer to avoid performance overheads by eliminating the unnecessary performance beyond the requirements.

8.1 Future Works

The future of computing industry is the integration of heterogeneous and mobile services in smart environments. Thus, a unified application mapping approach is an immediate requirement, which can generate the task allocation for applications, running partially on heterogeneous interconnection platforms simultaneously while performing the multi-objective optimizations like reducing the inter-task communications time and making the system thermal efficient. The next future direction should be a simulation environment for smart environments comprising of real application benchmarks and heterogenous computing platforms with a variety of task allocation techniques.

Bibliography

[1] Krste Asanovic et al. *The Landscape of Parallel Computing Research: A View from Berkeley.* Technical report, U.C.Berkeley, 2006.

[2] N. D. Liveris and P. Banerjee. *Power aware interface synthesis for bus-based SoC designs.* Design, Automation and Test in Europe Conference and Exhibition , 2004, pp.864-869.

[3] C. Addo-Quaye. *Thermal-aware mapping and placement for 3-D NoC designs.* In Proceedings of IEEE International SOC Conference, 2005, pp.25-28.

[4] Regan, Keith. *Intel Puts 4 GHz Processor on Back Burner.* E-Commerce Times, Oct 15, 2004. Online at http://www.ecommercetimes.com/story/37360.html

[5] *Texas Instruments.* http://www.ti.com/

[6] *Intel Corporation.* http://www.intel.com/

[7] *Tilera Corporation.* http://www.tilera.com/

[8] D. Brooks, R. P. Dick, R. Joseph, L. Shang. *Power, Thermal, and Reliability Modeling in Nanometer-Scale Microprocessors.* IEEE Micro, vol. 27, no. 3, 2007, pp.49-62, doi:10.1109/MM.2007.58

[9] B. Clifford Neuman. *Scale in Distributed Systems.* In Casavant, T. and Singhal, M. (eds.), Readings in Distributed Computing Systems, IEEE Computer Society Press, 1994, pp.463-489.

[10] Liang Guang. *Hierarchical Agent-based Adaptation for Self-Aware Embedded Computing Systems.* Ph.D. thesis. University of Turku, Finland. December 2012.

[11] D.L. Eager, J. Zahorjan, E.D. Lazowska. *Speedup versus efficiency in parallel systems.* IEEE Transactions on Computers, vol.38, no.3, 1989, pp.408-423.

[12] David P. Rodgers. *Improvements in multiprocessor system design.* In Proceedings of the 12th annual International Symposium on Computer Architecture (ISCA), IEEE Computer Society Press,1985, pp.225-231.

[13] M.B. Taylor. *Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse.* In proceedings of 49th ACM/EDAC/IEEE Design Automation Conference (DAC), 2012, pp.1131-1136.

[14] L. Olvera et al. *Translator training and modern market demands.* En Perspectives Studies in Translatology. Vol. 13, no. 2, 2005, pp.132-142.

[15] *Accelerating Product Time to Market.* Industry Insights from Kodak, 2009.

[16] Keutzer, K.; Newton, A.R.; Rabaey, J.M.; Sangiovanni-Vincentelli, A. *System-level design: orthogonalization of concerns and platform-based design.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol.19, no.12, 2000, pp.1523-1543.

[17] *International Technology Roadmap for Semiconductors (ITRS).* 2010.

[18] Andrew S. Tanenbaum, Maarten van Steen *Distributed Systems: Principles and Paradigms (2nd Edition).* Prentice-Hall, Inc., 2006.

[19] Singh, H.; Ming-Hau Lee; Guangming Lu; Kurdahi, F.J.; Bagherzadeh, N.; Chaves Filho, E.M. *MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications.* IEEE Transactions on Computers, vol.49, no.5, 2000, pp.465-481.

[20] Gokhale, M.; Cohen, J.; Yoo, A.; Miller, W.M.; Jacob, A.; Ulmer, C.; Pearce, R. *Hardware Technologies for High-Performance Data-Intensive Computing.* Computer, IEEE Computer Society, vol.41, no.4, 2008, pp.60-68.

[21] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, and Radu Marculescu. *On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches.* ACM Trans. Des. Autom. Electron. Syst. vol.12, no.3, article no.23, 2008, pp.23:1-23:20.

[22] Camille Jalier, Didier Lattard, Ahmed Amine Jerraya, Gilles Sassatelli, Pascal Benoit, and Lionel Torres. *Heterogeneous vs homogeneous MPSoC approaches for a mobile LTE modem.* In Proceedings of the Design, Automation and Test in Europe (DATE), 2010, pp.184-189.

[23] Ethiopia Enideg Nigussie. *Exploration and Design of High Performance Variation Tolerant On-Chip Interconnects.* Ph.D. thesis. University of Turku, Finland. August 2010.

[24] A.W. Topol et al. *Three-dimensional integrated circuits.* IBM Journal of Research and Development, 50(4), 2006, pp.491-506.

[25] Pavlidis, V.F.; Friedman, E.G. *3-D Topologies for Networks-on-Chip.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.15, no.10, 2007, pp.1081-1090.

[26] A.M.Rahmani, K. Latif, P. Liljeberg, J. Plosila, H. Tenhunen. *Research and Practices on 3D Networks-on-Chip Architectures.* In proceedings of 28th IEEE Norchip Conference, 2010, ISBN: 978-1-4244-8971-8.

[27] A. Y. Weldezion, M. Grange, D. Pamunuwa, Z. Lu, A. Jantsch, R. Weerasekera and H. Tenhunen. *Scalability of network-on-chip communication architecture for 3-D meshes.* In Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS), 2009, pp.114-123.

[28] *Stratix III Device Handbook* 2007, Altera.

[29] *International Technology Roadmap For Semiconductors.* 2005 Edition. Design.

[30] Modelsim, http://www.model.com.

[31] J.-L. Dekeyser et al. *Model Driven Engineering for Regular MPSOC Co-design.* Reconfigurable Communication-centric SoCs (ReCoSoC), 2006.

[32] G. de Jong. *A UML-Based Design Methodology for Real-Time and Embedded Systems.* Design, Automation and Test in Europe Conference and Exhibition, 2002, pp.776-778.

[33] K. Lahiri, A. Raghunathan, S. Dey. *Design Space Exploration for Optimizing On-Chip Communication Architectures.* IEEE Transactions on Computer-aided Design og Integrated Circuits and Systems, VOL. 23, NO. 6, 2004, pp.952-961.

[34] V. Leppänen, T. Seceleanu, O. Nevalainen. *Communication Scheduling for the SegBus Platform.* Proceedings of the IEEE International SOC Conference (SOCC), Sept. 2007.

[35] I.E.G. Richardson, J. E. Richardson H.264 and MPEG-4 Video Compression. Wiley, John & Sons, Inc. October 2003.

[36] T. Seceleanu. *The SegBus Platform - Architecture and Communication Mechanisms.* Journal of Systems Architecture (2006), doi:10.1016/j.sysarc.2006.07.002

[37] S. Srinivasan, L. Li, N. Vijaykrishnan. *Simultaneous Partitioning and Frequenct Assignment for On-chip Bus Architectures.* In *Proceedings of Design Automation andd Test in Europe Conference*, 2005, pp.218-223.

[38] A.D. Swaminathan, T. Seceleanu. *Interrupt Communication on the SegBus platform.* Proceedings of the IEEE International System on-chip Conference, Austin, TX, USA, 2006, pp.229-232.

[39] D. Truscan et. all. T.Seceleanu, H. Tenhunen, J. Lilius. *A Model-Based Design Process for the SegBus Distributed Architecture.* 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 2008, pp.307-316.

[40] Junyan Tan; Fresse, V.; Rousseau, F. *Generation of emulation platforms for NoC exploration on FPGA.* 22nd IEEE International Symposium on Rapid System Prototyping (RSP), 2011, pp.186-192.

[41] Ye Lu; McCanny, J.; Sezer, S. *Generic Low-Latency NoC Router Architecture for FPGA Computing Systems.* International Conference on Field Programmable Logic and Applications (FPL), 2011, pp.82-89.

[42] *Stratix III Device Handbook* 2007, Altera.

[43] Modelsim, http://www.model.com.

[44] ARM AMBA Specification and Multilayer AHB Specification (rev 2.0). www.arm.com.

[45] C. Anderson, J.-L. Baer. *Two techniques for improving performance on bus-based multiprocessors.* HPCA '95: Proceedings of the 1^{st} IEEE Symposium on High-Performance Computer Architecture, 1995, pp.256-275.

[46] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone and A. Sangiovanni-Vincentelli. *Metropolis: An Integrated Electronic System Design Environment* Computer, vol. 36, no. 4, 2003, pp.45-52.

[47] A. Bakshi, V. K. Prasanna and A. Ledeczi. *MILAN: A Model Based Integrated Simulation Framework for Design of Embedded Systems* SIGPLAN Not., vol. 36, no. 8, 2001, pp.82-93.

[48] F. Baslett, T. Jermoluk, and D. Solomon. *The 4D-MP Graphics Superworkstataion: Computing+Graphics=40MIPS+40MFLOPS and 100,000*

Lighted Polygons per Second In Proceedings of the COMPCON'88, 1988, pp.468-471.

[49] S. Cho, G. Lee *Reducing coherence overhead in shared-bus multiprocessors* Euro-Par, Vol. II 1996. pp.492-497

[50] F. Dahlgren. *Boosting the Performance of Hybrid Snooping Cache Protocols* 22^{nd} Annual International Symposium on Computer Architecture (ISCA), 1995, pp.60-69.

[51] W. Dally. *Route packets, not wires: on-chip interconnection networks* DAC - Design Automation Conference, 2001, pp.684-689.

[52] M. Ebrahimi, M. Daneshtalab, M.H. Neishaburi, S. Mohammadi, A. Afzali-Kusha, J. Plosila and H. Tenhunen. *An Efficent Dynamic Multicast Routing Protocol for Distributing Traffic in NOCs* Design, Automation and Test in Europe, 2009, pp.1064-1069.

[53] K. Latif, M. Niazi, T. Seceleanu, H. Tenhunen, S. Sezer. *Application development flow for on-chip distributed architectures.* Proceedings of the 21^{st} IEEE International SoC Conference (SOCC), 2008, pp.163-168.

[54] Z. Lu. *Design and Analysis of On-Chip Communication for Network-on-Chip Platforms.* Ph.D. thesis. Royal Institute of Technology, March 2007.

[55] E. McCreight. *The Dragon Computer System: An Early Overview* Technical report, Xerox Corp., 1984.

[56] C. Nicopoulus, V. Narayanan, C.R. Das. *Network-on-Chip Architectures: A Holisitic Design Exploration.* Springer 2009, ISBN 978-90-481-3030-6.

[57] M. F. S. Oliveira, a. Eduardo W. Bri F. A. Nascimento, F. R. Wagner, *Model Driven Engineering for MPSoC Design Space Exploration* In *Proceedings of the 20^{th} annual conference on Integrated circuits and systems design*, ACM, 2007, pp.81-86.

[58] M. Papamarcos, J. Patel. *A Low Overhead Coherence Solution for Multiprocessors with Private Cache Memories* In *Proceedings of the 11^{th} Annual International Symposium on Computer Architecture*, 1984, pp.348-354.

[59] A. Pimentel, L. Hertzbetger, P. Lieverse, P. van der Wolf, E. Deprettere. *Exploring Embedded-systems Architectures with Artemis* Computer, vol. 34, no. 11, 2001, pp.57-63.

[60] F. A. Samman, T. Hollstein, M. Glesner. *Multicast parallel pipeline router architecture for network-on-chip.* In *Proceedings of Design, Automation and Test in Europe*, 2008, pp.1396-1401.

[61] A.D. Swaminathan, T. Seceleanu. *Interrupt Communication on the SegBus platform.* Proceedings of the IEEE International System on-chip Conference, 2006, pp.229-232.

[62] D. Truscan, T.Seceleanu, H. Tenhunen, J. Lilius. *A Model-Based Design Process for the SegBus Distributed Architecture.* 15^{th} Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 2008, pp.307-316.

[63] T. Seceleanu. *The SegBus Platform - Architecture and Communication Mechanisms.* Journal of Systems Architecture vol. 53, no. 4, 2007, pp.151-169. DOI= http://dx.doi.org/10.1016/j.sysarc.2006.07.002

[64] T. Seceleanu, V. Leppänen, O. Nevalainen. *Improving the Performance of Bus Platforms by Means of Segmentation and Optimized Resource Allocation.* EURASIP Journal on Embedded Systems, vol. 2009, Article ID 867362, 14 pages, 2009. doi:10.1155/2009/867362.

[65] T. Seceleanu, I. Crncovik, C. Seceleanu. *Transaction level control for application execution on the SegBus Platform.* In *Proceedings of the 33^{rd} Annual IEEE International Computer Software and Applications Conference (COMPSAC)*, 2009, pp.537-542. doi:10.1109/COMPSAC.2009.78

[66] C. Seceleanu, A. Vulgarakis, P. Pettersson, *REMES: A Resource Model for Embedded Systems.* In *Proceedings of the 14^{th} IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2009, pp.84-94.

[67] A. S. Tranberg-Hansen, J. Madsen. *A Service Based Component Model for Composing and Exploring MPSoC Platforms* In *Proceedings of IEEE International Symposium on Applied Sciences in Bio-Medical and Communication Technologies (ISABEL)*, 2008, pp.1-5.

[68] A. L. Varbanescu, H. Sips, and A. Van Gemund *PAM-SoC: A Toolchain for Predicting MPSoC Performance* Lecture Notes in Computer Science, vol. 4128 LNCS, 2006, pp.111-113.

[69] Y. Zhang, Z. Lu, A. Jantsch, L. Li, M. Gao. *Towards hierarchical cluster based cache coherence for large-scale network-on-chip.* In *Proceedings of the 4^{th} IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, 2009, pp.119-122.

[70] A. Pimentel, C. Erbas, and S. Polstra *A Systematic Approach to Exploring Embedded System Architectures at Multiple Abstraction Levels* IEEE Transactions on Computers, vol. 55, no. 2, 2006, pp. 99-112.

[71] Claas Cornelius, Hendrik Bohn, Dirk Timmermann *Service-oriented Approaches for the Operation of large on-chip Networks* 24th NORCHIP Conference, 2006, pp. 183-186, ISBN: 1-4244-0772-9

[72] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, Vijaykrishnan Narayanan, Mazin S. Yousif, and Chita R. Das. *A Novel Dimensionally-Decomposed Router for On-Chip Communication in 3D Architectures* In *Proceedings of the 34th annual International Symposium on Computer Architecture (ISCA)*, 2007, pp.138-149.

[73] Jongman Kim; Nicopoulos, C.; Dongkook Park; Narayanan, V.; Yousif, M.S.; Das, C.R. *A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks* In *Proceedings of the 33rd annual International Symposium on Computer Architecture (ISCA)*, 2006, pp. 4-15.

[74] Dally, W.J. *Virtual-channel flow control* IEEE Transactions on Parallel and Distributed Systems, vol.3, no.2, 1992, pp.194-205.

[75] Shekhar Borkar. *Designing reliable systems from unreliable components: The challenges of transistor variability and degradation.* IEEE Micro, vol.25, no.6, 2005, pp.10-16.

[76] A. Jantsch and H. Tenhunen (Eds.) *Networks on Chip.* Kluwer Academic Publishers, Boston; 2003. ISBN 1-4020-7392-5

[77] Rikard Thid, Ingo Sander, Axel Jantsch. *Flexible Bus and NoC Performance Analysis with Configurable Synthetic Workloads.* In *Proceedings of the 9th EUROMICRO Conference on Digital System Design*, 2006, pp.681-688.

[78] Alok Choudhary, Ian Foster, Rick Stevens. *Multimedia Applications and High-Performance Computing.* IEEE Parallel and Distributed Technology, vol. 3, no. 2, 1995, pp.2-3.

[79] Matlab/simulink, http://www.mathworks.com.

[80] G. Guindani et al. *NoC Power Estimation at the RTL Abstraction Level* In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2008, pp.475-478.

[81] Srinivasan Murali, Giovanni De Micheli *Bandwidth-Constrained Mapping of Cores onto NoC Architectures.* In *Proceedings of Design Automation and Test in Europe (DATE)*, 2004, pp.896-903.

[82] J.Hu, R.Marculescu *Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints.* In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2003, pp.233-239.

[83] Mikkel B. Stensgaard, Jens Sparsø. *ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology.* In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2008, pp.55-64.

[84] I. Walter, I. Cidon, and A. Kolodny, D. Sigalov. *The Era of Many-Modules SoC: Revisiting the NoC Mapping Problem* Second International Workshop on Network on Chip Architectures (NoCArc), 2010, pp.43-48.

[85] Tiberiu Seceleanu, Ville Leppänen, Olli S. Nevalainen. *Device allocation on the SegBus platform based on communication scheduling cost minimization.* Proceedings of the IEEE International SOC Conference (SOCC), 2007, pp.191-196.

[86] Umit Y. Ogras, Radu Marculescu. *"It's a small world after all": NoC performance optimization via long-range link insertion.* IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2006, pp.693-706.

[87] D. Truscan et. all. *A Model-Based Design Process for the SegBus Distributed Architecture.* 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 2008, pp.307-316.

[88] G. Ascia, V. Catania, M. Palesi. *Multi-objective mapping for mesh-based NoC architectures.* International Conference on Hardware/Software Codesign and System Synthesis (CODES + ISSS), 2004, pp.182-187.

[89] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi. *Online NoC Switch Fault Detection and Diagnosis Using a High Level Fault Model.* In *Proceedings of 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2007, pp.21-29.

[90] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th annual international conference on Supercomputing (ICS)*, 2006, pp.187-198.

[91] N. Banerjee, P. Vellanki, and K.S. Chatha. A power and performance model for network-on-chip architectures. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, volume 2, 2004, pp.1250-1255.

[92] Xuning Chen and Li-Shiuan Peh. Leakage power modeling and optimization in interconnection networks. In *Proceedings of International Symposium on Low Power Electronics and Design (ISLPED)*, 2003, pp.90-95.

[93] Caroline Concatto, Debora Matos, Luigi Carro, Fernanda Kastensmidt, Altamiro Susin, Erika Cota, and Marcio Kreutz. Fault tolerant mechanism to improve yield in NoCs using a reconfigurable router. In *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes (SBCCI)*, 2009, pp.26:1-6.

[94] C. Constantinescu. Trends and challenges in vlsi circuit reliability. IEEE Micro, vol. 23, no. 4, 2003, pp.14-19.

[95] T. Dumitras and R. Marculescu. On-chip stochastic communication [soc applications]. In *Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp.790-795.

[96] M. Koibuchi et al. A Lightweight Fault-Tolerant Mechanism for Network-on-Chip. In *International Symposium on Networks-on-Chip (NoCS)*, 2008, pp.13-22.

[97] D. Fick, A. DeOrio, G. Chen, V. Bertacco, D. Sylvester, and D. Blaauw. A highly resilient routing algorithm for fault-tolerant NoCs. In *Design, Automation Test in Europe Conference Exhibition*, 2009, pp.21-26..

[98] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester. Vicis: a reliable network for unreliable silicon. In *Proceedings of the 46th Annual Design Automation Conference (DAC)*, 2009, pp.812-817.

[99] Arthur Pereira Frantz, Fernanda Lima Kastensmidt, Luigi Carro, and Erika Cota. Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk. In *Test Conference, 2006. ITC '06. IEEE International*, 2006, pp.1-9.

[100] C. Grecu, A. Ivanov, R. Saleh, E.S. Sogomonyan, and Partha Pratim Pande. On-line fault detection and location for noc interconnects. In *Proceedings of 12th IEEE International On-Line Testing Symposium (IOLTS)*, 2006. doi: 10.1109/IOLTS.2006.44

[101] G. Guindani, C. Reinbrecht, T. Raupp, N. Calazans, and F.G. Moraes. NoC power estimation at the rtl abstraction level. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2008, pp.475-478.

[102] Donghyun Kim, Kangmin Lee, Se joong Lee, and Hoi-Jun Yoo. A reconfigurable crossbar switch with adaptive bandwidth control for networks-on-chip. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2005, pp.2369-2372.

[103] Ying-Cherng Lan, Shih-Hsin Lo, Yueh-Chi Lin, Yu-Hen Hu, and Sao-Jie Chen. BiNoC: A bidirectional NoC architecture with dynamic self-reconfigurable channel. In *Proceedings of 3rd ACM/IEEE International Symposium on Networks-on-Chip (NoCS)*, 2009, pp.266-275.

[104] K. Latif, A. M Rahmani, K.R. Vaddina, T. Seceleanu, P. Liljeberg, and H. Tenhunen. Enhancing performance of noc-based architectures using heuristic virtual-channel sharing approach. In *Proceedings of 35th IEEE Annual Computer Software and Applications Conference (COMPSAC)*, 2011, pp.442-447.

[105] T. Lehtonen, P. Liljeberg, and J. Plosila. Fault tolerance analysis of NoC architectures. In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS)*, 2007, pp.361-364.

[106] Teijo Lehtonen. On fault tolerance methods for networks-on-chip. *Ph.D. Thesis, University of Turku, Finland.* 2009.

[107] P. Lotfi-Kamran, A. M. Rahmani, M. Daneshtalab, A. Afzali-Kusha, and Z. Navabi. EDXY - A low cost congestion-aware routing algorithm for network-on-chips. *J. Syst. Archit.*, vol. 56, no. 7, 2010, pp.256-264.

[108] Giovanni De Micheli Luca Benini. *Networks On Chips: Technology And Tools.* Morgan Kaufmann Publishers, 2006.

[109] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of 31st Annual International Symposium on Computer Architecture*, 2004, pp.188-197.

[110] S. Murali, C. Seiculescu, L. Benini, and G. De Micheli. Synthesis of networks on chips for 3d systems on chips. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2009, pp.242-247.

[111] M.H. Neishaburi and Z. Zilic. ERAVC: Enhanced reliability aware NoC router. In *Proceedings of 12th International Symposium on Quality Electronic Design (ISQED)*, 2011, pp.1-6.

[112] M.H. Neishaburi and Zeljko Zilic. Reliability aware NoC router architecture using input channel buffer sharing. In *ACM Great Lakes Symposium on VLSI*, 2009, pp.511-516.

[113] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, vol. 26, no. 2, 1993, pp.62-76.

[114] C.A. et al. Nicopoulos. Vichar: A dynamic virtual channel regulator for network-on-chip routers. In *39th Annual IEEE/ACM International Symposium on Microarchitecture, (MICRO)*, 2006, pp.333-346..

[115] S. Pasricha and N. Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 2008.

[116] A.-M. Rahmani, K. Latif, K.R. Vaddina, P. Liljeberg, J. Plosila, and H. Tenhunen. Congestion aware, fault tolerant, and thermally efficient inter-layer communication scheme for hybrid NoC-bus 3D architectures. In *Fifth IEEE/ACM International Symposium on Networks on Chip (NoCS)*, 2011, pp.65-72.

[117] A.M. Rahmani, A. Afzali-Kusha, and M. Pedram. NED: A Novel Synthetic Traffic Pattern for Power/Performance analysis of Network-on-chips Using Negative Exponential Distribution. In *Journal of Low Power Electronics (American Scientific Publishers)*, vol. 5, 2009, pp.396-405.

[118] Rohit Sunkam Ramanujam, Vassos Soteriou, Bill Lin, and Li-Shiuan Peh. Design of a high-throughput distributed shared-buffer NoC router. In *Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 2010, pp.69-78.

[119] Jih-Sheng Shen, Chun-Hsian Huang, and Pao-Ann Hsiung. Learning-based adaptation to applications and environments in a reconfigurable network-on-chip. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010, pp.381-386.

[120] K. Rao Vaddina, E. Nigussie, P. Liljeberg, and J. Plosila. Self-timed thermal monitoring of multicore systems. In *12th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2009, pp.246-251.

[121] Mojtaba Valinataj, Siamak Mohammadi, Juha Plosila, Pasi Liljeberg, and Hannu Tenhunen. A reconfigurable and adaptive routing method for fault-tolerant mesh-based networks-on-chip. *AEU - International Journal of Electronics and Communications*, vol. 65, no. 7, 2011, pp.630-640.

[122] Brian Patrick Towles William J Dally. *Principles and Practices of Interconnection Networks*. The Morgan Kaufmann Series in Computer Architecture and Design, 2004.

[123] P.M. Yaghini, A. Eghbal, H. Pedram, and H.R. Zarandi. Investigation of transient fault effects in an asynchronous NoC router. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2010, pp.540-545.

[124] T.T. Ye, L. Benini, and G. De Micheli. Analysis of power consumption on switch fabrics in network routers. In *Proceedings of 39th Design Automation Conference (DAC)*, 2002, pp.524-529.

[125] Z. Zhang, A. Greiner, and S. Taktak. A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip. In *45th ACM/IEEE DAC Conference*, 2008, pp.441-446.

[126] A.E. Zonouz, M. Seyrafi, A. Asad, M. Soryani, M. Fathy, and R. Berangi. A fault tolerant NoC architecture for reliability improvement and latency reduction. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, 2009, pp.473-480.

[127] Li, F. and Nicopoulos, C. and Richardson, T. and Yuan X. and Narayanan, V. and Kandemir, M. *Proceedings of the 33rd International Symposium on Computer Architecture* Design and Management of 3D Chip Multiprocessors Using Network-in-Memory, 2006, pp.130-141.

[128] Hsieh, A.-C. and Hwang, T. and Chang, M.-. and Tsai, M.-H. and Tseng, C.-M. and Li, H.-C. *TSV redundancy: architecture and design issues in 3D IC* Proceedings of the Conference on Design, Automation and Test in Europe, 2010, pp.166-171.

[129] Lotfi-Kamran, P. and Rahmani, A.-M. and Daneshtalab, M. and Afzali-Kusha, A. and Navabi, Z. *EDXY - A low cost congestion-aware routing algorithm for network-on-chips* Journal of Systems Architecture, vol. 56, no. 7, 2010, pp.256-264.

[130] Koopman, P. and Chakravarty, T. *Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks* Proceedings of the International Conference on Dependable Systems and Networks, 2004, pp.145-154.

[131] Carloni, L.P. and Pande, P. and Xie, Y. *Networks-on-chip in emerging interconnect paradigms: Advantages and challenges* Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip, 2009, pp.93-102.

[132] Rahmani, A.-M. and Kamali, I. and Lotfi-Kamran, P. and Afzali-Kusha, A. and Safari, S. *Negative Exponential Distribution Traffic Pattern for Power/Performance Analysis of Network on Chips* Proceedings of the International Conference on VLSI Design, 2009, pp.157-162.

[133] Rahmani, A.-M. and Afzali-Kusha, A. and Pedram, M. *NED: A Novel Synthetic Traffic Pattern for Power/Performance Analysis of Network-on-Chips Using Negative Exponential Distribution* Journal of Low Power Electronics, vol. 5, no. 3, 2009, pp.396-405.

[134] Weldezion, A.Y.; Grange, M.; Pamunuwa, D.; Zhonghai Lu; Jantsch, A.; Weerasekera, R.; Tenhunen, H. *Scalability of network-on-chip communication architecture for 3-D meshes," Networks-on-Chip* 3rd ACM/IEEE International Symposium on NoCS, 2009, pp.114-123, doi: 10.1109/NOCS.2009.5071459

[135] Tanenbaum, Andrew S. *Modern operating systems.* Upper Saddle River, NJ: Pearson Prentice Hall. ISBN 0-13-600663-9.

Turku Centre for Computer Science
TUCS Dissertations

1. **Marjo Lipponen**, On Primitive Solutions of the Post Correspondence Problem
2. **Timo Käkölä**, Dual Information Systems in Hyperknowledge Organizations
3. **Ville Leppänen**, Studies on the Realization of PRAM
4. **Cunsheng Ding**, Cryptographic Counter Generators
5. **Sami Viitanen**, Some New Global Optimization Algorithms
6. **Tapio Salakoski**, Representative Classification of Protein Structures
7. **Thomas Långbacka**, An Interactive Environment Supporting the Development of Formally Correct Programs
8. **Thomas Finne**, A Decision Support System for Improving Information Security
9. **Valeria Mihalache**, Cooperation, Communication, Control. Investigations on Grammar Systems.
10. **Marina Waldén**, Formal Reasoning About Distributed Algorithms
11. **Tero Laihonen**, Estimates on the Covering Radius When the Dual Distance is Known
12. **Lucian Ilie**, Decision Problems on Orders of Words
13. **Jukkapekka Hekanaho**, An Evolutionary Approach to Concept Learning
14. **Jouni Järvinen**, Knowledge Representation and Rough Sets
15. **Tomi Pasanen**, In-Place Algorithms for Sorting Problems
16. **Mika Johnsson**, Operational and Tactical Level Optimization in Printed Circuit Board Assembly
17. **Mats Aspnäs**, Multiprocessor Architecture and Programming: The Hathi-2 System
18. **Anna Mikhajlova**, Ensuring Correctness of Object and Component Systems
19. **Vesa Torvinen**, Construction and Evaluation of the Labour Game Method
20. **Jorma Boberg**, Cluster Analysis. A Mathematical Approach with Applications to Protein Structures
21. **Leonid Mikhajlov**, Software Reuse Mechanisms and Techniques: Safety Versus Flexibility
22. **Timo Kaukoranta**, Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization
23. **Gábor Magyar**, On Solution Approaches for Some Industrially Motivated Combinatorial Optimization Problems
24. **Linas Laibinis**, Mechanised Formal Reasoning About Modular Programs
25. **Shuhua Liu**, Improving Executive Support in Strategic Scanning with Software Agent Systems
26. **Jaakko Järvi**, New Techniques in Generic Programming – C++ is more Intentional than Intended
27. **Jan-Christian Lehtinen**, Reproducing Kernel Splines in the Analysis of Medical Data
28. **Martin Büchi**, Safe Language Mechanisms for Modularization and Concurrency
29. **Elena Troubitsyna**, Stepwise Development of Dependable Systems
30. **Janne Näppi**, Computer-Assisted Diagnosis of Breast Calcifications
31. **Jianming Liang**, Dynamic Chest Images Analysis
32. **Tiberiu Seceleanu**, Systematic Design of Synchronous Digital Circuits
33. **Tero Aittokallio**, Characterization and Modelling of the Cardiorespiratory System in Sleep-Disordered Breathing
34. **Ivan Porres**, Modeling and Analyzing Software Behavior in UML
35. **Mauno Rönkkö**, Stepwise Development of Hybrid Systems
36. **Jouni Smed**, Production Planning in Printed Circuit Board Assembly
37. **Vesa Halava**, The Post Correspondence Problem for Market Morphisms
38. **Ion Petre**, Commutation Problems on Sets of Words and Formal Power Series
39. **Vladimir Kvassov**, Information Technology and the Productivity of Managerial Work
40. **Frank Tétard**, Managers, Fragmentation of Working Time, and Information Systems

41. **Jan Manuch**, Defect Theorems and Infinite Words
42. **Kalle Ranto**, Z_4-Goethals Codes, Decoding and Designs
43. **Arto Lepistö**, On Relations Between Local and Global Periodicity
44. **Mika Hirvensalo**, Studies on Boolean Functions Related to Quantum Computing
45. **Pentti Virtanen**, Measuring and Improving Component-Based Software Development
46. **Adekunle Okunoye**, Knowledge Management and Global Diversity – A Framework to Support Organisations in Developing Countries
47. **Antonina Kloptchenko**, Text Mining Based on the Prototype Matching Method
48. **Juha Kivijärvi**, Optimization Methods for Clustering
49. **Rimvydas Rukšėnas**, Formal Development of Concurrent Components
50. **Dirk Nowotka**, Periodicity and Unbordered Factors of Words
51. **Attila Gyenesei**, Discovering Frequent Fuzzy Patterns in Relations of Quantitative Attributes
52. **Petteri Kaitovaara**, Packaging of IT Services – Conceptual and Empirical Studies
53. **Petri Rosendahl**, Niho Type Cross-Correlation Functions and Related Equations
54. **Péter Majlender**, A Normative Approach to Possibility Theory and Soft Decision Support
55. **Seppo Virtanen**, A Framework for Rapid Design and Evaluation of Protocol Processors
56. **Tomas Eklund**, The Self-Organizing Map in Financial Benchmarking
57. **Mikael Collan**, Giga-Investments: Modelling the Valuation of Very Large Industrial Real Investments
58. **Dag Björklund**, A Kernel Language for Unified Code Synthesis
59. **Shengnan Han**, Understanding User Adoption of Mobile Technology: Focusing on Physicians in Finland
60. **Irina Georgescu**, Rational Choice and Revealed Preference: A Fuzzy Approach
61. **Ping Yan**, Limit Cycles for Generalized Liénard-Type and Lotka-Volterra Systems
62. **Joonas Lehtinen**, Coding of Wavelet-Transformed Images
63. **Tommi Meskanen,** On the NTRU Cryptosystem
64. **Saeed Salehi**, Varieties of Tree Languages
65. **Jukka Arvo**, Efficient Algorithms for Hardware-Accelerated Shadow Computation
66. **Mika Hirvikorpi**, On the Tactical Level Production Planning in Flexible Manufacturing Systems
67. **Adrian Costea**, Computational Intelligence Methods for Quantitative Data Mining
68. **Cristina Seceleanu**, A Methodology for Constructing Correct Reactive Systems
69. **Luigia Petre**, Modeling with Action Systems
70. **Lu Yan**, Systematic Design of Ubiquitous Systems
71. **Mehran Gomari**, On the Generalization Ability of Bayesian Neural Networks
72. **Ville Harkke**, Knowledge Freedom for Medical Professionals – An Evaluation Study of a Mobile Information System for Physicians in Finland
73. **Marius Cosmin Codrea**, Pattern Analysis of Chlorophyll Fluorescence Signals
74. **Aiying Rong**, Cogeneration Planning Under the Deregulated Power Market and Emissions Trading Scheme
75. **Chihab BenMoussa**, Supporting the Sales Force through Mobile Information and Communication Technologies: Focusing on the Pharmaceutical Sales Force
76. **Jussi Salmi**, Improving Data Analysis in Proteomics
77. **Orieta Celiku**, Mechanized Reasoning for Dually-Nondeterministic and Probabilistic Programs
78. **Kaj-Mikael Björk**, Supply Chain Efficiency with Some Forest Industry Improvements
79. **Viorel Preoteasa**, Program Variables – The Core of Mechanical Reasoning about Imperative Programs
80. **Jonne Poikonen**, Absolute Value Extraction and Order Statistic Filtering for a Mixed-Mode Array Image Processor
81. **Luka Milovanov**, Agile Software Development in an Academic Environment
82. **Francisco Augusto Alcaraz Garcia**, Real Options, Default Risk and Soft Applications
83. **Kai K. Kimppa**, Problems with the Justification of Intellectual Property Rights in Relation to Software and Other Digitally Distributable Media
84. **Dragoş Truşcan**, Model Driven Development of Programmable Architectures
85. **Eugen Czeizler**, The Inverse Neighborhood Problem and Applications of Welch Sets in Automata Theory

86. **Sanna Ranto**, Identifying and Locating-Dominating Codes in Binary Hamming Spaces
87. **Tuomas Hakkarainen**, On the Computation of the Class Numbers of Real Abelian Fields
88. **Elena Czeizler**, Intricacies of Word Equations
89. **Marcus Alanen**, A Metamodeling Framework for Software Engineering
90. **Filip Ginter**, Towards Information Extraction in the Biomedical Domain: Methods and Resources
91. **Jarkko Paavola**, Signature Ensembles and Receiver Structures for Oversaturated Synchronous DS-CDMA Systems
92. **Arho Virkki**, The Human Respiratory System: Modelling, Analysis and Control
93. **Olli Luoma**, Efficient Methods for Storing and Querying XML Data with Relational Databases
94. **Dubravka Ilić**, Formal Reasoning about Dependability in Model-Driven Development
95. **Kim Solin**, Abstract Algebra of Program Refinement
96. **Tomi Westerlund**, Time Aware Modelling and Analysis of Systems-on-Chip
97. **Kalle Saari**, On the Frequency and Periodicity of Infinite Words
98. **Tomi Kärki**, Similarity Relations on Words: Relational Codes and Periods
99. **Markus M. Mäkelä**, Essays on Software Product Development: A Strategic Management Viewpoint
100. **Roope Vehkalahti**, Class Field Theoretic Methods in the Design of Lattice Signal Constellations
101. **Anne-Maria Ernvall-Hytönen**, On Short Exponential Sums Involving Fourier Coefficients of Holomorphic Cusp Forms
102. **Chang Li**, Parallelism and Complexity in Gene Assembly
103. **Tapio Pahikkala**, New Kernel Functions and Learning Methods for Text and Data Mining
104. **Denis Shestakov**, Search Interfaces on the Web: Querying and Characterizing
105. **Sampo Pyysalo**, A Dependency Parsing Approach to Biomedical Text Mining
106. **Anna Sell**, Mobile Digital Calendars in Knowledge Work
107. **Dorina Marghescu**, Evaluating Multidimensional Visualization Techniques in Data Mining Tasks
108. **Tero Säntti**, A Co-Processor Approach for Efficient Java Execution in Embedded Systems
109. **Kari Salonen**, Setup Optimization in High-Mix Surface Mount PCB Assembly
110. **Pontus Boström**, Formal Design and Verification of Systems Using Domain-Specific Languages
111. **Camilla J. Hollanti**, Order-Theoretic Mehtods for Space-Time Coding: Symmetric and Asymmetric Designs
112. **Heidi Himmanen**, On Transmission System Design for Wireless Broadcasting
113. **Sébastien Lafond**, Simulation of Embedded Systems for Energy Consumption Estimation
114. **Evgeni Tsivtsivadze**, Learning Preferences with Kernel-Based Methods
115. **Petri Salmela**, On Commutation and Conjugacy of Rational Languages and the Fixed Point Method
116. **Siamak Taati**, Conservation Laws in Cellular Automata
117. **Vladimir Rogojin**, Gene Assembly in Stichotrichous Ciliates: Elementary Operations, Parallelism and Computation
118. **Alexey Dudkov**, Chip and Signature Interleaving in DS CDMA Systems
119. **Janne Savela**, Role of Selected Spectral Attributes in the Perception of Synthetic Vowels
120. **Kristian Nybom**, Low-Density Parity-Check Codes for Wireless Datacast Networks
121. **Johanna Tuominen**, Formal Power Analysis of Systems-on-Chip
122. **Teijo Lehtonen**, On Fault Tolerance Methods for Networks-on-Chip
123. **Eeva Suvitie**, On Inner Products Involving Holomorphic Cusp Forms and Maass Forms
124. **Linda Mannila**, Teaching Mathematics and Programming – New Approaches with Empirical Evaluation
125. **Hanna Suominen**, Machine Learning and Clinical Text: Supporting Health Information Flow
126. **Tuomo Saarni**, Segmental Durations of Speech
127. **Johannes Eriksson**, Tool-Supported Invariant-Based Programming

128. **Tero Jokela**, Design and Analysis of Forward Error Control Coding and Signaling for Guaranteeing QoS in Wireless Broadcast Systems
129. **Ville Lukkarila**, On Undecidable Dynamical Properties of Reversible One-Dimensional Cellular Automata
130. **Qaisar Ahmad Malik**, Combining Model-Based Testing and Stepwise Formal Development
131. **Mikko-Jussi Laakso**, Promoting Programming Learning: Engagement, Automatic Assessment with Immediate Feedback in Visualizations
132. **Riikka Vuokko**, A Practice Perspective on Organizational Implementation of Information Technology
133. **Jeanette Heidenberg**, Towards Increased Productivity and Quality in Software Development Using Agile, Lean and Collaborative Approaches
134. **Yong Liu**, Solving the Puzzle of Mobile Learning Adoption
135. **Stina Ojala**, Towards an Integrative Information Society: Studies on Individuality in Speech and Sign
136. **Matteo Brunelli**, Some Advances in Mathematical Models for Preference Relations
137. **Ville Junnila**, On Identifying and Locating-Dominating Codes
138. **Andrzej Mizera**, Methods for Construction and Analysis of Computational Models in Systems Biology. Applications to the Modelling of the Heat Shock Response and the Self-Assembly of Intermediate Filaments.
139. **Csaba Ráduly-Baka**, Algorithmic Solutions for Combinatorial Problems in Resource Management of Manufacturing Environments
140. **Jari Kyngäs**, Solving Challenging Real-World Scheduling Problems
141. **Arho Suominen**, Notes on Emerging Technologies
142. **József Mezei**, A Quantitative View on Fuzzy Numbers
143. **Marta Olszewska**, On the Impact of Rigorous Approaches on the Quality of Development
144. **Antti Airola**, Kernel-Based Ranking: Methods for Learning and Performace Estimation
145. **Aleksi Saarela**, Word Equations and Related Topics: Independence, Decidability and Characterizations
146. **Lasse Bergroth**, Kahden merkkijonon pisimmän yhteisen alijonon ongelma ja sen ratkaiseminen
147. **Thomas Canhao Xu**, Hardware/Software Co-Design for Multicore Architectures
148. **Tuomas Mäkilä**, Software Development Process Modeling – Developers Perspective to Contemporary Modeling Techniques
149. **Shahrokh Nikou**, Opening the Black-Box of IT Artifacts: Looking into Mobile Service Characteristics and Individual Perception
150. **Alessandro Buoni**, Fraud Detection in the Banking Sector: A Multi-Agent Approach
151. **Mats Neovius**, Trustworthy Context Dependency in Ubiquitous Systems
152. **Fredrik Degerlund**, Scheduling of Guarded Command Based Models
153. **Amir-Mohammad Rahmani-Sane**, Exploration and Design of Power-Efficient Networked Many-Core Systems
154. **Ville Rantala**, On Dynamic Monitoring Methods for Networks-on-Chip
155. **Mikko Pelto**, On Identifying and Locating-Dominating Codes in the Infinite King Grid
156. **Anton Tarasyuk**, Formal Development and Quantitative Verification of Dependable Systems
157. **Muhammad Mohsin Saleemi**, Towards Combining Interactive Mobile TV and Smart Spaces: Architectures, Tools and Application Development
158. **Tommi J. M. Lehtinen**, Numbers and Languages
159. **Peter Sarlin**, Mapping Financial Stability
160. **Alexander Wei Yin**, On Energy Efficient Computing Platforms
161. **Mikołaj Olszewski**, Scaling Up Stepwise Feature Introduction to Construction of Large Software Systems
162. **Maryam Kamali**, Reusable Formal Architectures for Networked Systems
163. **Zhiyuan Yao**, Visual Customer Segmentation and Behavior Analysis – A SOM-Based Approach
164. **Timo Jolivet**, Combinatorics of Pisot Substitutions
165. **Rajeev Kumar Kanth**, Analysis and Life Cycle Assessment of Printed Antennas for Sustainable Wireless Systems
166. **Khalid Latif**, Design Space Exploration for MPSoC Architectures

Turku Centre *for* Computer Science

Joukahaisenkatu 3-5 B, 20520 Turku, Finland | www. tucs.fi

University of Turku
Faculty of Mathematics and Natural Sciences
- Department of Information Technology
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Division for Natural Sciences and Technology
- Department of Information Technologies

ISBN 978-952-12-2976-3
ISSN 1239-1883

Khalid Latif

Design Space Exploration for MPSoC Architectures

Khalid Latif

Design Space Exploration for MPSoC Architectures