Runtime Adaptive System-on-Chip Communication Architecture

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik

der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Mohammad Abdullah Al Faruque

aus Tangail, Bangladesh

Tag der mündlichen Prüfung: 22.07.2009

Erster Gutachter: Prof. Dr.-Ing. Jörg Henkel

Zweiter Gutachter: Prof. Dr.-Ing. Jürgen Becker

Mohammad Abdullah Al Faruque
Hans-Pfitzner-Str.7
76227, Karlsruhe

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Mohammad Abdullah Al Faruque

Acknowledgment

First of all, I sincerely thank my adviser, Professor Dr. Jörg Henkel, for his erudite guidance during my whole period of research. I am fortunate to have an adviser with great vision, who is always interested in exploring new research directions and ideas. He gave me complete freedom in performing research, and still made sure that my various works are neatly tied together. He has given excellent personal support to all his students and provided an enjoyable working environment.

I am also very grateful to Professor Dr. Jürgen Becker for accepting to be my co-examiner and providing valuable feedback.

I would like to thank my colleague Thomas Ebi for his continuous support in my research work. I present my gratitude to other colleagues Talal Bonny, Lars Bauer, and Muhammad Shafique for their support through discussion and positive feedback regarding my research work. I again thank all the other members of our chair, "Chair for Embedded Systems" for their help and support during my research work and thesis writing.

I am grateful to all my students for their hard and fruitful work, particularly Gereon Weiss, Xing Ye, Rudolf Krist, Manuel Hammerich, Thomas Schröder, and Thorsten Vogel.

I thank my parents for their constant love, support, and prayers. My parents always dreamed that one day I will finish my PhD. I believe this piece of work was only possible because of their prayer and wishes. I also thank my elder sister and younger sister for their loving and caring attitude throughout my student life. I would not miss the chance to thank my grandparents for their best wishes.

Finally, I present my deepest gratitude to my wife and my daughter. All of my achievements contain an invisible part of their contribution. I am grateful for all the sacrifices that my wife made to support my research. The quality of this work would not have been achieved without her priceless sacrifices.

I dedicate this thesis to all of my family members.

List of Publications

Publications Included in This Thesis

Journal Publication

[J.1] M. A. Al Faruque, J. Henkel: *"QoS-Supported On-chip Communication for Multi-Processors"*, in International Journal of Parallel Programming **(IJPP'08)**, Volume 36, Number 1, Pages: 114-139, February, 2008.

Conference Publications

[C.1] M. A. Al Faruque, T. Ebi, J. Henkel: *"Configurable Links for Runtime Adaptive On-chip Communication"*, in IEEE/ACM Design Automation and Test in Europe **(DATE'09)**, Munich, Germany, Pages: 256-261, April, 2009.

[C.2] M. A. Al Faruque, T. Ebi, J. Henkel: *"ROAdNoC: Runtime Observability for an Adaptive Network on Chip Architecture"*, in IEEE/ACM International Conference on Computer-Aided Design **(ICCAD'08)**, San Jose, California, USA, November, 2008.

[C.3] M. A. Al Faruque, R. Krist, J. Henkel: *"ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication"*, in 45th IEEE/ACM/EDA Design Automation Conference **(DAC'08)**, Anaheim, California, USA, Pages: 760-765, June, 2008. **(Received a European Network of Excellence on High Performance and Embedded Architecture and Compilation, HiPEAC Paper Award)**

[C.4] M. A. Al Faruque, J. Henkel: *"Minimizing Virtual Channel Buffer for Routers in on-chip Communication Architectures"*, in IEEE/ACM Design Automation and Test in Europe **(DATE'08)**, Munich, Germany, Pages: 1238-1243, March, 2008.

[C.5] M. A. Al Faruque, T. Ebi, J. Henkel: *"Run-time Adaptive on-chip Communication Scheme"*, IEEE/ACM International Conference on Computer-Aided Design **(ICCAD'07)**, San Jose, California, USA, Pages: 26-31, November, 2007.

[C.6] M. A. Al Faruque, J. Henkel: *"Transaction Specific Virtual Channel Allocation in QoS Supported On-chip Communication"*, IEEE International Conference on Application-specific Systems, Architectures and Processors **(ASAP'07)**, Montreal, Canada, Pages: 48-53, July, 2007.

[C.7] M. A. Al Faruque, G. Weiss, J. Henkel: *"Bounded Arbitration Algorithm for QoS-Supported On-chip Communication"*, IEEE/ACM International Conference on Hardware/-Software Co-Design and System Synthesis **(Codes+ISSS'06)**, Seoul, South Korea, pages: 76-81, October, 2006.

Workshop Publication

[W.1] M. A. Al Faruque, X. Ye, G. Weiss, J. Henkel: *"QoS-Supported Configurable Networks on Chip"*, Future Interconnects and Network on Chip **(NOCS'06)** Workshop co-located with IEEE/ACM Design Automation and Test in Europe **(DATE'06)**, Page: 42, Munich, Germany, March, 2006 (poster, abstract).

Abstract

Diverse and during runtime varying workloads and/or constraints in embedded systems require runtime adaptivity to provide a high degree of efficiency during any operation mode/scenario. Design-time decisions can often only cover certain scenarios and fail in efficiency when hard-to-predict system scenarios occur. Reliability concerns associated with upcoming technology nodes further strengthen the necessity towards considering runtime adaptivity in all possible parts of the future system-on-chips. In this thesis, the first approach of an adaptive system-on-chip communication architecture is presented. The adaptive system provides adaptivity both in the *system-level* as well as in the *architecture-level*. The *system-level* adaptation is provided using a runtime application mapping. The *architecture-level* adaptation is implemented by using several novel methodologies to increase the resource utilization of the underlying silicon fabric, i.e. sharing the *Virtual Channel Buffers* (VCBs) among different output ports, changing the routing at runtime, and changing the supported bandwidth between adjacent links using configurable links at runtime. To achieve successful runtime adaptation, runtime observability is a prerequisite, as it provides necessary system information gathered on-the-fly. Therefore, a comprehensive runtime observability infrastructure for the proposed adaptive system is also presented in this thesis.

The proposed adaptive system-on-chip communication architecture is capable of supporting deadlock-free data transmission and meets required bandwidth guarantees for parallel transactions. Therefore, it is built on top of a novel *Quality-of-Service* (QoS)-supported *Networks-on-Chip* (NoC) architecture. The link arbitration algorithm, the *Bounded-Arbitration-Algorithm* (BAA) proposed in this thesis manages the flow-control mechanism at *transaction-level* and provides 100% guarantee on demanded bandwidth. The advantages of *BAA* are demonstrated by means of a complete MPEG4 video decoder case study analysis and under certain constraints it is shown that *BAA* achieves a bandwidth utilization of up to 100% (97% on an average) with a guaranteed 100% bandwidth.

A runtime agent-based distributed application mapping is employed to achieve the *system-level* adaptation. To obtain a scalable mapping solution, the computational load is reduced by confining mapping to *clusters* which are a connected subset of NoC tiles. Exploration shows, the proposed runtime mapping obtains 10.7 times lower monitoring traffic compared to the state-of-the-art centralized mapping schemes proposed for a 64×64 NoC. The algorithm also requires less execution cycles compared to a non-clustered centralized approach. It achieves on an average 7.1 times lower computational effort of the mapping algorithm compared to the simple *Nearest-Neighbor* (NN) heuristics in a 64×32 NoC.

After the *system-level* has successfully set up a mapping instance, it is up to the *architecture-level* to configure each tile for the resulting transactions. The *2X-Link*, a novel link for the

architecture-level provides a higher throughput of up to 36%, with an average throughput increase of 21.3%, compared to the *Normal-Full-Duplex-Link* and keeps performance-related guarantees with as low as 50% of the *Normal-Full-Duplex-Link* capacity. The experiments show, when some links fail randomly, the NoC with the *2X-Links* may recover from those faults with an average probability of 82.2%, whereas those faults would be fatal for the *Normal-Full-Duplex-Links*. The on-demand buffer assignment presented as a part of the *architecture-level* of the adaptive system increases the buffer utilization and decreases the overall buffer use on an average of 42% in the case study analysis compared to a fixed buffer assignment. The runtime observability infrastructure that is integrated for the runtime system in the *architecture-level* is hardly intrusive, i.e. in worst case it may require a mere 0.7% of the total link capacity. It analyzes the communication architecture during runtime and self-adapts depending on the monitoring traffic on when and how a certain router should be configured for a certain transaction. The runtime observability infrastructure on an average increases the transaction success rate by 62% compared to having no runtime observability for the E3S benchmark suite with a hardware overhead of 46 slices in a Virtex II FPGA. The area overhead that stems from the *architecture-level* adaptation may be traded-off against the flexibility to select an available route using the *2X-Links* and on-demand buffer assignment (42% buffer saving) to that route for ensuring the QoS.

The QoS-supported NoC architecture is further explored for a case-study analysis to design an application-specific NoC. A novel methodology for design space exploration using a two-step methodology to minimize the number of *VCBs* is presented as a part of the application-specific NoC design in this thesis. On an average, 90.2% reduction in the number of *VCBs* compared to a fixed assignment for the E3S embedded application benchmark suite is achieved.

In summary, a runtime adaptive system-on-chip communication architecture on top of a QoS-supported NoC is presented in this thesis. The runtime adaptation is achieved by providing novel methodologies both in the *system-level* as well as in the *architecture-level*. The benefits of such an approach are presented with extensive experiments using representative embedded systems applications e.g. a robotic application, a set of multi-media applications, and the E3S embedded systems synthesis benchmarks suite.

Zusammenfassung

Vielfältige und zur Laufzeit variierende Auslastungen und/oder Einschränkungen eingebetteter Systeme erfordern eine Adaption des Systems zur Laufzeit, um einen hohen Grad der Effizienz während eines beliebigen Betriebsmodus/-szenarios zu gewährleisten. Entscheidungen, welche während der Entwurfszeit getroffen werden, können oft nur gewisse Szenarien berücksichtigen und scheitern im Falle von schwer vorhersehbaren Systemszenarien daran, die Effizienz aufrecht zu erhalten. Zuverlässigkeitsprobleme, welche verstärkt mit den kommenden Generationen der Halbleitertechnologie in Verbindung gebracht werden, verstärken den Bedarf an Laufzeitadaptivität in allen Teilen zukünftiger System-on-Chip Entwürfe. Diese Arbeit präsentiert den ersten Ansatz einer adaptiven Kommunikationsarchitektur für ein solches System-on-Chip. Dieses adaptive System bietet Adaptivität sowohl auf Systemebene als auch auf Architekturebene. Die Adaption auf Systemebene wird durch eine Zuordnung der Anwendungen auf die einzelnen Rechenknoten zur Laufzeit erzielt. Die Adaptivität auf Architekturebene wird durch verschiedene, neuartige Methodologien erreicht. Sie erhöhen die Auslastung der Ressourcen des zugrunde liegenden Siliziums, z.B. durch das dynamische Verteilen des Speichers der virtuellen Kanäle (*Virtual Channel Buffer*, VCB) an verschiedene Ausgangsports, eine Anpassung des Routings zur Laufzeit und das Ändern der möglichen Bandbreite zwischen benachbarten Routern zur Laufzeit. Um eine erfolgreiche Adaption zur Laufzeit zu ermöglichen, ist eine ständige Beobachtung des Systems erforderlich. Daher wird eine umfassende Infrastruktur zur Beobachtung des Systems zur Laufzeit im Rahmen dieser Arbeit entworfen und vorgestellt.

Die vorgestellte adaptive System-on-Chip Kommunikationsarchitektur, realisiert als Networks-on-Chip (NoC), unterstützt eine deadlock-freie Datenübertragung und garantiert benötigte Dienstgüten (*Quality-of-Service*, QoS) während paralleler Übertragungen. Das in dieser Arbeit vorgeschlagene Verfahren, der *"Bounded-Arbitration Algorithmus"* (BAA), regelt die Flusskontrolle auf Transaktionsebene und bietet eine 100-prozentige Garantie der verlangten Bandbreitenanforderungen. Die Vorteile des *BAAs* werden anhand einer umfangreicher Fallstudie eines MPEG4 Videodecoders analysiert. Hierbei wird gezeigt, dass der *BAA* unter den gegebenen Einschränkungen eine Bandbreitennutzung von bis zu 100% (im Durchschnitt 97%) erreicht, ohne an Dienstgüte zu verlieren (100% garantierte Bandbreite).

Es wird eine verteilte, agentenbasierte Zuordnung der Anwendungen auf die einzelnen Rechenknoten zur Laufzeit verwendet um die, Adaption auf Systemebene zu realisieren. Um die Zuordnung skalierbar zu halten, wird der Rechenaufwand reduziert, in dem die Zuordnung auf Cluster beschränkt wird. Ein Cluster besteht aus einer verbundenen Untermenge der Networks-on-Chip-Knoten. Untersuchungen zeigen, dass das vorgeschlagene Verfahren 10,7 mal weniger Datenübertragung zur Systemüberwachung verursacht, als ein zentral gesteuertes Verfahren,

wie es zum aktuellen Stand der Technik in 64 $\times$ 64 NoCs verwendet wird. Hinzu kommt, dass der Algorithmus im Vergleich zu Verfahren ohne Cluster weniger Rechenzeit benötigt. In Experimenten verursacht er im Durchschnitt 7,1 mal weniger Rechenaufwand im Vergleich zur einfachen *"Nearest-Neighbor"* (NN)-Heuristik in einem 64 $\times$ 32 NoC.

Nachdem auf Systemebene erfolgreich eine Anwendungszuordnung gefunden wurde, ist es eine Aufgabe auf Architekturebene, die resultierenden Verbindungen der Kacheln zu konfigurieren. *2X-Links*, eine neue Art von Verbindungen für Networks-on-Chips, ermöglicht einen bis zu 36% höheren Durchsatz (im Durchschnitt 21,3%) im Vergleich zu normalen Vollduplexverbindungen. Sie garantieren Verbindungen mit bis zu 50% der Kapazität von normalen Vollduplexverbindungen. Durch Simulationen wird gezeigt, dass ein NoC mit *2X-Links* mit einer durchschnittlichen Wahrscheinlichkeit von 82,2% zufällige Verbindungsausfälle umgehen kann, obwohl die Ausfälle für normale Vollduplexverbindungen unausweichlich wären. Die bedarfsabhängige Zuteilung der *VCBs* erhöht die Auslastung der einzelnen Buffer. In einer Fallstudie wird der Speicherbedarf im Vergleich zu einer festen Speicherzuteilungsstrategie im Durchschnitt um 42% reduziert. Die Infrastruktur zur Beobachtung des Systems zur Laufzeit, welche in die Architektur des Laufzeitsystems integriert ist, ist sehr leichtgewichtig – im schlimmsten Fall benötigt sie höchstens 0,7% der verfügbaren Verbindungskapazität. Sie analysiert die Kommunikation wären der Laufzeit und entscheidet eigenständig, wann und wie ein Router für gewisse Übertragungen konfiguriert sein sollte. Diese Maßnahme verbessert die Erfolgsquote der Übertragungen von Anwendungen aus der E3S-Benchmark-Suite um 62% im Vergleich zu einer Architektur ohne eine entsprechende Beobachtungsinfrastruktur. Sie benötigt einen Hardwareaufwand von 46 Slices in einem Virtex II FPGA. Der zusätzliche Hardwareverbrauch wird durch die Möglichkeit, verschiedene Routen – mit und ohne *2X-Links* – zu nehmen, der Speicherzuteilung zur Laufzeit (42% weniger Speicherbedarf) und durch die erlangte Dienstgüte ausgeglichen.

Die QoS-bietende System-on-Chip Kommunikationsarchitektur wird mittels einer Fallstudie weiter untersucht, um ein applikations-spezifisches NoC zu entwerfen. Eine neue Vorgehensweise, um mittels einer zweistufigen Methodologie die Zahl der benötigten *VCBs* zu minimieren, wird im Rahmen dieser Arbeit vorgestellt. Im Durchschnitt wird dadurch der für die "E3S embedded application benchmark suite" benötigte Speicher im Vergleich zu einer festen Speicherzuteilung um 90,2% reduziert.

Zusammenfassend präsentiert diese Arbeit ein laufzeitadaptives NoC, welchem eine System-on-Chip Kommunikationsarchitektur, welche QoS unterstützt, zugrunde liegt. Die Adaption zur Laufzeit wird durch neue Vorgehensweisen sowohl auf Systemebene als auch auf Architekturebene erzielt. Die Vorteile eines solchen Ansatzes werden durch zahlreiche Experimente aus repräsentativen Anwendungen für eingebettete Systeme dargestellt – z.B. einer Robotikapplikation, einer Klasse von Multimediaapplikationen und der "E3S benchmark suite".

Contents

Abbreviations

ACO	Ant Colony Optimization
AdNoC	Adaptive Networks-on-Chip
ADAM	Agent-based Distributed Application Mapping
ASIC	Application-Specific Integrated Circuit
ROAdNoC	Runtime Observability for Adaptive Networks-on-Chip
ARQ	Automatic Repeat Request
B-Frame	Bidirectional-Frame
BAA	Bounded-Arbitration-Algorithm
BE	Best-Effort
BW	BandWidth
CA	Cluster Agent
CAD	Computer-Aided Design
CM	Centralized Manager
CMOS	Complementary MetalOxideSemiconductor
CPU	Central Processing Unit
DSM	Deep Sub Micron
DSP	Digital Signal Processing
E3S	Embedded Systems Synthesis Benchmarks Suite
FIFO	First In First Out
Flit	Flow Control Unit
FPGA	Field-Programmable Gate Array
GA	Global Agent
GS	Guaranteed Service
GUI	Graphical User Interface
HF	Header Flit
I-Frame	Intra-Frame
IP	Intellectual Property
IPL	Image Processing Line
ITRS	International Technology Roadmap for Semiconductors
iQuant	Inverse Quantification of the MPEG4 Decoder
LB	Lower Bound
LC	Link Control
LUT	Look-Up-Table
MPEG4	Moving Picture Experts Group-Standard 4
MPSoC	Multi-Processor System-on-Chip

MUX	Multiplexer
MWD	Multi-Window Display
NACK	Negative ACKnowledgment
NI	Network Interface
NN	Nearest-Neighbor
NoC	Networks-on-Chip
NoCMS	NoC Monitoring Service
OA	Output Arbiter
OCP	Open Core Protocol
OCIN	System-on-Chip Interconnection Network
OD	Output Decoder
PF	Payload Flit
PE	Processing Element
P-Frame	Predicted-Frame
PCA	Physical Channel Arbiter
PDA	Personal Digital Assistant
PIP	Picture in Picture
QAP	Quadratic Assignment Problem
QoS	Quality-of-Service
QNoC	Quality-of-Service NoC
RAM	Random-Access Memory
RGB	Red, Green, and Blue
RR	Round-Robin
SDM	Space Division Multiplexing
SE	Storing Element
SHF	Secondary Header Flit
SoC	System-on-Chip
TF	Tail Flit
TG	Task Graph
TGFF	Task Graph For Free
TDD	Time-Division-Duplex
TDMA	Time Division Multiple Access
TTL	Time-To-Live
UB	Upper Bound
UMARS	Unified MApping, Routing, and Slot allocation
VBR	Variable-Bit-Rates
VC	Virtual Channel
VCA	Virtual Channel Arbiter
VCB	Virtual Channel Buffer
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VOPD	Video Object Plane Decoder
wXY	weighted XY

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Semiconductor industries have kept alive Moore's law since 1965 with the innovation of novel process technologies and this is expected to continue as long as our system requirements expand at the present rate. In the current process technology shift from 65nm to 45nm, the number of logic gates per square millimeter has increased from 700,000 to 1,4 million [85]. Intel also projects the availability of 100 billion transistors on a 300mm^2 die by 2015 [26]. Due to several practical limitations the process technology is no longer able to provide reliable silicon fabrics with the continuing technology scaling and therefore, reliable systems have to be built from un-reliable fabrics [6, 25, 126]. The burden of continuing on the technology roadmap to meet the increasing computational demand is now on the shoulders of *architecture* and *system-level* engineers. In the continuing process of novel architectures, a *Multi-Processor System-on-Chip* (MPSoC) made up of thousand of heterogeneous *Processing Elements* (PEs), i.e. different types of *instruction set processors* or *reconfigurable hardware* on such an architecture, plays an important role in satisfying the increased computational demand in an energy-efficient way. The design methodology for the MPSoC and its on-chip interconnection have already been shifted towards *Networks-on-Chip* (NoC) as it is envisioned that future MPSoCs will be predominantly communication-centric [14, 44, 69, 73, 87, 128, 174].

Extensive research had been done to develop area and energy efficient NoC since 2000 both in the industry and in the academia. Besides, several research breakthrough in academia to develop application-specific NoCs, i.e. *XPipe*[14], *Æthereal* [65], *Proteo* [145], etc. recently, several general-purpose NoCs such as *Tile64™*, an embedded multicore by Tilera [173] and an *80-core* general-purpose processor from Intel [76] have been fabricated. In a nutshell, the research in the domain of NoC has been mainly concentrated on application-specific NoCs [14, 123] and design-time parameterized general-purpose NoCs [66, 76, 173] (NoCs with respect to reconfigurable logic blocks are not considered here and will be discussed later). In order to achieve runtime flexibility for different scenarios, i.e. reliable communication in a MPSoC, the NoC must be extended with adaptive capabilities, e.g. the ability to change the traffic route at runtime in order to bypass faulty areas efficiently. This is the first work to propose a runtime adaptive system-on-chip communication architecture which together takes the reliability issues with other issues (e.g. the user-behavior) into consideration.

1.1 Background

Thousand of processor cores will be possible to integrate on a system-on-chip communication architecture [26]. In the *Deep Sub Micron* (DSM) technology in the late silicon era (nano-age),

many characteristics of the on-chip systems will change drastically and will have a major impact on the system-on-chip interconnect. The nano-age is characterized through novel effects that aggravate the design and architecture when migrating to upcoming technology nodes. Therefore, the challenges at physical-level at nano-age have to be tackled to achieve a successful system-on-chip communication architecture. Traditionally, shared bus-based architectures have been used as a system-on-chip interconnects for the MPSoCs [3, 96]. In such an architecture, only two communicating partners may access the bus at a single point of time. In case of multiple requests from different communicating partners, the requests are arbitrated using different arbitration mechanisms, e.g. round-robin, priority-based round-robin, etc. The bus-based architectures have evolved over the last couple of years from single shared buses to multiple bridged buses (hierarchical buses), and crossbars. The hierarchical buses, i.e. *AMBA multi-layer* [2], *STBus* [155], *SonicsMX* [149], etc. allow multiple buses to operate in parallel, however, all the communicating partners need to be connected to a single crossbar. These bus-based architectures suffer from scalability problems for a large number of connected processing elements.

In summary, these bus-based system designs have been seen as incapable of meeting the design challenges for the next generation systems mainly due to lack of scalability and the imbalance between gate delays and wire delays on-chip in the nano-age [44, 74, 96]. It has been studied in early works [44, 74] that every additional transaction on a bus increases the resistance, which in turn leads to higher energy consumption because of the necessary bus drivers. A long bus also comes with latency drawbacks which limit the possible system frequencies. Clock skew is an inherent synchronization problem of long links, as using the same clock for all components throughout the whole system will not be achievable with the growing future complexity. A hierarchical bus needs bridges to overcome synchronization problems, which are not feasible for more complex systems having thousand-core chips. Furthermore, in the *DSM* era, physical issues like crosstalk or energy dissipation are likely to appear. Additional mechanisms are therefore required to achieve the reliability of the communication. To tackle all these problems, the paradigm shift to communication-centric design using NoCs as a system-on-chip communication concept has emerged as an important step toward the MPSoC architecture [44, 69, 74, 87, 122]. The evolution of NoC from a simple shared bus structure is shown in Figure 1.1 (for simplicity, a NoC having a 2D regular Mesh topology is considered here).

1.2 Communication-centric Design: Networks-on-Chip Design Evolution

In the beginning of research in the domain of NoC, the major challenges had been to select the design related parameters for an area and energy efficient router architecture to facilitate packet-based communication. Several parameters are selected that influence the design and performance of the NoC. The parameters may be summarized as follows: *the topology*, *the link-width*, *the buffer size*, *the floorplanning*, *the routing algorithm*, *the switching strategy*, and *the application mapping*. Details of all these parameters are given in Chapter 4. Different technology and design-related issues considering these parameters have led NoC research broadly into the following directions: the application-specific NoCs, e.g. *Nostrum* [94, 118], *Xpipe*[14, 43], *QNoC* [23], the design-time parameterized general purpose NoCs, e.g. *Tile64™*[173], an *80-core* general-purpose processor from Intel [76], the reconfigurable NoCs, e.g. *DyNoC* [22],

CuNoC [88], *ReNoC* [156], *LiPaR* [141], etc. that are built on top of reconfigurable hardware, e.g. FPGA, and the adaptive NoC (proposed in the scope of this thesis).

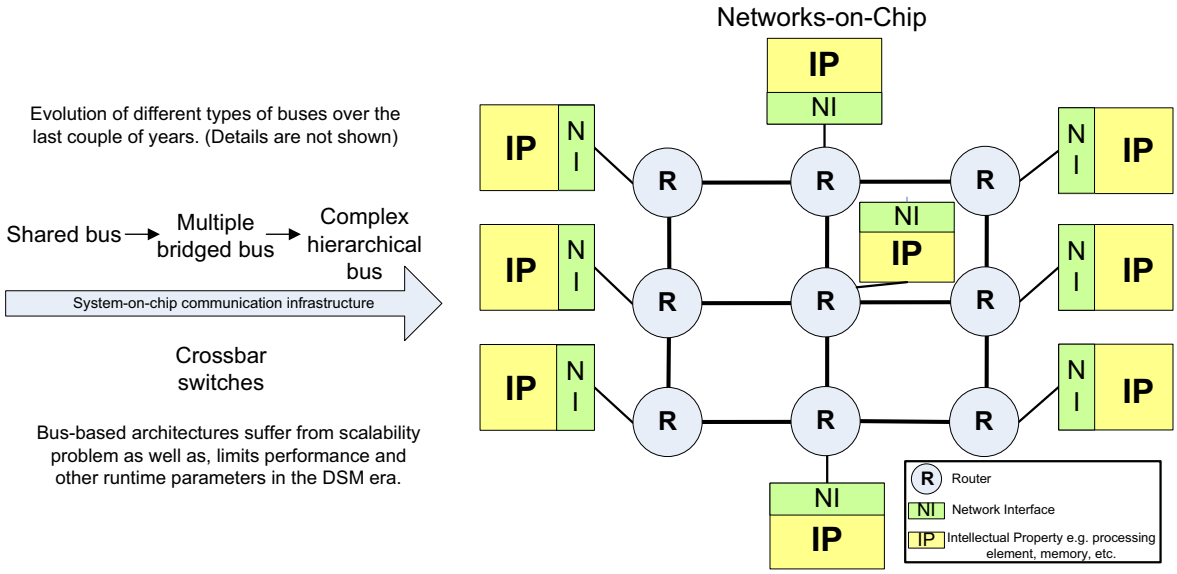


Figure 1.1: Evolution of the Networks-on-Chip

Application-specific Networks-on-Chip: Application-specific NoCs are generally tailor-made for a certain application or an application domain and fail in scenarios of hard-to-predict system behavior and/or in situations where user-behavior and reliability are concerns. An application-specific NoC is defined as a design-time parameterized architecture with a custom topology, a fixed routing scheme, a fixed number of allowed virtual connections at each output port, fixed supported bandwidth because of fixed channel width, a fixed floorplanning, and a static application mapping algorithm [14, 23, 78, 122]. Details of the application-specific NoC architecture, design methodology, tools, and other optimization are discussed in Chapter 4.

Design-time Parameterized General Purpose Networks-on-Chip: Design-time parameterized general-purpose NoCs are over-designed (e.g. number of virtual channels) considering different types of traffic scenarios and cannot adapt architectural parameters, i.e. buffer assignment or link capacity at runtime (lower resource utilization). Recently, several general-purpose NoCs such as Tile64^{TM}, an embedded MPSoC from Tilera [173] and an 80-core general-purpose processor from Intel [76] have been designed and fabricated. The runtime configuration of different communication-related parameters in the *architecture-level* is not considered in design-time parameterized general-purpose NoCs and therefore, they provide lower hardware utilization. Runtime application mapping may only be considered to achieve a general purpose behavior on such an over-designed general-purpose NoC [37].

Adaptive Networks-on-Chip: In general, the research in the domain of NoC has been concentrated on the application-specific NoCs and the design-time parameterized general-purpose NoCs, as well as NoCs on FPGA-type hardware (as discussed later). In order

to tackle hard-to-predict system behavior from different runtime scenarios, i.e. the user-behavior, reliability issues in a MPSoC, the NoC must be extended with adaptive capabilities, e.g. the ability to change the traffic route at runtime in order to bypass faulty areas efficiently with a higher resource utilization and flexibility. Self-organization considering the adaptivity has been a research focus in *Artificial Intelligence* and *Distributed Systems* communities [9, 95] for several years. *IBM's Autonomic Computing Initiative* [75] deals with self-organization of IT servers in networks. Self-organization in system-on-chip design is relatively new. The possibility to use the idea of adaptivity in the future system-on-chip design is discussed in [12, 75, 100].

To achieve self-adaptivity in the system-on-chip communication architecture, the adaptive NoC needs to provide adaptivity both in the *system-level* as well as in the *architecture-level*. The *system-level* adaptation may be provided using a runtime application mapping. Therefore, the assignment of different tasks of an application can be realized in different processing elements at runtime. At the same time, to achieve a higher resource utilization and flexibility, the architecture of the adaptive NoC needs to be integrated with novel methodologies of runtime parameterization. The runtime resource parameterization mainly exploits resource multiplexing mechanisms at varying workloads. In the scope of this thesis, a novel adaptive system-on-chip communication architecture, the *Adaptive Networks-on-Chip* (AdNoC) is presented to exploit all the benefits that can thus be achieved.

Networks-on-Chip on top of Reconfigurable Hardware: All the previously mentioned different types of NoC architectures are partly or fully static in design for a group of parameters. The adaptive NoC that is presented in this thesis is configurable in some parameters, i.e. *buffer assignment*, *routing*, *supported bandwidth*, and *mapping* but still static in its *topology*, *floorplanning*, *switching strategy*, *processing element types*, etc. at runtime. Therefore, these NoCs are not sufficiently flexible as a communication architecture on top of reconfigurable hardware, i.e. FPGA-based systems, where flexibility of the number of processing elements and their types are reconfigurable at runtime. The multi-core architectures built on top of FPGA-based fabrics demand a changing network which may be instantiated at runtime. A great deal of work has been done in this paradigm of NoCs for the reconfigurable hardware, i.e. in [22, 82, 83, 88, 103, 131, 141, 156].

1.3 Contributions of This Dissertation

In this thesis, a self-adaptive system-on-chip communication architecture that analyzes itself during runtime and self adapts when and how a certain router should be configured for a certain transaction is presented. **The novel contributions are as follows:**

1. *Adaptive Networks-on-Chip* **(AdNoC)**, an adaptive system-on-chip communication architecture is proposed in the scope of this thesis. The runtime adaptivity in the *AdNoC* architecture is employed both in the *system-level* as well as in the *architecture-level* (published in [52, 53, 54, 58]).

2. A scalable and configurable cycle-accurate *Quality-of-Service* **(QoS)-supported system-on-chip communication architecture** on top of which a case study analysis of an application-specific NoC, as well as the proposed *AdNoC* are built, are presented in details in this thesis (published in [57, 60]). A novel link arbitration algorithm named as ***bounded-arbitration-algorithm*** which provides *transaction-level* bandwidth guarantees considering the upper and the lower bounds of each transaction is used for the QoS-supported system-on-chip communication architecture (published in [59]).

3. The *system-level* adaptation is achieved using a runtime *Agent-based Distributed Application Mapping* **(ADAM)** algorithm for the next generation self-adaptive heterogeneous MPSoCs. The advantages, runtime behavior, and performance gain using the novel *ADAM* algorithm are presented in this thesis (published in [58]).

4. In the *architecture-level*, several novel methodologies to adapt the underlying interconnections on-demand in response to changing communication requirements imposed by an application, i.e. runtime application mapping request due to reliability issues or user behavior, are employed.

5. To provide on-demand interconnections, a novel adaptive routing algorithm **(*wXY-routing* algorithm)** that meets QoS requirements (bandwidth) is presented. The routing algorithm makes decisions locally at each router depending on the available bandwidth in each direction to the neighboring router. Dynamic connections are realized by re-assigning a certain number of buffer blocks to different output ports of a router on-demand. It also increases the resource utilization, especially buffer utilization, through **on-demand buffer block assignment** (published in [52]).

6. A runtime configurable link **(*2X-Link*)** at the *architecture-level* to compliment the adaptive system-on-chip communication architecture is presented. The *2X-Links* can adapt the link capacity by changing the direction at runtime on-demand, thereby increasing the resource utilization while considering the reliability issues. The building blocks of the *2X-Link* are two *half-duplex links* instead of the state-of-the-art *simplex* links (published in [54]).

7. To employ successful adaptation the communication architecture needs to be observed. Therefore, to provide runtime observability, a novel low cost *Runtime Observability infrastructure for the AdNoC* **(ROAdNoC)** is presented. It is highly flexible and hardly intrusive (published in [53]).

1.4 Dissertation Outline

The remainder of the thesis is outlined as follows: **Chapter 2** presents the state-of-the-art related works in the scope of system-on-chip communication architecture design. Different types of NoC architectures, design methodologies, design-space exploration, application mapping, energy and resource minimization, routing algorithms etc. are presented in short in this chapter.

In **Chapter 3**, the QoS-supported NoC that is the basic architecture of the proposed system-on-

chip communication is detailed. The mechanism to establish QoS on top of a packet-based communication is shown here. The link arbitration algorithm: the *bounded-arbitration-algorithm* proposed in the scope of this thesis is also presented with supporting case study analysis in this chapter.

Chapter 4 presents a complete design methodology of an application-specific NoC on top of the QoS-supported NoC architecture discussed in Chapter 3. Different parameters that may be customized at design time to build an energy/area efficient application-specific NoC are discussed with special emphasis on a novel methodology of buffer minimization. The buffer for an application-specific NoC may be minimized during application mapping as well by knowing the application-specific traffic model. Different case-study analysis are shown to further demonstrate the applicability of the proposed buffer minimization technique.

The proposed runtime *AdNoC* built on top of the QoS-supported NoC discussed in Chapter 3 as a part of the adaptive system-on-chip communication architecture is introduced in **Chapter 5**. Different definitions related to *AdNoC* architecture are introduced here. Details of the achieved adaptation both in the *system-level* and in the *architecture-level* are explained in the following chapters.

The *system-level* adaptation that performs a runtime application mapping is presented in **Chapter 6**. In the scope of this chapter, a novel runtime agent-based distributed application mapping algorithm (*ADAM*) is explained in detail. The complexity analysis as well as the flexibility and the overhead of such approach is also discussed in this chapter. Different case-study analysis to show the benefits of *ADAM* algorithm are presented in Chapter 9.

Chapter 7 detailed different novel methodologies for the *architecture-level* adaptation for the proposed *AdNoC* architecture. To achieve an area efficient flexible adaptation, different novel methodologies are used, i.e. on-demand buffer assignment to different output ports, adaptive routing, the *wXY-routing* algorithm for supporting varying workloads, and link reversal mechanism the *(2X-Link)* to allow runtime supported bandwidth adaptation. A comprehensive runtime observability infrastructure for the *AdNoC* (*ROAdNoC*) which is flexible (e.g. in choosing the routing path), hardly intrusive, and requires little additional overhead is also described in this chapter.

Tools, simulation environment, and the prototyping boards used to demonstrate the benefits of the proposed architecture are explained in **Chapter 8**. A SystemC-based simulator that has been developed as a part of the thesis work is also shown here. The VirtexII-based prototyping board for runtime hardware evaluation is presented in this chapter.

In **Chapter 9**, different case-study analysis to show the performance, area, and energy related results for the proposed *AdNoC* architecture are presented. Evaluation results for both the *system-level* adaptation and the *architecture-level* adaptation are demonstrated in this chapter. Different embedded applications, i.e. a robotic application, several multi-media applications, the E3S embedded benchmark suite, and artificial applications generated from the TGFF are used to show the benefits of the *AdNoC* architecture. Finally, the thesis is concluded together with future outlook in **Chapter 10**.

Chapter 2

Related Work: MPSoC Interconnections

Extensive works have been done in the domain of *System-on-Chip* (SoC) interconnection design for the *Multi-Processor System-on-Chip* (MPSoC) architectures. In this chapter, a short summary of the works those have highly influenced this thesis and have similarity on the domain of *Networks-on-Chip* (NoC) design are presented. In the rest of the chapters in various places several prior works on that particular novel contribution are discussed in details.

2.1 Bus-based Interconnection

In one facet of the bi-directional embedded system's development market, the digital convergence of multiple complex applications as well as new critical applications (software side) in single terminal demand for higher computational power. On the other facet of the development, the semiconductor industries allow to introduce thousands of processors or equivalent logic gates on a single chip (hardware side) to fulfill the computational demand [26]. We are therefore, in the era of complex MPSoC architectures. Typically, a MPSoC architecture is built by exploiting off-the-shelf standard components (the component-based design) and using multiple high performance hierarchical buses for establishing communication among components (bus-based system design).

It is already discussed in Chapter 1 that bus-based MPSoC design is not efficient and scalable for future such system design [44, 74]. Therefore, considering the limitations (see Table 2.1) of the bus-based architectures, the paradigm shift to the communication-centric design has emerged as an important step for the MPSoC architecture. Academic and industrial frameworks in the scope of MPSoC interconnections are briefly presented below (for a chronological overview see [144]). The industrial solutions can be divided into three main approaches:

- *Shared bus* has been traditionally used as a system-on-chip interconnections for the MPSoCs (ARM *AMBA* [3, 96], IBM *CoreConnect* [42], Sonics μ*Network* [149, 175]). In such an architecture, only two communicating partners may access the bus at a single point of time. In case of multiple requests from different communicating partners the request is arbitrated using different arbitration mechanisms, i.e. *Round-Robin* (RR), priority-based *RR*, etc. The bus-based architectures have evolved over the last couple of

Bus	**Networks-on-Chip**
Disadvantages	*Advantages*
Parasitic capacitance grows with every attached part, thus no scalability of performance is possible	(Structured) point-to-point, one-way links for any network size enable a better signaling
Difficulty of specifying the bus timing in DSM process	Global asynchronism of the parts enables pipelining of wires
Slow and problematic testability	Built-In Self Test (BIST) is fast and complete
Growing delay with every additional master and instance-specific bus arbiters are necessary	Distributed routing decisions are made possible and the same router can be scalable reinstantiated in the network
Bandwidth is limited and shared by all units attached to the bus	Scalability of the existing bandwidth through sharing the links
Advantages	*Disadvantages*
Fair scheduling and guaranteed latency through a single bus arbiter	(Unpredictable) network contention because of unpredictable communication behavior
Low silicon costs for bus	Significant silicon area costs for a whole system-on-chip network
Compatibility with most IPs	IPs need wrappers for transparent communication and synchronization of multi-processing
Simple and well understood concepts	Network design introduces new concepts for system designers

Table 2.1: Comparison of the bus-based and the NoC-based MPSoC architectures [19, 69, 96]

years from a single shared bus to a multiple bridged bus and crossbar (hierarchical bus). The hierarchical buses, i.e. *AMBA multi-layer* [2], *STBus* [155], *SonicsMX* [149], etc. allow multiple buses to operate in parallel however, all the communicating partners need to connect to a single crossbar. It has been used as an inexpensive solutions but suffers from face latency and scalability problems for a large number of connected cores.

- *Crossbar switches* provide a high bandwidth provision but suffer from a possible lack of scalability (Fulcrum Microsystems *Nexus* [115]).
- *Networks-on-Chip* are developed as a packet-based system-on-chip network that has to deal with all best-known network problems (STMicroelectronics *STNoC* [158], *Arteris* [4], *Silistix* CHAINworks [146] - before *CHAIN* [10] architecture of the Manchester university), *iNoC* [84], etc.

First academic projects in the field of NoC had been *SPIN* [69], *Proteo* [145, 164], the NoC of Dally [44], and *Æthereal* [65, 133]. Recent research resulted in architectures, i.e. *Nostrum* [94, 118], *Xpipe*[14, 43], *QNoC* [23, 135], etc.

The NoC paradigm allows to interconnect up to thousands of processor cores onto a single silicon die. The basic idea has been adopted from large-scale networks. Instead of connecting communication partners directly through one time-shared link, in a NoC the links can be shared by many transactions at the same time. Through this ability to handle parallel transactions, the problem of a bus as a bottleneck may be solved. In Table 2.1 as system-on-chip interconnections, NoCs and bus-based interconnections are compared considering the early works [69, 96].

2.2 Networks-on-Chip: Key Research Issues

NoCs as a system-on-chip communication architecture, its different design methodologies and key research problems have been discussed in [44, 69, 74, 87, 105, 122, 126, 128]. In [122], key research problems in NoC design are formalized. Here, authors have identified several parameters that need to be researched and parameterized for the application-specific NoC design. The key NoC research problems are in general: the topology synthesis problem, the channel width problem, the buffer sizing problem, the floorplanning problem, the routing problem, the switching problem, the scheduling problem, and the IP mapping problem. These research parameters from the perspective of the application-specific NoC design are discussed in Chapter 4. In [97], research challenges associated with some practical NoC implementation considering area and energy are shown.

Recently, in [126] authors have identified five broad research areas for the future NoC design: (1) system-on-chip interconnection network (OCIN) technology and circuits, (2) OCIN microarchitecture, (3) OCIN system architecture, (4) CAD and design tools for OCINs, and (5) evaluation and driving applications for OCINs. In the research area of OCIN technology and circuits, the CMOS technology roadmap and its affect on the OCIN need to be evaluated. OCIN microarchitecture research tries to find the answer of the micro-architectural solution for the system-on-chip routers and network interfaces to meet latency, area, and power constraints. OCIN system architecture research deals with the system architecture issues, e.g. topology, routing, flow control, and interfaces for system-on-chip networks. CAD and design tools for OCINs are needed to design system-on-chip networks and systems. Therefore, extensive research needs to be involved in this direction. Finally, research in evaluation and driving applications for OCINs tries to find the methodology to evaluate system-on-chip networks and to determine the dominant workloads (applications) for OCINs in next five to 10 years. In this paper, authors have also introduced the upcoming research challenges in NoC domain, i.e. programmability, power, application, reliability, variability, prototype development, standard benchmarks, etc.

2.2.1 Networks-on-Chip Architectures

NoC as discussed earlier, is promising for an interconnection fabric in MPSoC. Figure 2.1 shows the evolution of NoCs envisaged by the *International Technology Roadmap for Semiconductors* (ITRS) until the end of the next decade. Application-specific NoC design has been mainly a great research interest since the beginning of this decade. Several application-specific NoC architectures [20, 23, 32, 43, 65, 72, 92, 107, 133, 145, 157, 170] and different research issues, i.e. topology customization [1, 15, 33, 81, 109, 123, 127, 152], buffer minimization [55, 56, 78, 139], routing algorithms [45, 79, 132], application mapping [70, 77, 98, 104, 110, 111, 112, 113, 143], design methodologies and automation tools [57, 60, 67, 86, 89, 94, 114, 178] have been discussed in previous works. Different types of NoC architectures, i.e. the application-specific NoC, the design-time parameterized general-purpose NoCs, the adaptive NoCs, and the reconfigurable NoCs are already discussed in Chapter 1.

Recently, several NoC-related companies, i.e. *iNoC* [84], *Arteris* [4], *Silistix* [146] are selling tools to design application-specific NoCs. *iNoC* provides interconnection IP for ASIC and FPGA-based system. This IP portfolio enables seamless SoC connectivity, compliant with the

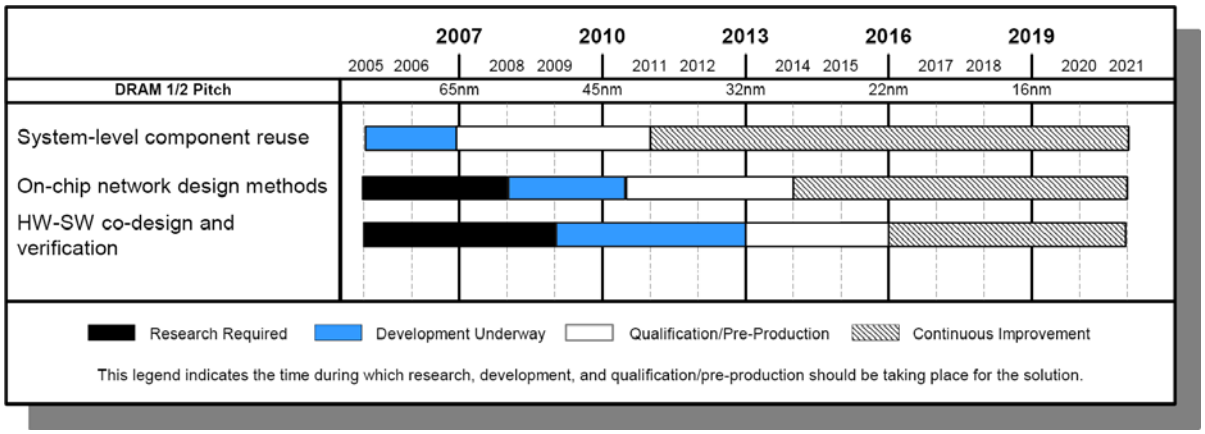


Figure 2.1: NoC as the future system-level design solution [85]

strictest requirements in bandwidth, latency, power, and area. *iNoC* also supplies a design tool for the application-specific NoCs. *Arteris* includes the following networking concepts for the NoC design: separation of computation and communication, packet-based communication, *Quality-of-Service* (QoS) based routing, and flexible NoC topology. *Silistix* provides *CHAINworks*, a suite of software tools for the design and synthesis of customized system-on-chip interconnect using self-timed clockless circuits.

2.2.2 Topology Customization

Lot of work has been done in the area of topology customization [1, 15, 33, 81, 109, 123, 127, 152] for designing an application-specific NoC. Some of the most influencing works in this research area are discussed in the following.

In paper [15], an advanced NoC architecture, called *Xpipe*, targeting high performance and reliable communication for system-on-chip interconnections is presented as one of the first works related to application-specific NoC design. *Xpipe* consists of a library of soft macros (switches, network interfaces, and links) that are design-time parameterizable so that application-specific architectures can be instantiated and synthesized. Links can be pipelined with a flexible number of stages to decouple link throughput from its length and to get arbitrary topologies. Moreover, a tool called *XpipeCompiler*, which automatically instantiates a customized NoC from the library of soft network components is used in this paper to test the *Xpipe-based* synthesis flow for application-specific communication architectures. Different features of *Xpipe* architecture may be found in [14, 16, 43, 86, 157].

In [123], authors have presented a methodology to automatically synthesize an architecture where a small number of application-specific long-range links are inserted on top of a given regular 2D Mesh topology. This way, they have achieved both the regularity as well as application-specific partial customization. In paper [152], a novel heuristic-based technique consisting of *system-level* physical design and interconnection network generation that generates custom low power NoC architectures for application-specific MPSoC is presented. They have shown that their technique has a low computational complexity, and consumes only 1.25 times the power consumption, and 0.85 times the number of router resources compared to their own previous

work considering an optimal MILP based technique [154] where computational complexity is not bounded.

In [109], efforts have been made to reduce the area cost of the application-specific NoCs by using network partitioning techniques. The *Fiduccia-Mattheyses* (FM) algorithm is adopted to formulate the partitioning problem for the application core graph. Their experiments show that their technique is a superior way to reduce the application-specific NoC area compared to other topology generation techniques. Authors in [127] present a customization scheme of the network topology for the *STNoC*, the NoC developed by STMicroelectronics. They started topology customization from a ring topology and a given application. Finally, their approach delivers an application-specific NoC with a customized topology in terms of performance, area, and energy overhead. They have shown that the generated *STNoC* custom topologies provide a reduced cost with respect to the *spidergon* topology. The work presented in [33] provides a methodology for generating an energy optimized application-specific NoC topology that supports both point-to-point and packet-switched networks. A prohibitive greedy iterative improvement strategy is used to explore the design space in their methodology. Their experiments show that a combination of on-chip and point-to-point networks (a hybrid network) achieve approximately 25% lower energy consumption (with a maximum of 37%) than a state-of-the-art min-cut partition based topology generator for different applications.

Work in [1] presents a component library for flexible construction of interconnection architectures to enable the creation of application development platforms. NoC topology optimization is included by describing the methodologies used by an effective design automation tool. The included cost functions of the tool capture the parameters contributing to the performance and energy consumption of asynchronous interconnections. Work in [81] shows a methodology to synthesize energy-efficient NoCs by adopting a multicommodity formulation to unify network topologies, physical embedding, and wire style optimizations. They utilize a variety of interconnect wire styles to achieve high performance low power system-on-chip communication. Their methodology could achieve a power saving up to 35% for a varity of applications. More works related to NoC topology customization can be found in [87]. In this thesis, a regular 2D Mesh architecture is used, no topology customization is considered both for the application-specific NoC and for the adaptive NoC on top of the proposed QoS-supported system-on-chip communication architecture.

2.2.3 Quality-of-Service-supported Networks-on-Chip

NoC design requires to adapt a set of parameters and these parameters need to be integrated with the QoS requirements for designing an embedded real-time system. Different issues related to QoS in bigger networks are summarized in [68, 181]. QoS parameters and its definition are extremely diverse and application-specific in embedded systems. The definition becomes more complicated while considering multiple applications on a single device. Current embedded systems offer multiple applications running concurrently or sequentially in the time domain. All these different applications have different requirements. These requirements need to be supported during runtime on such devices. To consider the requirements of multiple applications each application needs to be analyzed carefully and individually from a QoS perspective. Gen-

erally, the requirements of an embedded system application (e.g. real-time video telephoning) is time critical and driven by the current work load on that device. These quality limits are also sometimes restricted by the audio and visual perception of the human senses. Therefore, the underlying QoS parameters: latency, throughput and jitter must be provided by the system. Generally, the lower and the upper bounds can be measured for each type of transaction. Besides this performance phenomena, other services/transactions like security data and memory read/write or internet browsing may have varying quality requirements, e.g. reliability. The levels of reliability (i.e. lossless data transactions) also add more parameters to the final system design.

In [73, 133, 180], the issues of QoS, in terms of guaranteed *BandWidth* (BW), low latency, jitter, in-order transmission and lossless data transmission have been addressed. To meet these issues with minimum overhead is still an open research challenge. Formally, system-on-chip QoS in communication can be broadly classified into two categories: (I) *Time related performance guarantees* (i.e. bandwidth, latency and jitter) and (II) *Data-flow related reliability guarantees* (e.g. in-order data transmission, lossless data transmission). Early novel works [20, 145] have partly or fully ignored the time related QoS issues. In [14], data-flow related guarantees are emphasized. The need to support time-related guarantees and to provide flexible and/or hard guarantee for a time-critical system are very important issues. To differentiate among traffic requirements, *service classes/levels* have been introduced. The service classes in a system for different communication pairs have been roughly classified in [133] as: *Best-Effort* (BE) service and *Guaranteed Service* (GS). According to this classification, state-of-the-art works may be classified into two categories: *BE-oriented NoC* and *GS-oriented NoC*.

BE-oriented NoCs provide a higher resource utilization and treat all the communicating pairs equally and thus ignore the hard real-time guarantees. In *GS-oriented NoC*, the state-of-the-art approaches to provide time related guarantees are classified as (1) Establishing a connection prior to the communication and reserving resources and then sharing time slots among multiple available transactions (i.e. *Æthereal*) [65]. The contention among different data traffic patterns is fully handled during the connection establishment time; (2) Implementing different service classes and exploring the advantages of wormhole routing and *Virtual Channels* (VCs). A *virtual circuit*, a virtual connection, is built by exploiting prioritized service classes (i.e. QNoC) [23, 135].

The connection-oriented technique provides the contention-free routing by reserving the resources in advance. Inefficient resource reservation leads to unused bandwidth. The network is underutilized in *Variable-Bit-Rates* (VBR) transactions. The connection setup stage is an overhead for such architectures and thus, they suffer from lack of scalability. The second method performs and implements quality on top of packet-switched communication, generally using the wormhole routing scheme. It provides better performance results for *VBR* [71, 72], provides higher relative guarantees, but may fail in tight requirements. The problem also lies in the specification and granularity of the service classes which leads to unused bandwidth. These approaches use a simple *RR* link arbitration.

Existing service-class-related works can be broadly classified into two parting directions. The first group of the architectures presented in [23, 71, 72] have used a fixed number of service

classes. Each service class has a particular *Virtual Channel Buffer* (VCB) in each output port. They do not provide a 100% tight guarantee for each of the transaction (*QNoC* [23, 135] provides on an average slightly more than 99% guarantee but fails to achieve a 100% guarantee, which is a mandatory constraint for a safety critical embedded system), and these architectures do not have a clear and fine-granular classification for each transaction. Therefore, the bandwidth is not allocated fairly to increase resource utilization. The arbitration of the *VCBs* to the output port is dependent on their corresponding service class. Concurrent transactions use the packet-based *RR* ordering of input ports for the awaiting transmission of packets within the same service class. The basic arbitration principle among different service classes is: once a higher priority packet appears on one of the input ports, transmission of the current packet is preempted and the higher priority packet gets through.

The second group of QoS-supported service-class-associated NoCs is presented in [92, 170]. In this group the differentiation of transactions bandwidth requirement is done on the spatial domain. In a link of available bandwidth b, by allocating n number of *VCBs* (out of m *VCBs* in total) a total of $(b/m \times n)$ bandwidth may be provided. The virtual channels are arbitrated in a *RR* fashion. It consumes a lot of buffers in terms of *VCB* implementation for fine granularity and suffers from starvation for some transactions. In [170], apart from [92], *VCBs* assigned for *BE* traffic can be taken by *GS* at connection establishment time.

The proposed QoS-supported system-on-chip communication architecture in this thesis has taken the advantages of both the service-class-based and the connection-based approaches (see Chapter 3) and it has some novel architecture level differences compared to the current service-class-related state-of-the-art NoC works. However, the proposed QoS-supported system-on-chip communication architecture uses a wormhole switching scheme like [23, 71, 72, 92, 170] and thus utilizes virtual channel implementation.

2.2.4 Buffer Minimization

Buffers are used to facilitate the flow control mechanism in the router. In QoS-supported NoC where a *wormhole-based* routing scheme is used, the logical connections are implemented using buffers. In a typical system-on-chip router, buffers take the premier share of the silicon area of the NoC and consequently their size should be kept as minimum as possible. Work in [173] shows that the buffer area amounts to about 60% of the entire router area. Work in [122] shows that by increasing the buffer size at each input channel from 2 to 3 words, the router area of a 4x4 NoC increases by 30% or more. Again, depending on the network workload, increasing the buffer size may reduce the network latency by orders of magnitude. Therefore, due to diverse traffic patterns exhibited from application-specific NoCs, it might be beneficial to allocate more buffering resources only to the heavy loaded (traffic hotspot) areas. Authors in [78] show that the values for average packet latency that may be obtained for the same total amount of buffering space are very different.

Early major work [78] in the area of buffer reduction for application-specific NoC has proposed a methodology to provide application-specific buffer size allocation but did not consider the QoS issues. The proposed methodology assumes *deterministic routing*, *store-and-forward* or *virtual cut-through* switching schemes, and a *poisson* traffic distribution for all packets injected

in the network. Under these assumptions, the authors derive the blocking rate of each individual channel and then add more buffering resources only to the higher utilized channels. In [139], authors have presented different system-on-chip buffer properties and then have described an optimized buffer implementation methodology for the *Proteo* networks-on-chip architecture. They further have reported gate-level area estimation and have analyzed the performance of the network and buffer utilization across the network. In [34], authors have investigated the impact of FIFO size on the interconnect throughput for single source and single sink interconnect scenarios. Work in [41] has presented an algorithm to find the minimal decoupling buffer sizes for a NoC using TDMA and credit-based end-to-end flow-control where the performance constraints of the applications running on the system are maintained. Their exploration shows that their method results in a 84% reduction of the total NoC buffer area. Authors in [172] presented a zero-efficient buffer design as well as an error control scheme. They have acheived up to 43% energy saving using a 90nm CMOS process technology.

In [55, 56], buffer minimization using a two-step methodology to optimize the number of *VCBs* in a QoS-parameterized application-specific NoC architecture are presented. In Chapter 4, this methodology is explained in detail. As a first step, a multi-objective mapping algorithm that considers both the communication volume and the number of virtual channel buffers during NoC mapping based on a modified *Ant Colony Optimization* (ACO) algorithm is used. This approach is orthogonal to any application driven traffic model. In the second step, the traffic characteristics of the application is considered. It is observed that a wide range of digital media applications follow the *poisson distribution* and therefore, an analytical approach is provided to optimize further the number of virtual channel buffers depending on QoS parameter q and the application-specific traffic model. On the other hand, there are a lot of real applications which exhibit traffic patterns which are very different compared to the *poisson model* therefore, the derivation of analytical models for performance evaluation becomes much more difficult [167]. The research in this direction is also very challenging.

2.2.5 Runtime Adaptivity in Networks-on-Chip Design

It is already discussed that embedded systems applications have grown considerably in size and complexity in recent years. Therefore, such a complex embedded system must be able to handle also those situations efficiently that were unpredictable at design time. In this case, the system needs to adapt itself to the new situation and therefore, it needs to be designed with the capability of self-adaptiveness in mind. In the domain of self-organization, IBM's Autonomic Computing Initiative [75] deals with self-organization of IT servers in networks. Recently, the idea of adaptivity/self-organization has been borrowed for the future SoC design [12, 75, 100, 138].

As far as NoCs go, adaptivity would be very beneficial since a fixed topology with a fixed number of buffers etc. are most often only efficient in certain scenarios and represents an overdesign in all other cases (for details see Chapter 5). NoCs suffer from the high overhead a designer has to pay. Therefore, commercial deployment of NoCs is restricted to some specific projects. Mainstream deployment of NoCs in MPSoCs is with so-far-proposed approaches rather unlikely. The state-of-the-art NoCs [14, 23, 133] though presenting a scalable communication infrastructure are still too inflexible to dynamically support the communication among modules

in a system with heavily varying workloads and/or changing constraints. A static NoC is defined as a design-time parameterized architecture with a fixed routing scheme and a fixed number of allowed virtual connections at each output port. They are generally tailor-made for a certain application or application domain. In the following, a review of the related work that deals with on-demand interconnection schemes in different problem spaces that are closest to the proposed novel concept in presented.

The *FLUX* network [169] has been proposed to establish interconnection on-demand before or during program execution by adapting the physical network. While creating point-to-point connections, the deployed links need to be reserved. Hence, several links must be present to be able to adapt to arbitrary applications. The connections implement a circuit switching network approach and are non-scalable. The required number of links increases exponentially relative to the number of processing elements. In [150], authors have proposed a power-aware network whose links are turned *on* and *off* in response to bursts and dips of traffic on-demand. In their approach, future traffic characteristics are assumed to be predicable based on recent traffic patterns. Hard-to-predict events like, for instance, rapid motion in a video sequence cannot efficiently be treated. In [21, 22, 82, 83, 88, 103, 131, 141, 156], dynamic communication infrastructures among dynamically placed modules on a reconfigurable device are presented. The system is built on top of a reconfigurable hardware, e.g. on an FPGA. All these approaches are limited to reconfigurable devices. It is not scalable to faster and efficient ASIC while the proposed runtime adaptive system-on-chip communication architecture is orthogonal to any hardware implementation.

2.2.6 Application Mapping onto Networks-on-Chip

The application mapping onto a NoC architecture can be broadly classified into two categories: (1) design-time application mapping for the application-specific NoCs and (2) runtime application mapping for the adaptive and the general purpose NoC architectures. In this thesis, both the design-time application mapping for the application-specific NoC detailed in Chapter 4 and runtime application mapping required for the adaptive system-on-chip communication architecture detailed in Chapter 6 are presented.

Several design-time application mapping algorithms have been proposed in early works. In [77], a mapping using the *branch-and-bound* algorithm has been proposed. Here, the communication related energy consumption has been used as a goal function. The energy consumption is inherently minimized by reducing the amount of communication volume. *Genetic* algorithm based mapping algorithms have been introduced in [98, 143]. In these papers, efforts are made on the modification of the *genetic algorithm* to fit into NoC mapping. In [98], the optimization criteria is overall-execution time. In [143], cycle crossover and coordinate crossover are used to make sure that the offspring is legal. [77, 143] determine a network assignment which is designed to minimize the power consumption by reducing the communication volume. In [70], authors have proposed a heuristic-based (UMARS) mapping algorithm by keeping mostly communicating tasks together to each other and to the center of the NoC. In general, all these previous works considered only the communication volume as the basis of their optimization. The minimization of communication volume is definitely an important criteria to be minimized but in this thesis other criteria (e.g. buffer area) are also addressed to be minimized during

application-specific NoC mapping. Work in [5] presents a similar approach (both considered a multi-objective optimization during mapping) to multi-objective exploration of the mapping space of a Mesh-based application-specific NoC architecture. Based on evolutionary computing techniques (*genetic algorithm* is used here), they try to obtain the *Pareto* mappings that optimize performance and power consumption. In this thesis, in Chapter 4, a mapping algorithm that considers the amount of communication volume and at the same time considers the minimization of buffer space in a QoS supported application-specific NoC are presented. This NoC mapping is done in the *offline* stage of the application-specific NoC design.

The adaptive and the general-purpose systems, which changes its configuration over time, require a (re-)mapping/runtime mapping of the application. Possible reasons for the necessity of a runtime application mapping are listed in Chapter 6 (the proposed runtime application mapping algorithm, the *Agent-based Distributed Application Mapping* (ADAM) is orthogonal to any scheme that requires runtime application mapping [58]). Few research work has been conducted in the domain of runtime application mapping for MPSoC. In [147], authors extend the *MinWeight* algorithm proposed in [28] for solving the problem of runtime task assignment on heterogeneous processors. The task graphs are restricted to a small number of vertices or a large number of vertices with degree no more than two. Authors in [31] investigate the performance of several mapping heuristics promising for runtime use in NoC-based MPSoCs with dynamic workloads, targeting NoC congestion minimization. The work presented in [37] and [38] propose an efficient technique for runtime application mapping onto a homogeneous NoC platform with multiple voltage levels. Their work is limited to a homogeneous architecture and a separate control network besides the data network is used which represents an extra overhead in terms of area and energy consumption. All the state-of-the-art runtime application mapping works [31, 37, 38, 147] have used a *Centralized Manager* (CM) for conducting the mapping job which is not scalable in the era of hundreds or even thousands cores that may soon be integrated on a MPSoC. It suffers from a single point of failure, larger volume of *monitoring traffic* (*monitoring-traffic* is the traffic which is caused by collecting information about the state of the tiles, during each instance of mapping), central point of communication around the *CM* (hot-spot), and scalability issues.

The concept of task migration is an integral part of the runtime application mapping. The study of task migration to move a currently executing tasks between different processors which are connected by a network has already been a research focus in the distributed and parallel computing domain [137, 148]. Now it is used to facilitate runtime application mapping in adaptive heterogeneous MPSoC. Work presented in [17, 119] discuss the issues related to the task migration concept for MPSoC design, i.e. the cost to interrupt a given task, save its context, transmit all data to a new IP, and restart the task in the new IP. In the proposed *ADAM* algorithm, this approach is used though the detail of task migration is out of the scope of this thesis.

2.2.7 Runtime Traffic Observability

A runtime observability scheme to achieve a successful adaptation is used in the proposed system-on-chip communication architecture in this thesis. In order to assure a certain degree of

QoS (e.g. guarantees in performance and required bandwidth), a feedback of the current system state must be available. This can be achieved through runtime observability in an adaptive system-on-chip communication architecture. If a runtime observability infrastructure comes with only a small hardware overhead and some small communication overhead that would then more than compensate the degree of freedom achieved using a successful adaptation. Within this thesis an event-based NoC monitoring component at *architecture-level* is presented. In this thesis, *runtime observability* is denoted as a complete infrastructure and *monitoring* as a hardware component attached to each tile that offers runtime observability. The prime challenges for runtime observability are scalability, flexibility, non-intrusiveness, real-time capabilities, and cost. In order for the monitoring components to be as non-intrusive as possible, they need to keep their interference with normal system execution, so-called *probe effects* [90] at a minimum. An example of these effects would be the sending of monitoring packets (the traffic that is generated during runtime observation of the system state are described in details later in this thesis in Chapter 7) through the regular point-to-point data links between routers. If these packets are injected too rapidly, they demand resources which otherwise may have been used for regular traffic. It is therefore, necessary to limit *monitoring traffic* by keeping its bandwidth usage and occurrence frequency minimal.

In general, it may be stated that observability capabilities for system-on-chip communication have not been proactively investigated in the NoC domain. In [120], authors have mentioned an operating system controlled system-on-chip runtime collection of traffic statistics at the *Network Interface* (NI) to optimize the usage of communication resources in a NoC using a centralized resource management scheme. In [39, 40], authors have presented a generic event-based NoC Monitoring Service (NoCMS) for *Æthereal*. Their observability infrastructure is not designed specifically to detect faults in network traffic during runtime adaptation but instead just to gather statistics (debugging) on the NoC behavior. In [165], authors further used the monitoring probes proposed in [39, 40] for a new communication service to control congestion. In a nutshell, the *Æthereal* monitoring framework is not used to adapt the underlying system-on-chip communication architecture. Runtime observability is also not included in general-purpose NoCs (e.g. [66, 76, 173]) as these architectures do not adapt at the *architecture-level* to increase resource utilization. Recently, authors in [126] have also focused on self-monitoring components for NoCs considering reliability factors.

2.3 Conclusion

In summary, this chapter presents some of the most important related works in the domain of NoC design that help to explain the thesis. The related works presented here are further mentioned in various places in the rest of the thesis to demonstrate the specific novel contribution of this thesis.

Chapter 3

QoS-supported System-on-Chip Communication Architecture

The increasing complexity of *Multi-Processor System-on-Chip* (MPSoC) has invoked a paradigm shift in the system-on-chip communication architecture design. In the previous chapters it is discussed, that a *Networks-on-Chip* (NoC)-based architecture is seen to be the most promising solution for the future MPSoC. A NoC is a collection of tiles and each tile is commonly made up of three components (see Definition 4): a *router*, a *Network Interface* (NI), and the *Intellectual Property* (IP). The IP may be composed of either *Processing Elements* (PEs), i.e. CPUs or *Storing Elements* (SEs), e.g. RAM. IPs are mainly third party components and their implementation is not considered to be a part of NoC design. Therefore, the communication scheme needs to be transparent to allow the deployment of diverse IPs in the given NoC architecture. The transparency is achieved through the NI. The NI is located between an IP and its router and is responsible for translating data from the IP representation into a format understood by the network. This may be done by encapsulating the IP data with necessary header information needed by the routers. The routers are responsible for choosing the route, that is followed by the data in the network. In short, the process is accomplished by decoding the header information and by making a decision based on the implemented routing algorithm.

One of the major challenges in NoC design is the large design space spanned by an extensive set of (partly) independent parameters. Only a set of carefully adapted parameters will allow unveiling the potential benefits of a NoC. The adapted parameters have to be integrated with the *Quality-of-Service* (QoS) requirements for designing a time-critical embedded system. The definition of QoS is extremely diverse and application specific in embedded systems. The requirements of an embedded system application (i.e. TV broadcasting in a PDA) is time critical and driven by the current work load on that device. Those quality limits are also restricted by human audio-visual perception. The underlying QoS parameters latency, throughput and jitter in such performance phenomena, bounds must be provided by the system. Other sorts of services like security data and memory read/write or internet browsing may have varying quality requirements. The levels of reliability (i.e. lossless data transactions) also add more parameters to the system design. Considering the importance of QoS, a QoS-supported system-on-chip communication architecture is presented in this chapter. The basic router architecture considering the QoS issues are kept similar with some basic assumptions for the application-specific NoC presented in Chapter 4 and the adaptive NoC presented in Chapter 5. Both architectures use a *wormhole-based* routing algorithm, virtual channels implemented using buffer blocks,

pipelined architecture, and a route established mechanism for each transaction from the source to the destination during data transmission. The architecture is explained in detail later in this chapter. Specific features related to the application-specific NoC are presented in Chapter 4 and related to the adaptive NoC in Chapter 5.

The novel contributions in the scope of this chapter are as follows:
(1) A scalable and configurable cycle-accurate QoS-supported NoC as a system-on-chip communication architecture, on top of which a case study analysis of an application-specific NoC, as well as the proposed adaptive NoC are built are presented in details here (published in [57, 60]).
(2) In this chapter, a novel link arbitration algorithm named as *bounded-arbitration-algorithm* which provides *transaction-level* bandwidth guarantees considering the upper and the lower bounds of each transaction is detailed (published in [59]).

3.1 Definitions

The application characterization graph and other associated definitions that are necessary for the explanation of the QoS-supported NoC architecture are described below:

Definition 1: An application *Task Graph* (TG) is a directed graph $G = (T, F)$, where T is the set of all tasks (computational module), $t_i \in T$ describing a task and $f_{i,j} \in F$ represents the data transmission from task t_i to t_j. Each $f_{i,j}$ has associated a value $V(f_{i,j})$ describing the communication volume in bits transmitted between the tasks t_i and t_j. Moreover, each t_i may be annotated with several information, e.g. execution time on different types of processing elements, energy consumption depending on the type of the processing elements, task deadline, etc. The *task graph set* G is the set of all task graphs G_k.
Definition 2: A *Service Class* (SC), $SC = Normalized(BW)$ is a normalized value associated with each flow $f_{i,j} \in F$ in $G = (T, F)$ explaining range of bandwidth requirement between communicating partners. The SC is determined during application exploration and may be influenced by other QoS design parameters, i.e. latency, jitter, etc .
Definition 3: A *Service Class associated Task Graph* (SCTG) is a directed graph $SG(T, F) \subset G = (T, F)$, where the vertices ***T*** represent the set of partitioned computational modules referred to as tasks $t_i \in T$ and ***F*** represents the flows between tasks (similar to Definition 1). Each flow $f_{i,j} \in F$ between tasks t_i and t_j is associated with a communication volume $v(f_{i,j})$ and a corresponding service class assignment $SC_{i,j}$. The service class assignment between t_i and t_j is evaluated by the designers according to the QoS parameters.
Definition 4: A *tile*, N, is composed of mainly three mutually exclusive subsets, N_{IP}, N_{NI} and N_R which represent the set of Intellectual Properties (IPs), i.e. heterogeneous PEs, NI and routers respectively.
Definition 5: A *Virtual Channel* (VC) is a unidirectional logical or virtual connection between the tile N_i and N_j multiplexed with other ***VCs*** across the physical channel $PY_{i,j}$. Each ***VC*** is realized by an independently managed pair of message buffers referred to as ***Virtual Channel Buffer*** (VCB).
Definition 6: A NoC's *Physical Network* (PN) is a directed graph $PN = (N, V, B_t, r)$. N is a set of tiles n_i and $v_{i,j} \in V$ represent an edge, the physical channel between two tiles n_i and

n_j. Each tile has a current buffer configuration at time t, $b_{i,t} \in B_t$ represents the state of a buffer assignment to individual output ports. The network also has a routing function r which determines the routes taken. In case of an application-specific NoC design, both B_t and r are fixed at design time but may change for the adaptive NoC architecture.

Definition 7: The *application mapping* function is a function $MP_t : G \mapsto PN$ which maps an application onto the physical NoC network either at design time for the application-specific NoC or at runtime for the adaptive NoC. The function changes in discrete time intervals.

Definition 8: A *routing* function $r : N \times N \rightarrow V$, $r : (n_i, n_k) \mapsto v_{i,j}$ returns a route $v_{i,j}$ away from the current *PE* (n_i) given the input port for each transaction and the destination n_k.

Definition 9: A *transaction* (TR) is a message that needs to be sent from a source tile N_i to destination tile N_j at time t. The size of the message varies considering different parameters, i.e. time, application, source task, QoS, etc . The frequency and the size of the message determines the required bandwidth for the communicating partners. The transaction is divided into packets and packets are further divided into flits for the *wormhole-based* routing.

Other definitions that are necessary to understand the concept of the proposed adaptive NoC architecture are explained in Chapter 5.

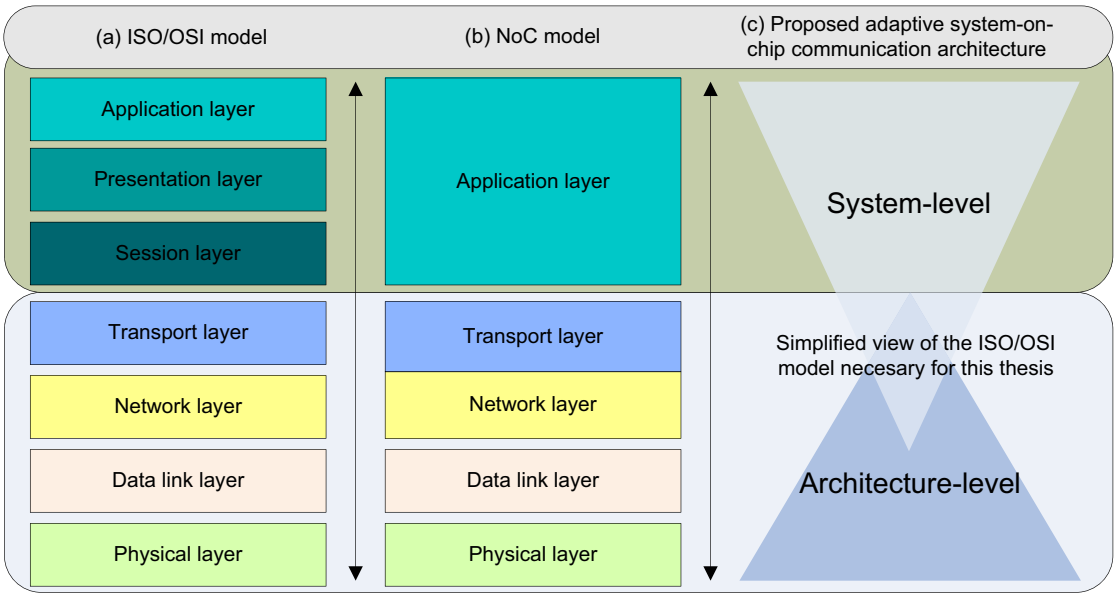


Figure 3.1: Adaptation of the ISO/OSI reference model

3.2 Packet-based Communication

There are two common ways of transmitting data through a network, *circuit switching* and *packet switching* [134]. With *circuit switching*, a complete route from the source to the destination is allocated to a connection and therefore, no intermediate router processing is performed (a very simple switch is used instead of a router). This dedicated route results in very low latency comparable to that of bus based systems. However, since the entire route is allocated to

the connection, this blocks all links included in the route. Therefore, *circuit switching* severely limits the number of simultaneous connections by the network. In general, a single connection may not require the maximum link bandwidth capacity resulting in an inefficient bandwidth utilization.

In a *packet switching* communication, a transaction is divided into several smaller data units (packets) and these are then transmitted separately through the network. To accomplish this, each packet needs to be buffered at each router during route traversal. The necessary header information is decoded and evaluated to decide on the subsequent route to be taken by the packet. *Packet switching* does not reserve the entire route. Therefore, links are not blocked by one transaction and because of resource multiplexing, bandwidth utilization is increased. *Packet switching* however also comes with some drawbacks, i.e. in *circuit switching*, bandwidth and latency of communication are guaranteed due to dedicated links but this is not the case in *packet switching* when multiple packets share links. Ensuring bandwidth and latency constrains to be met requires additional QoS mechanisms [101] in *packet switching*. In this thesis, a hybrid approach by using a connection-oriented approach on top of packet based communication is used to ensure QoS.

A packet-based communication architecture and its protocol stack can be logically modeled in layers. The ISO/OSI [162] model is such a widespread reference model where hierarchical communication layers use the services of lower layers to build more abstract, improved services. Originally the model consists of seven layers (see Figure 3.1 (a)). In NoC design, it is also common practice to adapt a hierarchical communication model because of complexity, reusability, and even for energy reasons. For the system-on-chip communication architecture seven layered models appear to be oversized and therefore, several recent works have envisaged three to five layered models. An adaptation to a three layered model of NoC with a focus on energy consumption is done in [13]. This *micro-network* stack consists of the *software*, *architecture&control* and *physical layer*. *The Nostrum Backbone* [107] introduces five layers, merging the three top layers to one (see Figure 3.1 (b)). The *physical layer* in system-on-chip communication architectures handles the reliability issues, i.e. data loss, crosstalk, energy dissipation, etc. The *data link layer* implements the switching technique and the medium access control. Routing, flow control and error control are the predominant tasks of the *network layer*. The *transport layer* provides a transport of data with a defined QoS. Above mentioned four layers are considered as the *architecture-level* (see details of *architecture-level* in Chapter 5) part in the scope of this thesis (see Figure 3.1 (c)). The PEs attached to tiles provide the *session layer*, the *presentation layer*, the *application management layer*, and part of the *network layer* and the *transport layer*. These layers are considered as the *system-level* (see details of *system-level* in Chapter 5) of the adaptive NoC architecture (see Figure 3.1 (c)).

3.2.1 Wormhole Switching

In the proposed packet-based communication scheme, flow control occurs in two steps: (1) at *transaction-level*: a transaction which may be composed of several packets travel from the source to the destination after a route is established between these two communicating partners.

This flow control provides required QoS at *transaction-level*. (2) at *packet-level*: the transfer of a packet across the physical channel between the adjacent routers may take several cycles or steps. Therefore, a *packet-level* flow control mechanism which is generally termed as switching strategy is required.

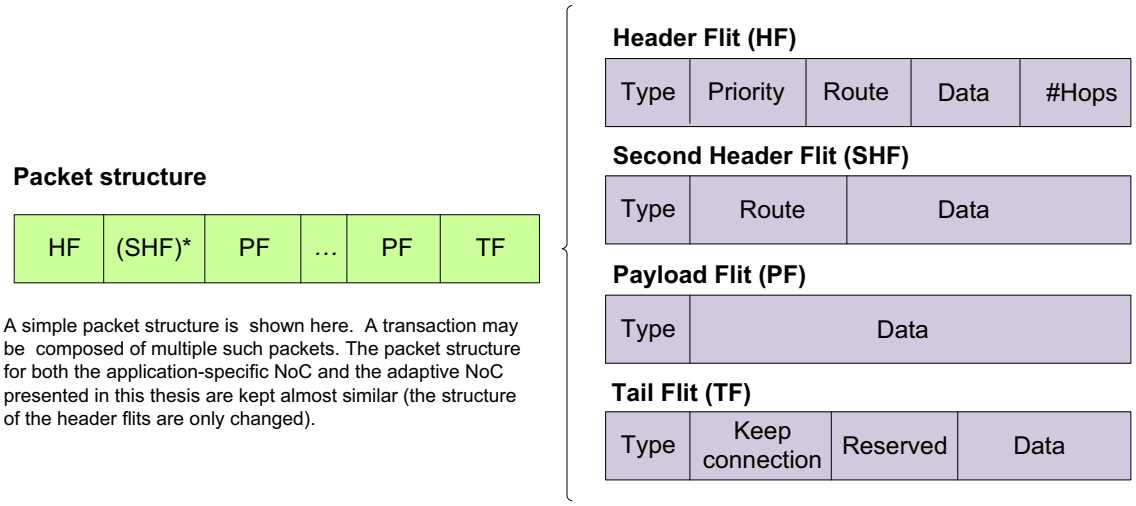


Figure 3.2: Flit composition of a packet for the application-specific NoC

There have been several switching strategies studied for the system-on-chip communication architecture design, i.e. *wormhole switching*, *store-and-forward switching*, *virtual cut-through switching*, etc. In this work, *wormhole-based* switching strategy is employed because of its performance, easy implementation and buffer space requirement [47]. Each packet may be broken into message flow control units named as *flits* (flow units). These subdivisions of the packets are all of the same length: typical sizes include 32 bits [59] to 96 bits [39]. Due to channel width constraints multiple physical channel cycles may be required to transfer a single flit. The amount of data unit that can be send within a single cycle across a link is termed as *phits*. In this thesis, the term *flits* and *phits* have been used interchangeably and kept similar in size. The packet format comprising different types of flits is presented in the following subsection. Generally, there are four types of flits (multiple types of *header flit* are possible). The first one named as the *header flit*, includes the routing information. The routers then set up the route which the *header flit* and all subsequent flits will take. Following the *header flit*, the *payload flits* then transmit the majority of the packet payload until the packet is finished by a *tail flit*. In wormhole routing, buffering only has to be done on the *flit-level* and not on the *packet-level* [47]. Since all except the *header flit* do not contain routing information, it is required that routing on the flit level is static and is not altered by a router for the duration of the packet route traversal. Therefore, each flit of a packet must follow the same route.

Flow control mechanism in the *packet-level* of the *wormhole switching* is implemented using a *Go-Back-N ARQ* [168] (Automatic Repeat ReQuest) using *Negative ACKnowledgments* (NACK). Therefore, successfully transmitted flits do not need to be acknowledged. However, once a transmission error has occurred, the receiving router sends a NACK to the previous router. This then sends a *Go-Back-N* signal to the *VCB* invoking a resending of the previous N

packets. The value N may be determined by the number of cycles of the processing delay n and the link propagation delay m. Thus N is at most $2m + n$.

3.2.2 Packet Structure

The topology used for the network is a regular 2D Mesh. This facilitates the NoC to be highly scalable and simplifies the design of the routing algorithms. Packets are generally composed of four types of flits for both the application-specific NoC and for the adaptive NoC: a *Header Flit* (HF), a following *Second Header Flit* (SHF), *Payload Flit* (PF), and a *Tail Flit* (TF) (see Figure 3.2 for the application-specific NoC packet structure and Figure 3.3 for the adaptive NoC packet structure). The first flit of a packet is the *HF* which contains the information needed by the network to process the packet. The *HFs* are different for the application-specific NoC and for the adaptive NoC. In the application-specific NoC, the *HF* contains a type field which defines its type, a priority field, and the routing information. The size of those fields are design-time parameterized. The routing information consists of direction bits for every *positional router* (the term *positional router* is used in this thesis to highlight the need of different orientations of input and output ports in irregular NoC topologies) and a field that represents the number of hops the packet has to travel. If there is not enough space for the routing information in one *HF*, the following header flits *SHF* may contain such routing information. In a smaller network, the available space in the *HF* for routing information is sufficient. A *HF* may also contain data if there is enough space available.

Header flit parts	**Size** $n \times m$	**Size** 4×4[§] **2 Priorities, 8 BW Slots**
Type	2 bit	2 bit
Priority	$\log_2$ (Priorities)	1 bit
Destination	$\log_2(n \times m)$	4 bit
Source	$\log_2(n \times m)$	4 bit
Transaction ID	source + 2	6 bit
TTL	$\log_2(n + m + 2x)$[†]	4 bit
BW	$\log_2$ (BW Slots)	3 bit

[§]rest of the bits of the 32 bit flit are free and not shown in Figure 3.3

[†] x = number of misroutes

Table 3.1: Size of the header flit contents in an n x m adaptive NoC and in a 4 x 4 adaptive NoC

In an adaptive NoC, the *HF* includes the packet priority and type (similar to *HF* in the application-specific NoC), its destination address, a unique transaction identifier, a *Time-To-Live* (TTL) counter, bandwidth requirements (in slots), and the transaction source. An example of their length for the *HF* is given in Table 3.1 for both a general $n \times m$ adaptive NoC and for an exemplary 4×4 adaptive NoC. The *HF* is followed by *PFs* which carry most of the packet contents. Usually, there are several *PFs*. The last flit to arrive is the *TF*, which signals the end of the packet. The *TF* also contains a *keep-connection* flag. If this is set, the router keeps resources allocated to the transaction for future packets.

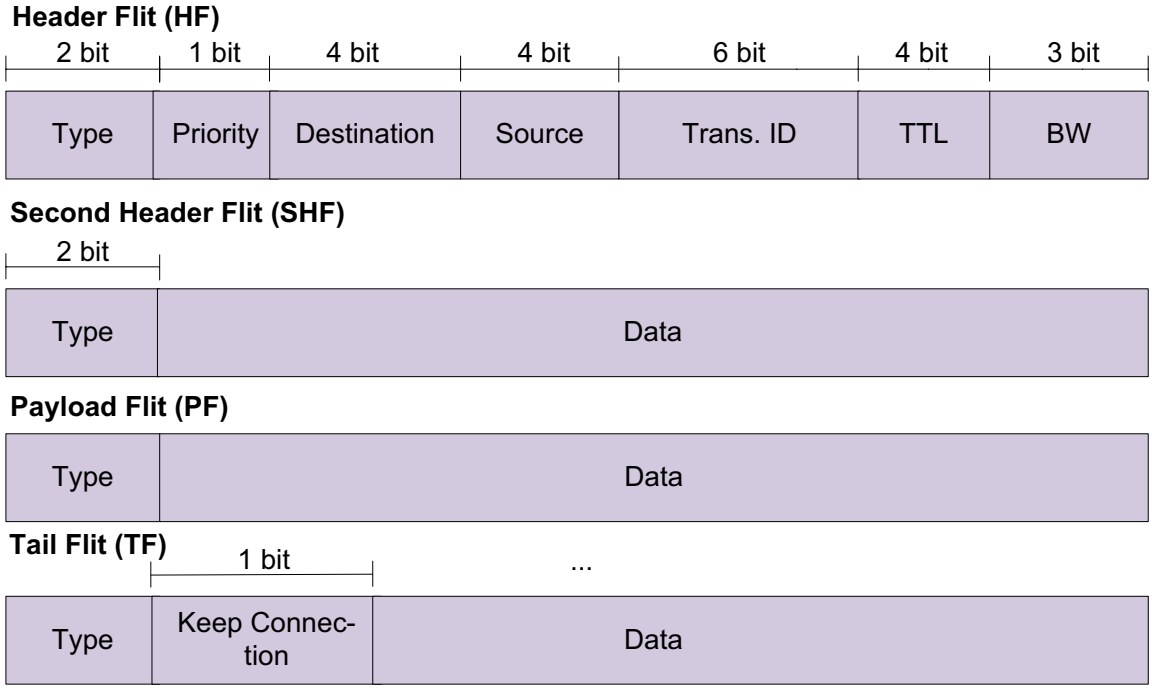


Figure 3.3: Exemplary flit composition of a 4 x 4 adaptive NoC

3.2.3 Pipeline Stages of the Router Architecture

The NoC architecture needs to be simple and parameterizable. The proposed QoS-supported NoC is pipelined like the state-of-the-art *Xpipe* architecture [14]. Several design related parameters, i.e. topology, routing, switching, etc. have already been discussed earlier. The links between two routers are pipelined. Therefore, irregular topology with a favorable routing algorithm may also be designed within the frame-work. The topology customization using irregular topology has not been included in the scope of this thesis.

To keep the NoC predictable in terms of cycle counts, the *router-to-router* connections and also the *router-to-IP* connections are cycle-accurate. To demonstrate the basic QoS-supported NoC architecture, one output port unit for a simple *positional router* is shown in Figure 3.4. Both the application-specific NoC (see Chapter 4) and the adaptive NoC (see Chapter 5) are designed on top of this basic QoS-supported architecture. The *Input Decoder* (ID) decodes the incoming packets for the appropriate output port. *Output Arbiter* (OA) arbitrates the output directions among the incoming packets. The following steps implement the QoS during the transmission. To allow a higher operating frequency, the router structure is deeply pipelined forming several pipeline stages at the output port. No error control unit is added in the architecture as the network is assumed to be reliable. Hence, no end-to-end retransmission at *packet-level* is required and thus, the throughput and the latency bounds can be measured cycle accurately. The two necessary stages for a guaranteed transaction, the *admission control stage* and the *packet control stage* are also highlighted in the figure. The *Virtual Channel Arbiter (VCA)* implements the admission control by runtime *VCB* assignment and the *Physical Channel Arbiter/Link Control*

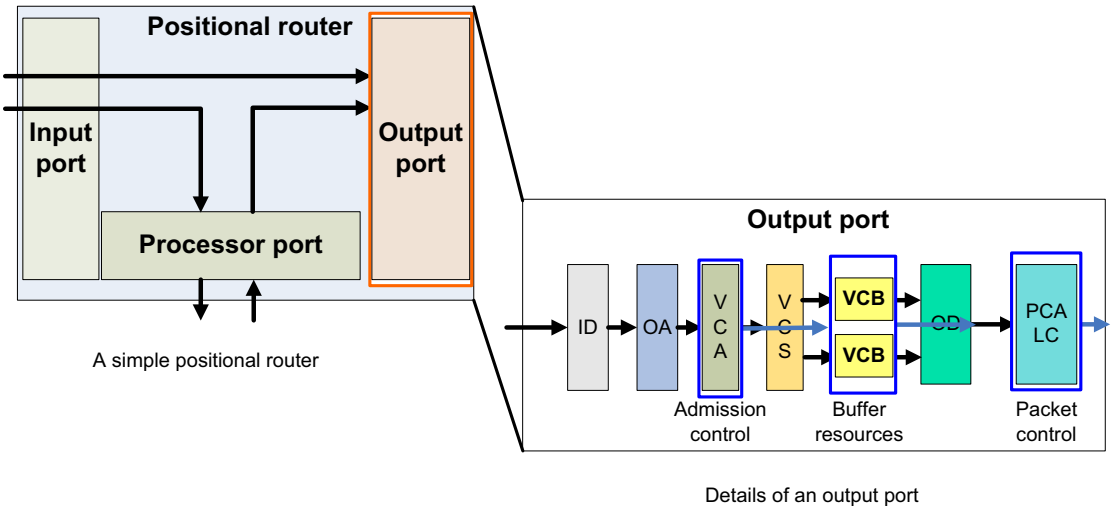


Figure 3.4: The basic QoS-supported router architecture

(PCA/LC) handles the link arbitration (a novel link arbitration algorithm named as *bounded-arbitration-algorithm* is presented later in this chapter).

In Figure 3.5, an exemplary transmission of a packet for an application-specific NoC is presented. An IP core initiates a transmission by utilizing its implemented protocol, i.e. OCP [121]. The NI accepts the incoming request and packetizes the data. Therefore, a classification of the incoming message is made. A service class represented by a priority in the *HF* and the connection management as a bit in the *TF* are determined. The route to the destination IP is looked-up in a routing table located in the NI and added to the header information (in the adaptive NoC the route is calculated distributively in each intermediate router rather being calculated in NI centrally). A *route-length* defines the number of hops the packet has to travel to the destination. It is decreased by every router along the route. On reaching its destination this parameter equals zero. With this number of hops the router is able to receive the direction of the packet at the next hop. The direction information consists of bits that define an output direction of the *positional router*. Every output direction of a router can be selected by a unique bit-sequence. If the packet is scheduled for the transmission out of the buffer, a request has to be sent first to reserve a *VCB* in the output port of the router. Every *VCB* or NI needs to be reserved in advance to keep the order of the data. The procedure is the same for sending to *VCBs* or NIs. Therefore, the process of a request is only described for the *VCBs*.

The shared resources in such a QoS-supported architecture are: (1) links and (2) buffers. The buffers are used to implement the *VCs*. *VCBs* are highlighted in Figure 3.4. *VCs* differentiate among the transactions during the communication. The links are shared among all the transactions for bandwidth allocation. In this architecture, the resource sharing of links and its utilization increment in terms of fair bandwidth allocation are considered. The buffer utilization is increased for the application-specific NoC by minimizing the number of *VCBs* at design time (see Chapter 4) and for the adaptive NoC by using a central pool of *VCBs* and then assigning those *VCBs* at runtime (see Chapter 7).

Similarities	
	Application-specific NoC and **Adaptive NoC**
Quality-of-service	Both the application-specific NoC and the adaptive NoC supports quality-of-service. The QoS is implemented by VCB assignment in each output port of the routers from the source to the destination and the arbitration algorithm provides the necessary flow control mechanism by the proposed bounded-arbitration-algorithm. A connection is established for each transaction.
Pipelined architecture	A pipelined architecture is used for both the application-specific NoC and the adaptive NoC. The basic architecture shown in Figure 3.4 is reused also for the adaptive NoC presented in Chapter 5.
Switching strategy	Both use wormhole switching strategy.
Topology	In general, a 2D Mesh topology is used for both the architectures.
Floorplanning	No floorplanning is considered in the scope of the thesis.

Differences or addition (the adaptive NoC) in the router architecture		
	Application-specific NoC	**Adaptive NoC**
Packet structure	The complete route from the source to the destination is given in the header flit of the first packet (see Figure 3.2).	The header flit contains the source and destination address (see Figure 3.3).
Virtual channel buffer	They are fixed to each output port at design time. They may be minimized at design time.	A central pool of VCBs are used.VCBs are assigned to each output port on-demand.
Routing algorithm	Source-based deterministic routing.	Distributed routing algorithm.
Bandwidth	Two simplex links are used to form the bi-directional communication between adjacent routers.	Two *half-duplex links* are used to form the bi-directional links. Bandwidth may be configured at runtime.
Mapping algorithm	Design-time mapping algorithms are used.	Runtime application mapping is used (see Chapter 5).
Monitoring	No monitoring hardware component is required.	Monitoring hardware to support *architecture-level* adaptation is added.

Table 3.2: Similarities/differences of the application-specific and the adaptive NoC architectures

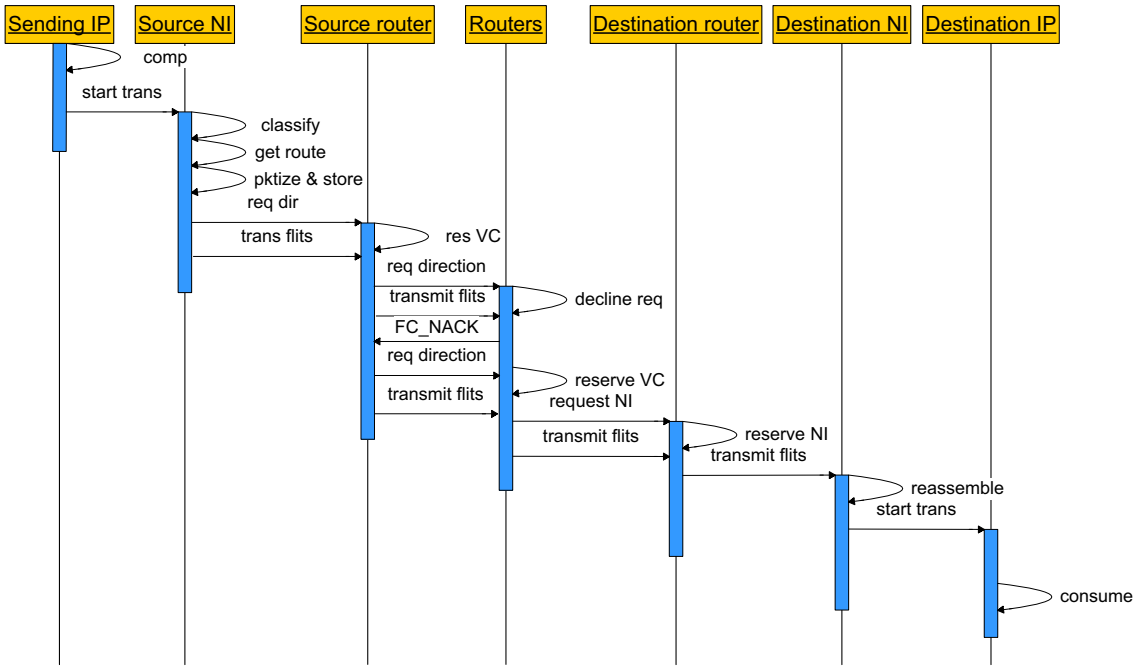


Figure 3.5: Example of a packet transmission and requesting directions

3.3 Guaranteed Communication on Top of the Packet-switched Network

The proposed system-on-chip communication architecture is built to support QoS on top of a packet based communication. As mentioned earlier, the architecture uses a *wormhole-based* routing algorithm and multiple pipeline stages are used for data transmission in each router. To accomplish multiple parallel transaction over a single physical interconnect between adjacent routers, *VCs* are used. *VCs* are realized by implementing *VCBs* in each output port of a router. The guaranteed transaction in terms of bandwidth and latency is implemented by exploiting the concept of a connection-establishment from the source to the destination. Each transaction allocates a single *VCB* in each router from the source to the destination throughout the complete route. The *VCB* assignment is performed at runtime during connection-establishment time. The link arbitration algorithm manages the flow control for each transaction depending on its requirement. Some parameters and features that are changed or added in the adaptive NoC are given as follows:

- *Packet structure*: a source-based routing algorithm is used in the application-specific NoC but a distributed routing algorithm is used in the adaptive NoC. Therefore, the *header flit* of the application-specific NoC contains the complete route before the transaction starts. On the other hand, in the adaptive NoC, the *header flit* only contains the source and the destination address and the route is calculated during the first *header flit* transmission in a distributed way in each router.

- *Distributed routing algorithm*: in the adaptive NoC at each router a *wXY-routing* algorithm is implemented. The routing algorithm is explained in detail in Chapter 7.
- *Link reversal mechanism*: The bi-directional links between two adjacent routers in the application-specific NoC are implemented using two simplex links but in the adaptive NoC the bi-directional links between two adjacent routers are implemented using two *half-duplex links*. Details are explained in Chapter 7.
- *Virtual channel buffer*: In an application-specific NoC, the *VCBs* are assigned to each output port at design time. The number of *VCBs* are also customized at design time depending on the application. On the other hand, in an adaptive NoC, the *VCBs* need to be assigned to each output port on-demand at runtime. Therefore, a central pool of *VCBs* are used in each router and these are assigned to each transaction and output port on-demand at runtime.
- *Mapping algorithm*: a design time application mapping algorithm presented in Chapter 4 is used for the application-specific NoC but for the adaptive NoC a runtime application mapping algorithm presented in Chapter 6 is used.
- *Monitoring component*: a monitoring component is implemented to allow successful *architecture-level* adaptation (see Chapter 7) for the adaptive NoC. In an application-specific NoC, no monitoring component is used or implemented.

An overview comparing the application-specific NoC and the adaptive NoC architecture on top of the proposed QoS-supported system-on-chip communication architectural design principles is summarized in Table 3.2. Two main components that are used to establish QoS are as follows:

1. *VCB* assignment for each transaction over the complete route (for details see Chapter 4).
2. Link arbitration algorithm that manages the flow control mechanism in each link (for details see Section 3.4).

A short overview of these components are described in the following.

Virtual Channel Buffer Assignment: *VC* implementation in *wormhole-based* routers increases the link resource utilization by multiplexing several logical transactions over a single physical interconnect between adjacent routers. *VCBs* are used to store data in the intermediate routers during data transmission for each transaction. A connection-oriented approach at *transaction-level* is used to establish QoS (a connection is established from the source to the destination tile by assigning a single *VCB* in each router throughout the complete route for each transaction). For the application-specific NoC presented in Chapter 4 the *VCB* are fixed at each output port during design time. The number of *VCB* assignment at each output port may be customized at design time and therefore, the area requirement for the router may be optimized. The *VCB* minimization at design time for the application-specific NoC is presented in detail in Chapter 4.

On the other hand, to allow different parallel transactions on each output port on-demand, the *VCB* assignment to each output port is performed at runtime for the adaptive NoC architecture. Therefore, a central pool of *VCB* is used and from there *VCBs* are assigned

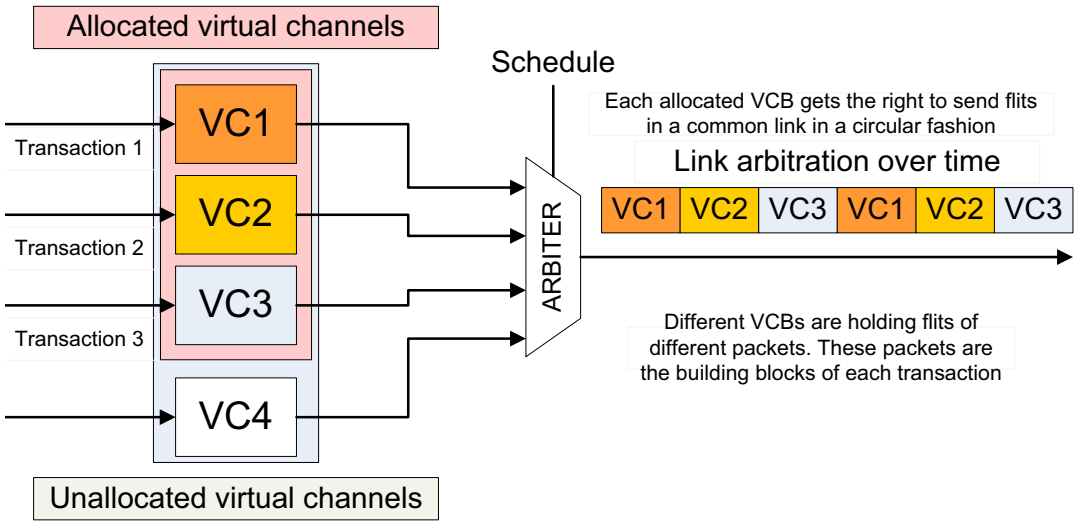


Figure 3.6: Round-robin link arbitration

to each output port at runtime. It increases the resource utilization and facilitates more concurrent parallel transactions. Details are explained in Chapter 7.

Algorithm 1 RR: Round-robin link arbitration algorithm

VC: Number of virtual channels
VC_{res}: Number of reserved virtual channels at time t
P_i: Priority of the i^{th} transaction
ST_{max}: Slot table size in number of cycles
ST_{avl}: Slot available in number of cycles
$VC_k^{delay\_jitter}$: Virtual channel k latency sensitive and jitter allowed
VC_k^{jitter}: Virtual channel k latency sensitive and jitter allowed
C_p^{max}: Upper bound # of cycles assigned to priority p
C_p^{min}: Lower bound # of cycles assigned to priority p
i^{th}: Entry in the Slot Table: $ST(i)$
C_p: Cycles assigned to priority P

1: **for all** $VC_{res_i} \in VC_{res}$ **do**
2: Assign slots to $ST(next\_slot) \leftarrow i$
3: $++next\_slot$
4: **end for**

Link Arbitration Algorithm: Link arbitration manages the *transaction-level* flow control during data transmission and therefore keeps the required bandwidth guarantees for the transactions. There have been several link arbitration algorithms used in the state-of-the-art works, i.e. *Round-Robin* (RR) [91], *Time Division Multiple Access* (TDMA) [41, 136], etc. The work presented in [91, 92] have assumed that all the *VCs* are statically allocated and predetermined by the *centralized manager*. In Figure 3.6, there are four *VCs* and three transactions. If the physical channel bandwidth is b then the granularity of the bandwidth reservation can only be *b/4, b/3, b/2* and b. Again, *QNoC* [23] provides on an average slightly more than 99% guarantee but fails to achieve 100% guarantee, which is

mandatory constraint for a safety critical embedded system. Work presented in [23] and [72] implement arbitrations depending on the service classes. Output line traffic within a service class is arbitrated using a *RR* approach (see Figure 3.6 and Algorithm 1). The higher prioritized packets get through as long as there is data in the buffer corresponding to the service class. In this fashion, this approach provides *relative guarantee*, not *tight guarantees* compared to the whole architectures. Common to current works in NoC similar to the proposed QoS-supported NoC architecture is that efficient resource utilization together with supporting hard guarantees is not fully considered.

In the scope of this thesis, a novel link arbitration algorithm based on TDMA approach is used. The proposed link arbitration algorithm named *Bounded-Arbitration-Algorithm* (BAA) increases the resource utilization by treating individual transaction separately by broadly classifying them to several service classes determined during application exploration.

3.4 Bounded-Arbitration-Algorithm

The link arbitration algorithm, *BAA*, used in the scope of presented QoS-supported system-on-chip communication architecture considers both *TDMA-like* connection-oriented and service class based approaches. The algorithm has adopted fine-granular service class specification, considering the lower and the upper bounds for each transaction type and therefore, avoids resource underutilization and erroneous resource reservation. The approach also provides flexible connections for the required service classes assigned to each transaction.

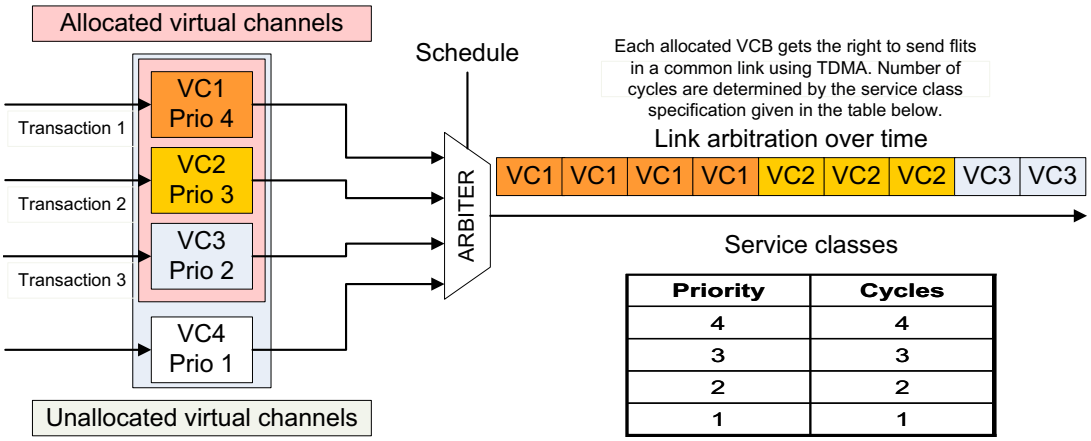
Service	Latency	Bandwidth (BW)	Jitter	Example	LB	UB
Priority H	*low*	-	-	*short message*	n_{l_H}	n_{h_H}
Priority X_1	-	*Fixed guarantee*	*No jitter/fixed*	*video 80%BW*	n_{l_1}	n_{h_1}
Priority X_2	-	*Fixed guarantee*	*jitter allowed*	*data 50%BW*	n_{l_2}	n_{h_2}
... ...	... ...	... ...	... ...	... ...	... ...	... ...
Priority X$_n$	-	*Fixed guarantee*	*No jitter*	*streaming audio*	n_{l_n}	n_{h_n}
Priority L	*lowest*	*Not fixed*	*acceptable*	*short memory access*	n_{l_L}	n_{h_L}

Table 3.3: Fine-grained service class specification

3.4.1 Fine-grained Quality-of-Service Specification

The support of diverse types of traffic patterns with the maximum link utilization is the principal goal of the QoS-supported system-on-chip communication architecture. In the scope of this thesis, support for guaranteed bandwidth, low latency, jitter, and a scalable framework for the service class assignment during application exploration at design time are specified. In an application-specific NoC, the application is well known, therefore the design space exploration is limited but for the adaptive NoC the design space becomes larger for all the possible types

of application. Therefore, in an adaptive NoC a trade-off is considered during service class specification for each transaction. Previous work like [133] has summarized the service classes in: *Best Effort* (BE) and *Guaranteed Service* (GS). Work in [23], *QNoC* architecture has identified 4 different types of service classes, namely, *Signaling*, *Real-Time*, *Read-Write (RD/WR)* and *Block Transfer*. So for all real time transactions with varying communication requirements, (i.e. bandwidth) *QNoC* treats the transaction in the same fashion. However, all these works underestimate the fine granularity of the bandwidth assignment. To overcome these shortcomings, the proposed approach offers a scalable and application-specific service class specification explored at design time. In Table 3.3, an exemplary service class specification for a multi-media application is shown. Each of these service classes specifies the *Lower Bound* (LB) and the *Upper Bound* (UB) cycles in the scheduling table (scheduling table and slot table are used interchangeably) to satisfy *variable-bit-rate* data transactions. The jitter problem is handled by keeping the lower and the upper bounds the same. Generally, the highest priority is kept for the messages having emergency requirements, i.e. interrupts, security messages.



Priority	Cycles
4	4
3	3
2	2
1	1

Figure 3.7: TDMA: Time division multiple access-like link arbitration

3.4.2 BAA on Top of the TDMA-like Link Arbitration

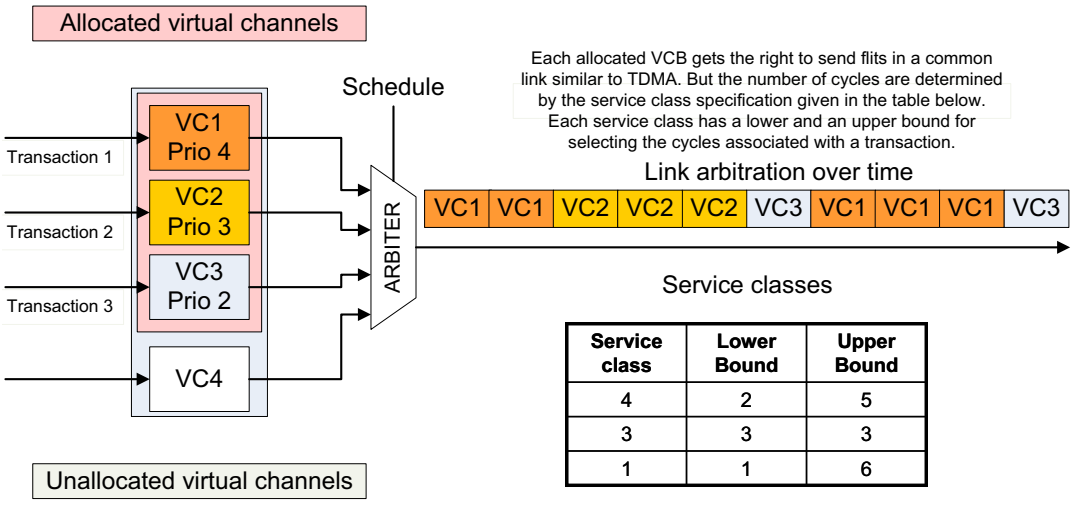
Considering all the above mentioned QoS requirements, in Algorithm 2 and in Figure 3.7, the link arbitration algorithm similar to the state-of-the-art *TDMA-like* fixed arbitration is presented. This algorithm provides service class specification as well as tight guarantees in terms of bandwidth and jitter. Algorithm 2 exploits the advantage of *TDMA* together with an independent service class specification. Each type of service class gets a fixed number of cycles to send data at a specified time on the link. The diversity of data transactions under different traffic loads is not taken into account. It cannot configure services adaptively while meeting the lower bound. These situations may lead to false reservation according to miss-use of available resources during average execution time. In the presence of available bandwidth, whole the bandwidth can be assigned to the lowest priority (like *Æthereal)*[133]). But a situation of having no available best effort traffic will lead to bandwidth underutilization for the higher prioritized traffic.

Algorithm 2 TDMA: Time division multiple access-like link arbitration algorithm

Other variables are defined in the previous Algorithm 1
P_i: Priority of the i^{th} transaction
C_p: Cycles assigned to priority P

1: **for all** $VC_{res_i} \in VC_{res}$ **do**
2: **for all** C_{p_i} **do**
3: Assign slots to $ST(next\_slot) \leftarrow i$
4: $++next\_slot$
5: **end for**
6: **end for**

To avoid the latency problem and to use the unused bandwidth by any available transaction, a novel link arbitration algorithm, called *bounded-arbitration-algorithm* is proposed in Algorithm 3 (see also Figure 3.8). Unused bandwidth can be shared by every present concurrent transaction, but still considering the service class specification. This tends to increase the throughput of the network. The less time a resource is (exclusively) reserved, the chance of contention for the resources by the concurrent transactions decreases. Thus, the algorithm is not only providing low latency for privileged transactions but also a possible better utilization for other transactions/applications using the QoS-supported system-on-chip communication architecture.



Service class	Lower Bound	Upper Bound
4	2	5
3	3	3
1	1	6

Figure 3.8: Proposed bounded-arbitration-algorithm

The algorithm implements the service class specification together with the higher and the lower bounds slot allocation. In this algorithm, all transactions first meet their lower bound requirements and then go for filling the unused bandwidth approaching to the upper bound. The selection of the *VCs* up to the upper bound, specified by the priority of the stored packets is done depending on the service class parameters, specially considering latency and jitter. The algorithm ensures the lower bound for any sort of transaction but certainly on the service class basis. Besides offering unused bandwidth to only the lower priority transactions, it tends to meet the upper bound requirements for some flexible (i.e. latency sensitive and jitter allowed) transactions.

Algorithm 3 BAA: Bounded-arbitration-algorithm

1: $slot\_assigned \leftarrow FALSE$
// Fill Scheduling Table SC with lower bound number of slots
2: **for all** $VC_{res_i} \in VC_{res}$ **do**
3: **for all** $C_{p_i}^{min}$ **do**
4: Assign slots to $ST(next\_slot++) \leftarrow i$
5: **end for**
6: **end for**
// Filling SC with latency sensitive transactions that allows jitter
7: **while** $\neg slot\_assigned$ **do**
8: $slot\_assigned \leftarrow TRUE$
9: **for all** $VC_{res_j} \in VC_j^{delay\_jitter}$ **do**
10: **if** $SLOT_j < C_p^{max}\ \ AND\ \ ST_{avl}$ **then**
11: Assign slots to $ST(next\_slot++) \leftarrow j$
12: Increase the allocated slots for $VC_{res_j}\ \ to\ \ SLOT_j$ ++
13: $slot\_assigned \leftarrow FALSE$
14: **end if**
15: **end for**
16: **end while**
// Filling SC with transactions that allows jitter (rest **BE transactions**)
17: **while** $\neg slot\_assigned$ **do**
18: $slot\_assigned \leftarrow TRUE$
19: **for all** $VC_{res_k} \in VC_k^{jitter}$ **do**
20: **if** $SLOT_k < C_{max}^p\ \ AND\ \ ST_{avl}$ **then**
21: Assign slots to $ST(next\_slot++) \leftarrow k$
22: Increase the allocated slots for $VC_{res_k}\ \ to\ \ SLOT_k$ ++
23: $slot\_assigned \leftarrow FALSE$
24: **end if**
25: **end for**
26: **end while**

3.4.3 Bound Analysis

Building a time-critical embedded system is always a challenging task. The system design starts from the application exploration. This exploration provides the average, lower and best case scenario for the running system. From this analysis the designer restricts the design to obey the bounds. The following bound analysis is only valid when the application and its transactions are well known at design time. The throughput bounds (in bits/cycle) for every transaction may be formulated as follows:

$$\textit{[L-B]}\ T_c^p = (low\_s_c^p/S) \times \textit{CW} \times \textit{DR [bits/cycle]} \tag{3.1}$$

$$\textit{[U-B]}\ T_c^p = (up\_s_c^p/S) \times \textit{CW} \times \textit{DR [bits/cycle]} \tag{3.2}$$

The throughput for a transaction ***c*** having the priority ***p*** is defined by T_c^p in the above equation. The number of allocated cycles for the transaction ***c*** over time for priority ***p*** is assigned both for the lower and the upper bounds, $low\_s_c^p$ and $up\_s_c^p$. The symbol *CW* stands for the channel width and *DR* stands for the data ratio, representing the protocol overhead. Similar to other architectures, the proposed QoS-supported system-on-chip communication architecture

also suffers from the protocol overhead, i.e. header data for the transmitted packets. The data ratio compared to the extra header information is also considered in the bound calculation.

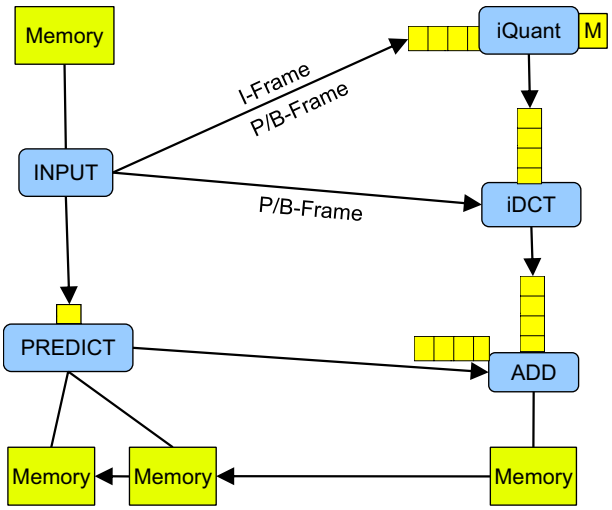


Figure 3.9: MPEG4 video decoder and its data flow graph

Similarly, the latency bounds can be summarized as:

$$L^i_{hop} = L^i_{vctolinkS} + L^i_{link} + L^i_{linktoVC_j} \tag{3.3}$$

$$L^p_c = \frac{(L_{IPtoSW} + L^i_{hop} + L_{NItoIP}) \times IP\_data}{T^p_c} \tag{3.4}$$

Here, L^i_{hop} is the latency of an individual hop *i*, calculated by summing up the latency from the virtual channel to the link ($L^i_{vctolinkS}$), the latency in the link (L^i_{link}), and the latency in link for the router *i* to the virtual channel, *j* ($L^i_{linktoVC_j}$). After attaining the latency of a single hop, the latency for an individual transaction may be calculated. Symbol L^p_c stands for the latency of a transaction having the priority p. This latency can be calculated by all the small parts that contribute to the latency along the route, the latency from the IP to the router switch (L_{IPtoSW}), the latency for each router *i* in the transaction route (L^i_{hop}), the latency from the NI to the IP (L_{NItoIP}). All sources of latency are then multiplied with the amount of data sent by the IP (IP_{data}) and divided by the throughput of the transaction having the priority *p* (T^p_c).

Injection:	Min.	Max.
MPEG4	400 MB/s	800 MB/s
Stimulus1	120 MB/s	400 MB/s
Stimulus2	180 MB/s	400 MB/s
Stimulus3	300 MB/s	600 MB/s

Table 3.4: Traffic modeling

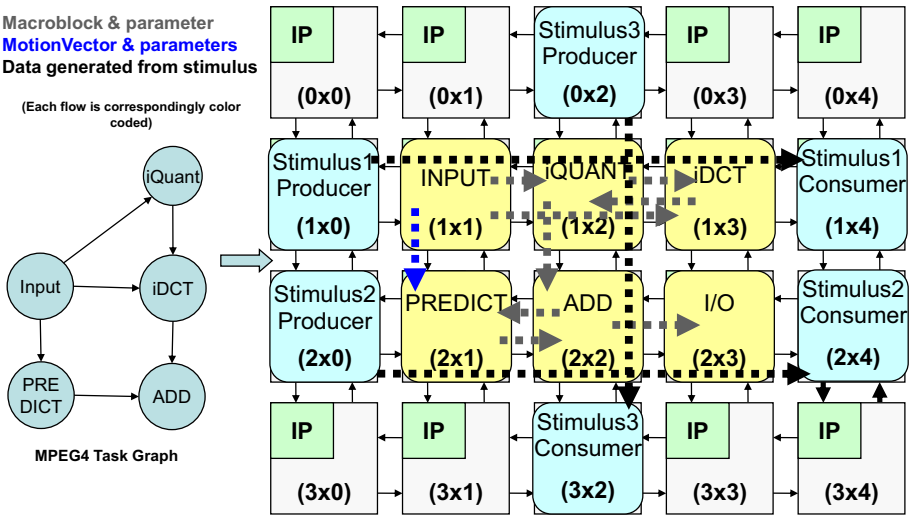


Figure 3.10: MPEG4 video decoder mapping onto a 5 × 4 Mesh

3.4.4 Evaluation of the Bounded-Arbitration-Algorithm

For the evaluation of the proposed *bounded-arbitration-algorithm*, an MPEG4 video decoder is implemented (only application-specific NoC is considered in this evaluation). In Figure 3.9, the task graph of the MPEG4 video decoder is shown. An MPEG4 encoded video is a sequence of *I-frames*, *P-frames*, and *B-frames*. An I-frame is not predicted. P-frames are predicted from the previous I- or P-frame. And finally, a B-frame is predicted from the previous and next I- or P-frame. B-frames are optional in compression. Figure 3.9 shows that an I-frame needs the tasks *INPUT*, *iQuant*, *iDCT*, and *ADD* for decoding. P- and B-frames require *INPUT/VLC*, *iQuant*, *iDCT*, *PREDICT*, and *ADD* tasks for decoding. The mapping of an MPEG4 video decoder onto a 5 × 4 Mesh NoC architecture is shown in Figure 3.10. Here, 6 inner tiles are used for the tasks of MPEG4. Other tiles are used for other *Stimulus* to create arbitrary traffic during the experiments. *Stimulus* created *Stimuli* go through the whole chip and then create concurrent communication together with the MPEG4 application. Thus it simulates the embedding of the MPEG4 in a larger application. In most of the cases, the smallest communication unit in MPEG4 is a *MacroBlock (MB)*, consisting of 16 × 16 pixels. The communication unit between the tasks INPUT and PREDICT is the so called *MotionVector (MV)*.

$$\alpha(t) = \mu = \frac{\rho_{'}(t) \times D}{\rho_{''}(t) \times W} \tag{3.5}$$

For the experiments, the traffic model described in Table 3.4 and in Equation 3.5 are taken. A random rate, μ keeps the injected traffic load between 30-100% of the worst case specified in the Table 3.4. Here D stands for the amount of data transferred between two tasks and W for the worst case (fastest computation) execution time. To keep the traffic application-specific, random variables $\rho_{'}$ and $\rho_{''}$ have been introduced, where $0 \leq \rho_{'} \leq 1$ and $1 \leq \rho_{''} \leq 2$. Experiments show, the injection rates of the *variable-bit-rate* data producing applications vary over time. For Example, the IP1x2 (iQuant) is producing a *MacroBlock* of 808 Bytes for IP1 × 3 (iDCT) in an interval depending on the prior processing of IP1x2 (Input). The analysis of the MPEG4

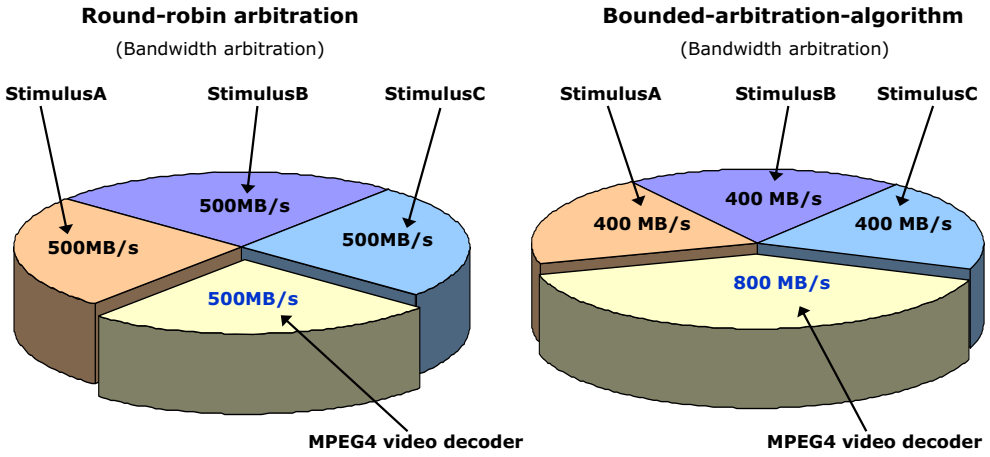


Figure 3.11: Transaction-specific bandwidth provides the lower bound guarantee

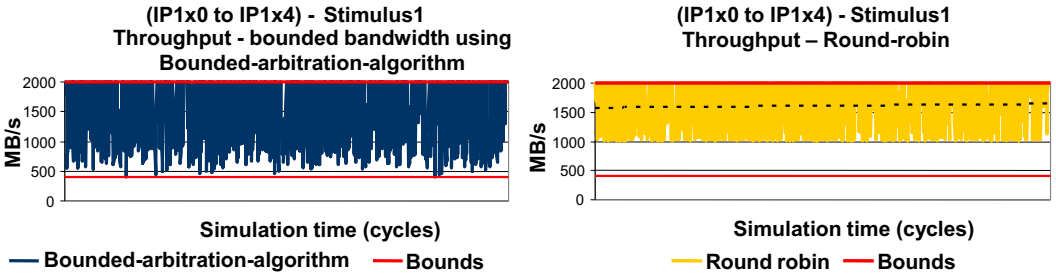


Figure 3.12: Throughput between RR and BAA

application shows that the computation times vary from 50-100%. Some observations considering the transactions *iQuant* → *iDCT* and *Stimulus1(Producer)*→ *Stimulus1 (Consumer)* are recorded in the following given figures.

An analytical result showing the comparison of traffic specification between the *RR* and the *BAA* is presented in Figure 3.11. In this experiment, link capacity of 2000 MB/s is considered. Link capacity is design time parameterizable for the application-specific NoC. The lower bound (minimum) requirements from the MPEG4 and *Stimulus* are assumed to be 800 MB/s and 400 MB/s respectively (different scenario compared to Table 3.4). The upper bound (maximum) requirements for both the MPEG4 and *Stimulus* are assumed 2000 MB/s. The *RR* arbitration scheme fails to allow all 4 concurrent transactions to meet their minimum requirements. The *BAA* supports application specific bandwidth assignment and that is why it can allow all 4 transactions to meet at least their lower bounds.

Figure 3.12 depicts the direct comparison of the throughput of the low prioritized *Stimulus1* transaction for the *BAA* and the *RR* algorithms. It is shown that the *BAA* supplies a fluctuating throughput of 400-2000 MB/s but the *RR* arbitration provides a throughput that oscillates in the interval of 1000-2000 MB/s. These observations can be explained as follows: the *BAA* algorithm arbitrates only the minimum bandwidth specified in the service class description of

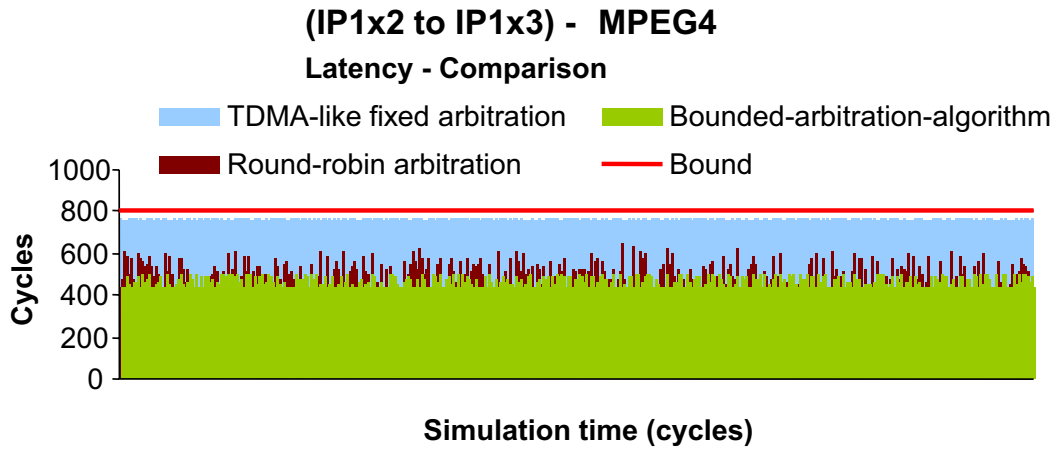


Figure 3.13: Latency comparison for TDMA-like fixed, RR and BAA algorithms

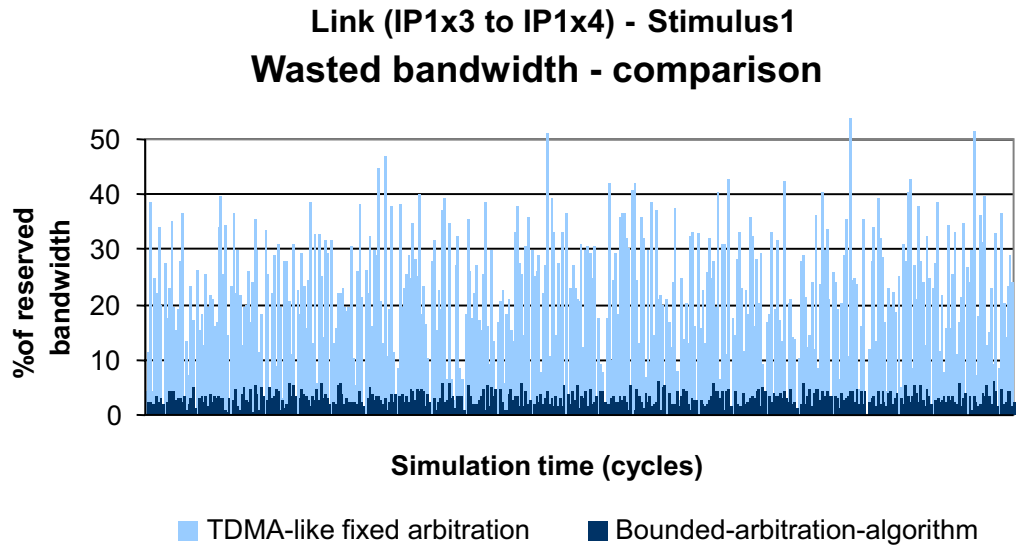


Figure 3.14: Resource utilization: TDMA-like fixed and BAA

the *Stimulus1's* priority to the transaction, if both transactions are present. Thus it privileges the MPEG4 over the *Stimulus1* transaction as it is specified. In contrast to that, the *RR* arbitration does not consider any priority or classification. Therefore, the low prioritized transaction gets the same portion of the bandwidth as the latency-sensitive data. With a growing number of concurrent transactions, high classified transactions might fail their deadline, because low prioritized data is arbitrated the same way. This comparison shows that *BAA* is able to arbitrate transactions separately depending on the classification of the transaction type.

Figure 3.13 shows the difference among the fixed arbitration scheme (TDMA-like) (see Algorithm 2), the *RR* (see Algorithm 1), and the *BAA*. The graph shows that *BAA* provides the lowest latency for the latency-sensitive MPEG4 transactions. Even though *RR* provides a low latency, (least is higher than the one in the *BAA*) with more concurrent transactions the *RR* would not be as competitive because of the equal arbitration of all transactions. *TDMA-like* fixed arbitration

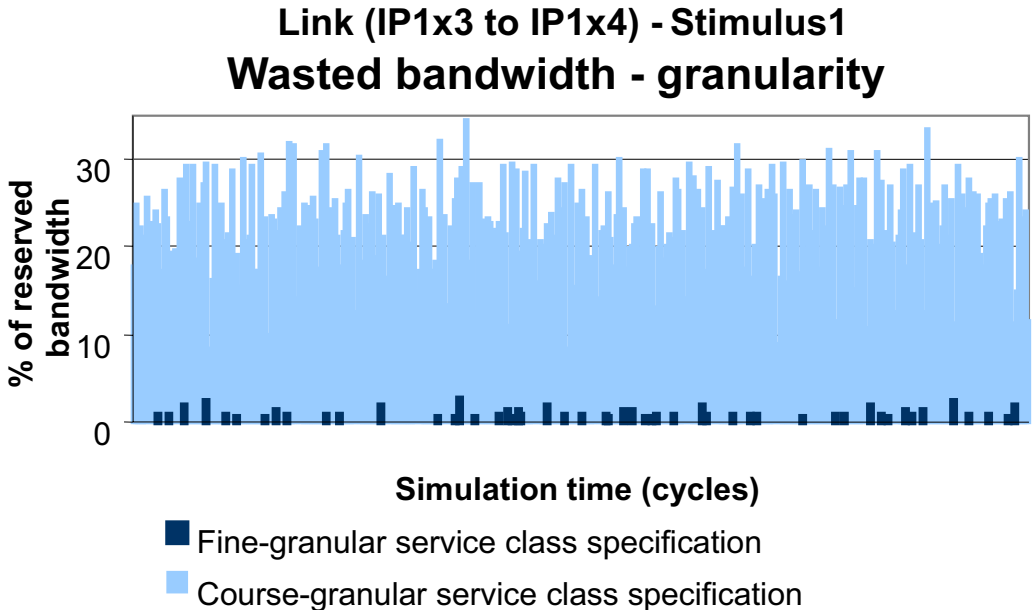


Figure 3.15: Fine-granular service class specification

algorithm provides predictable constant throughput and latency bounds but has no ability to adapt to the present traffic.

The waste of bandwidth for a static reservation of resources compared to a more adaptive, *BAA*, is shown in the Figure 3.14. It can be seen in the figure that the wasted bandwidth of the fixed approach is on an average near 25% because of the unpredictability of the applications injection rate. The *BAA* provides a good resource utilization by keeping the bandwidth waste significantly low around 3% over the whole simulation time. These results show the importance of bandwidth adaptation during runtime. Whereas the *BAA* shares unused bandwidth among other transactions, the fixed approach wastes it. This leads to a huge resource underutilization by wasting $\frac{1}{4}$ of the reserved bandwidth in this experiment. For the high costs of resources in a system-on-chip, such a waste of resources is inevitable. An adaptive approach like the introduced *BAA* is mandatory. It can be derived that the *BAA*, presented in this thesis provides a very high *resource utilization of 97%*. It represents an approach that is able to adapt to *variable-bit-rate* traffic patterns and considers service-based classifications.

The possible granularity of the traffic classification is crucial for the utilization of the network. The *Stimulus1* is producing data with 20% of the bandwidth. One classification reserves 25%, the other one 20%. *BAA* is used, but with no jitter. The result in Figure 3.15 shows that, the course-granular scheme exhibits a high waste of bandwidth. On an average 20% of the reserved bandwidth are wasted. This conforms with the deviation of 5% in the traffic specification, because 5% is 20% of the defined 25% of bandwidth. The waste of bandwidth by the fine-granular specification is on an average negligible with less than 0.10%. The result emphasizes the importance of a specific definition of the traffic requirements. Architectures that are not offering a fine-granular service specification suffer from a bad resource utilization in diverse traffic scenarios. However, the approach of this work with the introduced design allows a fine-

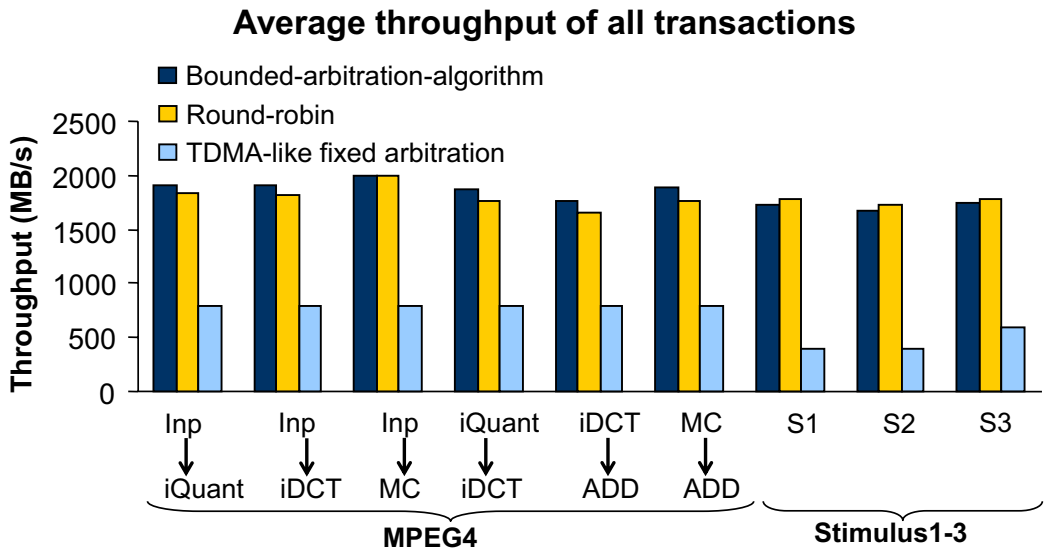


Figure 3.16: Overall throughput comparison for MPEG4 video decoder and stimulus

granular service class specification.

The advantage of the *BAA* in terms of higher throughput, low latency and more resource utilization compared to the *TDMA-like* fixed algorithm and to the simple *RR* algorithm are summarized in Table 3.5, Figure 3.16, and Figure 3.17. Because of the static reservation the *TDMA-like* fixed algorithm provides constant throughput which is determined by the reserved worst case requirements. *RR* offers a high throughput for the MPEG4 and the *Stimuli* transactions. But especially the throughput of the higher prioritized MPEG4 transactions is less than in the *BAA* simulation. The *BAA* assigns the most throughput to the latency-sensitive MPEG4 transactions. Thus the throughput of the *Stimuli* is not as high as utilizing *RR*. In case of more concurrent transactions along the same route, *RR* would share the throughput equally among all the transactions. Such an arbitration does not consider any service classification and might fail in meeting the deadlines of each individual transmission. The throughput results show the advantage of the *BAA* that arbitrates the traffic as defined in the service class specification.

The average latency in cycles per thousand bytes is presented for the three arbitration schemes. Utilizing the *TDMA-like* fixed algorithm, the highest average latency for every transaction is obtained. Because of the static reservation, always only a reserved fraction of the bandwidth is used, even if the full bandwidth would be available. This leads to constant but very high latency for the *TDMA-like* fixed algorithm, measured in cycles. Comparing *RR* and *BAA* it can be derived that the average latency in both arbitration schemes is quite low. But considering the traffic classification the *BAA* provides the best results. The latency-sensitive MPEG4 transactions using the *BAA* have lower latency cycles than with *RR* arbitration. In the *RR* scheme, the low classified *Stimuli* have a lower latency than utilizing *BAA*. But this results in the higher average latency of the MPEG4 transactions which are classified as latency-sensitive and privileged. Thus *RR* does not consider any service-based separation of the traffic. *BAA* provides a methodology to keep the latency for specific transactions high, by maximizing the latency of

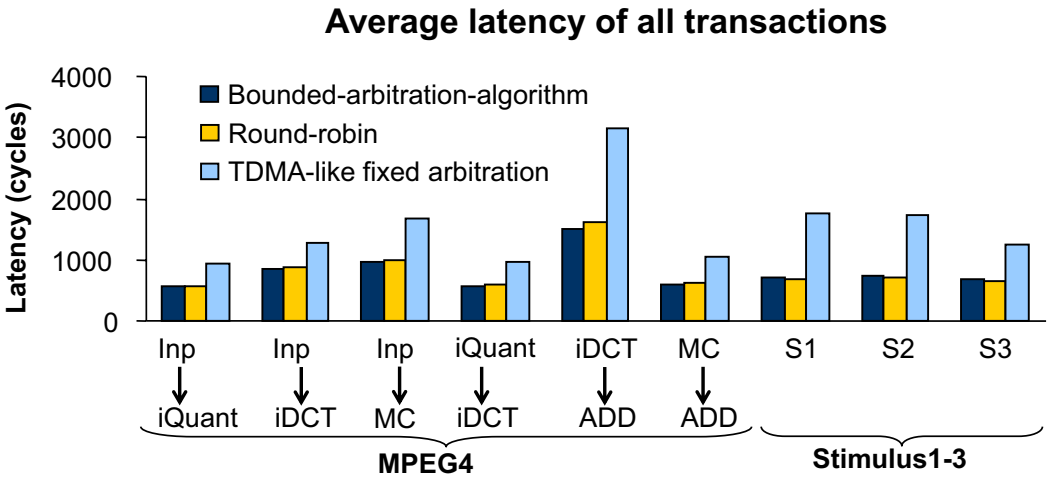


Figure 3.17: Overall latency comparison for MPEG4 video decoder and stimulus

traffic with a low priority. Table 3.5 also summarizes that the *BAA* considering service classification provides a higher network utilization of about 97% together with higher throughput and low latency.

3.5 Conclusion

The adaptive system-on-chip communication architecture introduced in Chapter 5 is built on top of the QoS-supported NoC architecture presented in this chapter. The flow control mechanism to provide QoS at *transaction-level* using a novel link arbitration algorithm named as *bounded-arbitration-algorithm* is explained in detail here. The importance of classifying each transaction at its fine granular level with lower and upper bounds of services is introduced in this chapter. It is already discussed that a scalable system-on-chip interconnect with bandwidth guarantee is beneficial for system-on-chip real-time MPSoCs. However, it carries the burden of typically wasting a large amount of bandwidth when the guaranteed bandwidth is not used by one or more interconnect channels. As discussed, state-of-the-art communication architectures do not provide satisfactory solutions, either they lack 100% guarantee or they come with a relatively large average bandwidth waste.

It is demonstrated that the proposed link arbitration algorithm that provides QoS at *transaction-level* is able to provide a 100% guarantee of bandwidth with an average waste of only 3% (i.e. 97% utilization) a value that has not been achieved by others so far. The advantages of *BAA* is evaluated by a case study of an MPEG4 video decoder under varying *Stimuli* scenarios.

Transactions	Average throughput MB/s			Average latency cycles/1000B			% Bandwidth waste		
	Fixed (TDMA-like)	RR	**BAA**	Fixed (TDMA-like)	RR	**BAA**	Fixed (TDMA-like)	RR	**BAA**
Input− >iQuant	800	1840	**1901**	939	575	**561**	90	0	**0**
Input− >iDCT	800	1822	**1900**	1272	883	**850**	83	1	**1**
Input− >PREDICT	800	2000	**2000**	1680	986	**978**	89	2	**2**
iQuant− >iDCT	800	1756	**1875**	962	599	**573**	90	0	**0**
iDCT− >ADD	800	1662	**1767**	3148	1620	**1499**	57	1	**1**
PREDICT− >ADD	800	1768	**1890**	1051	626	**590**	55	1	**1**
Stimulus1	400	1788	**1732**	1754	676	**709**	23	4	**8**
Stimulus2	400	1730	**1673**	1717	703	**739**	25	4	**4**
Stimulus3	600	1777	**1737**	1236	648	**668**	19	3	**4**

Table 3.5: Comparison of BAA compared to RR and TDMA-like fixed link arbitrations

Chapter 4

Road to Adaptive Networks-on-Chip from Application-specific Design

Application-specific *Networks-on-Chip* (NoC) design has been a great research interest since the beginning of this decade. Several application-specific NoC architectures [20, 23, 32, 43, 65, 72, 92, 107, 133, 145, 157, 170] and different research issues, i.e. topology customization [1, 15, 33, 81, 109, 123, 127, 152], buffer minimization [55, 56, 78, 139], routing algorithms [45, 79, 132], application mapping [70, 77, 98, 104, 110, 111, 112, 113, 143], design methodology, and automation tools [57, 60, 86, 89, 94, 114, 178] have been discussed in previous works. In general, customization of different NoC parameters starting from the application or application-domain to a highly optimized communication-centric Multi-Processor System-on-Chip (MPSoC) architecture at design time is termed as the application-specific NoC. In an application-specific NoC, the customization is mainly performed in three domains of NoC design (1) NoC *architecture-level* parameters, i.e. *buffer*, *topology*, *link-width*, and *floorplanning*, (2) *communication paradigm*, i.e. *routing algorithm*, *switching strategy*, and (3) *application mapping*, i.e. *task and communication scheduling*, *task and PE mapping*. Details of all these parameters with a general design flow of an application-specific NoC are described in the following sections.

The novel contributions in the scope of this chapter are as follows:
(1) A detailed study to formulate the parameters for building networks-on-chip architectures are shown in this chapter. These customization parameters are later evaluated to construct the novel adaptive system-on-chip communication architecture.
(2) A case-study analysis to build an area-efficient application-specific NoC by reducing buffer at design time is detailed in this chapter.
(3) To optimize the number of *Virtual Channel Buffers* (VCBs) in a *Quality-of-Service* (QoS)-supported application-specific NoC architecture, a two-step methodology is provided. The two steps are given below:
Step 1: A multi-objective mapping algorithm that considers both the communication volume and the number of *VCBs* during NoC mapping based on a modified *Ant Colony Optimization* (ACO) algorithm is used. This approach is orthogonal to any application driven traffic model (published in [55]).
Step 2: In the second step, the traffic characteristics of the application is considered. It is observed that a wide range of digital media applications follow the *poisson distribution* and therefore, an analytical approach to optimize further the number of *VCBs* depending on QoS parameter q and the application-specific traffic model is presented (published in [56]).

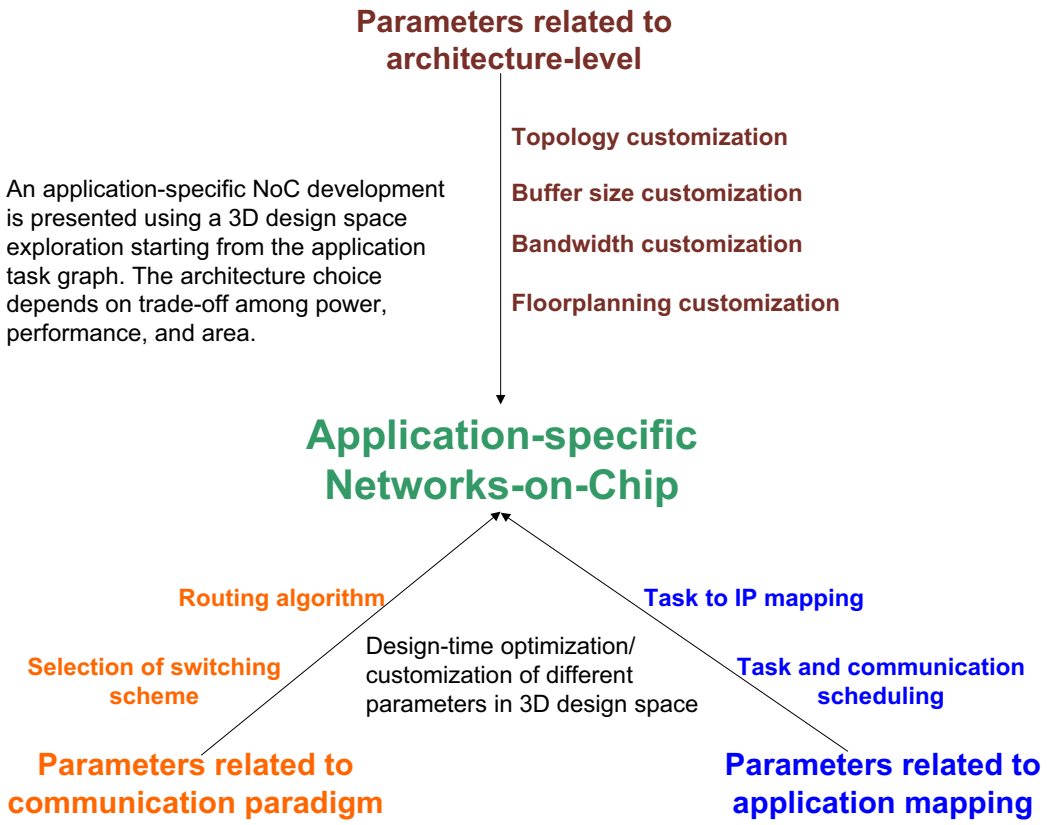


Figure 4.1: 3D design-space exploration for the application-specific NoC design

4.1 Parameters to be Customized for the Application-specific Networks-on-Chip

In this section, a summary of all the possible parameters that can be customized at design time for an application-specific NoC design are presented. A similar overview is also presented in [122]. An application-specific NoC design begins from the application task graph TG (see Definition 1 in Chapter 3) and then the application is mapped onto an optimized NoC architecture. In the following, the application-specific NoC, $ANOC$ is defined:

Application-specific Networks-on-Chip: The system-on-chip communication architectural design choice starting from an application to a 3D design-space exploration, considering the triple $ANoC = (X, Y, Z)$ may be defined as application-specific NoC. Figure 4.1 shows different parameters in a 3D view that needs to be customized at design time. The variables in the triple $ANoC = (X, Y, Z)$ are explained as follows:

- The parameter X in the triple $ANOC$ is a quadruple $X = (To, Bu, BW, Fl)$, the *architecture-level* parameters, those are required to design the router for the application-specific NoC. In the quadruple, To is the *custom topology*, Bu is the *buffer* assigned to each port, BW is the *supported bandwidth* in each output port which may be calculated from the given *link-width*, and Fl represents the *floorplan-*

ning of the final MPSoC depending on different sizes of the tiles.

- The parameter Y consists of two parameters for selecting the type of *communication paradigm*, $Y = (Ro, Sw)$, where Ro is the *routing algorithm* selected for the NoC considering the application. The routing in general is of two types (considering only the deterministic routing) (1) *source-based routing* and (2) *distributed routing*, e.g. the *XY-routing* algorithm. Second parameter in selection of the *communication paradigm* is the *switching strategies* (Sw), i.e. *store-and-forward*, *virtual cut-through*, *wormhole*, etc.

- The third dimension in the 3D design-space for the application-specific NoC design, is the *application mapping* to the NoC platform. Several parameters, i.e. energy, performance, etc are used as the cost function for the application mapping. It also has two parameters to be customized, $Z = (Sc, Mp)$, where Sc is *the task and communication scheduling* and Mp is *the mapping of the task on the corresponding precessing element* inside each tile.

All these parameters those need to be customized at design time to generate an application-specific NoC are described in short, in the following subsections. A detailed study regarding all these parameters may be found in [43, 47, 65, 77, 87, 122].

4.1.1 Architecture-level Parameters

The hardware-related parameters those comprises the design of the router for the application-specific NoC are considered in this category. The parameters are: (1) *topology selection*, (2) *link-width*, (3) *buffer size*, and (4) *floorplanning* of the NoC.

Topology Customization: In the *architecture-level* design process of an application-specific NoC, one of the most important steps is to choose a suitable NoC topology for a particular application and then mapping of that application on the selected topology [86, 114]. There are several standard topologies onto which the application may be mapped. Those may be classified as (1) *direct topology*, e.g. Mesh (see Figure 4.2 (a)), Torus, Hypercube, etc and (2) *indirect topology*, e.g. 3-stage clos, butterfly (see Figure 4.2 (b)), etc [47]. In a *direct topology* in each router, only one *Processing Element* (PE) is connected and on the other hand in a *indirect topology* several PEs may be connected to a router. The selection of an appropriate topology for the application-specific NoC *architecture-level* design depends on several design objectives, i.e. network latency, throughput, QoS, power consumption, area, etc.

The simplicity and easy *floorplanning* have always motivated towards selection of a regular grid-like topology, i.e. Mesh [23, 24, 59]. While the topology is selected, the NoC design problem is reduced to the rest of the other issues, i.e. routing algorithm, buffer assignment, application mapping, etc. Several experiments have shown that such a regular grid-like topology does not meet the design constraints or even may end up with infeasible solutions [122]. Therefore, to achieve a better design-space exploration, a group of

standard topologies is also explored to find an optimal solution for the given application [93, 114].

Recently, the limitations of the standard topology selection have motivated the researchers and designers for exploration in the direction of flexible customized topology (see Figure 4.2(c)). The benefits of a customized flexible topology for an application-specific NoC are multifold: the communication workload presented in an application can be exploited and therefore can be optimized for the architecture [124, 130, 153], the wastage of silicon area because of different sizes of tiles may be saved, resource utilization per transistor basis may be increased, etc. Several approaches concerning its design challenges have been already discussed in the previous works [1, 33, 43].

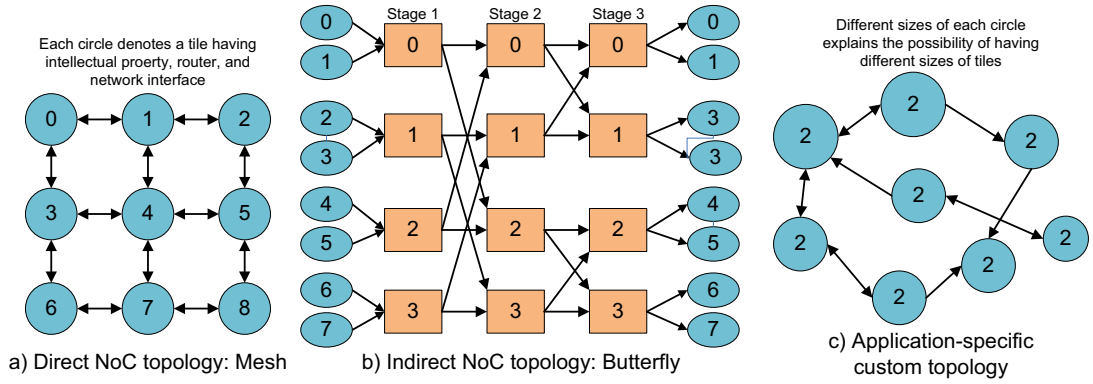


Figure 4.2: Different types of topology for the application-specific NoC design

Link-Width Customization: The width of the communication link determines the *supported bandwidth* of that particular link. The bandwidth calculation in relation to the *link-width* is calculated as follows:

$$BW = f_{ch} \times W \tag{4.1}$$

In Equation (4.1), BW stands for the *supported bandwidth* of the link which is calculated considering the link operating frequency f_{ch} and the width of the links in number of bits is shown as W. In general, if the *link-width* W increases then it decreases the communication latency. The side affect of the increased *link-width* W is higher area in the NoC implementation. The choice of an appropriate *link-width* influences the NoC parameters, i.e. resource utilization, area, energy, etc [47, 60, 122].

Buffer Size Customization: *Buffer* is required to store the transactions in the intermediate routers from the source to the destination to resolve the issues, like. route selection, congestion, etc. Depending on the router implementation *buffer* may be used in the input ports as well as in the output ports. An estimation shows that the *buffer* area amounts to about 60% of the entire router area [173]. In [122], authors have shown that by increasing the *buffer* size at each input port from 2 to 3 words increases the overall router area by 30% for a 4×4 NoC (an input buffering scheme together with *wormhole-based* routing is used in this architecture). Therefore, overall use of the *buffer* has to be reduced for

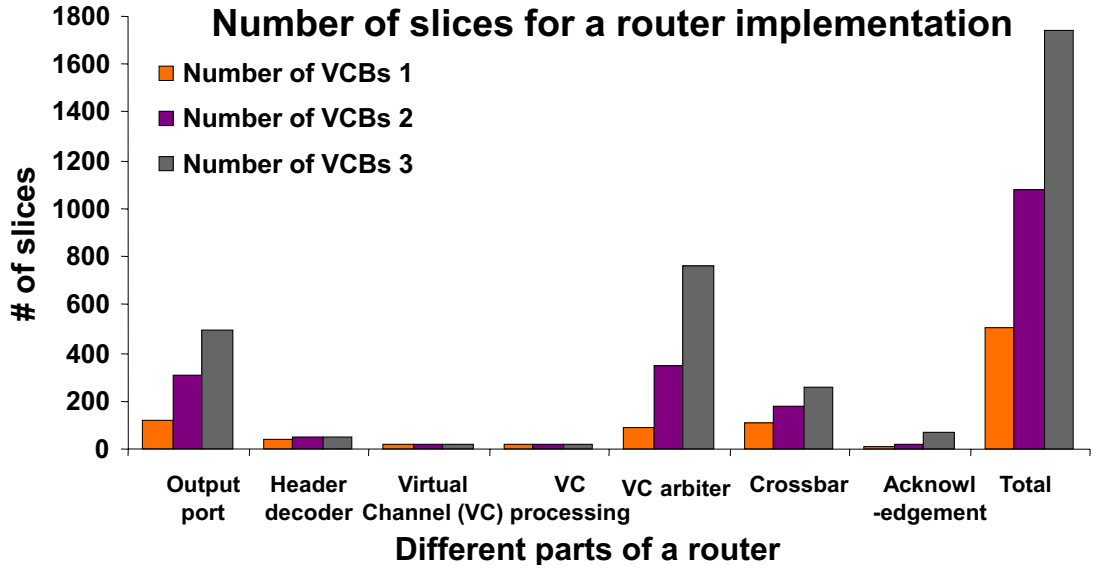


Figure 4.3: Area effects of the increasing virtual channel buffers

designing a system-on-chip communication architecture and it is of prime importance. On the other hand, using more *buffers*, the reduced latency for the transactions and accomodation of more parallel transactions may be achieved.

In the scope of this thesis, an application-specific NoC on top of the QoS-supported system-on-chip communication architecture presented in Chapter 3 is detailed in this chapter (see Section 4.3). Special emphasis has been given to reduce the size of the buffer at design time in this work. Figure 4.3 shows the effect of increasing the buffer that is needed to implement the *Virtual Channels* (VCs) in the proposed architecture.

Floorplanning Customization: The *floorplanning* is a part of the *architecture-level* parameters that may be customized at design time for the application-specific NoC. The *floorplanning* for the regular topologies is well studied and has been addressed in [179]. But on the other hand, if the size of the tiles in the NoC varies with the selection of an arbitrary topology then the customization of the *floorplanning* becomes a very important step in the NoC design process. *Floorplanning* needs to consider several issues, i.e. placement of routers and repeaters for a latency insensitive operation, routability of the global network link for higher performance, the coupling effects, etc. [30, 86, 110].

4.1.2 Communication Paradigm Customization

The *communication paradigm* guides the transaction flow after the application is mapped onto the NoC architecture. Generally, the parameters that are required for the flow control of the transactions are: the *routing algorithm* and the *switching strategy*. The selection of the *communication paradigm* directly influences the physical parameter of the NoC architecture, e.g. buffer as well as the performance of the NoC, i.e. latency, throughput, QoS. An overview of these parameters is given below.

Routing Algorithm Selection: One of the most important parameters for the NoC design is the selection of the *routing algorithm*. Endless number of *routing algorithm*s have been proposed for the system-on-chip communication architectures for the last couple of years (for details see [47]). The selection of the *routing algorithm* greatly affect the NoC design parameters specially performance, energy, and area. Therefore, a trade off among area, energy, and performance is considered.

Routing algorithms can be broadly classified depending on the number of destinations into two distinct categories: (1) *unicast routing*, single destination and (2) *multicast routing*, multiple destinations [47]. Most of the early works have mainly concentrated in *unicast routing* because of its higher demand from application side [24, 43, 77]. There have been very few works dealing with *multicast routing* for the system-on-chip communication architecture [102, 140]. Depending on where the routing decisions are taken an *unicast routing* may be mainly classified as: (1) *source-based routing* and (2) *distributed routing*. In a *source-based routing*, the routing decision is taken in the source of the transaction and the complete route is determined before data transmission starts. On the other hand, in a *distributed routing*, the route is determined in a deterministic way while the packets inside a transaction are traveling across the network. Both these *source-based* [43] and *distributed* [132] *routing algorithm* have been studied in early works for the system-on-chip communication architectures. All these *routing algorithms* can be *deterministic* [116] or *adaptive* [7, 62, 79, 142]. *Deterministic routing* algorithms always provide the same route for a given source and a destination and *adaptive routing* algorithms use several information, i.e. network traffic, available network resources, etc [47]. For an application-specific NoC design selection of the *deterministic routing* algorithm is more appropriate as the given application is well studied and therefore, the *routing algorithm* may be customized by using the state-of-the-art techniques, i.e. traffic splitting [113]. Situations where the traffic scenarios may not be predicted beforehand, i.e. during faults, demand for the *adaptive routing* algorithms [7, 48, 49]. In the scope of this thesis, a *deterministic routing* algorithm is used for the application-specific NoC and an *adaptive routing* algorithm named as the *wXY-routing* algorithm for the adaptive system-on-chip communication architecture explained later in Chapter 7.

The *routing algorithms* whatever the type is need to be livelock, deadlock, and starvation free [47, 62]. In practice, deadlock and livelock detection and recovery mechanisms are expensive and they may effect the latency and throughput of the network. Deterministic and partially adaptive algorithms [79] based on turn model are free from deadlock problems, but on the other hand, fully *adaptive routing* algorithms require extra precaution, e.g. *Time-To-Live* (TTL) counters are used in the proposed adaptive system-on-chip communication architecture in Chapter 7.

Switching Strategy Selection: The second parameter in the *communication paradigm* selection, that is closely coupled with the *routing algorithm*, is the *switching strategy*. The *switching strategy* determines the flow control mechanism of the packet inside a router. It influences the required buffer to store the intermediate data during transmission [47]. There are several types of switching strategies for the system-on-chip communication architectures, e.g. *store-and-forward*, *wormhole*, *virtual-cut-through*, etc. Among the above

mentioned switching strategies *wormhole switching* has been proved to be most promising for the system-on-chip communication architectures because of its lower buffer resource and latency requirements [20, 23, 43, 44].

In general, *wormhole switching* provides better network performance i.e latency or average throughput compared to a pure circuit switching [106, 133] under the presence of dynamic traffic [122] (examples of circuit switched NoC can be found in [27, 176]). In an application-specific NoC design, the application is well studied and therefore, a circuit switching may provide 100% QoS compared to a *wormhole switching*. Therefore, circuit switching may be used for an application-specific NoC design where QoS is of prime importance. In the proposed application-specific NoC presented in this thesis, a connection-oriented approach over *wormhole-based* switching is presented. The connection-oreinted approach exploits the concept of circuit switching in the transaction level and therfore, provides QoS.

4.1.3 Mapping of the Application

The application is mapped onto the NoC architecture at design time and this mapping influences several parameters during the execution of the applications, i.e. QoS, performance in terms of throughput and latency, etc. The parameters those may be included in the architecture exploration for system-on-chip communication architecture considering the mapping of the application are: *task and communication scheduling* and *task to PE mapping*. A short outline of these parameters is given below.

Scheduling Algorithm Selection: During the *application mapping*, the problem of the *task and communication scheduling* needs to consider the computation and the communication together. The scheduling is very important for real-time and DSP-applications. The *task and communication scheduling* problem is a classical problem in computer science and has been also studied for the applications that are mapped onto the NoC architectures. Besides the classical problems, in the NoC domain the inter-processor communication delay together with the congestion issues need to be considered [122]. Several works have been done in this respect, i.e. an architectural support for compile-time scheduling in the system-on-chip communication [99], energy minimization by first allocating more slack to those tasks which have larger impact on energy and performance constraints [80], etc. In [80], authors have used a *level-based* scheduling for tasks and communication transactions in parallel and then have used a search and repair procedure iteratively to improve the solution by fixing the deadline misses in the schedule.

Application Task Graph to Processing Element Mapping: The application task graph onto PE mapping can be described as follows: an application is described by a set of tasks and those are already bounded and scheduled to a particular PE in the earlier stage and then the function of the *application mapping* is to assign these tasks topologically to the networks. Several parameters, i.e. energy, performance, buffer area, etc are considered as the cost function for the *application mapping*. In an application-specific NoC, the mapping is done offline at design time. In practice, the stages: *task to PE binding* and then

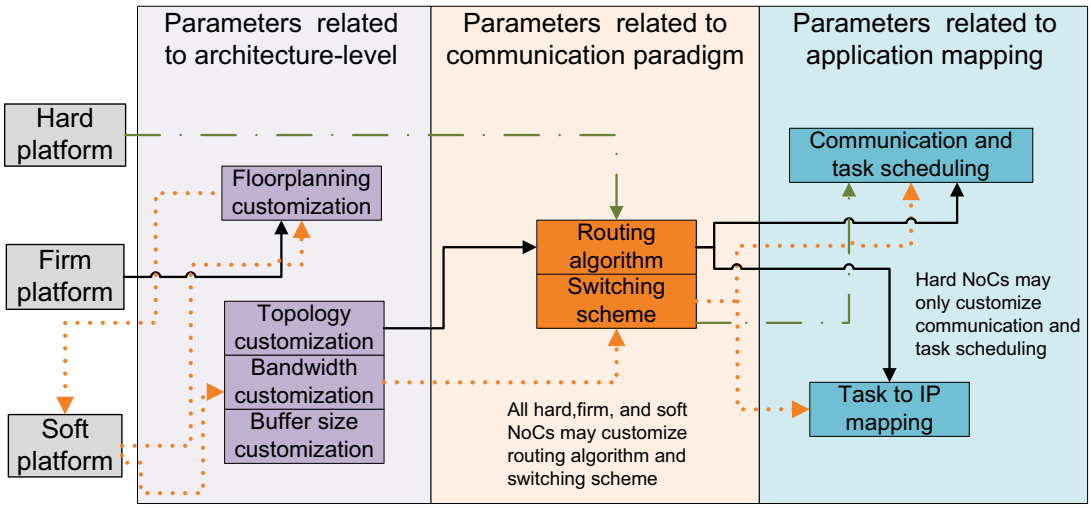


Figure 4.4: Design flow for an application-specific NoC motivated from [122]

mapping physically to a given NoC architecture happen in a feedback loop. Generally, the mapping is performed after the binding and then the NoC is simulated to gather results on the cost matrices and given to the binding process for further modification.

There have been several works for design-time mapping algorithm. In [77], a mapping using *branch-and-bound* algorithm has been proposed. The authors have used the communication related energy consumption as a goal function. *Genetic algorithm* based mapping algorithms have been introduced in [98, 143]. [77, 143] determine a network assignment which is designed to minimize the power consumption by reducing the communication volume. All these previous works considered mainly the communication volume as the basis of their optimization. Work in [5] presents an approach to multi-objective exploration of the mapping space of a Mesh-based network-on-chip architecture. Based on evolutionary computing techniques (*genetic algorithm* is used here), they try to obtain the *pareto mappings* that optimize performance and power consumption. In the scope of this thesis, the importance of buffer minimization during *application mapping* is presented and therefore, a multi-objective optimization algorithm to optimize the amount of total communication volume and the amount of buffer using a modified *ACO* algorithm is presented. The performance of *ACO* algorithm and its suitability over other algorithms for optimization are given in [55, 159]. Within this thesis, the *ACO* algorithm is adapted as the first step of the two-step methodology to optimize the amount of buffer for an application-specific NoC that is built on top of the QoS-supported system-on-chip communication architecture presented in Chapter 3. Details are explained in Section 4.3.

4.1.4 Design Flow for an Application-specific NoC

In this subsection, a summarized design flow for an application-specific NoC considering the concept of *hard*, *firm*, and *soft* NoC platform as presented in [122] is discussed. The *hard* NoC

has already a selected NoC architecture and all the computation and communication components are pre-designed. Therefore, the application-specific NoC design challenge or the customization is only possible in the *communication paradigm* parameters (see $Y = (Ro, Sw)$ in 3D design-space exploration parameters in Section 4.1) and in the *application mapping* parameters (see $Z = (Sc, Mp)$ above in Section 4.1). In a *firm* NoC platform, the PE may be plugged-in onto different tiles and thus the *floorplanning* may be customized in some extents. After that similar to the hard NoC platform the *communication paradigm* parameters and the *application mapping* parameters may be customized. Finally, the *soft* NoC, the most flexible NoC may be customized for all the parameters described above during the 3D design-space exploration for the given application. Figure 4.4, similar to the flow presented in [122], shows the design flow of all possible types of the application-specific NoC architectures.

4.2 Novel Application-specific NoC Architecture

In this section, the case study analysis to build an application-specific NoC on top of the QoS-supported system-on-chip communication architecture presented in Chapter 3 is described. For better understanding of the rest of the chapter, a short summary, of the QoS-supported NoC, as the system-on-chip communication architecture is repeated in this section (for details see Chapter 3). To show the novelty of the NoC, a comparison is drawn to the state-of-the-art architectures (service-class-based approaches [23, 72, 92, 170]) which are similar to the proposed architecture.

The QoS-supported NoC architecture that is presented in this thesis has taken the advantages of both service-class-based and connection-based approaches and it has some novel architecture level differences compared to the current service-class-related state-of-the-art NoC architectures. The presented methodology uses wormhole *switching strategy* like [23, 72, 92, 170] and thus utilize *VC* implementation. Previous works related to service-class-based architectures used a fixed number of *VCBs* in each output port and thus do not consider an application-specific *VCB* assignment. It is already discussed in Chapter 3, that existing service-class-related architectures may be broadly classified into two parting directions. The first group of the architectures presented in [23, 72] have used a fixed number of service classes (see Figure 4.5 (a)). Each service class has a particular *VCB* in each output port. They do not provide 100% tight guarantee for each of the transaction, and these architectures do not have a clear and fine-granular classification for each transaction. Therefore, the bandwidth is not allocated fairly to increase resource utilization. The arbitration of the *VCBs* is dependent on their corresponding service class to the output port. Concurrent transactions use the packet-based *round-robin* ordering of input ports for the awaiting transmission of packets within the same service class.

The second group of QoS-supported service-class-associated NoCs (see Figure 4.5 (b,c)) is presented in [92, 170]. In this group, the differentiation of transactions bandwidth requirement is done in the spatial domain. In a link of available bandwidth b, by allocating n number of *VCBs* (out of m *VCBs* in total) a total of $(b/m \times n)$ bandwidth can be provided. The *VCs* are arbitrated in a *round-robin* fashion. It consumes a lot of buffer in terms of *VCB* implementation for fine granularity and suffers from starvation for some transactions. In [170], shown in Figure 4.5 (c),

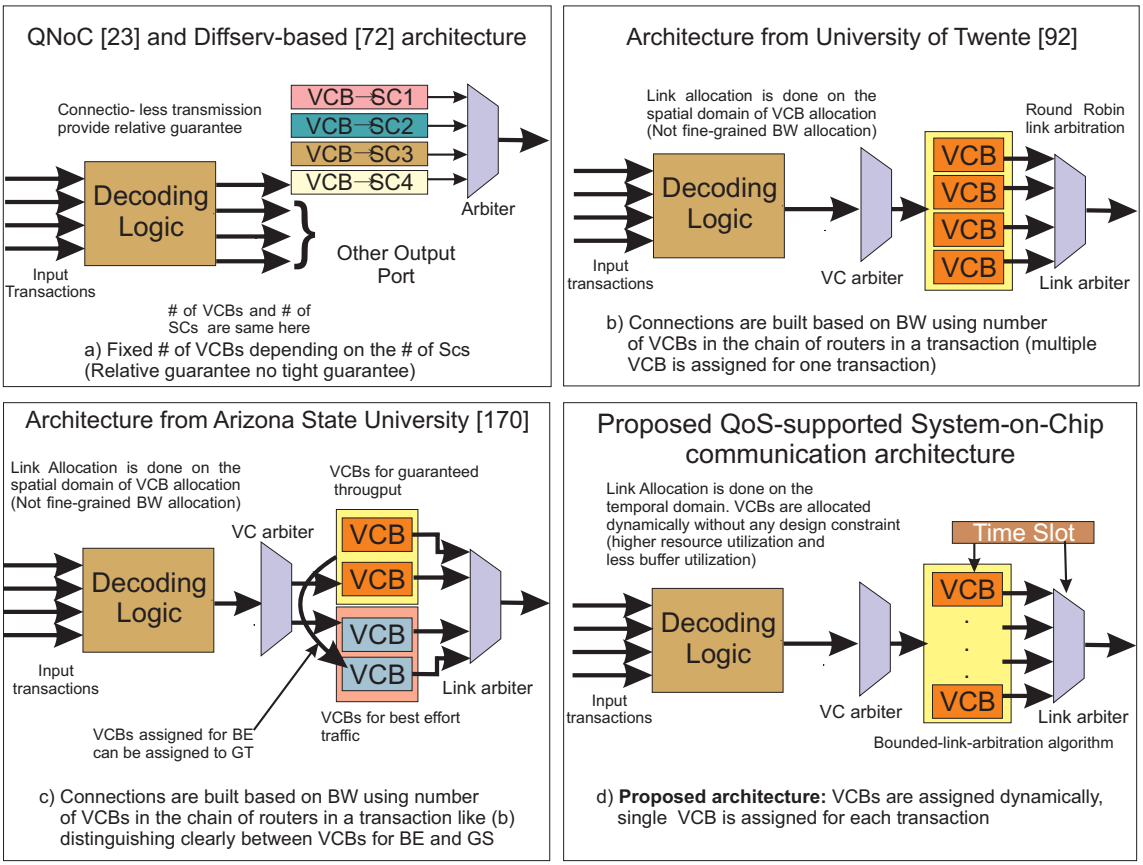


Figure 4.5: Different state-of-the-art service-class-based architectures

apart from [92] shown in Figure 4.5 (b), *VCBs* assigned for *Best Effort* (BE) traffic can be taken by *Guaranteed Service* (GS) at connection establishment time.

To overcome all these problems, an architecture that arbitrates on the temporal domain, has application-specific fine-granular service-class specification, thus higher resource utilization, and no particular static assignment of *VCBs* is proposed in Chapter 3. The number of *VCBs* is constrained in each output port by the number of allowed concurrent transactions, some of them allowing flexible transactions. The connections are freed after each unit transaction (after every packet transmission the *VCB* is freed to be allocated by other concurrent transactions). The size of each unit transaction depends on the application behavior and is customizable at design time for application-specific NoC. All these *VCBs* are arbitrated in the time domain and provide buffer minimization considering all transactions in an application at design time and fine-grained bandwidth assignment per transaction. The arbitration on the bandwidth for competing transactions is done using a modified *Time Division Multiple Access* (TDMA) approach described as in Chapter 3. For the current buffer analysis it is assumed that the complete application behavior including the traffic characteristics is known a priori (there have been several prior research for system-on-chip traffic modeling, e.g. in [151, 166]).

In the proposed application-specific NoC, for each transaction only one *VCB* is needed, not

multiple *VCBs* like in [92, 170]. It is already discussed that *VCBs* are used to facilitate the flow control mechanism in the router and take the principal share of the silicon area of the NoC [78, 173], and consequently their size should be minimized.

In the following of this chapter, a novel two-step methodology to optimize the number of *VCBs* for an application-specific NoC built on top of the QoS-supported system-on-chip communication architecture described in Chapter 3 is presented. The two steps are given below:

Step 1: A multi-objective mapping algorithm that considers both the communication volume and the number of *VCBs* during application mapping based on a modified *Ant Colony Optimization* (ACO) algorithm is used. This approach is orthogonal to any application driven traffic model.

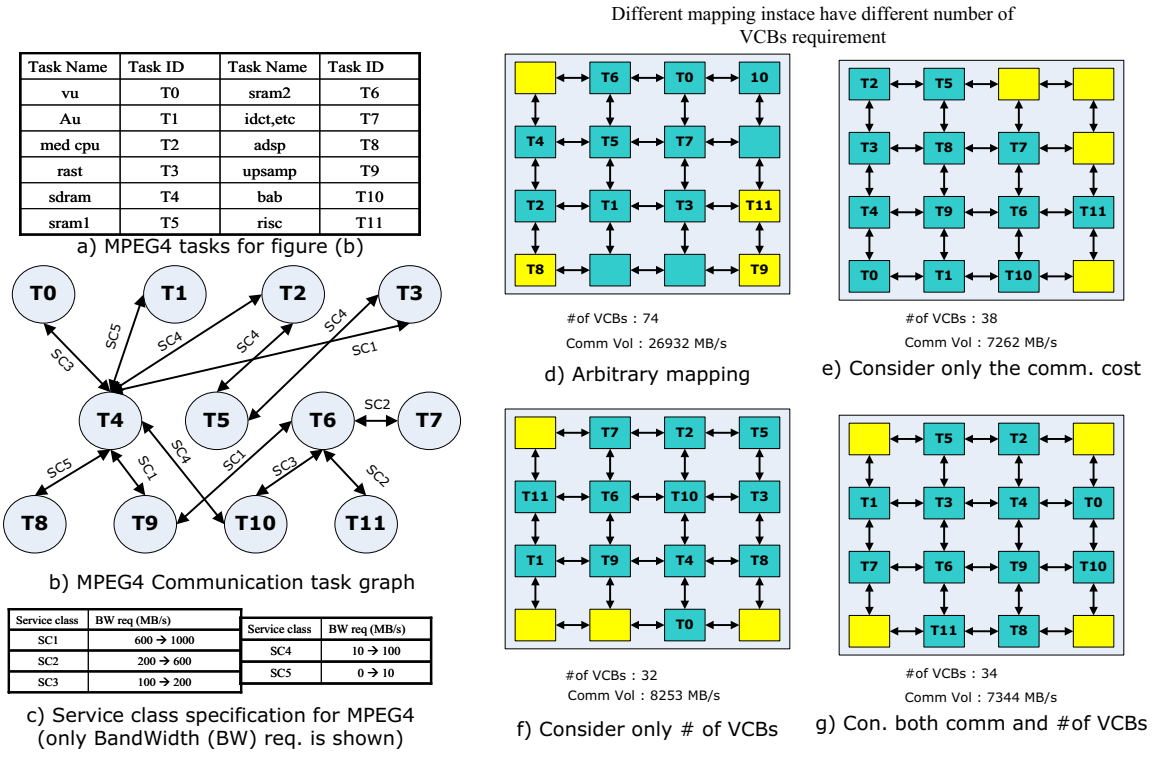
Step 2: In the second step, the traffic characteristics of the application is considered. It is observed that a wide range of digital media applications follow the *poisson distribution* and therefore, an analytical approach to optimize further the number of *VCBs* is provided depending on QoS parameter q and the application-specific traffic model.

4.3 Application-specific Virtual Channel Buffer Assignment

VCBs are used to store intermediate data for each transaction. In the proposed application-specific NoC architecture one *VCB* per router output port for each transaction along the complete path is used. A connection is built using a chain of *VCBs* along the route of the transaction. Below it is explained how these number of *VCBs* in each output port may be optimized using a two-step methodology.

To demonstrate the motivation of the application-specific *VCB* assignment besides minimizing the total communication volume at design time during application mapping is presented by means of an MPEG4 video decoder case study analysis in Figure 4.6. This is the first step of the proposed approach and it takes no assumptions on the traffic model and freeing/releasing buffer strategy. Figure 4.6 (a,b,c) show the *Service Class associated Task Graph* (SCTG) (see Definition 3 in Chapter 3) of the MPEG4 video decoder.

Figure 4.6 (d) provides an arbitrary mapping of tasks onto a 4×4 Mesh NoC. It can be calculated that the needed number of the *VCBs* is 74 and the total communication volume is 26,932 MB/s and therefore, higher area and energy consumption than the optimal value (given budget). The detailed explanation to calculate the total number of *VCBs* and the total communication volume is given later. The number of *VCBs* is directly proportional to the area [55] and the leakage power [35]. The total communication volume is proportional to the communication related energy consumption [77, 98, 143]. Therefore, the goal for the application-specific NoC design is to keep the amount of the *VCBs* and the total communication volume as low as possible and to consider them both at the mapping time.



Task Name	Task ID	Task Name	Task ID
vu	T0	sram2	T6
Au	T1	idct,etc	T7
med cpu	T2	adsp	T8
rast	T3	upsamp	T9
sdram	T4	bab	T10
sram1	T5	risc	T11

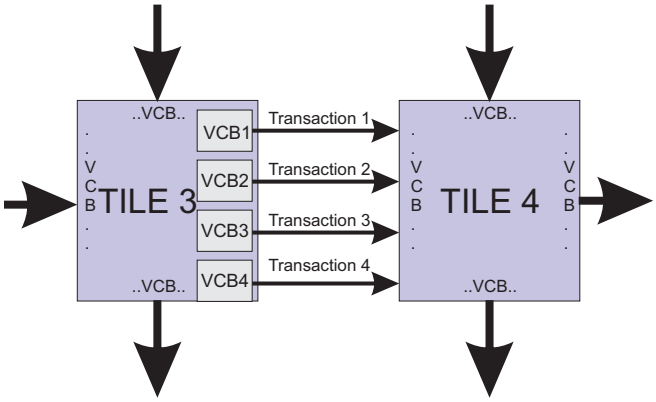
Service class	BW req (MB/s)
SC1	600 → 1000
SC2	200 → 600
SC3	100 → 200

Service class	BW req (MB/s)
SC4	10 → 100
SC5	0 → 10

Figure 4.6: Motivating example: MPEG4 video decoder mapped onto a 4 x 4 NoC

Figure 4.6 (e) shows the mapping of the tasks to the tiles considering only communication volume similar to the approaches [77, 98]. Here, the communication volume is 7,262 MB/s which is an optimal value and the number of the *VCBs* is 38. Now in Figure 4.6 (f), it can be seen that if we optimize only for the *VCBs* then we have an optimal number of *VCBs* (32 *VCBs*) but the total communication volume increases to 8,252 MB/s. Therefore, it is clear that both these parameters need to be considered jointly during mapping. Using a very simple cost function presented later in this chapter, a trade off can be achieved between the number of *VCBs* and the total communication volume. In Figure 4.6 (g), it requires 34 *VCBs* and the communication volume is 7,344 MB/s which are near optimal solution. The combined optimization goal function is a designer's choice.

Lets now motivate the second step of the proposed *VCB* reduction strategy. In Figure 4.7 (a), different concurrent transactions between two tiles are shown. The traffic model and the parameters to fit into the *poisson distribution* is shown in Figure 4.7 (b,c). The probability of use each *VC* is shown in Figure 4.7 (d). It is assumed that the *VCBs* can be freed and be assigned at run-time, and thus depending on the probability, the designer can decide to put required amount of buffers in the corresponding output port. In this example, 2 *VCBs* can be used easily instead of 4 as *VCB3* and *VCB4* have lower probability (0.57% and 0.02% respectively) to be used during concurrent transmission. Detail formulation and result analysis are given later in this chapter. Therefore, if the traffic model is known priori and the communication task graph is given then the *VCB* assignment can be carefully modelled at design time.



a) Transactions between two tiles

Transactions	Expected Arrival in T(λ)	Service Class
Trans 1	1	SC2
Trans 2	50	SC1
Trans 3	0.5	SC3
Trans 4	1	SC2

c) Parameters for the poisson model

Service Class (SC)	packet Length (flits)	Average Inter Arrival Time (cycles)
SC1	4	1000
SC2	250	50000
SC3	900	100000

b) Traffic patterens of the transactions

Virtual Channel Buffer (VCB)	Probability of Use P(VCB)
VCB 1	100%
VCB 2	5,13%
VCB 3	0.57%
VCB4	0.02%

d) Probability to use virtual channel buffer

Figure 4.7: VCBs reduction using an analytical approach

4.3.1 Minimizing Virtual Channel Buffer during Application Mapping

In this subsection, the first step of the two-step methodology to optimize the number of *VCBs* in an application-specific NoC is presented. Application mapping is one of the most important steps to build the application-specific NoC and is described in 4.1.3 during presenting the 3D design-space exploration for such architectures.

4.3.1.1 The Optimization Criteria

Generally, in a NoC scenario, the application mapping to a set of PEs, the communication volume [77, 98, 143] is considered as the optimization criteria to optimize the total communication-related energy consumption. All these above mentioned works are not optimizing for multiple criteria and do not consider the effect of increased number of *VCs*. In Equation (4.2), only the minimization of the communication volume is shown. A multi-objective optimization criteria considering both the total communication volume as well as the number of *VCBs* as the first step to optimize the number of *VCBs* in an application-specific NoC is presented in the scope of this chapter. Equation (4.4) considering both the optimization objective shown in Equation (4.2) and Equation (4.3) gives the cost function of the proposed optimization algorithm for the application to PE mapping. In the following, the specification of the two optimization criteria are summarized.

- **Total communication volume** of a given mapping configuration is:

$$vol_{tot}(TG_{vol}, map) = \sum_{i=0}^{N_{ts}-1} \sum_{j=0}^{N_{ts}-1} dst(i,j)\, vol_{i,j} \tag{4.2}$$

$$\forall i,j \in \{0,\ldots,N_{ts}-1\} : dst(i,j) = |x_i - x_j| + |y_i - y_j|$$
$$\forall i \in \{0,\ldots,N_{ts}-1\} : x_i = ts_i \bmod N_{noc}$$
$$\forall i \in \{0,\ldots,N_{ts}-1\} : y_i = ts_i \operatorname{div} N_{noc}$$

  where TG_{vol} is the application task graph containing the communication volume for every task, $vol_{i,j}$, $\forall i,j \in \{0,\ldots,N_{ts}-1\}$. The map data object contains the representation of task mapping. The elements of the mapping data object are values ts_i, where i stands for the *"task identification number"* and the value stored in ts_i represents the *"tile identification number"* where the task is being mapped onto.

  In Equation 4.2, the total communication volume is calculated by accumulating the communication volume of each flow, $vol_{i,j}$ between two different tasks ts_i and ts_j in the task graph. $vol_{i,j}$ is multiplied by the *Manhattan distance* between these two tasks $dst(i,j)$.

- **Total number of *VCBs*** of a given mapping configuration is:

$$vcb_{tot}(TG_{vol}, map) = \sum_{i=0}^{N_{ts}-1} \sum_{j=0}^{N_{ts}-1} dst(i,j)\, \sigma(vol_{i,j}) \tag{4.3}$$

$$\forall x \in int, x \geq 0 : \sigma(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \end{cases}$$

  where the $\sigma()$ function is the positive integer values.

The two optimization criteria are combined to the total *cost function* (optimization criteria, cost function, and goal function are used interchangeably in this chapter) for the application mapping:

$$c = \alpha \frac{vcb_{tot}(TG_{vol}, map)}{vcb_{tot,unmap}(TG_{vol})} + (1-\alpha)\frac{vol_{tot}(TG_{vol}, map)}{vol_{tot,unmap}(TG_{vol})} \tag{4.4}$$

The variables $vcb_{tot,unmap}(TG_{vol})$ and $vol_{tot,unmap}(TG_{vol})$ are almost the same as the two target functions defined above in Equation 4.2 and Equation 4.3. Only the distance between the tasks is not considered when calculating them:

$$vol_{tot}(TG_{vol}) = \sum_{i=0}^{N_{ts}-1} \sum_{j=0}^{N_{ts}-1} vol_{i,j} \tag{4.5}$$

$$vcb_{tot}(TG_{vol}) = \sum_{i=0}^{N_{ts}-1} \sum_{j=0}^{N_{ts}-1} \sigma(vol_{i,j}) \tag{4.6}$$

Thus, the resulting values can be used as a value to normalize the partial target which then can be better compared with the other targets, as it is done in the cost function c.

4.3.1.2 Optimization Algorithm

In the proposed approach, an optimization algorithm that can consider both the communication volume as well as the number of *VCBs* is explored. The algorithm has to be computationally efficient and to provide better optimal solution. Exploration during the evaluation of the proposed methodology shows the *ACO*, one of the *Swarm Intelligence* (SI) systems as the best solution for such problem space [159, 160]. The comparison to other algorithms to *ACO* is discussed later in this chapter. *ACO*, introduced by Moyson and Manderick and furthermore developed by Marco Dorigo [159] is a probabilistic technique for solving computational problems which may be reduced to find good paths through graphs. They were inspired by the behavior of *biological ants* in finding paths from the colony to a food source. In reality, ants initially wander randomly and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path they are likely not to keep traveling at random but instead follow the trail. Therefore, returning and reinforcing the trail if they find food. Over time, however, the pheromone trail starts to evaporate thus reducing its strength. More pheromone is evaporated if more time is required by an ant to travel down and come back. A shorter path may be observed by comparing the density of the pheromone. Pheromone evaporation has also the advantage of avoiding the convergence to a *local optima*. If there is evaporation at all then the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained. Therefore, when one ant finds a good path from the colony to a food source then other ants are more likely to follow that path and positive feedback eventually leaves all the ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with *"simulated ants"* walking around the graph representing the problem to solve.

4.3.1.3 Problem Formulation

Application mapping to the PE inside the NoC is a *Quadratic Assignment Problem (QAP)*. Therefore, the problem formulation considering [159] is given first. The formulation of the problem is given as below:
Given *n* tasks from an application and *n* tiles in a NoC (the number of tasks in an application and the number of tasks may be different), two $n \times n$ matrices $A = [a_{i,j}]$ and $B = [b_{r,s}]$, where $a_{i,j}$ is the distance between tiles *i* and *j* and $b_{r,s}$ is the communication flow between two tasks *r* and *s* is considered. Therefore, the *QAP* may be stated as follows:

$$\min_{\psi \in S(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{i,j} a_{\psi_i, \psi_j} \tag{4.7}$$

Here, *S(n)* is the set of all mapping solutions for *n* tasks and ψ_i gives the corresponding mapped tile for the task *i* in the current solution $\psi \in S(n)$. Here $b_{ij} a_{\psi_i, \psi_j}$ describes the cost contribution of simultaneously assigning task *i* to tile ψ_i and task *j* to tile ψ_j. The formulation of the *QAP* may be made as an integer optimization problem with a quadratic objective function. Let x_{ij} be a binary variable which takes value **1** if task *i* is mapped onto tile *j* and **0** otherwise. Then the

problem can be formulated as:

$$minimize \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n} a_{i,j}b_{k,l}x_{i,k}x_{j,l} \tag{4.8}$$

Here, $\sum_{i=1}^{n} x_{i,j} = 1$, $\sum_{j=1}^{n} x_{i,j} = 1$ and $x \in 0, 1$.

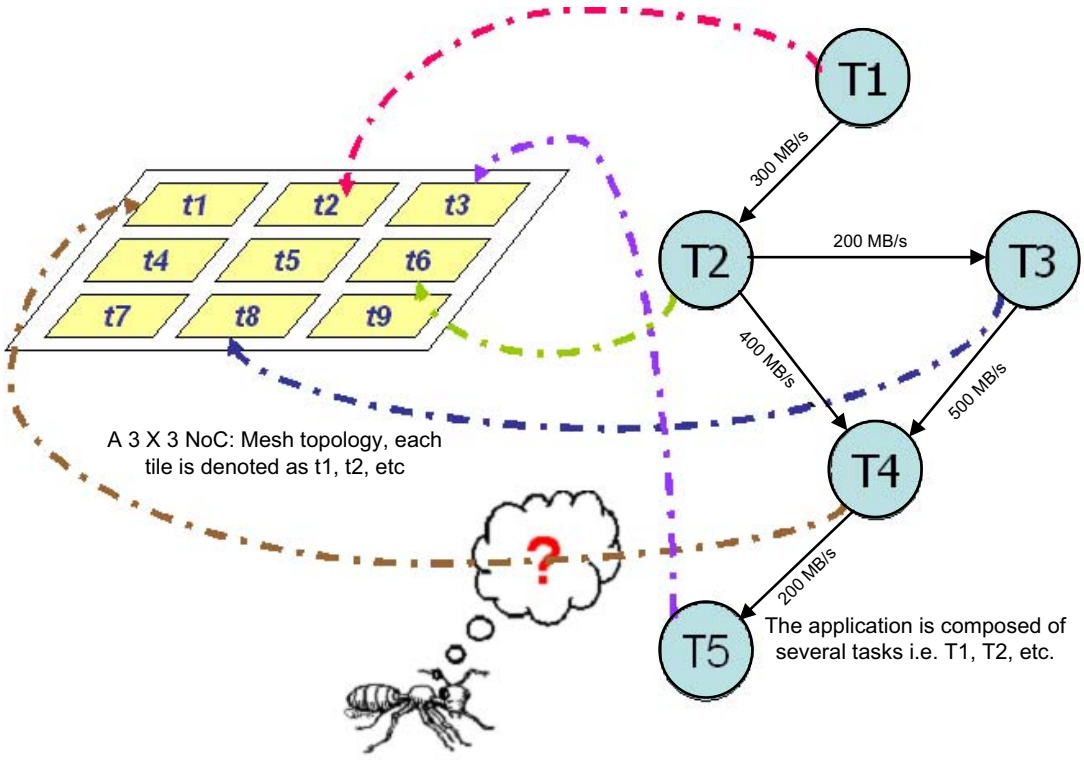


Figure 4.8: Exemplary application mapping onto a 3 x 3 NoC using ACO

4.3.1.4 Solution Construction

The *QAP* is a *NP-hard* optimization problem [159]. Therefore, to solve the *QAP*, heuristic algorithms may be used to achieve very high quality solutions in short computation time. Several such heuristic algorithms have been proposed which include algorithms like *genetic algorithms*, simulated annealing, tabu search, depth search, etc. It is shown in [159] that, for structured instances, *ACO* algorithms are one of the best performing algorithms. Therefore, it is already mentioned in 4.3.1.2 that, *ACO* is used to solve the application mapping problem considering the cost function given in 4.3.1.1. An application mapping scenario using *ACO* is shown in Figure 4.8, where each task should be mapped onto one tile in a 3×3 Mesh NoC, e.g. task *T1* should be mapped onto tile *t2*, task *T3* to tile *t5*, etc. The function for the ants is to find the best way to map all the tasks to the corresponding tiles. In this mapping algorithm, *Max-Min Ant System* (MMAS) is chosen because it has higher convergence speed than other ant systems

and it can avoid search stagnation by limiting the range of the pheromone trail strengths to the interval $[t_{min}, t_{max}]$ [159].

In Algorithm 4 (see Line 1 to Line 12), the probability (pheromone) between the tasks and corresponding mapped tiles are initialized. Then the ants will map the tasks to the tiles one by one considering the mapping probability. After all the ants finish their mapping, only the best ant which obtains the most optimal results can update the pheromone. After several iterations an optimal solution is found. The *fitness* function calculation for the algorithm is given below:

$$fitness = \sum_{i=1}^{n} \sum_{j=1}^{n} c \times d(\Theta_i, \Theta_j) \tag{4.9}$$

Here c is described in Equation 4.4 and $d(\Theta_i, \Theta_j)$ is the *Manhattan distance* (number of tiles) between Θ_i and Θ_j.

$$d(\Theta_i, \Theta_j) = d(\Theta_i, \Theta_j)_x + d(\Theta_i, \Theta_j)_y = |(\Theta_i - \Theta_j)\%M| + |(\Theta_i - \Theta_j)/M| \tag{4.10}$$

To compare the performance and usability of *ACO* it is compared to *Genetic Algorithm* (GA) [98] and to *List-Based* (LB) heuristics for application mapping [70]. A heuristic to keep the *LB* very simple by putting the most communication intensive tasks closer to each other around the center of the tile based architecture is used. *LB* is not suitable for multiple-objective optimization. Again to compare *ACO* with *branch-and-bound*, it is observed that *ACO* does not use *upper bound* or *lower bound* to trim probability solutions. It just uses the best ants to update the pheromone to influence the next iteration. Therefore, it does not cause exhaustive enumeration like *branch-and-bound* [159].

GA uses crossover and mutation to breed new population, but in application mapping, the principle of crossover and mutation should be carefully selected, otherwise, illegal solution is generated. *ACO* only uses local search to find the local optimal solution thus changes between two solutions will not cause illegal results. Therefore, the local search can be used very safely. *ACO* enjoys advantages like positive feedback, distributed-computation and fast convergence while local optima may be avoided by pheromone evaporation. The performance of *ACO* compared to *GA* and *LB* considering only the amount of communication volume as the cost function is shown in Figure 4.9 for different sizes of applications. The applications are created from *TGFF* [46]. The result shows that, *ACO* provides better optimization result than *GA* and *LB* and also it has a higher convergence rate compared to *GA* especially for larger applications.

4.3.2 Probabilistic Analysis

In order to further optimize the number of *VCBs* after the first step, the application-driven traffic model is carefully analyzed at design time. It is observed that a wide range of embedded system applications follow a standard statistical model i.e *poisson distribution*, *gaussian distribution*, etc. for their corresponding traffic modeling and this behavior can be captured at design time during application exploration [78]. A *VCB* optimization methodology depending on the given

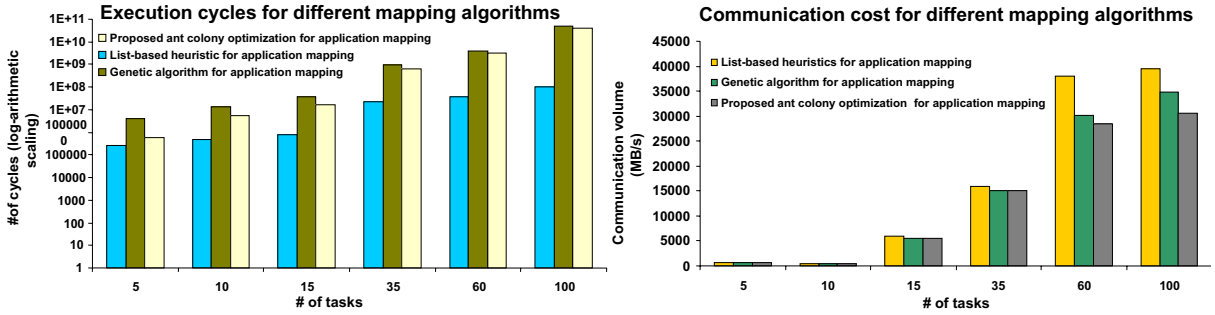


Figure 4.9: Comparison of ACO with the GA and the LB algorithms

application traffic model (*poisson distribution* is considered here) is given below. The presented methodology does not only constrained to *poisson distribution* model but depending on the traffic behavior of the given application, i.e. *gaussian distribution*, etc. similar analysis may be done.

4.3.2.1 Traffic Modeling

Multimedia applications, for instance, exhibit much more predictable traffic behavior than that found in traditional networks. The *poisson distributed model* is a mathematical model commonly used in network analysis. It correctly models events which occur independently at random intervals. In a NoC, traffic patterns are generally more regular for a wide variety of applications. In wide range of digital media applications such as a motion recognition application of a robot called *Image Processing Line* (IPL) [8], for instance, packets are sent from one task to another in fairly regular intervals making it easy to determine well-defined average intervals. In cases where a task does not send anything over one interval and does in another, it is simply considered the sending interval. The *poisson distribution* is then calculated for that interval. Packet arrival in the proposed model is assumed to be *poisson distributed*. Thus, the probability of X packets arriving in a given time interval T can be expressed by Equation 4.11. The value of λ_{TR} is the number of arrivals of a particular transaction expected in the interval.

$$P_{\lambda_{\mathrm{TR}}}(X = k) = \frac{\mathrm{e}^{-\lambda_{\mathrm{TR}}} \lambda_{\mathrm{TR}}^{k}}{k!} \tag{4.11}$$

where:

$$\begin{aligned} \lambda_{\mathrm{TR}} &= \frac{\mathrm{T}}{\text{Average arrival time}} \\ k &= \text{Arrivals occurring in T} \end{aligned}$$

The *poisson distribution* only provides information on the number of packets expected in an interval. For some analysis, such as to predict *VCB* use, it is necessary to make a prediction on expected arrival times. Packet arrival behavior can be analyzed using the *poisson distribution's*

respective *poisson process* shown in Equation 4.12. Here $P((N(t+\tau) - N(t)) = k)$ denotes the probability for $N = k$ events to occur in the arbitrary time interval of length τ. This allows us to make predictions on an event's expected time.

$$P_{\lambda_{\mathrm{TR}}}((N(t+\tau) - N(t)) = k) = \frac{\mathrm{e}^{-\lambda_{\mathrm{TR}}\tau}(\lambda_{\mathrm{TR}}\tau)^k}{k!} \tag{4.12}$$

Algorithm 4 Application-specific VCB assignment

$ant_{best,glob}$: Globally decided best ant at time t

initialize($pher\_tab$)
// initial values of the entries are $\frac{1}{N_{tiles}}$
while !exit_condition **do**
// ACO iterations can be limited by time or by maximum number of iterations
  initialize(ant_{pop})
  // generates a new ant population
  construct_solutions($ant_{pop}, TG_{vol}, pher\_tab$)
  // assign ants' tasks to tiles
  calculate_fitness(ant_{pop})
  // cost_function: $c = \alpha\frac{\#VCB}{\#VCB_n} + (1-\alpha)\frac{Comm_{vol}}{Comm_{vol,n}}$
  local_search(ant_{pop})
  evaporate($pher\_tab$)
  drop_pheromone($pher\_tab, ant_{pop}$)
  if fitness($ant_{best,pop}$) < fitness($ant_{best,glob}$) **then**
    $ant_{best,glob} \leftarrow ant_{best,pop}$
  end if
end while
Use the probabilistic approach after the application mapping using *ACO*
for all tiles t_i **do**
// All the tiles in the NoC
  if there are any transmission with the East tile **then**
  // connections in the East
    Use the probabilistic approach to optimize VCB depending on q
  end if
  if there are any transmission with the West tile **then**
  // connections in the West
    Use the probabilistic approach to optimize VCB depending on q
  end if
  if there are any transmission with the North tile **then**
  // connections in the North
    Use the probabilistic approach to optimize VCB depending on q
  end if
  if there are any transmission with the South tile **then**
  // connections in the South
    Use the probabilistic approach to optimize VCB depending on q
  end if
end for

4.3.2.2 VCB Reduction Considering Quality-of-Service

The required number of *VCBs* to satisfy a QoS parameter q is now analyzed, which defines the probability for successful transmission in terms of meeting the provided QoS bounds on throughput and latency in the proposed architecture. This translates to the QoS-supported packet transmission and thus needs to be analyzed. Here, the *VCB* reduction is done after the first step in the design-space exploration to optimize the number of *VCBs*, where the number of *VCB* for concurrent connections and also designer defined flexible provision for transmission are already decided.

In the scope of this work, the QoS parameter *q* is defined to be a probabilistic value of the time-related QoS parameter latency and throughput *Q(l,t)*. If a router architecture $\Re(B, DL)$, (*B* represents buffer elements and *DL* represents decoding logic) can provide bounded latency and throughput, means 100% of its required QoS value, then *q* is assumed to be 100%. The value of *q* is parameterised by the number of *VCBs*, as if there are rooms for every transaction considering the worst case: concurrent transactions. Every unit transaction will get a *VCB* in each output port if it is demanded after the first step of the application-specific *VCB* assignment. The availability of the *VCB* is guaranteed, because in the first step the buffer assignment is done depending on the total number of worst-case concurrent transactions. The value of *q* less than 100% influences the packet blocking time in the buffer assignment for concurrent transactions and thus cannot meet the 100% guarantee.

Depending on all these properties of the router architecture and the application traffic model (*poisson distribution*), the required number of *VCBs* using the following probabilistic analysis is now calculated. The expected values of the *poisson distributions* $\mathrm{E}(X) = \lambda$ to calculate the probability of TR_a claiming the first *VCB* is used:

$$P(\mathrm{VCB}_1 = \mathrm{TR}_a) = \frac{\lambda_{\mathrm{TR}_a}}{\sum\limits_i \lambda_{TR_i}} \tag{4.13}$$

The assignment of subsequent *VCBs* is then the conditional probability of the assignment of previous *VCBs*. Therefore, taken the probability for the first *VCB* it is examined the *poisson processes* of each transaction for the time it is occupied. Since the *VCBs* are *filled up*, it is assumed them to be initially empty. Therefore, τ is only proportional to the TR in the first *VC*.

$$\tau = \frac{(p_a)(2m+n)}{\mathrm{T}}$$

With a given unit time interval *T*, the packet size in flits of $\mathrm{TR}_a = p_a \times (2m+n)$ is the number of cycles that a flit remains inside the buffer. The probability of TR_b occupying the second *VCB* is then:

$$P(\mathrm{VCB}_2 = \mathrm{TR}_b \mid \mathrm{VC}_1 = \mathrm{TR}_a) = (1 - \mathrm{e}^{-\lambda_{\mathrm{TR}_b}\tau}) \times P(\mathrm{VCB}_1 = \mathrm{TR}_a)$$

where:

$$\tau = \frac{\min(p_b, p_a)(2m+n)}{\mathrm{T}}$$

Here, $P(\mathrm{VCB}_2 = \mathrm{TR}_b)$ is calculated from Equation 4.12. To calculate $P(\mathrm{VCB}_2 = \mathrm{TR}_b)$, it is first calculated $P(\mathrm{VCB}_2 \neq \mathrm{TR}_b)$ by taking the value $k = 0$ in the equation and then it may be found, $(1 - P(\mathrm{VCB}_2 \neq \mathrm{TR}_b))$ = $(1 - \mathrm{e}^{-\lambda_{\mathrm{tr}_b}\tau})$. Now for the nth *VC*:

$$P(\mathrm{VCB}_n = \mathrm{TR}_a \mid \mathrm{VCB}_{n-1} = \mathrm{TR}_b) = (1-\mathrm{e}^{-\lambda_{\mathrm{TR}_a}\tau}) \times P(\mathrm{VCB}_{n-1} = \mathrm{TR}_b \mid \mathrm{VCB}_{n-2} = \mathrm{TR}_c) \tag{4.14}$$

In order to meet the QoS requirements, the usage of a *VCB* must be guaranteed for a given percentage q contention-less packet transmission. To accomplish this, its usage may not exceed $(1 - q)$. That is, a *VCB* is not required if:

$$\bigcup_{a,b}(P(\mathrm{VC}_n = \mathrm{TR}_a \mid \mathrm{VC}_{n-1} = \mathrm{TR}_b)) < (1 - q) \tag{4.15}$$

Therefore, to optimize the number of *VCBs* in the first step, a multi-objective mapping function during application mapping considering the worst case concurrent transactions that provides 100% QoS, is used. To further optimize the number of *VCBs*, the application-specific traffic model is analyzed. In the second step, depending on the QoS value the designer can find an optimal solution to assign number of *VCBs*.

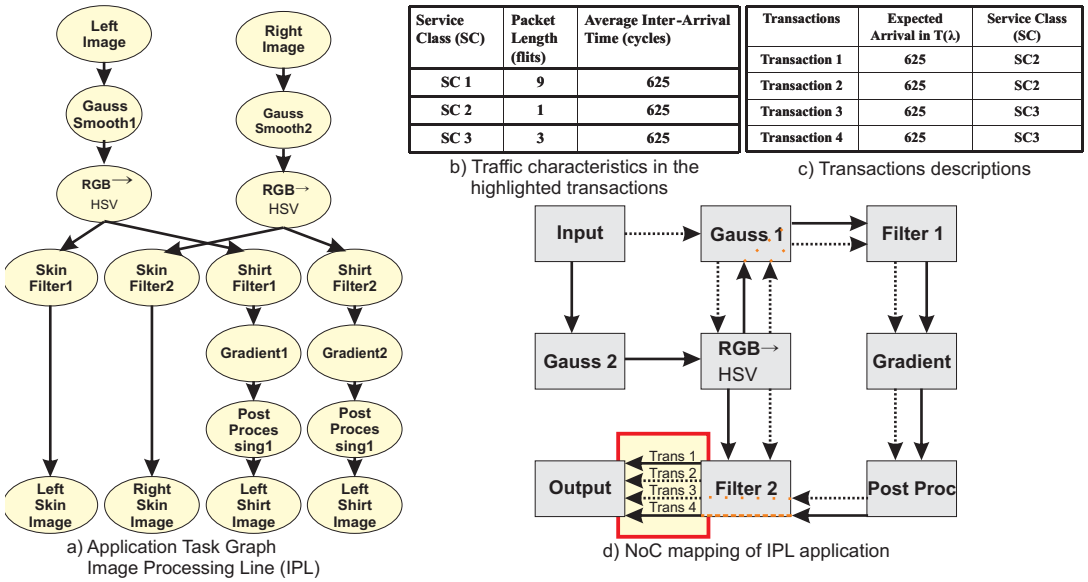
The pseudo code of the *VCB* reduction methodology is presented in Algorithm 4. In this algorithm, Line 1 to Line 12 present the pseudo code for the *ACO-based* mapping algorithm. A global pheromone table, $pher\_tab$ is initialized by an uniform value considering the number of tiles, N_{tiles}. Ants in the algorithm can update the pheromone table and pheromone evaporation is also used to overcome the local minima in the optimization algorithm. Line 13 to Line 27 presents the probabilistic approach based on the traffic model and QoS parameter. It is applied after a optimum mapping is achieved using the proposed mapping algorithm.

4.3.3 Evaluation of the Proposed Methodology

Transactions	$P(\mathrm{VC}_1 = \mathrm{TR}_x)$
TR_1	25.0%
TR_2	25.0%
TR_3	25.0%
TR_4	25.0%

Table 4.1: Likelihood of transaction i occupying VC_1

An *IPL* [8] application, shown in Figure 9.2 in Chapter 9 for the case study analysis to reduce the number of *VCBs* is used here. The application shown in Figure 4.10 (a), is mapped onto a 3×3 NoC architecture shown in Figure 4.10 (d) to gain performance increase in terms of



Service Class (SC)	Packet Length (flits)	Average Inter-Arrival Time (cycles)
SC 1	9	625
SC 2	1	625
SC 3	3	625

Transactions	Expected Arrival in T(λ)	Service Class (SC)
Transaction 1	625	SC2
Transaction 2	625	SC2
Transaction 3	625	SC3
Transaction 4	625	SC3

Figure 4.10: IPL mapping and VCBs reduction through probabilistic analysis

Transactions	$\mathrm{VC}_1 = \mathrm{TR}_1$	$\mathrm{VC}_1 = \mathrm{TR}_2$	$\mathrm{VC}_1 = \mathrm{TR}_3$	$\mathrm{VC}_1 = \mathrm{TR}_4$
$P(\mathrm{VC}_2 = \mathrm{TR}_1)$	0.0%	0.4%	1.17%	1.17%
$P(\mathrm{VC}_2 = \mathrm{TR}_2)$	0.4%	0.0%	1.17%	1.17%
$P(\mathrm{VC}_2 = \mathrm{TR}_3)$	0.4%	0.4%	0.0%	1.17%
$P(\mathrm{VC}_2 = \mathrm{TR}_4)$	0.4%	0.4%	1.17%	0.0%

Table 4.2: Likelihood of transaction i occupying VC_2

more frames decoding per second (from 5 fps in uniprocessor system to 25 fps in a MPSoC). For the experimental setup some assumptions are made: the image size is 320×200 pixels, it decodes 25 frames per second what means 1 pixel per 625 ns, each flit payload size is 24 bits (1 RGB pixel), robot left eye and the right eye process image parallely and the *VCB* depth is of $(2m + n)$.

In a fixed *VCB* assignment approach, the NoC needs 96 *VCBs*. After using the first step of the proposed *VCB* reduction methodology the number of *VCBs* can be optimized to 20 for the presented optimized mapping. The second step of the methodology depends on the analysis of two neighboring tiles. For this application it is only shown the *VCB* reduction for the tiles *Filter 2* and *Output*. From *Filter 2* to *Output* there are 4 *VCBs*. The traffic characteristics for all these connections are also shown in Figure 4.10 (b,c). Two transactions are of *Service class 2* (see Definition 2 in Chapter 3) and the other two are of *Service class 3*. The detailed calculations are shown in Table 4.1, Table 4.2, and Table 4.3. Depending on the probabilistic calculations, the number of *VCBs* can be optimized to 2 from 4 for this physical link as *VCB3* and *VCB4* has the probability of use 0.44% and 0.01% respectively and these can be neglected in terms of provided QoS budget (this budget may be provided by the designer depending on the application characteristics). Other connections can be also calculated using the same rules. In

Table 4.1, using the Equation 4.13 the probability to have at least 1 *VCB* is shown (100%). The expected probability to use a second *VCB* is 9.07% and the other values are shown in Table 4.3 using Equation 4.14.

Virtual Channel Buffer (*VCB*)	Probability of use, P(*VCB*)
VCB_1	100%
VCB_2	9.07%
VCB_3	0.44%
VCB_4	0.01%

Table 4.3: Probability to use each virtual channel buffer

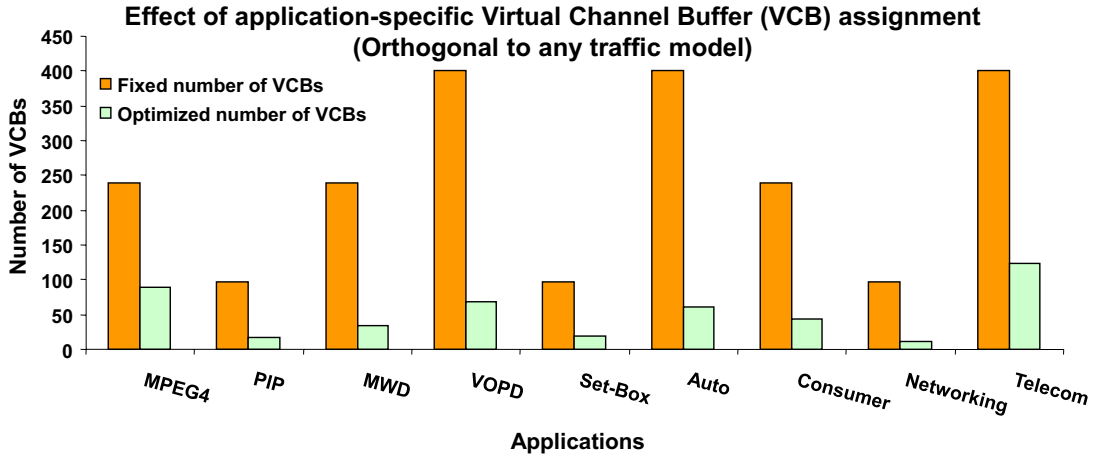


Figure 4.11: Effect of the application-specific VCB assignment for the E3S benchmark suite

In Figure 4.11, the buffer savings, taking only the *VCBs* as the optimization criteria for a collection of multi-media applications and the E3S benchmark suite [50], compared to the fixed number of the *VCB* assignment approach is shown. The saving can be on an average of 90.2% for the applications. The number of *VCBs* in each output port for a fixed approach is taken 4 which is comparable to the QNoC [23, 24] architecture.

In Figure 4.12, the effect of multiple goal functions in application mapping for the E3S benchmark suite [50] is shown. The bar chart shows the normalized number of *VCBs* and the lines show the normalized amount of total communication volume for three different optimization criteria. The results show different optimized mappings considering only the communication volume, only the number of *VCBs*, and both respectively. Table 4.4 shows that, for a collection of multimedia and E3S embedded applications a multi-objective goal function during application mapping can optimize the number of *VCBs* and the communication volume near to an optimal solution compared to considering these parameters separately (the first step of the proposed methodology).

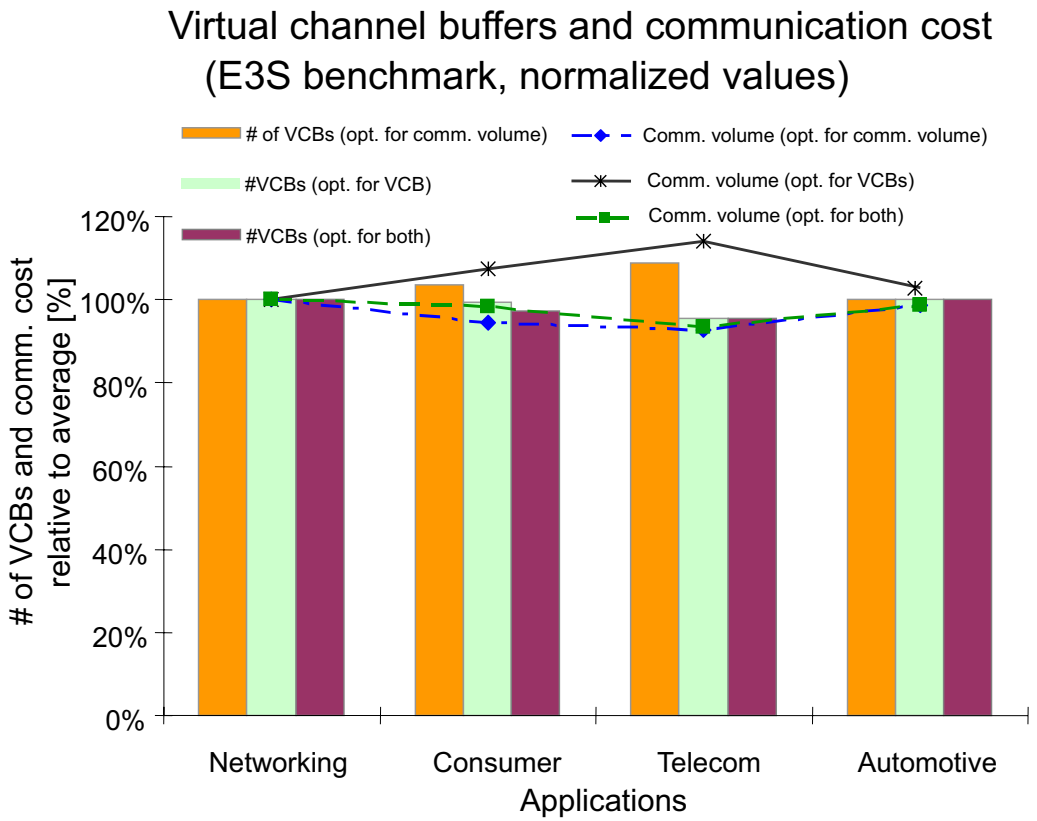


Figure 4.12: Effect of the multi-objective goal function for the E3S benchmark suite

4.4 Road to Runtime Adaptation: Parameters

In this section, the parameters that may be adapted at runtime depending on the need of the adaptive system-on-chip communication architecture are discussed. The parameters that are possible to be customized for an application-specific NoC design are presented in Section 4.1. The best case for an application to be executed onto a NoC would be to optimize for all the presented parameters, but at runtime it is not possible to customize all the parameters. An extensive exploration has been done depending on the requirements of the adaptive system-on-chip communication architecture to find the runtime parameters that may be adapted. A detailed explanation of the requirement of such architecture and its parameters are explained in the following chapters in Chapter 5, Chapter 6, and Chapter 7. The parameters that are identified in the scope of this thesis may also be viewed as a 3D design-space exploration graph similar to one shown in Figure 4.1.

Similar to the 3D design-space exploration for the application-specific NoC, the *architecture-level parameters* for the runtime adaptive NoC can be presented as $X = (Bu, BW)$, where Bu represents the *buffer* inside the router and BW is the *supported bandwidth* of the links. The *communication paradigm* is presented as $Y = (Ro)$, where Ro is the *routing algorithm*. Both these two dimensions together form the *architecture-level* part of the proposed, runtime *Adaptive Networks-on-Chip* (AdNoC) presented in the following chapters. The third dimension,

Application	#VCs (comm_v)	#VCs (vc)	#VCs (both)	c_cv (comm_v)	c_cv (vc)	c_cv (both)
VOPD	42	37	37	4618	6026	5025
MPEG4 decoder	39	34	34	7246	8545	7457
PIP	9	9	9	661	653	646
MWD	17	17	17	1451	1490	1429
Set-Top Box	11	11	11	900	990	923
Auto	26	26	26	136	142	136
Consumer	17	16	16	52	59	54
Networking	9	9	9	44	44	44
Telecoms	41	36	36	129	159	130

Table 4.4: Tasks and service class specification

the *application mapping* is presented as $Z = (Mp)$, where Mp is the *runtime application mapping* and it falls in the *system-level* part of the proposed *AdNoC*. In the following, these parameters that have been identified to be customized at runtime are explained in short.

Runtime Virtual Channel Buffer Assignment: Single *VCB* for each transaction in each output port is used in the proposed *AdNoC* architecture similar to the application-specific NoC presented earlier in this chapter. In the application-specific NoC, the *VCBs* are tightly coupled to each output port at design time and therefore, it is not possible to change the number of *VCBs* in each output port at runtime. To accomplish runtime assignment, a central pool of *VCBs* has been used instead of tightly coupled *VCBs* in each output port for the proposed *AdNoC*. The proposed on-demand *VCB* assignment increases the resource utilization of the buffer and thus compensate the extra hardware used to achieve this flexibility. Details are explained in Chapter 7.

Runtime Bandwidth Configuration: It is explained in 4.1.1 that the bandwidth of a link is determined by the *link-width*. At runtime it is not possible to increase the *link-width* and therefore, a novel technique named as the *2X-Link* has been designed to change the *supported bandwidth* by using two *half-duplex links* instead of two *simplex links*. The proposed *2X-Links* can operate in three different bandwidth modes: (1) both in one direction (both have X unit of bandwidth) (2) both in reverse direction (first one has 2X and the second one has 0 units of bandwidth) and (3) both in opposite direction (first one has 0 and the second one has 2X units of bandwidth). Details are explained in Chapter 7.

Runtime Adaptive Routing Algorithm: The source-based deterministic *routing algorithm* may not be used for the *AdNoC* as the application is changed over time in such an architecture. Therefore, an adaptive *routing algorithm* based on the *XY-routing* algorithm named as the *wXY-routing* algorithm is presented in the thesis. In the *wXY-routing* algorithm, the route is calculated in a distributed way by considering the available bandwidth in each output port. The deadlock and starvation problem are also solved by using a *TTL* counter for each transaction. Details are explained in Chapter 7.

Runtime Application Mapping Algorithm: The design-time mapping algorithms discussed earlier may not be used for the adaptive system-on-chip communication architecture, (*AdNoC*), because an adaptive system that changes its configuration over time requires

a (re-)mapping/runtime mapping of applications. Therefore, in the *system-level* part of the *AdNoC*, a runtime agent-based distributed application mapping algorithm named as *ADAM* is employed. Details are explained in Chapter 6.

4.5 Conclusion

A case study analysis and a complete design flow of an application-specific NoC on top of the proposed QoS-supported system-on-chip communication architecture presented in Chapter 3 are presented here. In the scope of the chapter, the novel contribution to reduce the *VCB* using a two-step scheme is also presented. It is shown that, on an average 90.2% reduction in the number of *VCBs* compared to the state-of-the-art QNoC , a fixed allocation for the E3S embedded application benchmark suite using the first step may be acheived. In the second step, the reduction depends on the designer, the QoS parameter and the application-specific traffic model. The complete two-step *VCB* reduction scheme by means of a robotic application (Image Processing Line) case-study analysis is also demonstrated here. Finally, this chapter has introduced the road to design of the novel *AdNoC* using the design-space exploration knowledge of an application-specific NoC on top of the similar QoS-supported system-on-chip communication architecture.

Chapter 5

AdNoC: Runtime Adaptive Networks-on-Chip

Diverse and during runtime varying workloads and/or constraints in embedded systems require runtime adaptivity to provide a high degree of efficiency during any operation mode/scenario. Design-time decisions can often only cover certain scenarios and fail in efficiency when hard-to-predict system scenarios occur. In this thesis, the first approach of an adaptive system-on-chip communication architecture is presented. The adaptive system provides adaptivity both in the *system-level* as well as in the *architecture-level*. The *system-level* adaptation is provided by using a runtime agent-based distributed application mapping (for details see Chapter 6). The *architecture-level* adaptation is implemented by using several novel methodologies to increase the resource utilization (silicon fabric), i.e. sharing the *Virtual Channel Buffer* (VCB) among different output ports, changing the supported link bandwidth, etc. (for details see Chapter 7). To achieve the successful adaptation, the runtime observability is a prerequisite as it provides necessary system information gathered on-the-fly. Therefore, a comprehensive runtime observability scheme (for details see Chapter 7) for the proposed adaptive system-on-chip communication architecture is also presented in this thesis.

The novel contribution in the scope of this chapter is as follows:
The novel runtime *Adaptive Networks-on-Chip* (AdNoC), as an adaptive system-on-chip communication architecture is introduced in this chapter. The runtime *AdNoC* architecture is built on top of the presented QoS-supported NoC presented in Chapter 3. Runtime adaptation is achieved both in the *system-level* as well as in the *architecture-level* of the proposed *AdNoC* architecture (published in [52, 53, 54, 58]).

5.1 Motivation

Application specific *Networks-on-Chip* (NoCs) as discussed in Chapter 4, are design-time parameterized architectures with a custom topology, a fixed routing scheme, and a fixed number of allowed virtual connections at each output port [14, 52]. The application-specific NoCs are generally tailor-made for a certain application or an application domain and fail in scenarios of hard-to-predict system behavior and/or in situations where reliability is a concern. Some scenarios are as follows :

- The system constraints may change during runtime.
- The user of the system may change his pattern of how to operate/use the system.
- Smaller feature sizes in the nano age will invoke the reliability concerns. It will require building future reliable systems out of un-reliable components. For further details on the reliability concerns in upcoming technology nodes the reader is referred to [26].

All those scenarios – from the user behavior to the reliability issues – require designing systems with adaptivity capabilities in mind such that the system may allow application variations and may react on faulty situations accordingly. The implementation of a reliable communication-centric system-on-chip may, for example, depend upon the ability of the NoC to route traffic in such a way that it can efficiently bypass the faulty areas at runtime. Therefore, in the scope of this thesis, a self-adaptive system-on-chip communication architecture that analyzes itself during runtime and self adapts when and how a certain router should be configured for a certain transaction is presented. Adaptivity is required both in the *system-level* as well as in the *architecture-level* where it is realized through modification thereof, or even new paradigms in the architectural design. The software part in between the application and the underlying hardware layer executed in the *Processing Element* (PE) is considered as the *system-level* (see Definition 5) and the data transmission part which is implemented in the hardware as the *architecture-level* (see Definition 6). Changes in the user behavior, system constraints and/or reliability issues can be effectively compensated at the *system-level* by, for instance, dynamically (re-)mapping a running application at runtime. *Architecture-level* modifications on the other side may help to increase the resource utilization at runtime [52] besides the main purpose of providing flexibility.

5.2 Advantages of the Adaptive NoC over the Application-specific NoC

In Table 5.1 (symbol "(-)" stands for showing the disadvantages and "(+)" for showing the advantages), the qualitative advantages of the *AdNoC* over the application-specific NoC are summarized. A table comparing the architectural difference between the application-specific NoC and *AdNoC* is already given in Chapter 3 (see Table 3.2). The *AdNoC* architecture presented in this thesis is flexible, efficient, and reliable. Due to the fact that the *AdNoC* supports the adaptive routing and the runtime application mapping, it may support the varying workloads for the embedded systems application. It uses resource multiplexing by reusing different parts of the router and therefore, higher resource utilization is achieved in an *AdNoC*. The higher resource utilization minimizes the design-time estimation for the hardware resources. Therefore, the over-design may be avoided for the *AdNoC* architecture compared to the design-time parameterized general purpose NoCs design.

	(Design-time) Application-specific NoC		(Runtime) Adaptive NoC (AdNoC)	
Flexibility	All the design related parameters, i.e. number of virtual channel, buffer depth are fixed at design time. No flexibility.	**(-)**	All the design-related parameters may be configured at runtime to meet the requirements of the different tasks. Therefore, high flexibility.	**(+)**
Routing & Mapping Algorithm	Routing and mapping are static and performed offline. Therefore, they are only efficient in specific scenarios.	**(-)**	Dynamic routing and runtime mapping are used. Therefore, may support varying workloads and/or constraints in systems.	**(+)**
Hardware Resources	Hardware resources are allocated to each part of NoC at design time. Therefore, no resource multiplexing (lower resource utilization).	**(-)**	Hardware resources may be shared and reused. Therefore, higher resource utilization and lower silicon area.	**(+)**
Efficiency	Design-time decisions only cover certain scenarios and fail in efficiency when hard-to-predict system scenarios occur.	**(-)**	The system may be adapted both at system-level as well as *architecture-level* during unpredicted scenarios.	**(+)**
Reliability	Static design and therefore, may not be able to recover from faults.	**(-)**	The system-level adaptation allows to move a complete task to another PE and therefore, may recover from faults. Architecture-level adaptation supports resource reuse to other parts. Higher resource utilization during faults.	**(+)**
Complexity of control logic	Simple implementation and the control management is easy to realize.	**(+)**	Increased complexity. The control requires additional hardware and consumes more power. Over-designed, hard-to-predict the initial hardware budget.	**(-)**

Table 5.1: Advantages/disadvantages of the adaptive and the application-specific NoCs

5.3 Runtime Adaptive Networks-on-Chip (AdNoC)

In this section, a short summary of the proposed *AdNoC* architecture is presented together with some related definitions that will help to understand the rest of the thesis.

5.3.1 AdNoC Specific Definitions

In addition to the general NoC-related definitions given in Chapter 3, the novel *AdNoC* architecture-related definitions are described in the following:

Definition 1: *Logical Network* (LN) at time t is a directed graph $L_t = (M, W)$, where M is a set of task groups m_i and $w_{i,j} \in W$ represents the set of connections between two task groups m_i and m_j. A task group m_i is a set of tasks scheduled to be executed on a particular PE. Therefore, *LN* is the subset of the task graph set G (see Definition 1 in Chapter 3) that are running at a specific time t.

Definition 2: The *task mapping function* is a function $l_t : T' \subseteq T \mapsto L_t$ which maps subset T' of each task graph T in the task graph set G (see Definition 1 in Chapter 3) to the logical network LN. Specifically T' can be equal to the entire task graph.

Definition 3: The *network mapping function* is a function $p_t : L_t \mapsto S \subseteq P$ which maps a logical network, LN onto a subset of the physical network, PN. The function changes in discrete time intervals.

Definition 4: The *buffer configuration* $b_{i,t}$ is the current buffer configuration of tile $n_i \in N$. Buffers are required to implement virtual connections between two tiles to realize multiple mutually exclusive transactions. The definition of *VC* and *VCB* are given in Chapter 3 (see Definition 5). The buffer configuration provides the current assignment of individual buffer blocks and thus the number of available *VCs* to each output port at time *t*. The set of current configurations is B_t.
Definition 5: The *system-level* of the adaptive system-on-chip communication architecture SL is a collection of the top three layers and partially the other two layers (the *transport layer* and the *network layer*) of the ISO/OSI model presented in Chapter 3 (see Figure 3.1), task mapping function l_t, and network mapping function p_t. The three layers are: the *application layer*, the *presentation layer*, and the *session layer*. All these functionalities are assumed to be running in the PEs of the *AdNoC* architecture.
Definition 6: The *architecture-level* of the adaptive system-on-chip communication architecture AL is the collection of the bottom four layers of the ISO/OSI model presented in Chapter 3 (see Figure 3.1) together with the routing function r presented in Chapter 3 (see Definition 8) and the buffer configuration $b_{i,t}$. The bottom four layers that are considered as a part of the *architecture-level* are: the *transport layer*, the *network layer*, the *data link layer*, and the *physical layer*.
Definition 7: The *system monitor* M is a hardware component inside the runtime observability infrastructure, which is used to collect, aggregate, and process system statistics. It spans from the top to the lower level of the entire architecture. The system monitoring abstraction is predominantly realized in the *system-level* in software and in the *architecture-level* in hardware.
Definition 8: A *cluster* is a subset $C_i \subseteq N$, where N is the set of tiles n_j that belong to the physical network PN presented in Chapter 3 (see Definition 6) and a *virtual cluster* Cv_i is a cluster where there are no fixed boundaries to decide which tiles are included and which tiles are not. It can be created, resized and destroyed at runtime.
Definition 9: An *agent* Ag is a computational entity, which acts on behalf of others. The construction of an agent is motivated from [18, 64] where agents are used for distributed network management. The properties of an agent in the proposed architecture are: an agent (1) is a smaller task closer to the system, (2) must do resource management, (3) may need memory to store state information for the resources, (4) must be executable on any PE, (5) must be migratable, (6) must be recoverable, (7) is able to be instantiated for a new cluster, and (8) is able to be destroyed if the cluster no longer exists. An agent-based mapping algorithm provides a flexible framework for runtime application mapping because it has the negotiation capability among the clusters distributed over the whole chip and is not dependent on the design-time parameters (see above properties).
Definition 10: A *cluster agent* $Ag^{C_i} \in Ag$ is an agent that is responsible for mapping operations in the cluster C_i. The *cluster agent* is located in the processing element $p_j^{C_i}$ where the index j of p_j suggests that the *cluster agent* can be located on any PE of the cluster. The Ag^{C_i} stores the information about the *cluster* that the agent is responsible for.
Definition 11: A *global agent* Ag^{glob} is an agent that stores the information for performing the mapping operations of a selected cluster. It stores information that describes the current usage of communication and computation resources for each cluster C_i and this information is used for selection and re-organization of the clusters. Ag^{glob} is movable and the stored information

is light-weight and easily recoverable (there are multiple instances of the *global agents*).
Definition 12: The application *mapping function* is given by $m : T \ni t_i \mapsto n_j \in N$ and the *runtime application mapping function* m_{run} maps the instance of *task graph set* G_t at time t to the physical network PN presented in Definition 6 of Chapter 3.
Definition 13: A *binding* is a function $b : T \ni t_i \mapsto tp_{PE} \in Tps$, where T is the set of all tasks of an application and Tps is the set of the PE types that are used on the physical network PN. The function assigns each task t_i of the TG to a favorable type of PE. After the binding operation is completed, the tasks are allowed to be mapped only to PEs of the type given by the binding function b.
Definition 14: The *Adaptive Networks-on-Chip* (AdNoC) is defined as the tuple $AdNoC = (PN, M, G_i, p_t, m, r, SL, AL, L_t, l_t)$ with the parameters as given above. The values of t are from discrete intervals. There may be multiple task graphs simultaneously (G_i) running on *AdNoC*. The definitions of G_i and PN are given in Definition 1 and Definition 6 respectively in Chapter 3.

5.3.2 Overview of the AdNoC Architecture

AdNoC, the proposed runtime adaptive system-on-chip communication architecture is capable of supporting deadlock-free data transmission and meets the required bandwidth guarantees for the concurrent transactions in system-on-chip communication for a network exposed to varying system constraints and/or mode switches. A *transaction-level* connection-oriented approach on top of a packet-based communication to provide the performance-related guarantees, i.e. bandwidth requirements for the critical transactions, as well as a pure best effort approach for the rest of the transactions (a hybrid approach) is used. As most NoCs, the architecture is pipelined, deploys wormhole-based distributed routing, and the topology is a regular 2D Mesh (for details see Chapter 3). *AdNoC* is divided into two main parts: the *system-level* (see Definition 5), and the *architecture-level* (see Definition 6). Figure 5.1 shows an overview of the complete *AdNoC* architecture. Details of the *system-level* and the *architecture-level* are presented in Chapter 6 and in Chapter 7 respectively.

5.3.3 System-level Adaptation

Adaptivity at *system-level* is deployed using the runtime application mapping algorithm in which a runtime agent-based (for the definition of agents see Definition 9) distributed application mapping is proposed. The detailed scheme is explained in Chapter 6. In summary, to obtain a scalable mapping solution the computational load is reduced by confining the mapping to *clusters* (see Definition 8) which are a connected subset of NoC tiles. The clusters have a variable size that can be adjusted during runtime and each cluster has one *cluster agent* (see Definition 10) which is responsible for (re-)mapping.

There are several reasons for (re-)mapping at different levels, e.g. user-behavior from the application-level or hardware faults from the *architecture-level*. The *cluster agent* first tries to find a suitable (re-)mapping for a mapping request. In the event that the *cluster agent* is not able to establish a

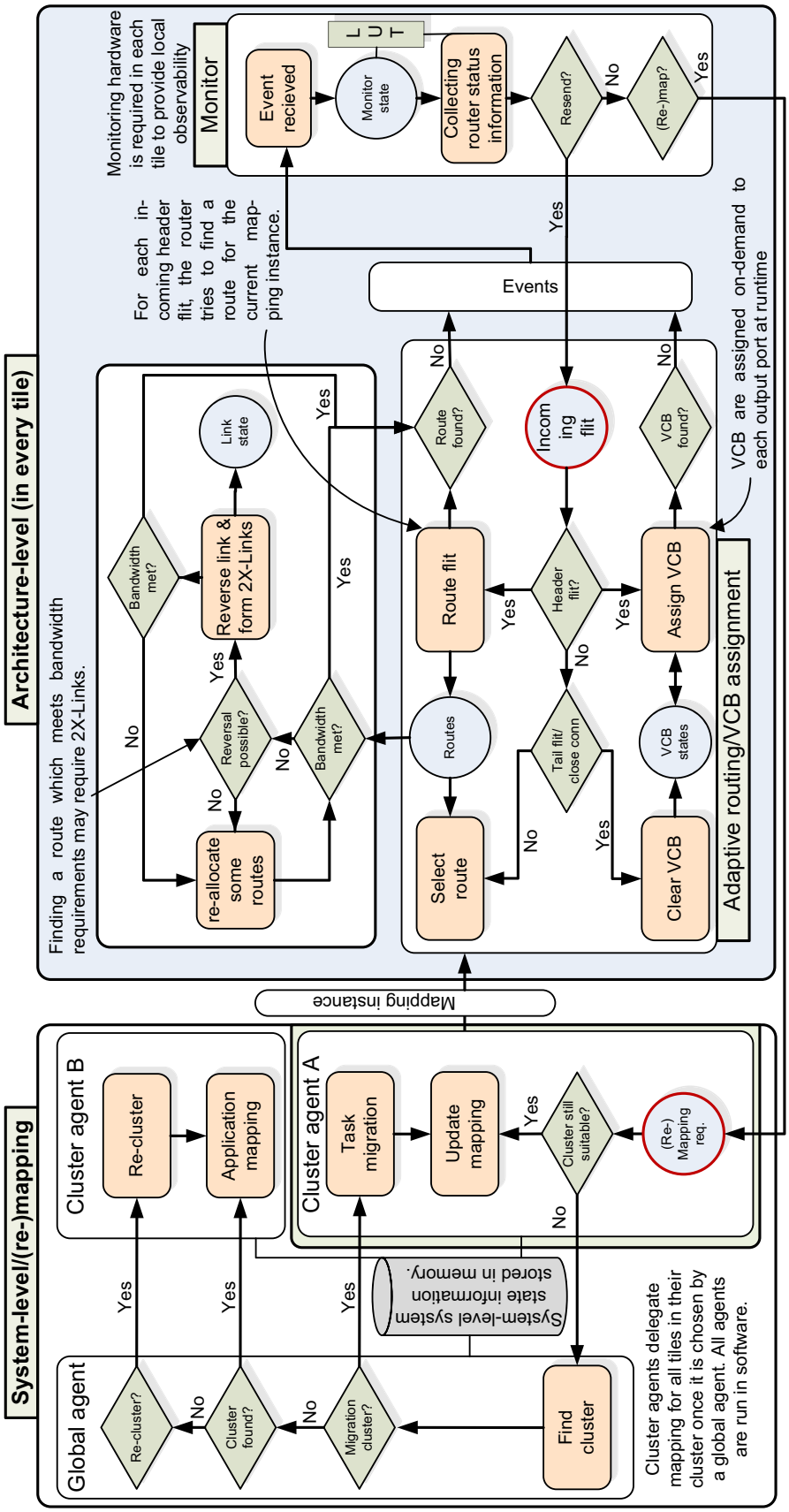


Figure 5.1: The adaptive system-on-chip communication architecture

mapping instance it informs a *global agent* (see Definition 11). Those are special agents which are in charge of selecting a different cluster, re-clustering, and coordination. The *global agent* then tries to resize the cluster associated with the *cluster agent*. If this fails, a different cluster is chosen and a new mapping is done. All agents are implemented in software and may be migrated to run on any PE in every tile within their deployment area (i.e. in a cluster or globally).

5.3.4 Architecture-level Adaptation

After the *system-level* has successfully set up a mapping instance, it is up to the *architecture-level* to configure each tile for the resulting transactions. Once a transaction arrives at a tile, all possible directions must first be checked for suitable routes. To find a valid route which can meet the bandwidth constraints for the transaction it may be needed to use the concept of the *2X-Link*. Up until now, the number of *VCBs* at one port has typically been fixed at design-time [14, 76] for the application-specific NoCs (see Chapter 4). With on-demand assignment [52], the *VCBs* are not tied to ports, but only to the router itself. The router may distribute the *VCBs* to any route as needed by assigning a transaction to the *VCBs* through the virtual channel arbiter and then assigning it to an output port. Details of the *architecture-level* adaptation are discussed in Chapter 7.

5.4 Conclusion

The novel idea of the adaptive system-on-chip communication architecture has been introduced in this chapter. The adaptive system provides adaptivity both in the *system-level* as well as in the *architecture-level*. The *system-level* adaptation and the *architecture-level* adaption parts are detailed in Chapter 6 and Chapter 7 respectively.

Chapter 6

System-level Adaptation in the Runtime Adaptive Networks-on-Chip

To solve the problem of mapping tasks to respective *Processing Elements* (PE), several design-time (offline) mapping algorithms have been proposed in related work. In [77], *branch-and-bound*-based, in [98] *genetic algorithm*-based, and in [70] *heuristic*-based mapping algorithms are proposed (details of these related works are already explained in Chapter 3). But an adaptive system that changes its configuration over time requires a (re-)mapping/runtime mapping of applications. Possible reasons for the necessity of a runtime application mapping are given below.

Let us motivate the need of a runtime distributed application mapping for the *Adaptive Networks-on-Chip* (AdNoC) architecture by means of a simple scenario. A 32×32 *Networks-on-Chip* (NoC) with a Mesh topology may be studied in this respect. Some events that may require a (re-)mapping at runtime for an adaptive system where design-time mapping algorithms fail are given below:

- Online detection of hardware faults.
- To minimize runtime system costs (i.e. to save energy because of the low battery status).
- When the user requirements change, e.g. the user wants to switch video playback to a higher resolution.
- When an adaptive system tries to configure the underlying NoC infrastructure (i.e. changing the routing algorithm and the buffer assignment in the *architecture-level* of the adaptive NoC architecture) and if it fails, then the mapping instance of the application needs to be changed.

The state-of-the-art runtime application mapping works [31, 38, 37, 147] have used a *Centralized Manager* (CM) (see Figure 6.1 (a)) for conducting the job of mapping which is not scalable in the context of hundreds or even thousands of cores that may soon be integrated on a *Multi-Processor System-on-Chip* (MPSoC). These architectures may bear the following problems:

- Single point of failure.
- Higher computational cost to calculate mapping inside *CM*.

- Large volume of *monitoring-traffic*. *Monitoring-traffic* is defined as the traffic which is caused by collecting information about the state of the tiles.
- Point of traffic congestion area (*hot-spot*) as every tile sends the status of the PE to the *CM* after every instance of mapping. It increases the chance of bottleneck around the *CM*.

To solve the problem of a static design-time mapping algorithm which may require a high computational effort, we need an algorithm that can perform a low-cost (execution time) mapping algorithm inside a *virtual cluster* (see Definition 8 in Chapter 5) constructed at runtime. The problems of a centralized mapping algorithm are solved in this thesis by using a distributed mapping inside each *virtual cluster*. This distributed mapping is accomplished by software modules that are autonomous, modifiable, and exhibit adaptation capabilities.

The novel contributions in the scope of this chapter are as follows:
(1) A runtime agent-based distributed application mapping for the next generation self-adaptive heterogeneous MPSoCs is presented in this chapter. The proposed application mapping algorithm is composed of two major parts: (1) a cluster selection and cluster reorganization at runtime and (2) a mapping algorithm inside a cluster at runtime (published in [58]).
(2) In the scope of this chapter, a runtime cluster negotiation algorithm that generates *virtual clusters* to solve the problems of the centralized mapping algorithm is provided.
(3) A low cost heuristic-based mapping algorithm in terms of execution cycle on any instruction set processor that minimizes the communication related energy consumption is presented.

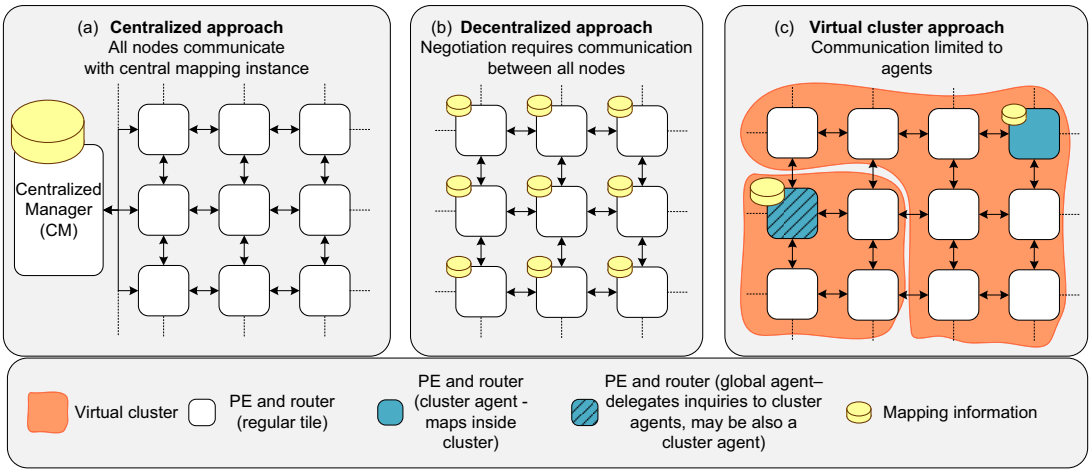


Figure 6.1: Various options for (re-)mapping

6.1 ADAM: Runtime Agent-based Distributed Application Mapping Algorithm

Adaptivity at the *system-level* is deployed using a runtime application mapping algorithm in which an *Agent-based Distributed Application Mapping* (ADAM) is proposed. An *agent* is a

computational entity, realizes in software, and acts on behalf of other entities (see Definition 9 in Chapter 5). In summary, to obtain a scalable mapping solution, the computational load is reduced by confining mapping to *virtual clusters* which are a connected subset of NoC tiles. The clusters have a variable size that may be adjusted during runtime and each cluster has one *cluster agent* (CA) which is responsible for (re-)mapping (see Figure 6.1 (c)).

There are several reasons as discussed earlier for (re-)mapping at different levels during application execution, e.g. user-behavior from the *application-level* or hardware faults stems from the *architecture-level*. The *CA* first tries to find a suitable (re-)mapping for a mapping request. In the event that the *CA* is not able to establish a mapping instance it informs a *Global Agent* (GA). These are special agents which are in charge of selecting a different cluster, (re-)clustering, and coordination. The *GA* then tries to resize the cluster associated with the *CA*. If this fails, a different cluster is chosen and a new mapping is done. All agents are implemented in software and may be migrated to run on any PE in every tile within their deployment area (i.e. in a cluster or globally).

6.1.1 Different Parts of the ADAM Algorithm

An overview of the *ADAM* algorithm is presented in Figure 6.2. The runtime application mapping in the proposed algorithm is achieved by using a negotiation policy among *CAs* and *GAs* of a certain instance of time distributed over the whole chip. In Figure 6.2, an application mapping request is sent to the *CA* of the requesting cluster which receives all mapping requests and negotiates to the *GAs*. There may be multiple instances of the *GAs* that are synchronized over time. The *GAs* have global information about all the clusters of the NoC in order to take decision to which cluster the application should be mapped onto. Possible replies to this mapping request are:

1. When a suitable cluster of the application exists then the *GAs* inform the requesting source *CA* and it asks the suitable destination *CA* for the actual mapping of the application.
2. When no suitable clusters are found by the *GAs* then the *GAs* report the next most promising cluster where it is possible to map the application after the task migration (it is called a candidate cluster). The destination *CA* or the candidate *CA* negotiates with the *GAs* to make this cluster suitable for the application mapping. The number of iterations is a design-time configuration parameter.
3. When neither suitable cluster nor candidate cluster for a task migration are found then the (re-)clustering concept is used. The (re-)clustering operation tries to acquire PEs from the neighboring clusters (for details see 6.1.3). If the requirements are met after (re-)clustering then the application may be mapped onto that cluster. This step is iterated for a number of times specified by the design-time configuration parameter.

If all the above mentioned options do not lead to a successful mapping (the application and the system constraints are not met), then the mapping request is refused and reported to the requester. The requester waits until some resources are freed to proceed with the mapping request.

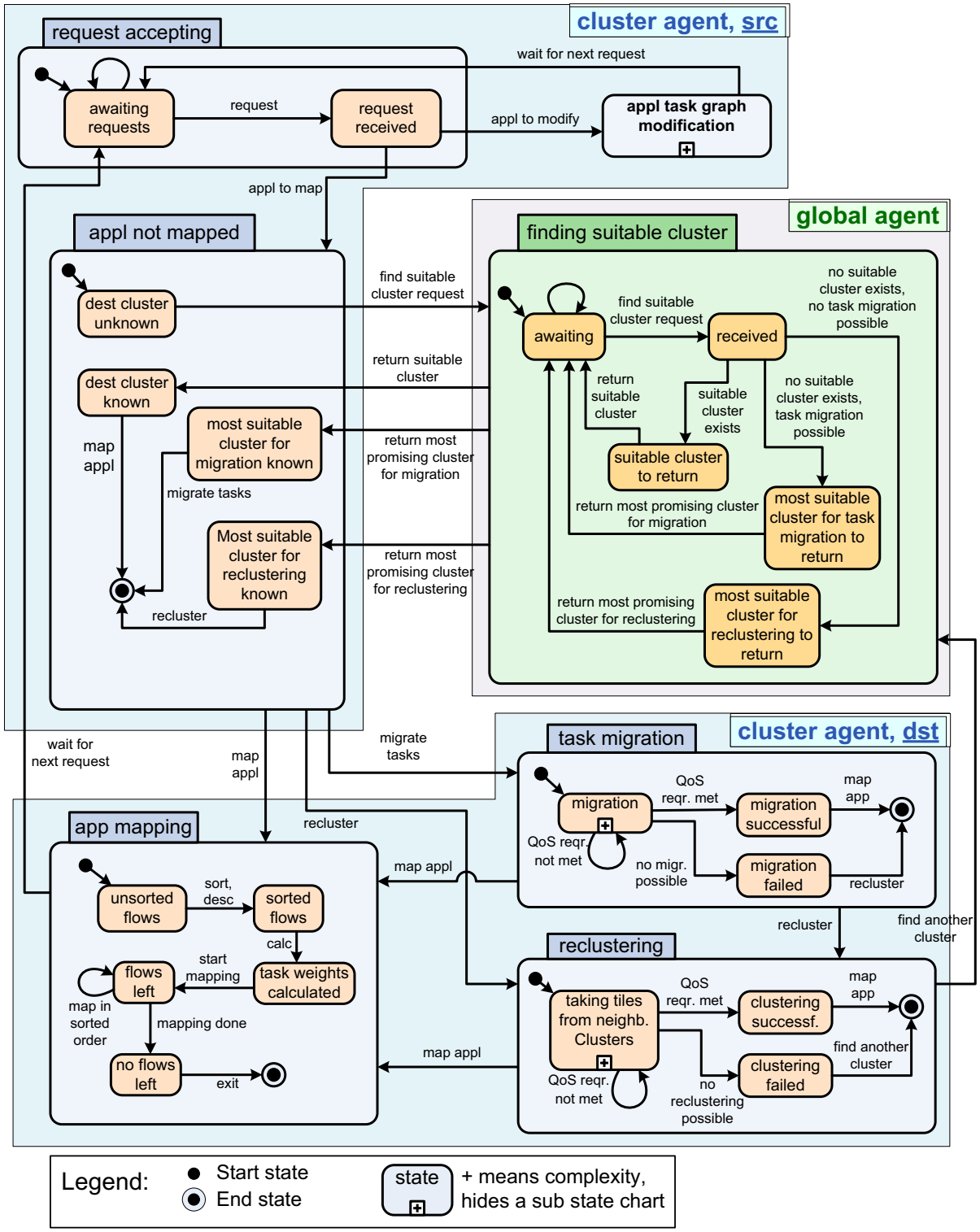


Figure 6.2: Flow of the ADAM algorithm

In the following subsections, the detailed algorithm of runtime the *ADAM* which has the following components are presented: (1) a cluster negotiation algorithm, (2) the (re-)clustering and the task migration methodology, and (3) a mapping algorithm inside a *virtual cluster* (component 1 and 2 are combined to a single component earlier in this chapter).

6.1.2 Cluster Negotiation Algorithm

The runtime suitable cluster negotiation algorithm (see Algorithm 5) is presented here. In this part of the algorithm, a suitable cluster is tried to be found, which is the first part of the *ADAM* algorithm before actual mapping of the application starts.

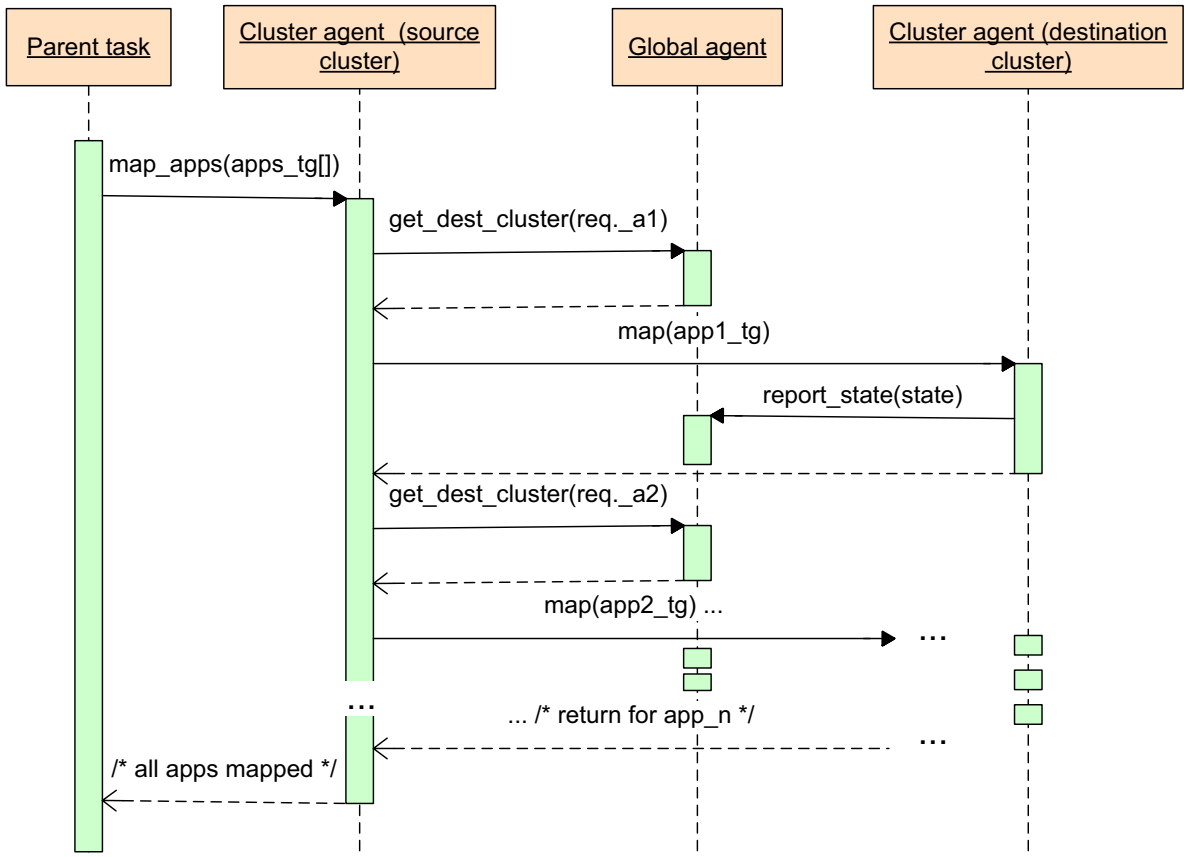


Figure 6.3: Cluster negotiation and application mapping inside the negotiated cluster

A simple sequence diagram showing a straight forward distributed application mapping (without considering the (re-)clustering and the task migration methodologies presented in 6.1.3) and the components which participate when applications get mapped onto the NoC are described in Figure 6.3. *Parent task* is a computational module that is able to start applications, create tasks etc. *Cluster agent (source cluster)* is the management unit that is responsible for the resource allocation of the cluster and contains the *parent task* (details of the *CA* are described in Definition 10 of Chapter 5). The *Cluster agent (source cluster)* receives mapping requests and

in this case it iterates over all applications contained in the set *apps_tg[]* and negotiates with the *GA* which cluster is best suitable to map the tasks of the particular application. The *GA* holds some configuration data about all the clusters, e.g. how many computational resources are available and how much communication capacity is free. When the *GA* has found a cluster, where the application will fit into, it returns its *identification number* to the *Cluster agent (source cluster)*, the requesting cluster. *GAs* are central instances and have overall information about the whole MPSoC (details of the *GA* are described in Definition 11 of Chapter 5). The *Cluster agent (source cluster)*, the requesting cluster sends a mapping request to the *Cluster agent (destination cluster)*, the destination cluster, whose *identification number* is returned by the *GA*. The *Cluster agent (destination cluster)* receives the task graph of the application that has to be mapped as an argument of the mapping request. If no suitable cluster is found then this algorithm tries to form a suitable cluster for the application mapping.

6.1.2.1 Data Structures for the Algorithms

Input and output data structures common for Algorithm 5 and Algorithm 6 are described in the following:

- The application communication task graph, *TG* (see Definition 1 in Chapter 3) with required computational resource profiles for each task and each flow is given by a set of entries: $entry = (id_{src}, id_{dst}, bw_{req}, lat, RR_{tp})$. Here, id_{src} and id_{dst} are the *identification number* of the source and destination tasks of the flow respectively, bw_{req} is the required bandwidth of the flow, lat is the communication latency, and RR_{tp} is the computational resource requirement on each PE type that is needed for a task to ensure a successful execution.

- The state information about all the clusters are stored in a summarized format by the *GAs* and the more detailed information about the individual cluster is stored by the *CA* of that particular cluster. Table 6.1 presents the data structure of the *cluster PE type look-up-table* which is an entry for each *CA* stored within the *GA* data structure. This table represents an individual *CA* holding the state information about the groups of tiles bounded by different PE types. This table is also referred as $nhist_c[]$ (described later), a tile *computational resource usage histogram* in the following algorithms. The memory requirements to store this table depends on several parameters: (1) $\#Tps$: the number of all possible PE types, (2) $\#N_{clu}$: the maximum number of tiles a cluster may contain, and (3) n_{cl_a}: the number of the *computational resource requirement classes* which is described later during introducing the data structure $nhist_c[]$. All tiles of a certain PE type within a cluster are composed to a group and this PE type oriented state information about the MPSoC is used to determine which types of PEs should be taken for the tasks. The entries for the PE types contain current usage of all the tiles that belong to the type and these usages are separated into several *computational resource requirement classes*. The entries of the Table 6.1 are described below:

  PE type identification number: tp_{PE} represent the *identification number* of the PE type. The memory necessary to store this field depends on the number of PE types

field	memory requirement	short description
tp_{PE}	$\log_2 \#Tps$	PE type identification number, $\#Tps$ = #of PE types
q_tiles	$\log_2 \#N_{clu}$	N_{clu} maximum #of tiles in a cluster
r_usg_{tot}	$\log_2 \#N_{clu}$	total computational resource used by the PE type
q_cl_0	$\log_2 \#N_{clu}$	#tiles in resource usage class $(0, \frac{1}{n}]$
q_cl_1	$\log_2 \#N_{clu}$	#tiles in resource usage class $(\frac{1}{n}, \frac{2}{n}]$
...		...
q_cl_n	$\log_2 \#N_{clu}$	#tiles in resource usage class $(\frac{n-1}{n}, 1]$

Table 6.1: Cluster PE type look-up-table: an individual CA

that are available on the MPSoC at design time, e.g. 20 PE types require 8 bits ($2^5 = 32 < 20 > 2^4 = 16$, 5 bits, aligned to Byte size) to store this data structure.

Cluster tile quantity: q_tiles describes the number of tiles inside a cluster of a particular PE type. The memory size required for this field depends on the maximum number of tiles those can be accommodated in a cluster ($\#N_{clu}$) (design-time parameterized).

Computational resource usage: r_usg_{tot} is required by the tasks that are running on all PEs of the particular PE type inside a cluster. It is important to sort the Table 6.1 by resource availability which is described below in Algorithm 6 (see also Figure 6.7 (d)). The memory requirement for this field is determined by the maximum number of the tiles that may be accommodated in a cluster ($\#N_{clu}$).

Computational resource usage histogram: The entries $q\_cl_k \in \{q\_cl_0, \ldots, q\_cl_n\}$ represent the actual *computational resource usage histogram* that contains the number of tiles of the particular PE types that are assigned to a *computational resource usage class* given by k in q_cl_k: $(\frac{k-1}{n_{cl_a}}, \frac{k}{n_{cl_a}}]$, where $k \in \{1, \ldots, n_{cl_a}\}$ and n_{cl_a} is the number of the *computational resource usage classes*. The memory required to store this field is determined by the maximum number of tiles that a cluster can accommodate $\#N_{clu}$.

- **Energy Model:** The selection criteria in the cluster negotiation process is the minimum energy consumption during execution of a certain application. In embedded systems design, energy is considered to be one of the major parameters to be minimized. To make a binding decision, the amount of energy consumption for different PE types at different *computational resource requirement classes* is needed. To explain the energy model, an example from Figure 6.4 (b) is taken, where for the PE of type tp_2, the energy consumption is specified by two values: $tp_2 : (4X, 12X)$ that means that each PE of type tp_2 consumes 4 units of energy (static energy consumption) in a fixed time when it uses no processing resources and 12 units of energy when it consumes the complete PE resources and otherwise,

$$E = u \cdot (E_{[100\%]} - E_{[0\%]}) + E_{[0\%]} \tag{6.1}$$

This simple energy consumption to the computational resource requirement relation in

the energy model is used in current the *ADAM* algorithm. But the *ADAM* algorithm is orthogonal to any energy model. Any other fine-grained energy model may be used to replace this exiting model.

- Data structure $thist[]$ and $nhist_c[]$ store the *computational resource requirement histogram* and the *computational resource usage histogram* respectively. The data structure $nhist_c[]$ is also described by the Table 6.1. As an example, in Figure 6.4 (e), (f) these two data structures can be seen as tables. Each entry $thist[tp, k]$: gives the quantity of tasks of given type being in the *computational resource requirement class* $(\frac{k-1}{n_{cl_a}}, \frac{k}{n_{cl_a}}]$ (n_{cl_a} is the number of classes) and each entry $nhist_c[tp, k]$: gives the actual quantity of tiles of given type being in resource usage class $(\frac{k-1}{n_{cl_a}}, \frac{k}{n_{cl_a}}]$.

$$thist[tp, k] = \#\{t \in T \mid b(t) = tp \wedge u(tp, t) \in (\frac{k}{n_{cl_a}}, \frac{k+1}{n_{cl_a}}]\} \tag{6.2}$$

$$k = \lfloor u(tp, t) \cdot n_{cl_a} \rfloor$$

$$nhist[tp, k] = \#\{n \in N_{clu} \mid tp(n) = tp \wedge u(tp, n) \in (\frac{k}{n_{cl_a}}, \frac{k+1}{n_{cl_a}}]\} \tag{6.3}$$

$$k = \lfloor u(tp, n) \cdot n_{cl_a} \rfloor$$

A simple example to calculate k for the task t_3 in Figure 6.4 (a) bounded to PE of type tp_2 is as follows: $n_{cl_a} = 5, k = \lfloor 22\% \cdot 5 \rfloor = 1 \Rightarrow t_3$ increases the counter of tasks bound to PE of type tp_2 that are in resource requirement class $(\frac{1}{5}, \frac{2}{5}]$.

- The *output* data structure are the cluster where the application is mapped onto and the binding $\forall t_i \in T : b(t_i) \in Tps$ (see Definition 13 in Chapter 5). In Algorithm 5, an array $b[t_i]$ is used for this purpose.

6.1.2.2 Algorithm Description

The pseudocode of the suitable cluster negotiation algorithm is presented in Algorithm 5. The loop spreading from Line 1 to 6 is executed for each task contained in the given communication task graph *TG*. In the loop body, the initial binding is calculated and stored in the list $b[]$ data structure. The criteria for this binding is only the specification of energy consumption of the tasks for each PE type. Therefore, only those PE types are chosen where the energy consumption have the lowest value (see Equation (6.1)). In Line 3, the $u[tp]$ data structure counts for each PE type, the total computational resources required by the tasks that are bounded to this particular PE type. This data structure is used for sorting during calculating the values from the table illustrated in Figure 6.4 (c). This table contains the computational resource requirement profile for the application. Line 4 shows the calculation of the *computational resource requirement class* identifier k. This identifier is used to assign the task to the corresponding *computational resource requirement class* using the requirement of the task that is bounded to a particular PE type. Finally, the result is stored in the *computational resource requirement histogram* $thist[]$.

Algorithm 5 Suitable cluster negotiation

input: TG, $\{nhist_c[] \mid c \text{ is cluster}\}$ Figure 6.4 (a,f)
output: c, $b[]$; (suitable cluster and binding) Figure 6.4 (f,g)
$u(tp,t)$: gives the computational resource requirement when the task t is bounded to tp Figure 6.4 (c)
$u[tp]$: gives the total computational resource requirement for the tasks bounded to given PE type in TG
$E(tp_j,t_i)$: computational energy consumed when task t_i is bound to type tp_j Figure 6.4 (d)
n_{loop}: constant, number of matching loop iterations

1: **for all** $t_i \in TG$ **do**
`// min energy binding Figure 6.4 (d) &` $thist$ `calculate & summarize` $u(tp)$ `in` TG
2: $b[t_i] = \min_{tp_j}\{E(tp_j,t_i) = u(tp_j,t_i)\cdot(E_{[100]}(tp_j) - E_{[0]}(tp_j)) + E_{[0]}(tp_j)\}$
`// initial binding stored in` $b[]$`, minimum energy Figure 6.4 (d)`
3: $u[b[t_i]] = u[b[t_i]] + u(b[t_i],t_i)$
`// consider the columns of computational resource requirement profile Figure 6.4 (c)`
4: k = $\lfloor u(tp_j,t_i)\cdot n_{cl_a}\rfloor$
5: $thist[b[t_i],k] = thist[b[t_i],k] + 1$ Figure 6.4 (e)
6: **end for**
7: sort $thist$ by $u[tp]$ desc
8: $tp_{max} = \max_{tp_j}\{u[tp_j]\}$
9: sort $\{c \subseteq N | c \text{ is a cluster}\}$ by $u_c[tp_{max}]$
10: **for all** $c \in N, c$ is a cluster **do**
11: sort $nhist_c$ by $u[tp]$ asc
12: match $thist[]$ and $nhist_c[]$ (Equation (6.4))
13: store $mismatch[c, i_{loop}] = (tp_j, k_{mis}, q_{tsk,mis})$
14: **if** $matched$ or $i_{loop} = n_{loop}$ **then**
15: leave loop
16: **end if**
17: **end for**
18: **if** $i_{loop} = n_{loop}$ **then**
19: **for all** $c \in N, c$ is a cluster, $(init : i_{loop} = 0)$ **do**
20: $(tp_j, k_{mis}, q_{tsk,mis}) = mismatch[c, i_{loop}]$
21: move $q_{tsk,mis} tasks$ with $\max_t\{u[b[t]]\}$ from tp_j to another PE types with $\min_{tp}\{E(tp, tasks)\}$
22: match $thist$ and $nhist_c$
23: **if not** $matched$ or $i_{loop} = n_{loop}$ **then**
24: restore $b[]$ to min energy binding; leave loop
25: **end if**
26: **end for**
27: **end if**
28: **if not** $matched$: find cluster and tasks to migrate
29: **if not** $matched$: find cluster and tasks to be taken from neighbors or share with neighbors
30: **return** $b[]$, c

In Line 7, the *computational resource requirement histogram* is sorted by the criteria of the computational resource requirement of all tasks bounded to the particular PE types $u[tp]$, calculated in Line 3, in a descending order. In Line 8, the type whose bounded tasks have the maximum resource requirement is selected. It is used below in Line 9 as the sorting criteria of the data structure containing references to all clusters. The more resources of a particular PE type is available on the cluster, the higher is the position of this PE type in the sorted cluster list.

Line 10 begins iteration over all clusters of the MPSoC and ends in Line 17. At first, the *computational resource usage histogram* is sorted in an ascending order inside the loop. The sorting criteria is the resource usage of all tiles of particular PE types. Therefore, the most available PE types are located in the beginning of the *computational resource requirement histogram* data structure.

In Line 12, the matching of *computational resource requirement histogram* $thist[]$ and the *computational resource usage histogram* $nhist_c[]$ is calculated (an example to show the matching is given later). Equation (6.5) is used for this matching. If the cluster is not suitable (no match occurs) then the PE of type tp_j, the resource requirement class identified by k_{mis}, and the quantity of mismatched tasks $q_{tsk,mis}$ are stored in the mismatch data structure in Line 13. If every group of tasks (grouped by PE types) has matched with the available computational resources, or the number of tries to find a suitable cluster given by n_{loop} is reached then the loop is terminated in Line 15.

During mismatch and reaching the maximum number of tries ($i_{loop} = n_{loop}$) another loop begins (Line 18 and Line 19). The loop iterates again over all the clusters available on the MPSoC and this time the binding is modified. At first, inside the loop the mismatch information is made local (Line 20). In Line 21, some tasks which are bounded to the critical PE types where not enough resources are available are rebounded to another PE types (considering the order of increasing energy consumption given by the energy consumption profile). Line 22 executes the matching of the histograms $thist[]$ and $nhist_c[]$ again. The loop may be left under the same criteria as above when the minimum energy bounded tasks are attempted to map (Line 23). While leaving the loop the binding is restored to the minimum energy binding in Line 24 because of the other operations later: the task migration and the (re-)clustering expect the minimum energy binding as the initial binding. In Line 28, the task migration is initiated, if still the matching is not successful. If the task migration is not successful either then the *(re-)clustering* is considered. On success the binding $b[]$ and the suitable cluster are provided as output in Line 30.

6.1.2.3 Histogram Matching during Cluster Negotiation

The *matching* of the two data structures $nhist_c[]$ and $thist[]$ shown in Equation (6.4) is the heart of the Algorithm 5. $nhist_c[]$ is sorted in the ascending order and $thist[]$ is sorted in the descending order. When the result of this term is *true*, then the cluster is suitable for the tasks of the application. During the cluster matching, the concept of task rebinding is also considered.

$$\forall i \in 1, ..., n_{cl_a} - 1 : \sum_{j=n_{cl_a}-i}^{n_{cl_a}-1} thist[tp, j] \leq \sum_{j=1}^{i} nhist_c[tp, j] \tag{6.4}$$

If there is no available cluster which may be selected as a suitable cluster for a given energy budget then possible clusters are searched where the task migration may be performed to retrieve suitable cluster for the application. The system has to decide which tasks may be migrated for each application at a given time. In this work, the task migration steps are not addressed in details and uses state-of-the-art methodologies [17, 31]. A short summary of the used task migration methodology is explained in 6.1.3.

If the migration of the tasks does not deliver the expected results to make a cluster suitable, then the last possibility to achieve this objective is the (re-)clustering operation. Therefore, all the possible candidate neighboring clusters are examined, if it does have the desired tiles to make the cluster suitable for the application mapping. If the condition is fulfilled then the tiles are examined if they are available to be given away to another cluster, or if it is possible to perform a task migration operation within the neighboring cluster to make the desired tiles available for the (re-)clustering.

The complexity of the cluster negotiation algorithm is $O(m + r \cdot \log r)$ where m is the number of tasks and r is the number of *virtual clusters*. Due to the low complexity, this part of the approach is suitable to be applied at runtime.

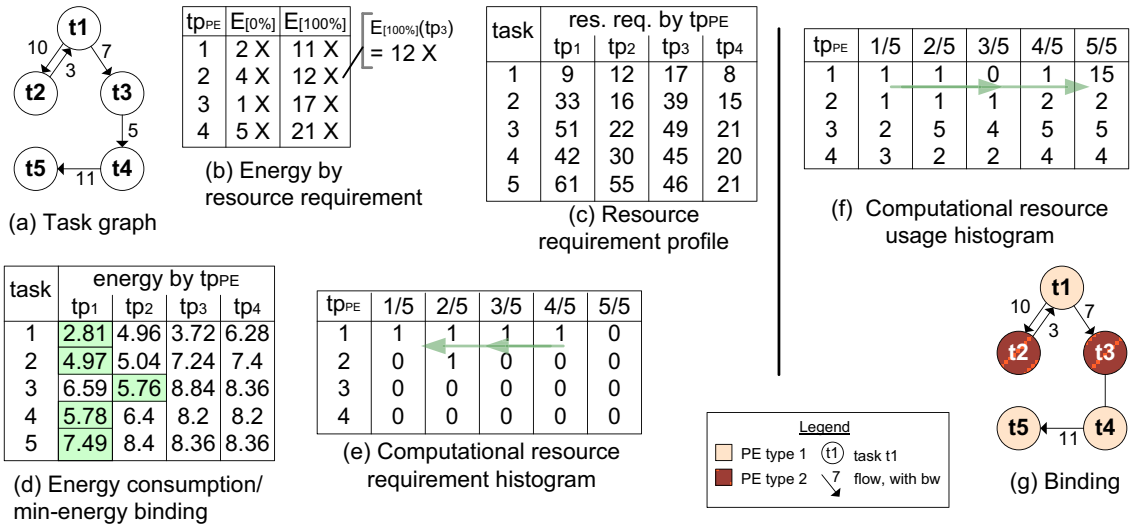


Figure 6.4: Suitable cluster and binding example

6.1.2.4 Exemplary Algorithm Execution

In Figure 6.4, an example for the cluster negotiation procedure is presented. The task graph of the application that is requested to be mapped is shown in Figure 6.4 (a). Each node of this graph corresponds to a task $t_i \in T$ and each edge corresponds to a flow $f_{i,j} \in F$ between two tasks t_i and t_j. The weights assigned to the edges corresponds to the bandwidth required by the inter task communication between the tasks that are connected by the edges.

The energy consumed by different PE types are presented in Figure 6.4 (b). In the table in Figure 6.4 (b), $E_{[0\%]}$ provides the energy consumption when the computational resource usage required by the tasks running on the PEs are $0\,\%$ and $E_{[100\%]}$ provides the energy consumption when the tasks running on the PEs use $100\,\%$ of the computational resources. Figure 6.4 (c) shows the computational resource requirements of all tasks $t_i \in T$, contained in the task graph for each PE type. As an example, task t_3 requires $51\,\%$ of computational resources, if it is mapped onto a PE of type tp_1, $22\,\%$ of the computational resource is required on a PE of type tp_2, and so on. Equation (6.1), Figure 6.4 (b), and Figure 6.4 (c) are used to calculate the actual required energy consumption for every tasks and for every types of PEs that the tasks can be bounded to at runtime. The result of this calculation is presented in Figure 6.4 (d). As an example, taking the values of the computational resource requirement of the task t_3 while bounded to the PE of type tp_1 ($u = 51\,\%$) from Figure 6.4 (c) and the values $E_{[0\,\%]}(tp_1) = 2$, $E_{[100\,\%]}(tp_1) = 11$ from Figure 6.4 (b) we can calculate the corresponding entry for Figure 6.4 (d).

$$E = u \cdot (E_{[100\%]} - E_{[0\%]}) + E_{[0\%]} = 51\,\% \cdot (11 - 2) + 2 = 6.59$$

Using the energy consumption values from Figure 6.4 (d) the minimum energy binding for the tasks of the application can be derived by simply selecting the lowest value of energy consumption for each task. These values are the highlighted entries of Figure 6.4 (d). The task t_3 for example is bounded to the PE of type tp_2 because t_3 consumes least energy when it is bounded to tp_2. This binding is used to classify the tasks of the application into *computational resource requirement histogram* ($thist[]$) in Figure 6.4 (e). The entry in the second row (tp_2) and the second column (class: $(\frac{1}{5}, \frac{2}{5}]$) contain the value 1, that means that there is exactly 1 task that is bounded to the PE of type tp_1 and classified to the second class ($k = 2$). Task t_3 requires $22\,\% \in (\frac{1}{5}, \frac{2}{5}]$ of the computational resources of a PE of type tp_2.

Figure 6.4 (e) presents the *computational resource usage histograms* ($nhist_c[]$) for a cluster. As an example, table entry (tp_3, cl_2) contains the value 5 which means that the cluster contains 5 tiles with a PE of type tp_3, where the currently running tasks use between $20\,\%$ and $40\,\%$ of the computational resources. In Algorithm 5, the data structures $thist[]$ (Figure 6.4 (e)) and $nhist_c[]$ (Figure 6.4 (f)) are compared. If these data structures match then the cluster is selected as a suitable cluster (6.4). The values of the row iterated in Figure 6.4 (e) are considered in the opposite direction compared to Figure 6.4 (f), that means the tiles with smallest resource usage are used first and continue from *left to right*, beginning with the class $(\frac{1}{5}, \frac{2}{5}]$.

At first, the entry from Figure 6.4 (e) with the largest computational resource requirements of the PE of type tp_1 is considered. $thist[tp_1, cl_4] = 1$ that means 1 task is bounded to tp_1 and requires an amount $\in (60\,\%, 80\,\%]$ of the computational resources. Now this value is compared to the entry of Figure 6.4 (f) representing tiles which can provide this computational resources. Here, $nhist_c[tp_1, cl_1] = 1$ that means tiles whose resources are used at maximum $10\,\%$. We see that the required resources of this step is available on this cluster: $thist[tp_1, cl_4] \leq nhist_c[tp_1, cl_1]$ therefore, we go to the next *computational resource requirement class* cl_3. According to (6.4) the following is checked: $thist[tp_1, cl_3] + thist[tp_1, cl_4] \leq nhist_c[tp_1, cl_2] + nhist_c[tp_1, cl_1]$ $= 2 \leq 2$. Each resource requirement/usage class for each PE type until the first mismatch occurs or all entries match is checked similarly. In this example in the third matching step a

mismatch occurs: $1 + 1 + 1 \leq 1 + 1 + 0$ which returns *false*. To handle this a rebinding of the corresponding tasks is done. Therefore, task t_3 which requires $33\,\%$ of the computational resources on PE of type tp_1 (causing there $4.9X$ energy consumption) is rebounded to tp_2 ($5.0X$ energy consumption). Type tp_2 is selected because the energy consumption is well below the given energy budget. Finally, Figure 6.4 (g) presents the returned binding and the selection of the cluster whose *computational resource usage histogram* is in Figure 6.4 (f).

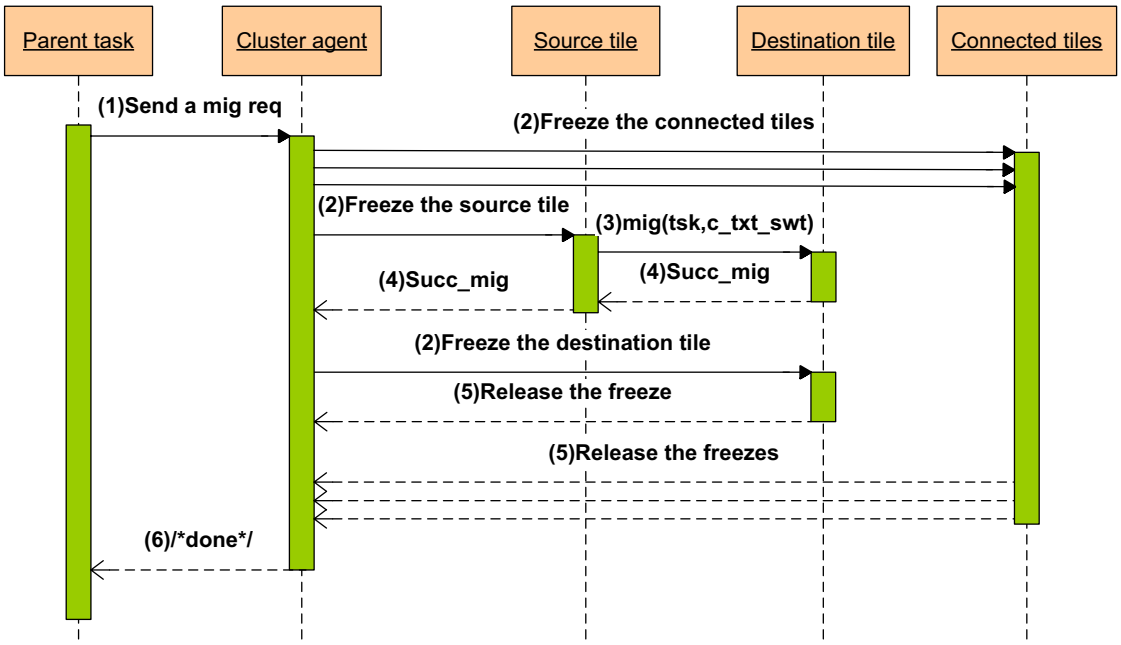


Figure 6.5: Task migration to support runtime application mapping

6.1.3 (Re-)clustering and Task Migration for the ADAM Algorithm

In case a suitable cluster cannot be found in Algorithm 5, it starts looking for the clusters which may support the task migration. The task migration as an integral part of the runtime application mapping algorithm is demonstrated roughly in Figure 6.5. Details of the task migration methodology are out of the scope of this thesis. The algorithm uses the approach presented in [119]. The *parent task* sends a task migration request to the *CA* and upon receiving the request it freezes the source tile, connected tiles to the source, and the destination tile for successful and transparent task migration. The task migration is performed with all local data within the executing task, the state of the task, and even the modified binary of the task (the binary of the application may need to be changed to make it compaitable for different instruction set processors). The feedback after the task migration is provided to the *CA*.

When the migration of the tasks does not deliver a suitable cluster, then the (re-)clustering operation shown in Figure 6.6 is invoked. First the neighboring clusters are negotiated if there are some unoccupied PEs that they might be given away to the requesting cluster. If no unoccu-

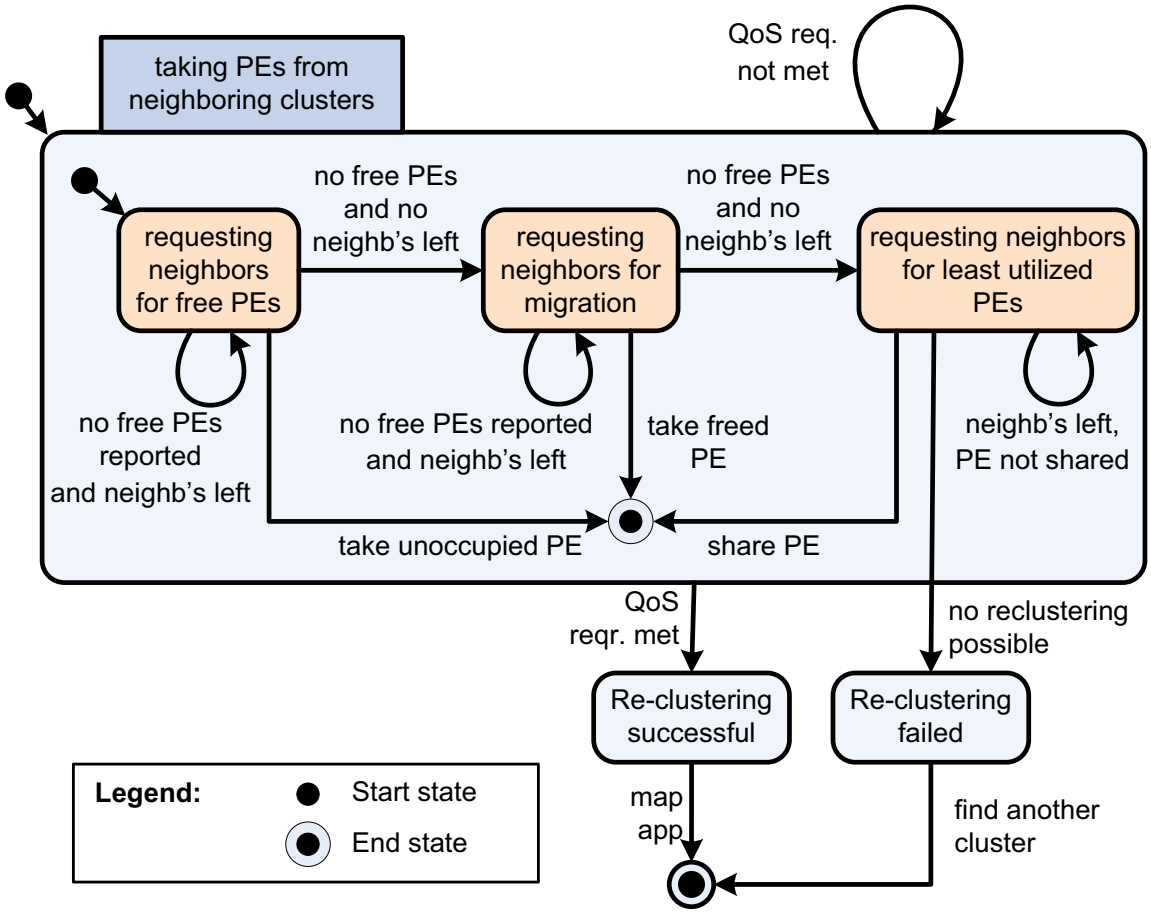


Figure 6.6: The (re-)clustering algorithm flow

pied PEs are available, the neighbors are requested to migrate tasks between PEs of that cluster without losing its performance and runtime constraints. If that is not successful either, then the neighboring cluster is requested for the least utilized PEs that may be shared with the requesting cluster. The task migration and the (re-)clustering methodologies have not been covered in the scope of this thesis in detail. The concept of the task migration and the (re-)clustering have only been used in the scope of this thesis and have been kept as a future work.

6.1.4 The Mapping Algorithm Inside a Cluster

The runtime application mapping algorithm inside a cluster managed by the *CA* is motivated by the static mapping algorithm presented in [70] as it is very light-weight in terms of execution cycles and provides near-optimal mapping solution.

6.1.4.1 Data Structures for the Algorithm

An offline mapping algorithm may be adapted without any changes to create an initial mapping, when applications are loaded onto NoC, comparable to the approaches from [70]. But afterward it has to provide additional functions to handle events that will occur at runtime. These functions will modify the task graph that is held in memory of the mapping module. This is solved by adding extra information contained within the TG ($entry = (id_{src}, id_{dst}, bw_{req}, lat, RR_{tp})$) explained in 6.1.2.1 to the resulting mapping data structure given below (this entry is a single element in the data structure $mpng$):

$$mpng = (id_{src}, id_{tile}, id_{dst}, id_{app}, RR_{tp}, vol, lat)$$

In the following, the data structures which are used by this part of the mapping algorithm are studied. The major data structures that are stored in the memory of the *CA* are shown in Table 6.1 for each cluster (explained in detail in 6.1.2.1), Table 6.2, and Table 6.3. The first two tables correspond to the variable $model$ of the Algorithm 6.

Table 6.1 is stored for each cluster by the *CAs*. It is a local copy of the entry in the globally stored table by the *GAs*. The detailed description of the table is given in 6.1.2.1. Table 6.2 describes an entry of the *cluser tile look-up-table* that is located in each cluster to define one of the data structures for the *CA*. It describes the usage of the tiles and of the links between the tiles or more precisely resource utilization of the output ports. The memory usage of this table relies on the following parameters: (1) $\#N$: the number of tiles of the complete *MPSoC*, (2) $\#Tps$: the number of different PE types, (3) $\#Lv$: the number of different *computational resource usage classes* of a tile or the number of the communication bandwidth usage levels of a link. (4) q_vc_{max} is a design parameter, that specifies the maximal number of the virtual channels.

field	memory requirement	short description
id	$\log_2 \#N$	tile identification number
tp_{PE}	$\log_2 \#Tps$	type of tile's PE
r_usg_{comp}	$\log_2 \#Lv$	resource usage
bw_{used}		communication bandwidth usage
l_N	$\log_2 \#Lv$	for the north link (output port)
l_E	$\log_2 \#Lv$	for the east link
l_S	$\log_2 \#Lv$	for the south link
l_W	$\log_2 \#Lv$	for the west link
q_vc		virtual channel quantity
l_N	$\log_2 q\_vc_{max}$	for the north link (output port)
l_E	$\log_2 q\_vc_{max}$	for the east link
l_S	$\log_2 q\_vc_{max}$	for the south link
l_W	$\log_2 q\_vc_{max}$	for the west link

Table 6.2: Entry of the cluster tile look-up-table

The entries of Table 6.2 are explained below:

Tile identification number The field id is used to identify the tiles in the table. It is needed because the *virtual clusters* are not fixed. A tile can be added or removed from the cluster. The size of this field is directly linked to the number of tiles, because each tile in the NoC has an unique *id*: $\log_2 \#N$. As an example when the MPSoC would have 4096 tiles, 12 bits are needed to store the *id* field and due to using Byte sized data structures it is 16 bits or 2 Bytes.

PE type The entry tp_{PE} of this table stores the type of the PEs that is located on the cluster. This information is needed during the execution of the mapping procedure because after binding each task to a PE type, the algorithm must know where tiles of this PE types are located. The size of this field depends on the number of the PE types that are available on the MPSoC, e.g. 20 PE types would require 5 bits ($2^5 = 32$) aligned to Byte size, exactly one Byte is required for this data.

Computational resources The entry r_usg_{comp} represents the computational resource usage of the tasks that are currently running on a particular PE in a *virtual cluster*. This field is required by the mapping algorithm to check if there are available resources for the new tasks to be mapped onto the PEs. This entry is used inside the cost function calculation ($RU(n_j)$ represents the computational resource usage on tile n_j) in Equation (6.5). The memory used by this entry depends on the granularity of the resource usage specifications which are specified by the parameter $\#Lv$.

Bandwidth The entries $bw_{used,l_{dir}}$ represent the communication bandwidth usage of each tile for each output port, where $dir \in \{N, E, S, W\}$ is the direction (North, East, South, and West) of the output ports. This information is used in Algorithm 6 to decide if there are enough available communication bandwidth for the task to be mapped onto a candidate tile. These entries are using the same resource usage specification granularity given by $\#Lv$.

Virtual channels The entries $q\_vc_{l_{dir}}$ correspond to the number of virtual channels used in the output ports ($dir \in \{N, E, S, W\}$). The memory required by this entry depends on the maximum number of virtual channels q_vc_{max} that is specified at design time.

field	memory requirement	short description
id_{src}	$\log_2 \#T_{tot}$	source task identification number
id_{tile}	$\log_2 \#N$	assigned tile identification number
id_{dst}	$\log_2 \#T_{tot}$	destination task identification number
id_{app}	$\log_2 \#apps$	application identification number
r_usg_{src}	$\log_2 \#Lv$	computational resource usage in steps of $\frac{1}{256}$
bw_{req}	$\log_2 vol_{max}$	$\log_2 \#$ volume
lat	$\log_2 lat_{max}$	latency

Table 6.3: Entry of the task-to-tile mapping look-up-table

Table 6.3 (*the task-to-tile mapping look-up-table*) corresponds to the variable $mpng$ (Algorithm 6). Here, the information about the tasks that are mapped onto the cluster and the corresponding communication flows between the tasks are stored. This data structure is used to

look up which tile a task is assigned to. The application communication task graph ($entry = (id_{src}, id_{dst}, bw_{req}, lat, RR_{tp})$) is transformed to this data structure when a mapping of an application is successful. The memory usage of this data structure depends on the following parameters: (1) $\#T_{tot}$: maximum number of tasks those can be supported by the MPSoC, (2) $\#N$: number of tiles of the MPSoC, (3) $\#apps$: maximum number of supportable applications, (4) $\#Lv$: number of *computational resource usage classes*, (5) vol_{max}: maximum communication volume of a flow between two tasks, (6) lat_{max}: maximum latency of a flow, and (7) $n_{flow,max}$: the maximum number of flows that are supportable by the MPSoC. The detailed description of the entries of Table 6.3 are given below:

Source task identification number id_{src} is used as the input parameter to find the mapped task. The required memory size for this entry is dependent on the maximum number of tasks that may be supported by the MPSoC $\#T_{tot}$. If the MPSoC supports 20,000 ($\in \{2^{14}, \ldots, 2^{15}\}$) tasks, this entry will require 15 bits or Byte-aligned 16 bits = 2 Bytes of memory.

Assigned tile identification number id_{tile} describes to which tile the task is mapped onto. It needs an amount of memory that is dependent on the total number of tiles provided by the MPSoC $\#N$.

Destination task identification number id_{dst} stores the *identification number* of the destination task to which the source task is connected to. For each edge of the TG an entry is needed to ensure that the graph information is not lost. The memory size requirement is the same as the id_{src}: $log_2 \#T_{tot}$.

Application identification number To keep the communication task graph information after mapping, also the application *identification number* id_{app} for each task is required. Without this information no mapping modification is possible. Memory size relies on the maximum number of applications those can be run on the MPSoC at once: $\#app$.

Computational resources r_usg_{src} is an entry that comes from the communication task graph (r_usg_{src} is similar to RR_{tp} and describe the computational resource requirement of the task on each PE type) and needs to be stored to provide the possibility to modify the mapping at runtime. Memory size relies on the resource usage granularity $\#Lv$ (see Table 6.2).

Communication bandwidth bw_{req} is the required communication bandwidth, that is needed by the communication flow between the source and the destination task. The memory size depends on the maximum communication volume vol_{max} that can be transmitted by a particular flow.

Latency lat is the required communication latency for a particular communication flow between the source and the destination tasks.

6.1.4.2 Heuristics for the Optimization

To decide to which tile of the PE type a task should be mapped onto, a heuristic is used and it is described by the following cost function $c(t, n)$ for the selection of a tile n_j for a given task t_i.

$$c(t_i, n_j) = \alpha \cdot (\mathrm{D}(n_j) + bw_{req,tot}(n_j) + RU(n_j)) + \beta \cdot \sum_{k \in T_{conn,mp}} d(k) \cdot \mathrm{vol}(k) \tag{6.5}$$

$$\text{where, } \mathrm{D}(n_j) = \frac{1}{\#tiles_{clu}} \sum_{l \in N} d(n, l)$$

The parameters that are used to describe the Equation (6.5) are explained below:

- $\mathrm{D}(n_j)$: average distance of a tile to all other tiles within a cluster.
- $d(n, l)$: Manhattan distance between the tiles n and l.
- $T_{conn,mp}$: set of all t_i, connected and mapped tasks.
- d_k: Manhattan distance between the mapped tasks.
- vol_k: the communication flow between the connected tasks.
- $bw_{req,tot}$: total bandwidth required by the tasks running on the tile.
- RU_{n_j}: the computational resources that are used by the running tasks on a PE.

The cost function considers the current state of the tiles where the tasks may be mapped onto. Equation (6.5) considers the additional resource requirements by the tasks when those are mapped onto the tiles. The distance to the already mapped tasks are considered as well, and the tasks those are connected to the already mapped tasks will be mapped now. The cost function has two components that are parametrized by the scalar coefficients α and β. The first part of the cost function is parametrized by α which considers the current state of the NoC but not the tasks from the TG.

$D(n)$ is used to determine the optimality of the position of the tile n, a lower value is achieved when the tile has a lower distance to every other tiles in the cluster. Lower distance means lower energy required to transfer data through this connection.

$bw_{req,tot}$ is used in the cost function because if only D is considered then the algorithm will try to map every tasks to the center of the cluster as they have a better D value. Therefore, $bw_{req,tot}$ produces a tradeoff while there are too many tasks with high communication bandwidth requirements to be mapped onto a tile.

RU_{n_j} guides the search for the tiles whose computational resources are more available.

The second part of the cost function is parameterized by β and uses the information from the TG to get good locations for the tasks. The sum of the products of $d(k) \cdot vol(k)$ accumulated over all tasks of the TG those are connected to the current task and the tasks those are already mapped on the MPSoC. Therefore, the tasks of the TG of an application are tried to keep together.

Algorithm 6 Runtime application mapping inside a cluster

TG: input data, application communication task graph
$mpng$: output data, mapping of tasks to tiles
$model$: state of the physical network
$Tps \in model$: types contained in model
tp_{PE}: type of a tile's PE, $tp_{PE} \in Tps_{model}$
$rs\_avail(tp_{PE})$: gives the available computational resources of all PEs of the given type tp_{PE}
binding: $\forall t_i \in TG : \exists b(t_i),\ b:$ see Definition 13 in Chapter 5
sorted: Tps_{model}, **asc**, **by** rc_avail(tp_{PE})
`// sorting by availability of PE types in he MPSoC`

1: **for all** $a \in Tps_{model}$ **do**
2: $f^a = \{f_{ij} \in F \mid \text{bound}(t_i, a) \vee \text{bound}(t_j, a)\}$
3: sort(f^a, **desc**, **by** bw_req($f_{ij} \in f^a$))
4: **for all** $f_{ij}^k \in f^a$ **do**
5: select tiles $n_{i'}, n_{j'} \in model$, **for** t_i, t_j **of** f_{ij}^k, **by** $\min(c_{mp})$
6: insert($n_{i'}, n_{j'}$ **to** $mpng$)
7: **end for**
8: **end for**
9: allocate($mpng$); update($model$ **by** $mpng$)

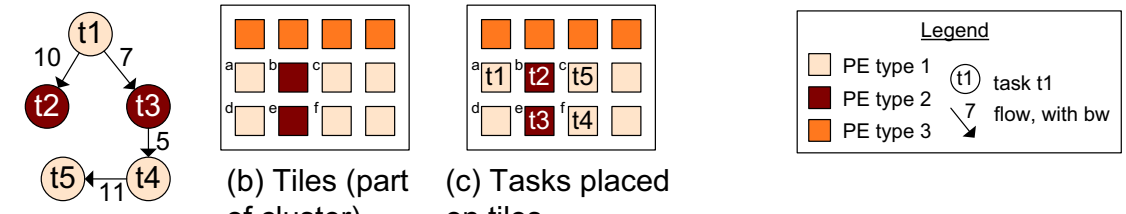
For simplicity, the computational resource availability is not considered in the order of the tasks while mapping. It is assumed that the required bandwidth of the inter task communication will correlate with the computational resource requirements (this assumption can be redefined in the future work). Therefore, the bandwidth is used to sort the data structure containing the flows whose connected tasks are mapped. The order of these flows within the data structure guides the mapping order of the tasks. In an unfavorable case, the less computational resource requiring tasks are mapped at first and block the computation-heavier tasks to be mapped on a better location. Observation shows, the computational resource requirement of a task is a less important criteria for the assignment of the tasks. Only if the tiles of the PE types where the tasks have to be bounded to are too critical then it has to be taken into account. This problem is not solved in this thesis adequately and considered as the future work. One possibility to handle this problem may be to consider the sorting criteria of the data structure f^a and an additional one that relies directly on the computational resource requirements of the tasks.

6.1.4.3 Algorithm Description

The pseudocode of the runtime application mapping algorithm inside a cluster is given in Algorithm 6. The *input data structure* is the communication task graph TG of the incoming application and the data structure $model$ of the MPSoC which stores the current state of the used computation and communication resources. The TG contains for each task its energy consumption for computation, which is fixed for a particular type of PE. The task to the PE binding is determined in the previous negotiation step between the *CA* and the *GA*. The TG also contains

the communication costs for each flow f_{ij} between the tasks t_i and t_j. The $model$ contains for each tile its current computational resource usage by the running tasks, the type of processing element of this tile tp_{PE}, and for each link the current bandwidth usage parameterized by the service classes. The *output* data structure is the mapping of tasks to the corresponding tiles of the $mpng$ and it is used to allocate the tiles physically on the MPSoC and to update the $model$ by the added application.

In the following Algorithm 6 is explained in detail. The loop in Line 1 begins to iterate over all PEs of types $a \in Tps_{model}$, the listing shows that the set of all PEs of types Tps_{model} must be sorted in an ascending order by the availability of the computational resources of those PE types. In Line 2, the set of flows f^a is created where each flow $f_{ij} \in F$ connects to a task that is bounded to the current PE of type a (Line 1). The binding is already done in Algorithm 5 during cluster negotiation. This flow set f^a is sorted in a descending order in Line 3 by the bandwidth requirements *bw_req*(f_{ij}) where, $f_{ij} \in f^a$. In Line 4, another loop starts iterating over the elements of the current set of flows $f_{ij} \in f^a$. Inside this loop body tiles $n_{i'}, n_{j'}$ are selected from the *cluster tile look-up-table* (Table 6.2) where the tasks connected by the current flow f_{ij} may be mapped onto (Line 5). To guide the quality of the mapping the cost function described in Equation (6.5) is used. The selected tiles are stored within the output data structure $mpng$ (Table 6.3). Finally in Line 9, the assignment of the mapping to the physical network takes place and the data structures holding the information about the MPSoC are updated.



(a) Task graph

(b) Tiles (part of cluster)

(c) Tasks placed on tiles

tp_i	avail_rs(tp_i)	ord(tp_i)
tp_1	1730 %	5
tp_2	210 %	1
tp_3	370 %	2
tp_4	530 %	4
tp_5	505 %	3

(d) available computational resources

flows	bw
1-2	10
1-3	7
3-4	5
4-5	11

(e) Flows by PE types

tasks	cmp_req
t1	30 %
t2	25 %
t3	29 %
t4	40 %
t5	38 %

(f) Required computational costs

PE	res. in use
a	10 %
b	40 %
c	20 %
d	10 %
e	33 %
f	25 %

(g) Current comp. resources in use by tasks on tiles

Figure 6.7: Example for a runtime application mapping inside a cluster

6.1.4.4 Exemplary Algorithm Execution

Algorithm 6 is demonstrated using an example in Figure 6.7. After the cluster negotiation part presented in 6.1.2, the *CA* is requested to map an application to the responsible cluster. At first, all types of PEs are iterated and the current type is referred to as a in the algorithm. For each type the set of flows f^a are considered. In Figure 6.7 (a), the communication task graph TG, whose tasks are bounded by respective PE types is shown. There are two groups of tasks: (1) bounded to *PE of type* tp_1: $\{t_1, t_4, t_5\}$ and (2) bounded to *PE of type* tp_2: $\{t_2, t_3\}$. These groups are sorted by the availability of the resources provided by the PEs of these types. The value assigned to the edges specify the communication volume between the connected tasks.

In Figure 6.7 (b), a part of the tiles of the selected cluster where the tasks will be mapped onto is presented. Figure 6.7 (g) shows the current computational resource usage by the tasks running on some of these tiles from Figure 6.7 (b). Figure 6.7 (f) presents the computational resource requirement for each task of the communication task graph. The availability of the computational resources (*avail_rs(*tp_i*)*) is presented in the second column of the Figure 6.7 (d). To determine these values the individual value of each PE of this type have been added up, e.g. all tiles of type 1 provide 1730% of the computational resources. The lines in this table are sorted and the sorting order depends on the available resources.

In Figure 6.7 (e), the first set of flows f^{tp_2} those are related to *PE type of 2* ($\{f_{12}, f_{13}, f_{34}\}$) and the corresponding bandwidth requirements of these flows are highlighted. Figure 6.7 (e) shows that, the flows are sorted by the required bandwidth within the current set (f^{tp_2}). In the following the mapping of the tasks considering the flows are shown (the result of the mapping is shown in Figure 6.7 (c)):

- The algorithm starts with the flow f_{12} and looks where the tasks t_1 and t_2 may be mapped onto. A good location for the assignment of t_1 is the tile [a] (see Figure 6.7 (f),(g)), for the next task t_2 the tile [b] is selected and so on (considers also the PE types and the binding of the tasks). This selection is added to the data structure $mpng$ that collects all these selections of tiles for the tasks.
- For the application mapping, the next flow f_{13} finds that task t_1 is already mapped therefore, the algorithm continues with the next task t_3 and assigns it to the tile [e].
- Similarly, for the flow f_{34}, tile [f] is selected for the task t_4.

Now the algorithm continues with the next set of flows f^{tp_1} which corresponds to the PE of type tp_1. For the flow f_{45}, tile [c] is selected for the task t_5. After all these mapping steps, the tasks have to be assigned physically as it is specified in the data structure $mpng$ and finally, the state of the network $model$ is updated.

6.1.5 Configuration Data for the Runtime ADAM Algorithm

The required configuration data those need to be stored in the *local memory* associated with the *CAs* and *GAs* for the runtime *ADAM* algorithm are discussed here. In the previous subsections,

the data structures that have to be transmitted between clusters, *CAs*, *GAs*, etc. have already been discussed. The data structures describing the state of the network are considered here again to answer the following questions:

- Which data structures have to be stored in the MPSoC to provide the runtime application mapping functionality?
- Where these data structures have to be stored?
- Which data structures have to be communicated between tiles?
- Which data structures are needed in the algorithms during application mapping?

In the following, these listed issues are discussed.

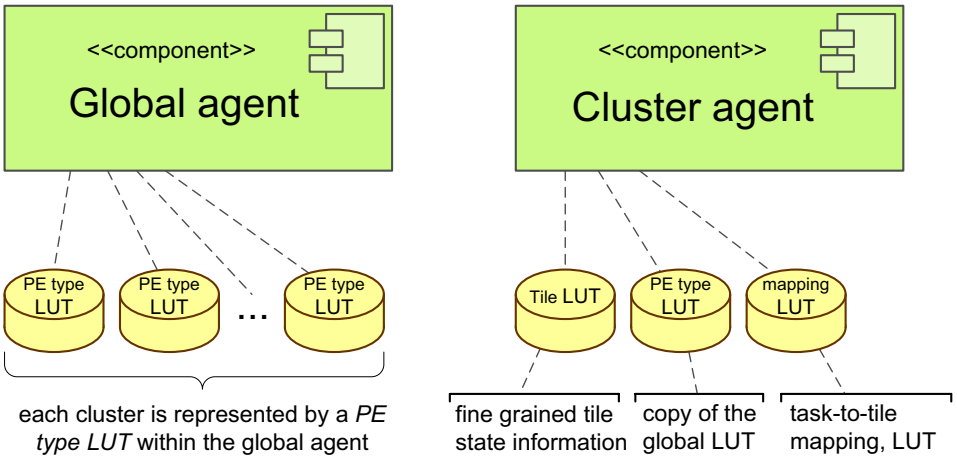


Figure 6.8: Configuration data for the runtime ADAM algorithm

6.1.6 Persistent Configuration Data

To provide the runtime application mapping functionality by the *ADAM* algorithm, some data structures about the current MPSoC state have to be kept persistent during the execution. The term *persistence* is used to emphasis the meaning of some information those have to be kept between different mapping requests. During runtime application mapping, some other temporary processing data are also generated. There are several *data structures* containing configuration data those needs to be stored to provide the mapping functionality. First of all, there must be some *global data structures* those describe the state of every cluster of the MPSoC. This data is in summarized format and does not contain fine grained details about a particular tile inside the clusters. This data structure is stored in the *PE type look-up-table* in Table 6.1 for each cluster. These cluster *PE type look-up-tables* for each cluster are stored within the *GAs*. The global part of the mapping algorithm (Algorithm 5) only uses this data structure to decide to which *CA* the application has to be assigned. *CAs* need detailed information about the resources provided by the cluster. In 6.1.4.1, the data structures those are needed by the mapping algorithm to perform

the mapping on the cluster level have already been explained . These data structures are: the *cluster tile look-up-table* in Table 6.2, the *cluster PE type look-up-table* in Table 6.1 which is a copy of the corresponding look-up-table stored within the *GA*, and the *task-to-tile mapping look-up-table* ($mpng$) presented in Table 6.3.

All the above mentioned configuration data structures have to be stored in the MPSoC. The evaluation depends on various architectural parameters of the MPSoC, e.g. the number of PE types and the number of tasks those can be mapped onto the MPSoC. Some assumptions are considered to present the following example to calculate the amount of memory required to store configuration data by the *GAs* and *CAs*. It is assumed that the NoC dimension is 64x64, therefore, $\#N = 4096$, the tiles of the MPSoC are organized in $qnt_{clu} = 32$ clusters that means the average cluster size is $sz_{clu} = 128$ tiles, the lower and upper bound of the cluster size is set to $sz_{clu,min} = 16$ and $sz_{clu,max} = 2^{16}$ tiles respectively, 16 ($= n_{Tps}$) different types of PEs are used, and $n_{cl_a} = 16$ classes in the *computational resource usage histogram* to classify the resource usage by the tasks running on the tiles are considered. The data structures within the *CAs* need some additional parameters: the resource usages are specified by a scalar data structure whose granularity is specified by n_{lv} and therefore, this parameter is set to $n_{lv} = 2^{16}$. The maximum number of tasks is $qnt_{tsk,max} = 2^{16}$ and the MPSoC is able to run 2^{16} applications in parallel.

Calculation of the Memory Requirements for a GA In the *GA*, there is a *cluster PE type look-up-table* for each cluster. The maximum number of clusters is given by $\#N/sz_{clu,min} = 4096/16 = 256$. Therefore, there must be enough memory to store 256 *cluster PE type look-up-tables*. A single *cluster PE type look-up-table* described in Table 6.1 has $n_{Tps} = 16$ entries and each entry needs the following amount of memory. The PE type id requires 1 Byte, the field q_{tiles} requires $\log_2 sz_{clu,max} = 16$ bit $= 2$ Byte, the field r_usg_{tot} needs 2 Bytes of memory, and the actual *computational resource usage histogram* that is stored by the fields $q\_cl_0, \ldots, q\_cl_n$ require $n_{cl_a} \cdot 2\,\text{Byte} = 16 \cdot 2\,\text{Byte}$ of memory. Therefore, the *GA* must provide memory space of

$$\frac{\#N}{sz_{clu,min}} \cdot n_{Tps} \cdot (len(q_{tiles}) + len(r\_usg_{tot}) + n_{cl_a} \cdot q\_cl_x) =$$

$$\frac{4096}{16} \cdot 16 \cdot (2\,\text{B} + 2\,\text{B} + 16 \cdot 2\,\text{B}) = 147.5\,\text{KB}$$

Calculation of the Memory Requirements for a CA Three data structures are stored by the *CAs*. The first data structure is the *cluster tile look-up-table* (Table 6.2). The maximum cluster size is $sz_{clu,max} = 2048$ tiles therefore, this is also the maximum number of entries in the *cluster tile look-up-table*. The field used for the tile id needs 2 Bytes $= \log_2 \#N$ of memory, the *PE type* field requires one Byte of memory, the field r_usg_{comp} needs 2 Bytes, each of the bandwidth fields needs 2 Bytes, and the virtual channel fields need one Byte each. Therefore, we can calculate the memory requirement of Table 6.2 as follows:

$$sz_{clu,max} \cdot (len(id) + len(tp_{PE}) + len(r\_usg_{comp}) + 4 \cdot len(bw_{used,x}) + 4 \cdot len(q\_vc)) =$$

$$2048 \cdot (2\,\text{B} + 1\,\text{B} + 2\,\text{B} + 4 \cdot 2\,\text{B} + 4 \cdot 1\,\text{B}) = 34.8\,\text{KB}$$

The second data structure is the *cluster PE type look-up-table* which is a copy of the globally stored data structure presented in (Table 6.1). Therefore, the memory required by one *cluster PE type look-up-table* is 576 Bytes.

The third data structure is the *task-to-tile mapping look-up-table*. The memory required by this data structure is calculated in the following. It is assumed that there are maximum number of entries which are stored in this table and in the worst case a cluster can contain the maximum number of tasks which may be mapped onto the tiles of the cluster. It is further assumed that, there is no cluster that is larger than the half of the MPSoC ($sz_{clu,max} = 2048$). Therefore, the maximum number of tasks that should be manageable by the *CA* is also the half of the maximum number of tasks mappable on the complete MPSoC ($qnt_{tsk,max,clu} = \frac{qnt_{tsk,max}}{2} = 2^{15}$). The field source task *identification number*, id_{src} requires $\log_2 qnt_{tsk,max} = 2$ Bytes, the field assigned tile *identification number*, id_{tile} requires 2 Bytes because the number of tiles is $n_{tiles,max} = 2^{12}$, the field for destination task *identification number*, id_{dst} requires 2 Bytes, the field id_{app} requires 2 Bytes, the field r_usg_{src} needs 2 Bytes to be able to store $n_{lv} = 2^{16}$ different states, and the fields bw_{req} and the lat require 2 Bytes of memory. All together the memory required by the *task-to-tile mapping look-up-table* in the worst case is calculated as follows:

$$qnt_{tsk,max,clu} \cdot (len(id_{src}) + len(id_{tile}) + \cdots + len(lat)) =$$

$$2^{15} \cdot 7 \cdot 2\,\text{B} = 458.8\,\text{KB}$$

Therefore, the memory required by a single *CA* is:

$$34.8\,\text{KB} + 576\,\text{B} + 458.8\,\text{KB} = 494.1\,\text{KB}$$

It must be considered that the presented amount of memory required by a *CA* is only in the worst case scenario, because the cluster size is the half of the MPSoC. If the cluster is resized, (on the (re-)clustering operation) the memory required by these data structures are changed. Therefore, the data structure *cluster tile look-up-table* will only contain the same number of entries as the number of tiles that belong to this cluster, the *task-to-tile mapping look-up-table* will require only the number of entires to be able to hold the mapping of the corresponding number of the single task mappings, e.g. 2^{15} entries are needed for a cluster that has the size of the half of MPSoC, but only $2048 = 2^{11}$ entries are needed for a cluster with $sz_{clu} = 128 = \frac{\#N}{32}$ tiles, $2048 = \frac{2^{16}}{32}$ and so on. Therefore, for this presented cluster the total required memory is:

$$(128 \cdot (6 \cdot 2\,\text{B} + 1\,\text{B})) + (16 \cdot 18 \cdot 2\,\text{B}) + (2048 \cdot 7 \cdot 2\,\text{B}) =$$

$$1664\,\text{B} + 576\,\text{B} + 28672\,\text{B} = 30.9\,\text{KB}$$

Considering the maximum memory size of almost 0.5 MB which is required to store the data structures, it is difficult to take another option other than the *local memory* of the particular PEs inside a tile. Therefore, the data structures used within the *ADAM* algorithm have been stored in the *local memory* of the tile, where the *GAs* as well as the *CAs* are located.

6.1.7 Collecting Status of Each Tile

The data structures that are stored inside the *CAs* in the *cluster tile look-up-table* are collected at runtime. These data structures have to be collected from each tile directly at runtime and therefore, those have to be kept available by the tiles. The tile architecture has to provide some status information which are stored directly on the tiles. The entry of the *cluster tile look-up-table* presented in Table 6.2 has to be located on each tile.

According to the previous experimental setup each tile has to store following amount of status information:

$$(len(id) + len(tp_{PE}) + len(r\_usg_{comp}) + 4 \cdot len(bw_{used,x}) + 4 \cdot len(q\_vc)) = 17\,\mathrm{B}$$

This status information is communicated from each tile to the corresponding *CAs* at runtime and is aggregated in the *cluster tile look-up-table*. The *CAs* summarize this raw status information and send the current status information about the MPSoC to the *GAs*. This information is stored in the *cluster PE type look-up-tables* that are stored both within the *CAs* and *GAs*. The amount of data that is needed to be transferred depends on the mapping request, more precisely, on the size of the application that needs to be mapped on the executing MPSoC. The entries of the data structures stored within the *CAs* and the *GAs* are modified when the status needs to be updated. Therefore, the status information of the modified tiles are updated in the *CAs* after the mapping is completed and from the *CA* the differences between the old status and the new state of the *cluster PE type look-up-tables*, representing the summarized cluster status are transported to the *GA*.

6.2 Conclusion

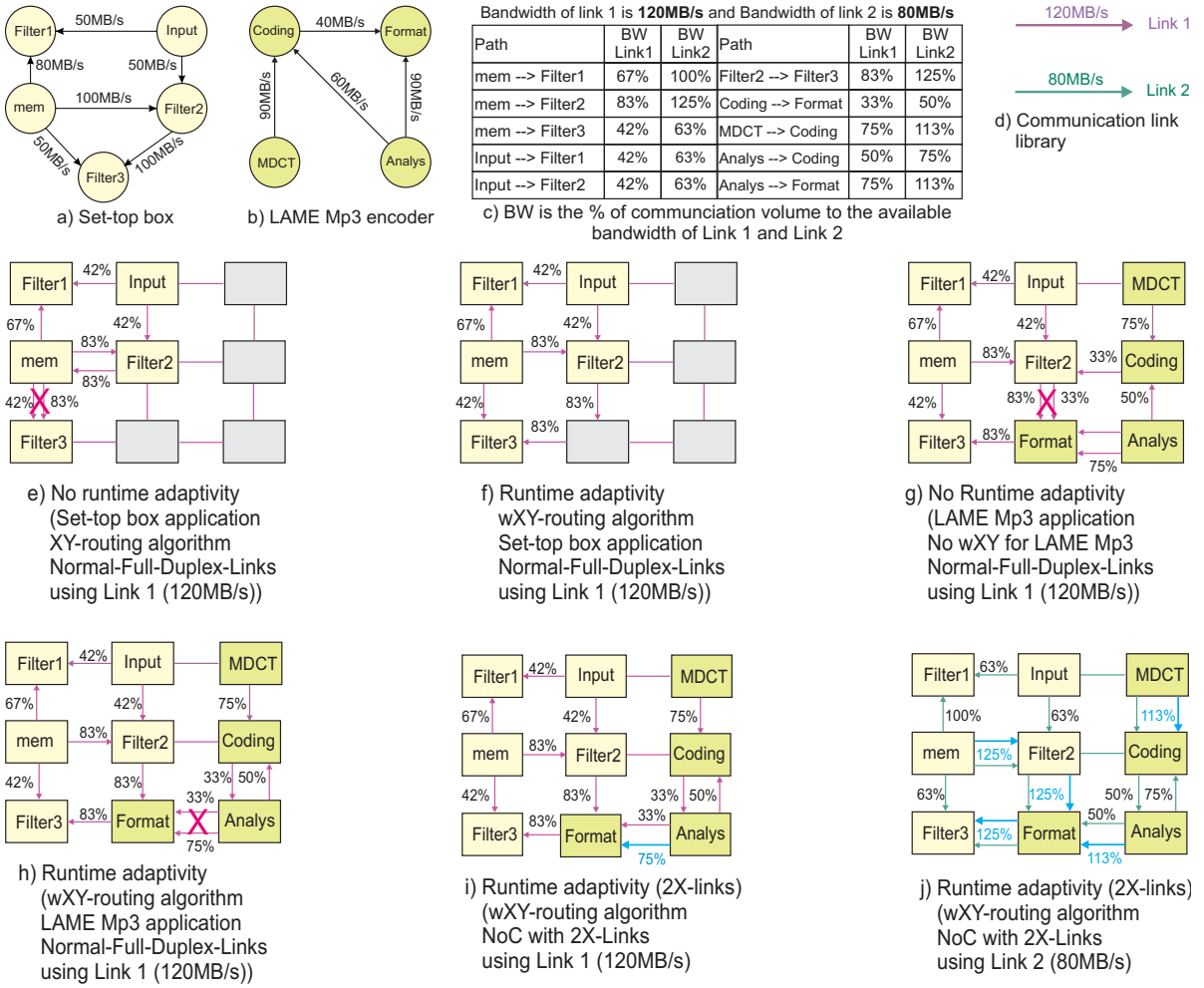
The first algorithm for a runtime application mapping in a distributed manner using an agent-based approach is detailed in this chapter. This algorithm is targeted for the adaptive NoC-based heterogeneous multi-processor systems. The proposed runtime application mapping algorithm, *ADAM* is included in the *system-level* part of the adaptive system-on-chip communication architecture. A detailed evaluation of the proposed algorithm is presented in Chapter 9.

Chapter 7

Architecture-level Adaptation in the Runtime Adaptive Networks-on-Chip

After the *system-level* part of the proposed adaptive system-on-chip communication architecture, has successfully set up a mapping instance, it is up to the *architecture-level* to configure each tile (the router part inside a tile) for the resulting transactions. The router parameters are configured to achieve higher resource utilization in terms of buffer utilization and to increase the number of successfully transaction meeting the bandwidth guarantees. Once a transaction arrives at a router inside a tile, all possible directions must first be checked for suitable routes. To find a valid route which can meet the bandwidth constraints for the transactions it may need to use the concept of *2X-Links* (details are explained in 7.1.3). The number of *Virtual Channel Buffers* (VCBs) per output port has typically been a design-time parameter [14, 76]. An on-demand buffer assignment where *VCBs* are tied to routers and not to individual output ports allows a router to distribute the *VCBs* as needed to any possible route. Therefore, the *VCBs* are assigned to transactions which are in turn assigned to an output port.

The novel contributions in the scope of this chapter are as follows:
(1) The *architecture-level* part of the runtime adaptive system-on-chip communication architecture (*AdNoC*) that adapts the underlying interconnects on-demand in response to changing communication requirements imposed by an application, e.g. after a runtime application mapping request due to reliability issues or user behavior.
(2) To provide on-demand interconnections, a novel adaptive routing algorithm that meets *Quality-of-Service* (QoS) requirements (bandwidth) is presented. The routing algorithm makes decisions locally at each router depending on the available bandwidth in each direction to the neighboring router. Dynamic connections are realized by re-assigning a certain number of buffer blocks to different output ports of a router on-demand. The on-demand buffer block assignment also increases the resource utilization, especially buffer utilization (published in [52]).
(3) A runtime configurable link (*2X-Link*) at the *architecture-level* to compliment the *AdNoC* architecture is presented. The *2X-Link* can adapt the link capacity by changing the direction at runtime on-demand, thereby increasing the resource utilization while considering the reliability issues. The building blocks of the *2X-Link* are two *half-duplex links* instead of the state-of-the-art *simplex* links (published in [54]).
(4) To achieve successful adaptation the communication architecture needs to be observed. Therefore, to provide runtime observability for realizing a successful on-demand adaptation, a



Path	BW Link1	BW Link2	Path	BW Link1	BW Link2
mem --> Filter1	67%	100%	Filter2 --> Filter3	83%	125%
mem --> Filter2	83%	125%	Coding --> Format	33%	50%
mem --> Filter3	42%	63%	MDCT --> Coding	75%	113%
Input --> Filter1	42%	63%	Analys --> Coding	50%	75%
Input --> Filter2	42%	63%	Analys --> Format	75%	113%

Figure 7.1: Motivational example to show the requirement of the architecture-level adaptation

novel low cost runtime observability infrastructure is presented. It is highly flexible and hardly intrusive (published in [53]).

7.1 Architecture-level Adaptation

A variety of functionalities that are included in the router architecture are considered for the *architecture-level* adaptation. The runtime adaptivity in the *AdNoC* architecture is achieved through configuration of different router-related parameters. The parameters that may be configured at runtime are summarized below:

- The routing algorithm used in the router implementation is not determined at design time, i.e. as with the *XY-routing* algorithm, but the routing decision is taken depending on the current traffic load of each output port in the individual router. A weight is calculated to choose a route depending on the availability of bandwidth and the possibility of assigning

a free *VCB* on that route.

- A concept of the *2X-Link* which can configure its transmission direction to three different modes given as: (1) both in one direction (2) both in the reverse direction and (3) both in opposite directions. This concept helps to increase the available bandwidth in a particular direction and therefore, influences the routing decision. This runtime scheme to configure the available bandwidth of a link between two adjacent routers increases the success rate for the transactions.
- An on-demand *VCB* assignment to each output port instead of a fixed assignment at design time is used in the router implementation of the *AdNoC* architecture. A centralized pool of buffer is used and therefore, buffer may be assigned to output ports on-demand for each transaction at runtime.

Details of all these runtime configurable parameters are explained later in this chapter.

7.1.1 Motivational Example Supporting Architecture-level Adaptation

For a motivational example of the proposed runtime *architecture-level* adaptation let us consider a 3×3 Mesh NoC, two applications: the *LAME MP3 encoder* and a *set-top box application* (Figure 7.1 (a,b)), and two macro libraries: the *Intellectual Property* (IP) library and the communication link library (Figure 7.1 (d)). The IP library includes 5 types of *Processing Elements* (PEs) which are not shown in the figure as it is not required to understand the example. Figure 7.1 (c) shows that using the current configuration (link library and applications) $Link1$ can meet the requirement of each transaction while $Link2$ fails (showing more than 100% bandwidth) in 5 of the given transactions considering no runtime adaptivity (here no *wXY-routing* algorithm or *2X-Link* is considered).

At any given time the NoC may be configured to run applications with heavily varying constraints as requested from the upper layer. The upper layer is the application layer which also includes the operating system and other associated system administering applications. All these parts are abstracted as the *system-level* part (see Definition 5 of Chapter 5) of the adaptive system-on-chip communication architecture (*AdNoC*) and is described in Chapter 6. Lets assume, at time t_0 the NoC is running the *set-top box application* in Figure 7.1 (e). In the current execution instance, all the transactions other than the transaction from *Filter2* (the task name is used to represent the tile where that particular task is mapped onto) to the *Filter3* are successful in conducting a deterministic *XY-routing* algorithm like [23, 61, 77] (in a 2D tile-based NoC the packet first goes in X direction and then in Y direction toward its destination). However, the transaction between the *Filter2* and the *Filter3* will fail due to the limited bandwidth availability in the link between the *mem* and the *Filter3*. This is denoted by the red X. Consequently, the router at *Filter2* is forced to try another route which is successful (using the proposed *wXY-routing* algorithm) shown in Figure 7.1 (f). With the routes, the routers supply corresponding *VCBs*, assigning the buffer to the corresponding output ports on-demand.

Generally, in application-specific NoCs or general-purpose NoCs the *VCBs* are assigned at design time to accommodate transactions but this can limit the accomodable transactions in a particular output port of a router. In this scenario, as *VCB* assignment is done from a central pool of *VCBs* therefore, the proposed adaptive routing algorithm (the *wXY-routing* algorithm) may be used without considering the availability of *VCBs* at a particular output port.

Now again assume that at time t_1, the *set-top box application* and the *LAME MP3 encoder* need to be executed in parallel. Similar to the *set-top box application*, using the *XY-routing* algorithm [61, 77], the traffic from *Coding* is routed through *Analysis* to *Format* in Figure 7.1 (g). This however fails, as the combined bandwidth required by the transactions exceeds the total link capacity. This is denoted by the red X. With the *wXY-routing* algorithm proposed in the scope of this thesis, however, the tasks are able to choose different routes but in this scenario, the algorithm is still unable to manage to find a valid route. The runtime (re-)mapping mechanism presented in the Chapter 6 can also be tried to solve this problem, but for this case it is investigated that the (re-)mapping does not work because no instance of mapping can meet the bandwidth constraints.

Under this presented scenario, the state-of-the-art mechanism proposed in [113], splitting the traffic across multiple routes between the source and the destination for each transaction (*traffic-splitting*) may be used to logically expand the link capacity. However, this approach leads to higher packet latency, flits rearranging, unpredicted network situations, and in extreme cases may cause an increase in communication bottlenecks. To remedy this, an ideal solution may be to physically expand the link capacity of a particular link at runtime. To accomplish this, the possibility of using link reversal mechanisms is investigated (the *2X-Link* is used), e.g. if the link from $Format$ to $Analysis$ can be reversed, then the bandwidth capacity in the opposite direction will be doubled and all the transactions will be successful (see Figure 7.1 (i)). It is further investigated that it is able to decrease the available bandwidth even more by using only $link2$ from the communication library while still being able to successfully accommodate all of the transactions (see Figure 7.1 (j)). Therefore, the link reversal mechanism is highly efficient in terms of *link-resource-utilization* compared to an adaptive system-on-chip communication architecture, i.e. *AdNoC* without runtime configurable links.

The example clearly shows that using an adaptive system-on-chip communication architecture considering both the adaptive routing algorithm and the *2X-Link* can increase the number of successful transactions and therefore, bring higher efficiency to the network. On the other side the novel on-demand *VCB* assignment uses resource multiplexing and therefore, increases *buffer-resource-utilization*.

7.1.2 Parameters for the Runtime Adaptation

The parameters that can be adapted at runtime in the communication architecture are already summarized in 7.1 and are as follows: (1) the adaptive routing algorithm [52], (2) the available bandwidth capacity of a link termed as the *2X-Link* [54], and (3) on-demand *VCB* assignment to the output ports considering the direction of the transactions [52]. Finally, a runtime observability component is included in the architecture to perform a successful adaptation and to gather

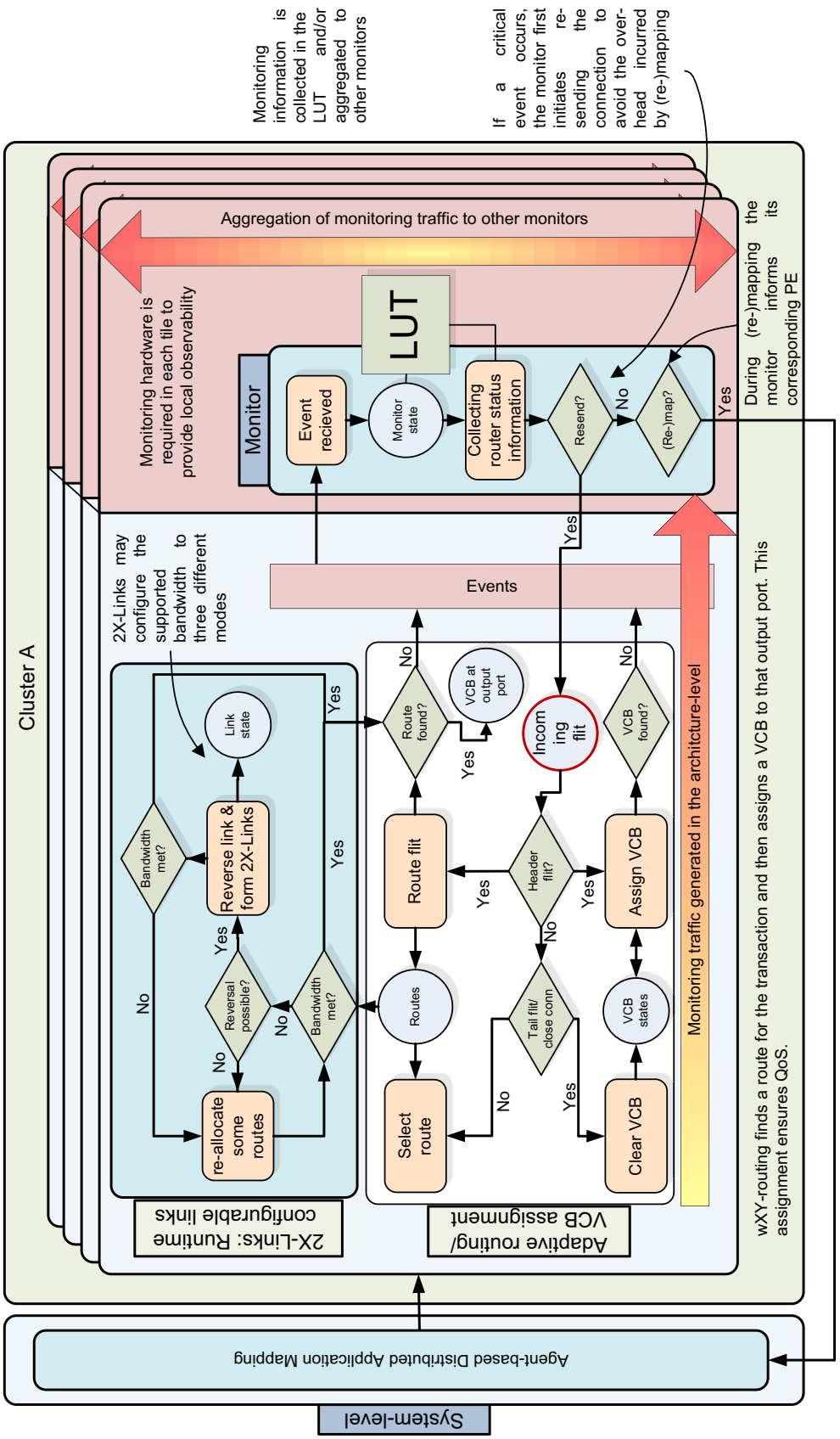


Figure 7.2: Functionality during architecture-level adaptation

unsuccessful events to assist for further steps [53]. A diagram showing the workflow of the adaptive router in the *architecture-level* is presented in Figure 7.2. At a given time *t*, application task graph is requested to be mapped by the *system-level* part onto the NoC architecture. In the *architecture-level*, apart from the system-on-chip transmission functionality the main purpose of the adaptive system-on-chip communication architecture (*AdNoC*) is to find a suitable route that meets all the requirements for the transactions, i.e. the bandwidth is met and then assigning *VCBs* on-demand on those directions for the current instance of mapping. Several possible situations which may occur while searching for a suitable route after a mapping instance has been passed down to the *architecture-level* are summarized below (see Algorithm 10):

- If the adaptive router can find a suitable route for all the transactions, then the *VCBs* are assigned according to the transaction direction and the *architecture-level* adaptation ends successfully.
- If the adaptive router cannot meet the performance-related guarantees for all the transactions of the current instance of mapping, the router searches for an appropriate link which may be reversed to form a *2X-Link* in its transmission direction to help performance-critical transactions. When the feedback value is *true*, then the *VCBs* are assigned accordingly.
- If no suitable link can be reversed for successful transactions, some routes may be (re-)allocated to relieve the link resource congestion. Since the computational complexity for the route allocation is less than for the (re-)mapping, it is at first tried to balance the load on each link. If all the performance-related guarantees can be met, the final state is reached.
- If both the link reversal and route (re-)allocation do not work, (re-)mapping is the only way to meet the performance-related guarantees as proposed in Chapter 6.

When the last option does not lead to a successful application execution that meets every performance-related guarantee, then the (re-)mapping request will be refused by the currently executing system. The *architecture-level* adaptation is performed with the assistance of runtime observability infrastructure that is an integral part of the proposed system-on-chip communication architecture.

7.1.3 Runtime Configurable Links (2X-Links)

Communication links in the state-of-the-art NoCs typically employ full-duplex communication (simultaneously bi-directional) using two simplex links [76, 117, 133, 173] (details of link types are explained later). In between two adjacent routers there are two uni-directional simplex links, i.e. one from *Router 1* to *Router 2* and another from *Router 2* to *Router 1*. Both of these uni-directional links together form the *full-duplex links*. The links are design-time parameterized (e.g. the *bit-width* of the link) and therefore, provide a fixed link capacity in terms of bandwidth for a certain clock frequency. It is observed that there may be several scenarios that may require configurable links (links which can adjust their supported bandwidth capacity by using the proposed *2X-Links*). Below some of those scenarios mentioned above are shown:

Algorithm 7 2X-Links for the adaptive system-on-chip communication architecture (AdNoC)

$avBW_{dir}$: Available bandwidth in the direction dir
$reqBW$: Bandwidth requested by transactions towards dir
BW_{dir}: Total bandwidth towards dir
$usedBW_{dir^{-1}}$: Bandwidth used in the opposite direction of dir
t: Direction of the link (if 2X-Links then it is 2 otherwise 1)
QoS: performance-related guarantees (bandwidth, latency, etc.)

```
1: t = 1
2: upon receiving a transaction and the destination do
3: if QoS are not met then
   // try to reverse links
4:    for all links with avBW_dir < reqBW do
5:       if usedBW_dir^-1 = 0 then
6:          reverse link
7:          t = 2
8:          avBW_dir += BW_dir^-1
9:          if QoS are now met then
10:            route to dir
11:            return
12:         else
13:            t = 1
14:         end if
15:      end if
16:   end for
17: end if
```

- As the number of possible simultaneous inter-task transactions increases in a particular link in one direction, it becomes more difficult to meet the bandwidth requirements of all transactions. Therefore, there might be situations where a certain link requires an increase in link capacity (only possible when all the transactions are flowing in a single direction).
- Sometimes, the bandwidth requirements of the tasks expand according to user requirement changes (e.g. the user wants to switch video playback to a higher resolution). If in such a scenario the current link capacity of one half of the duplex link cannot meet the requirement then the other half may be reversed to add more available bandwidth (doubled the link capacity in the proposed *2X-Link*).
- When a hardware fault in one direction of the link, e.g. in the link from *Router 1* to *Router 2*, occurs randomly, the transactions via this link in this direction will not succeed. In such a scenario, if the other half of the *full-duplex link* (link from *Router 2* to *Router 1*) has the ability to reverse its direction while free, it could transmit these transactions. Therefore, *2X-Link* increases the resource utilization and improves reliability factors through the runtime adaptation.

7.1.3.1 System-on-Chip Communication Links

Some link-related terms to explain the proposed *2X-Links* are discussed here:
A *simplex communication link* is a link where transactions are allowed only in one direction.

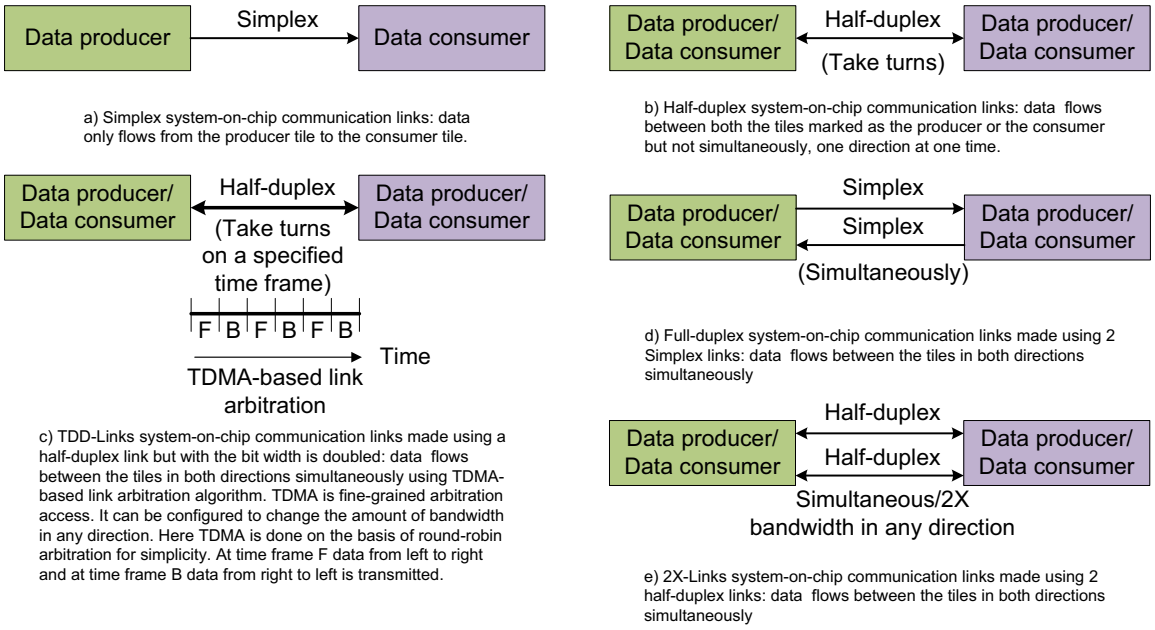


Figure 7.3: Different types of system-on-chip communication links

Figure 7.3(a) shows that data can only be transmitted from *Data producer* to *Data consumer*. A *half-duplex link* provides communication in both directions, but only in one direction at a time (not simultaneously) (see Figure 7.3(b)). A *full-duplex link* allows communication in both directions, and unlike *half-duplex link*, allows this to happen simultaneously (see Figure 7.3(d)). Two *simplex communication links* may be combined together to form a *full-duplex link*, e.g. links in the current NoC architectures [76, 117, 133, 173].

A *2X-Link* is a combination of two *half-duplex links* and therefore, their transmission can be configured to three different modes (both in one direction, both in the reverse direction, and both in opposite directions). Figure 7.3(e) shows the data transmission of the proposed *2X-Link*. For the runtime *AdNoC* architecture these *2X-Links* are used.

A *Time-Division-Duplex-Link* (TDD-Link) uses time division multiplexing to separate inward and outward transactions. It emulates full-duplex communication over a single *half-duplex link* (the bit width of the single *half-duplex link* used to form a *TDD-Link* is doubled in order to be able to compare it to the two *half-duplex links* used to form the *2X-Link*). Time division duplex has a strong advantage in the case where the asymmetry of the forward link and reverse link data speed is variable. As the amount of forward link data increases, more bandwidth can dynamically be allocated and as it shrinks, it can be taken away. The *TDD-Link* is implemented in the scope of this work to compare it to the proposed *2X-Link* (see Figure 7.3(c)).

7.1.3.2 Design and Implementation of the 2X-Links

If in a *Normal-Full-Duplex-Link* the link capacity is *"X"* then the proposed *2X-Link* may adjust its link capacity among *"Zero"*, *"X"*, and *"2X"*. The possibility of incorporating the *TDD-*

Link as a runtime configurable link in the *AdNoC* is also investigated and the *TDD-Link* is then compared to the proposed configurable *2X-Link*. The *2X-Link* has the advantages of lower area and fault-tolerance ability over the *TDD-Link*. Its limitations are: transmission can only be supported in one direction and the bandwidth granularity in the link is coarse-grained (0, X, and 2X). On the other hand, the *TDD-Link* may adjust its time slot to support bi-directional transactions and at the same time may configure the runtime link capacity in one direction in a fine-grained way but suffers from higher area overhead and fault-tolerance ability. Considering the area overhead and the fault-tolerance ability, the *2X-Link* is chosen as an integral part of the runtime adaptive system-on-chip communication architecture (*AdNoC*).

Algorithm 8 Weighted XY-routing algorithm at router-level

BW_{dir}: Total bandwidth towards dir
w_{dir}: Weight in each direction to find the route for a transaction
t: Direction of the link (if 2X-Link then it is 2 otherwise 1)
$reqBW$: Bandwidth requested by transactions towards dir
$avBW_{dir}$: Available bandwidth in the direction dir for a router

1: **upon** receiving a transaction and the destination **do**
2: **if** transaction in look-up-table from different source port **then**
`// look for potential loops`
3: loopRoute $\Leftarrow$ output port of other transaction
4: **end if**
5: **for all** output ports p_{dir} **do**
`// initialize all weights to zero`
6: $w_{dir} \Leftarrow 0$
7: **end for**
`// Calculation of the route distance relative to the destination`
8: $dx =$ | destination $x -$ current x|
9: $dy =$ | destination $y -$ current y|
10: **for all** p_{dir} with $avBW_{dir} > reqBW$ **and** loopRoute $\neq p_{dir}$ **do**
`// there is no loop and bandwidth available`
`// East and West output ports`
11: **if** p_{dir} points toward destination x **then**
12: $w_{dir} \Leftarrow avBW_{dir} \times dx + BW_{dir} \times t$
`// Weight calculation considers the existence of the 2X-Links`
13: **else if** $avBW_{dir}$ points away from destination x **then**
14: $w_{dir} \Leftarrow avBW_{dir} \times t$
15: **end if**
`// North and South output ports`
16: **if** $avBW_{dir}$ points toward destination y **then**
17: $w_{dir} \Leftarrow avBW_{dir} \times dy + BW_{dir} \times t$
18: **else if** $avBW_{dir}$ points away from destination y **then**
19: $w_{dir} \Leftarrow avBW_{dir} \times t$
20: **end if**
21: **end for**
`// route toward the port having highest weight`
22: route $= p_{dir}$ with $\max(w_{dir})$
`// save the route in the look-up-table`
23: look-up-table $\Leftarrow$ transaction $=$ route
24: **return** route

Principally, the concept of the *2X-Link* is not new and has been borrowed from *Telecoms* [51]

and *Wireless Sensor Networks* [29] research (in these domains it is termed as the link reversal mechanism). Link reversal algorithms have been studied experimentally and have been used in practical routing algorithms, e.g. TORA [129]. The circuit implementation and the *2X-Links* inside the complete *AdNoC* architecture are shown later in this chapter.

7.1.4 Weighted Routing Algorithm

To provide bandwidth guarantees in an adaptive system-on-chip communication architecture (*AdNoC*), the underlying communication interconnects needs to provide an adaptive route allocation strategy that is not static in function. In a NoC where runtime adaptation is not possible, the routing decision may be a distributed or source-based deterministic routing scheme. In a distributed deterministic routing scheme, the routing decision is determined locally at each router using predefined rules, e.g. the *XY-routing* algorithm in the *QNoC* [23] architecture. The source-based deterministic routing scheme (e.g. *Xpipe* [14]) keeps the complete route in its header information and needs the global view of the whole chip before execution or even at design time. That is why both schemes are not suitable for the *AdNoC* architecture where the group of the tasks and their mapping may change during runtime. Therefore, finding a route for a given logical network and physical mapping of an application are major challenges. The runtime route allocation algorithm is given in Algorithm 8.

For a requesting transaction, the route is checked in every possible direction and the *VCB* is assigned accordingly on-demand. The weighted XY-routing (*wXY-routing*) algorithm presented in Algorithm 8 assigns each output port a weight based on available bandwidth and dx, the x coordinate (columns) distance or dy, the y coordinate (rows) distance between the current and the destination nodes. This ideally gives the packet a maximum number of sensible routing choices along its route as it allows the packet to be routed toward its destination in both the x and y directions. The weight is also proportional to the available bandwidth. If the output port is chosen with the highest associated available bandwidth, the used bandwidth is distributed as evenly as possible among the output ports. Thus the other output ports are more likely to be able to accommodate future transactions. In the proposed *AdNoC* architecture, the concept of the *2X-Link* described in 7.1.3 is used to increase the available bandwidth of a particular link. Therefore, the weight calculation in the *wXY-routing* algorithm also inherently considers the advantages of using the *2X-Links*. By allowing all these values to contribute to the weight, the weight becomes a trade-off among these considerations.

The *wXY-routing* is described as follows: given is the tuple $\mathcal{P} = \{N, E, S, W\}$. Each $i \in \mathcal{P}$ has a weight w_i and available bandwidth $avBW_i$ with $avBW_i < t \times BW_{\max}$, $BW_{\max}$ being the maximum link capacity (simplex link) represented as bandwidth, and t indicates whether a link is a *2X-Link* or not. If the link is a *2X-Link* then $t = 2$, otherwise, $t = 1$. The current router coordinates are x, y. Each packet p has destination coordinates x_d, y_d and a required bandwidth $reqBW$. The weights are assigned as follows:

$$w_N = \begin{cases} avBW_N \times |y_d - y| + t \times BW_N & , y_d - y < 0 \\ 0 & , avBW_N < reqBW \\ avBW_N & , \text{otherwise} \end{cases}$$

$$w_E = \begin{cases} avBW_E \times (x_d - x) + t \times BW_E & , x_d - x > 0 \\ 0 & , avBW_E < reqBW \\ avBW_E & , \text{otherwise} \end{cases}$$

$$w_S = \begin{cases} avBW_S \times (y_d - y) + t \times BW_S & , y_d - y > 0 \\ 0 & , avBW_S < reqBW \\ avBW_S & , \text{otherwise} \end{cases}$$

$$w_W = \begin{cases} avBW_W \times |x_d - x| + t \times BW_W & , x_d - x < 0 \\ 0 & , avBW_W < reqBW \\ avBW_W & , \text{otherwise} \end{cases}$$

subject to:

$$\begin{cases} t = 2 & , if \text{ 2X-Link } is\ activated \\ t = 1 & , otherwise \end{cases}$$

The route r chosen is then:

$$r = \begin{cases} P\ (local\ processor\ port) & , x = x_d \text{ and } y = y_d \\ i \in \{N, E, S, W\} & , \text{else, with } w_i = \max_i(w_i) \end{cases}$$

The *wXY-routing* algorithm is designed to make meaningful routing decisions whenever possible. However, in certain worst-case scenarios, routing problems can arise which need to be dealt with.

A *livelock* situation occurs when packets are routed through a network without ever reaching its destination. The *wXY-routing* algorithm given in Algorithm 8 may lead to *livelock*. However, since the algorithm routes towards a destination whenever possible, *livelock* can only happen if a packet is continuously misrouted away from its destination. This is a sign that there are no available routes to the destination. *Livelock* may be dealt with using a *Time-to-Live (TTL)* counter. This counter is given an initial value equal to the *Manhattan distance* $+2m$, where m is the number of allowed misroutings. At each hop the *TTL* for a packet is decremented by one and when the counter reaches zero, the packet is removed from the network and a feedback message is sent to the sender.

Algorithm 9 On-demand buffer assignment

1: **upon** receiving a transaction and direction **do**
2: search for next free buffer $bf_{\text{free}} \in$ buffer pool B and not in buffer table
3: **if** bf_{free} found **then**
`// assign available buffer to current direction`
4: current buffer $bf_{\text{curr}} \Leftarrow bf_{\text{free}}$
5: buffer table $\Leftarrow bf_{\text{curr}} \rightarrow$ output port
6: **return** bf_{curr}
7: **else**
8: **return** no buffer available
9: **end if**

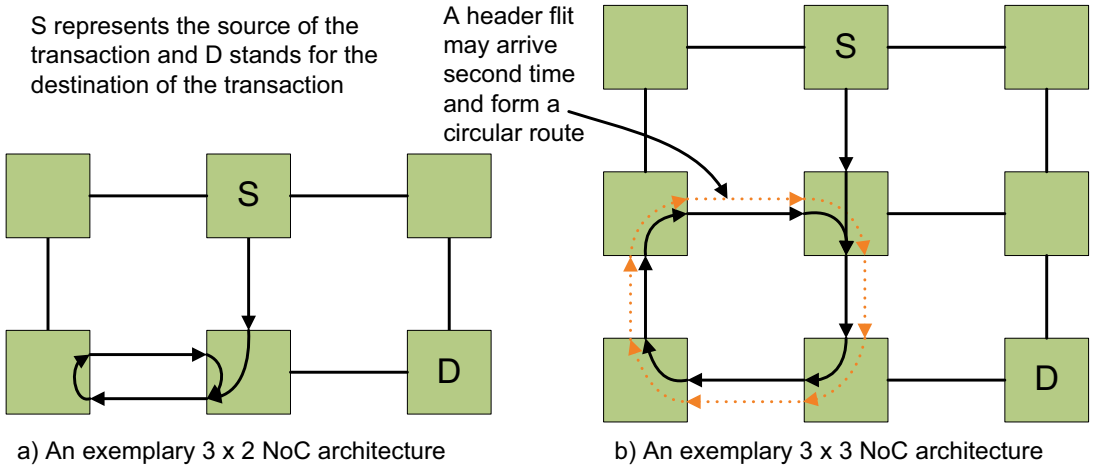


Figure 7.4: Examples of circular routing from the source S to the destination D

A scenario (*ping-pong effect*) which may lead to a *livelock* situation is shown in Figure 7.4 (a). In this example, after misrouting (i.e. to the bottom left router), the algorithm will tend to favor returning the packet directly backward towards the destination. This is generally not the desired behavior to avoid *livelock* since the previous router has already decided to route away from the destination. This is remedied in the *wXY-routing* algorithm by setting the weight of the output port corresponding to the input port equal to the lowest possible non-zero weight, if there is enough bandwidth, and 0, if there is not. The return route should not be avoided altogether as it is possible that there is absolutely no other way out of the router. This however leads to another problem: what happens when the *header flit* of the packet arrives at a router for the second time? It sees that its *transaction identification number* is already in the router and is routed in the same direction again forming a *circular-routing* (see Figure 7.4 (b)). Generally, every time a packet crosses its own route, the result is a *circular-routing* (see Figure 7.4 (a) and Figure 7.4 (b)). To avoid this problem in the *AdNoC*, the *transaction identification number* is tied to an input port. When a *header flit* crosses an already visited router in its route from a different input port then it generates two different transactions with the same *transaction identification number*. It is guaranteed that the second transaction is not routed to the previously chosen output port by setting its weight to 0. The router then sends out a *split-tail-flit* in the direction of the previous route for the first transaction and halts the transaction until the *split-tail-flit* arrives at the router. It deletes the first transaction and ties the second one to the input port of the first.

7.1.5 On-demand Buffer Assignment

When transmitting data over a packet-switched network, it is necessary to store parts of the data at each intermediate hop. In a *wormhole-based* router, this requires *VCBs* for each router through the complete route of each transaction. Up till now, the number of *VCBs* at one port has always been fixed at design time [14, 23, 72, 73, 92, 170]. With the on-demand assignment, the *VCBs* are not tied to ports, but only to the router itself. The router may distribute the *VCBs*

to any route as needed by assigning those to the according output ports. The scheme to assign buffers on-demand (at runtime) is given in Algorithm 9. Physically, it is realized by using a central pool of FIFOs connected to each output port through a crossbar matrix. Pointers need to be saved for each output port to remember the current state of the buffer assignment.

Algorithm 10 Runtime architecture-level adaptation

```
1: upon receiving a mapping instance and a transaction at runtime do
   // A connection is established in a distributed way for each transaction
   from the source to the destination.  In each router a route is determined
   and a VCB is then assigned towards that route.  To get a route the
   bandwidth may be configured using the 2X-Links
2: for all All transactions t_i do
3:    repeat
4:       if destination = processor port then
5:          route ⇐ processor port
6:       else
7:          if flit type ≠ head or transaction in look-up-table from same source port then
            // non-header flits are always in look-up-table
8:             route ⇐ look-up-table
9:          else
            // get route
10:            route ⇐ do weighted XY-routing algorithm and call Algorithm 8
11:            if route found then
               // assign buffer to route
12:               do runtime buffer assignment for found route call Algorithm 9
13:            else if possible to reverse the link then
14:               call Algorithm 7
15:               route ⇐ do weighted XY-routing algorithm and call Algorithm 8
16:               if route found then
                  // assign buffer to route
17:                  do runtime buffer assignment for found route call Algorithm 9
18:               end if
19:            end if
20:            if no route found or buffer assignment unsuccessful then
21:               collect router status information
22:               send information to higher level (see Algorithm 11)
23:            end if
24:         end if
25:         if flit type = tail and keep-alive not requested then
            // free buffer
26:            remove buffer from buffer table
27:            remove transaction from look-up-table
28:         end if
29:      end if
30:   until (complete route of the transaction is found)
31: end for
```

The benefits of such on-demand assignment is evident: through on-demand assignment, buffers are only assigned when needed meaning that *VCBs* can be reused by different ports. The obvious drawback of this method is that additional cycles are needed to retrieve the buffer pointers and the buffer contents. Experiment shows, however, that the latency for each transaction is

mainly dominated by the packet size but not by the distance in a pipelined *wormhole-based* communication. The processing time for sending data is fixed in each router at design time (and is therefore, constant). The latency in a pipelined architecture can be formulated as follows:

$$L_{trans} = F \times T_p + (D_{manh} - 1) \times T_p[cycles] \tag{7.1}$$

Here L_{trans} is the latency for each transaction, F is the number of flits in a packet, T_p is the number of cycles required to process and to send each flit, and D_{manh} is the *Manhattan distance* to the destination. The packet size F is the dominant factor in this equation as the T_p is constant and the change in the *Manhattan distance* D_{manh} due to the routing algorithm is negligible. The routing algorithm tends to keep the length of the route near to the *Manhattan distance* by assigning a very low weight to the reverse directions (directions pointed toward the source). Therefore, the runtime route allocation algorithm does not affect the end-to-end latency.

Figure 7.5 shows an exemplary scenario to showcase the runtime behavior using different transactions in one router. Transactions are represented by different sized rectangle boxes depending on their bandwidth use within that link. In this exemplary scenario, the different parameters that can be configured at runtime include: (a) the route chosen by the *wXY-routing* algorithm and (b) the on-demand *VCB* assignment. The concept of the *2X-Link* is not shown here as it is assumed that the weight calculation is abstracting it. Let us now discuss what happens at selected points in time T_i in Figure 7.5.

T_0: All four directions are occupied with four different transactions and therefore, buffers are also assigned. Some *VCBs* are un-assigned and also there is available bandwidth for other transactions in the South and the East directions.

T_1: Transaction T5 requests a route and weights are calculated until t_δ taking 4 cycles. A *VCB* is also assigned to the calculated direction before t_δ. By now, all five transactions meet their bandwidth requirements.

T_2: Transaction T1, T2, and T4 free their corresponding virtual channels and assigned buffers. Therefore, these *VCBs* and the released virtual channel bandwidth may be utilized by future transactions.

T_3: Four new transactions T1, T2, T4, and T6 request processing and they are granted resources for transmission.

T_4: Transactions T7 requests a route and *VCB* but due to unavailable buffer resources, the transaction cannot be granted. So, the requesting transaction has to wait or inform the upper layer through the system monitor (*ROAdNoC* presented in the next Section 7.2). This situation may also occur in case there is a *VCB* but no bandwidth left in any direction.

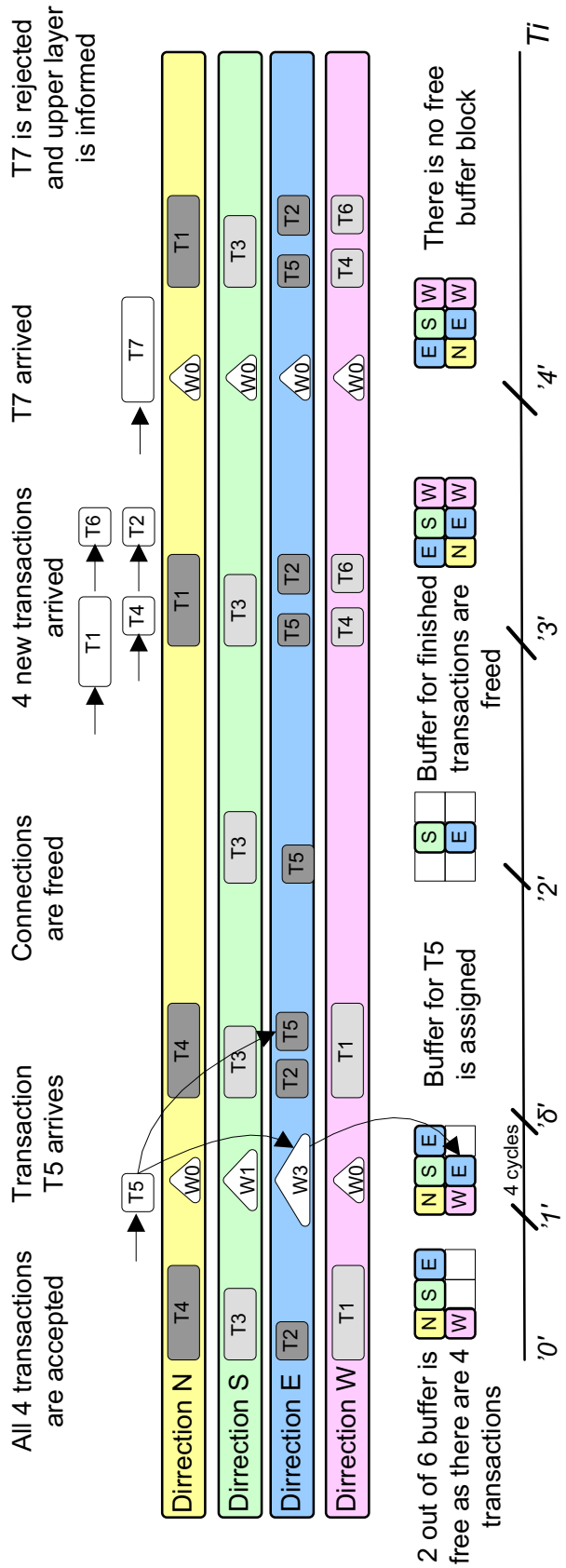


Figure 7.5: Scenario of the runtime adaptive architecture capabilities

7.2 Runtime Observability for the AdNoC Architecture (ROAdNoC)

In order to assure a certain degree of QoS (e.g. guarantees in performance and required bandwidth), a feedback of the current system state must be available. This can be achieved through runtime observability in the *AdNoC* architecture. If a runtime observability infrastructure comes with only a small hardware overhead and some small communication overhead that would then more than compensate the degree of freedom achieved using a successful adaptation. Within this thesis, an event-based NoC monitoring component at *architecture-level* that offers runtime observability is proposed. In the *AdNoC* architecture, *runtime observability* is denoted as a complete infrastructure and *"monitoring"* is a hardware component attached to each tile. The prime challenges for runtime observability are scalability, flexibility, non-intrusiveness, real-time capabilities, and cost. In order for the monitoring components to be as non-intrusive as possible, they need to keep their interference with normal system execution, so-called *probe effects* [90] at a minimum. An example of these effects would be the sending of monitoring packets through the regular point-to-point data links between routers. A monitoring packet is the traffic that is generated during runtime observation of the system state and is described later in this chapter. If these packets are injected too rapidly, they demand resources which otherwise may have been used for regular traffic. It is therefore necessary to limit monitoring traffic by keeping its bandwidth usage and occurrence frequency minimal. The *Runtime Observability for an Adaptive Networks-on-Chip* (ROAdNoC) infrastructure that supports successful *architecture-level* adaptation is implemented using a low cost and light-weight monitoring component inside each tile. The key parts of *ROAdNoC* infrastructure are described below.

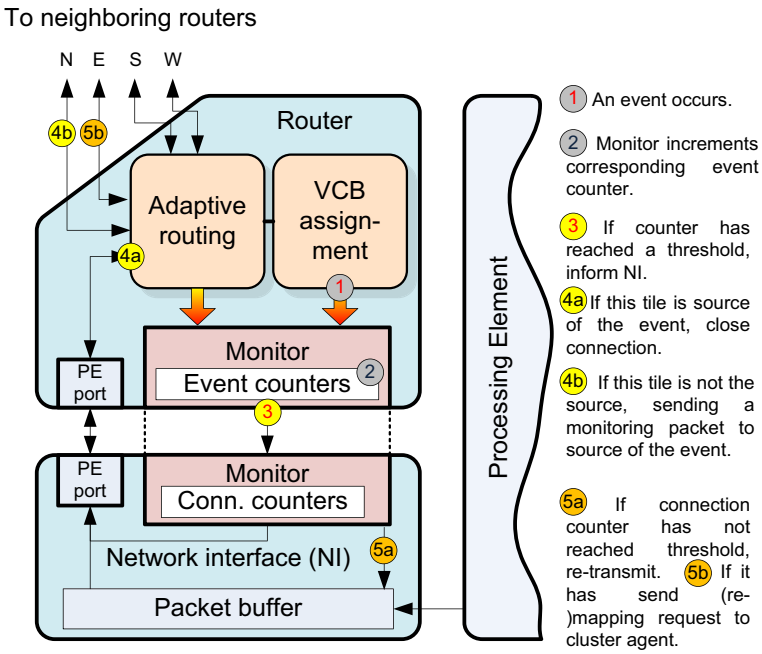


Figure 7.6: Overview of the monitoring component

7.2.1 Monitoring Events

Events are situations when successful traffic propagation is prevented in the communication architecture. The *ROAdNoC* infrastructure is event-based and events are caused by failures in a subsystem of an individual router, i.e. the adaptive routing algorithm or the on-demand transaction assignment to a *VCB*. The list of events is generic and can be enhanced depending on the architecture. The events that are necessary for the proposed adaptive system-on-chip communication architecture (*AdNoC*) are explained in the following (events that may be generated from the *2X-Links* are not considered in the current implementation):

- **TTL-expire-event:** In order to assure livelock-free routing, each packet is given a maximum *Time-To-Live* (TTL) hop count. If a packet fails to reach its destination within the *TTL*, it is removed from the network. The *TTL* is the *Manhattan distance* plus a given maximum number of misroutes.
- **No-route-found-event:** If the routing algorithm fails to find any available routes inside a router, the packet is removed from the network. This occurs when too many bandwidth slots (for details of bandwidth slot see Chapter 3) are already reserved in all directions, not leaving enough bandwidth to accommodate incoming packets.
- **No-buffer-event:** If the *Virtual Channel Arbitrator* (VCA) fails to find a free *VCB* to hold the incoming packet it is removed from the network.
- **Buffer-full-event:** Occurs when the *VCA* already has an assigned *VCB* to a transaction but cannot write to it because it is full. This does not result directly in packet loss but it is a sign of congestion in the network. This situation may be resolved automatically, however it should be observed and be reacted to if it persists over a prolonged period of cycles.

These events correspond to the *NoC alert events* in the monitoring system for Æthereal NoCs presented in Section 3. Unlike in *Æthereal* [39, 40], these events are used to identify the faults during NoC adaptation at *architecture-level* and are used to invoke the necessary steps to remedy it. The events given here are binary in nature; that is, either an event has occurred or not (except *buffer-full-event* which is invoked for a given threshold). This simplification eliminates the need for attributes to be supplied for events as with the monitoring component for *Æthereal* [39]. The *user-configuration-events* of *Æthereal* (high level communication configuration events such as *transaction-opened* and *transaction-closed*) are indirectly observed. However this is only done in order to set up the counters for each transaction and to free them when the transaction closes.

7.2.2 Design and Event Collection

The monitoring component of a router for the *ROAdNoC* infrastructure consists of a *Look-Up-Table* (LUT) containing a set of counters for each transaction going through the router. These are tied to events which can occur in the on-demand *VCB* assignment and in the adaptive routing

Algorithm 11 Aggregation and processing of the monitoring traffic

input: event $e = \{$event type t, transaction ID C, transaction source $S\}$
output: $null$
definitions: X: current router
E: event queue
LUT[transaction ID, event type]: look-up-table with event counters
τ_t: given threshold for events of type t
δ: given threshold for re-sending
s_c: send counter in S for C
NI: network interface
CA: cluster agent associated with X

1: get next event e from E
2: event_counter $\leftarrow$ LUT[C, t]
3: **increment** event_counter
4: **if** event_counter $> \tau_t$ **then**
5: **if** X != S **then**
6: **send** event message $e = \{t, C, S\}$ to S
7: **else**
8: **signal** NI: send tail flit from packet buffer for C to close conn.
9: **if** $s_c < \delta$ **then**
10: **signal** NI: re-send packet for C
11: **else**
12: **send** (re-)mapping message $\{remap$,S,$t\}$ to CA
13: **end if**
14: **end if**
15: LUT[C, t] $\leftarrow 0$
16: **else**
17: LUT[C, t] $\leftarrow$ event_counter
18: **end if**

parts of the router. These counters are incremented every time an event is reported, thereby collecting data on events.

The counters are stored in the *LUT* with the corresponding *transaction identification number* and the source address of a transaction. In particular, the source address can be the same address as the monitoring component if the corresponding PE is the source of the transaction. This is a special case, as the counters are not only incremented by events occurring within the router but also through messages received from monitoring components in other routers. In this case, the triggering part is the *Network Interface* (NI) in Table 7.1.

7.2.3 Aggregation and Processing

An adaptation fault occurs when an event counter reaches a certain value. The event aggregation and processing scheme is explained in Figure 7.6 and the functionality upon problem detection is given in Algorithm 11 with corresponding parts of the figure in the following description:

1. Monitoring component X detects a problem

Counter	Triggering Router Part	Response
TTL-expire-event	wXY-routing or NI†	directly
No-route-found-event	wXY-routing or NI†	directly
No-buffer-event	VCA/Buffer allocation or NI†	directly
Buffer-full-event	VCA or NI†	wait till over threshold

†Message from other monitor

Table 7.1: Event counters

2. Monitoring component X sends message to NI part
3. NI part of X looks if it is the source S of the transaction
   - NI of X sends message to S if it is not
4. If $X = S$, X tells NI to close its current transaction by sending a *tail flit*: freeing VCB and clearing the route
5. IF $X = S$, X examines its transaction counter and
   - Tells NI to resend data if a specified value is not reached, or
   - Sends message to upper layer requesting (re-)mapping

The aggregation is done through the *NI* by sending messages to the source of the transaction. The processing is done partially in the *NI* and in the cluster agent. *NI* takes care of re-transmission while the cluster agent is invoked if a (re-)mapping is needed. The time-line diagram which portrays a certain scenario of the *ROAdNoC* infrastructure can be seen in Figure 7.7.

T_1: The processing element PE_1 associated with router R_1 begins to send data to another PE. The *NI* assigns this transaction the *transaction identification number* C_1. Thus, the monitoring component M_1 of R_1 adds this transaction to its list of observed transactions. Here, only one exemplary counter is given in the monitoring component. This counter representing the event of *no-buffer* being available in a router is initialized to zero.

T_2: On its way to the destination the *header flit* of this transaction arrives at router R_2. R_2 is generally not reached after one hop; the *header flit* may already have been routed through other routers. Upon arrival of the *header flit*, C_1 is added to the transaction table of monitoring component M_2 at router R_2.

T_3: The flit then progresses through R_2 until it reaches the *VCA*. The *VCA* fails to allocate a free buffer for C_1 and this event is reported to M_2 which increments its *no-buffer* counter. The *VCA* then simply discards the *header flit* and all subsequent flits and does not send a *NACK* signal as it would if the buffer were already successfully assigned but simply full. This is done so that there is no blocking in previous routers and a *tail flit* can arrive to close the transaction.

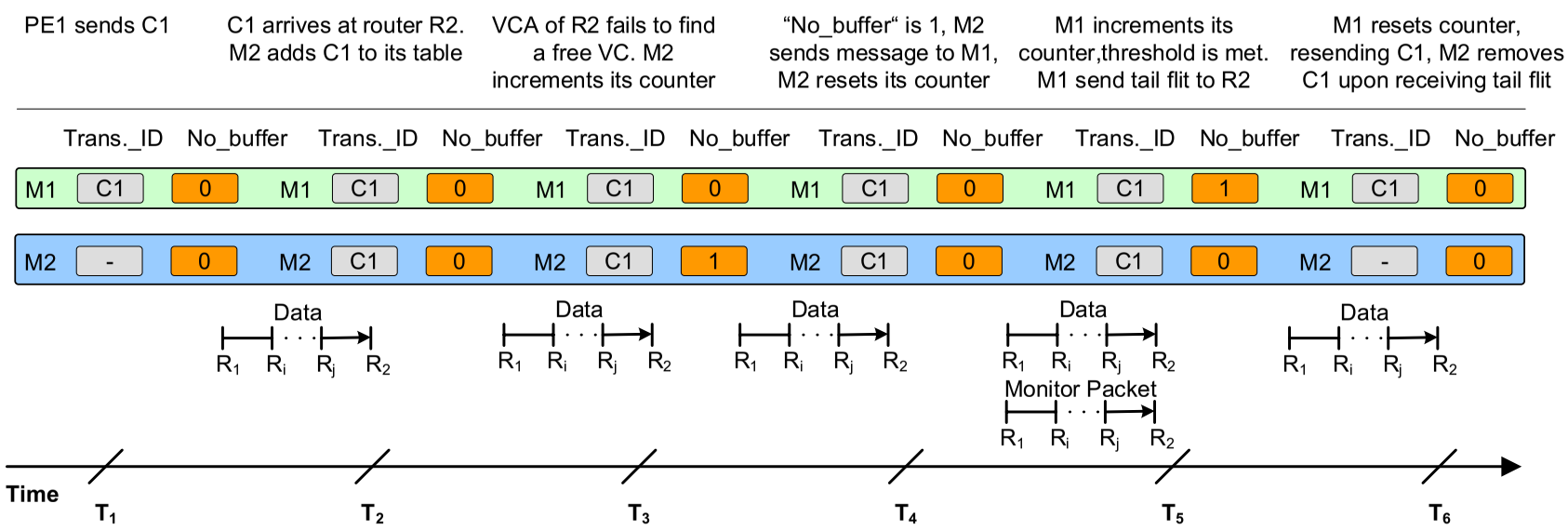


Figure 7.7: Runtime observability capabilities of the ROAdNoC infrastructure

T_4: For the *no-buffer* counter the threshold is one. That is, one *no-buffer-event* is enough to invoke a monitoring packet from the monitoring component. M_2 informs the sender of the transaction through its *NI*; in effect sending a monitoring packet to the monitoring component M_1 at time T_4. At the same time, M_2 resets its *no-buffer* counter for transaction C_1.

T_5: Once the monitoring packet arrives at M_1 it increments its own counter for C_1 and, since the threshold is met, it informs its *NI* to close the current transaction C_1. This is done by simply sending a *tail flit*.

T_6: M_1 has already reset its *no-buffer* counter and the *NI* is in the process of re-sending its data. The *tail flit* arrives at R_2 causing M_2 to remove C_1 from its transaction table.

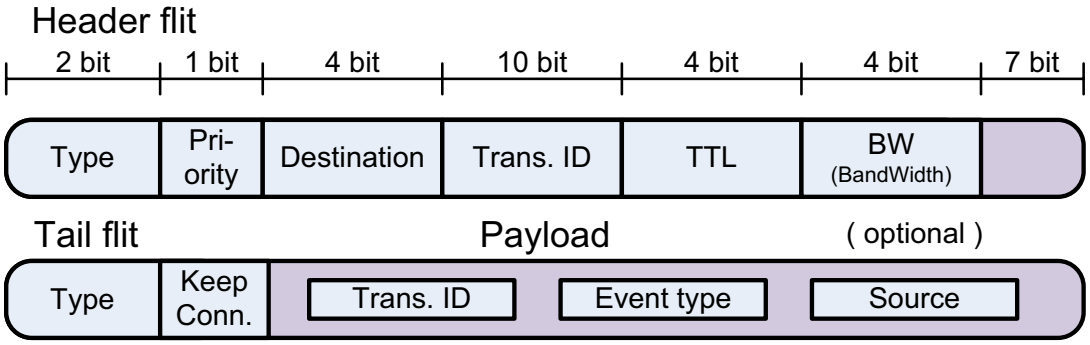


Figure 7.8: Flit composition of a monitoring packet

7.2.4 Monitoring Related Traffic

The monitoring component is situated partially between the router and the *NI* (see Figure 7.6). It is therefore able to interact with the *NI* to send its own packets over the regular communication network. This means, however, that the monitoring traffic must compete with regular transmissions for network resources. A monitoring packet must be of a higher priority than regular packets. This allows it to preempt another transaction in a *VCB*. When using only two priorities, one for regular traffic and one for monitoring traffic, a packet's priority requires a one-bit field in the *header flit*. For the rest of the fields, an exemplary monitoring packet in a 4 $\times$ 4 NoC is shown in Figure 7.8. It is two flits in size and is composed of a regular *header flit* for transmission plus a *tail flit* with payload data containing at least the triggering *transaction identification number* and the type of event. In addition, it may also contain information such as the source of the transaction which is used to provide the cluster agent with additional knowledge it may exploit during (re-)mapping. This is the minimum size since the routing algorithm implementation requires both a *header* and a *tail flit*; the *header flit* is required to create a route and the *tail flit* to close it. It also does not require a source field in the header since monitoring packets are not monitored themselves. The size of the monitoring packet can be calculated from the formulas given in Table 7.2.

Flit part	Size $n \times m$	Size 4×4 (2 priorities, 8 bandwidth (BW) slots)
Type	2 bit	2 bit
Priority	$\log_2$ (priorities)	1 bit
Destination	$\log_2 (n \times m)$	4 bit
TTL	$\log_2$ (n+m+2x†)	4 bit
BW	$\log_2$ (BW slots)	3 bit

†x = number of misroutes

Table 7.2: Flit size for a monitoring packet in an n x m NoC and in a 4 x 4 NoC

The frequency with which monitoring packets are generated is also very important to consider. These are event-based meaning they are only sent when an event occurs. Since events are only generated on faults during adaptation, there is no monitoring traffic when the network operates normally. Apart from the regular monitoring traffic, events can also eventually initiate (re-)mapping. (Re-)mapping comes with a high communication overhead. It is, however, also through observability that unnecessary (re-)mapping can be avoided compared to a scheme where any transaction fault automatically calls for (re-)mapping.

7.3 Adaptive Router Architecture for the AdNoC

The router of the *AdNoC* architecture is built on top of the router used for the QoS-supported NoC presented in Chapter 3. Figure 7.9 shows a simplified router diagram of the *AdNoC* architecture (only the parts that are interesting for understanding the *AdNoC* are highlighted and the other parts are kept similar to the Figure 3.4 in Chapter 3). The *Input decoder* is directly connected to the adjacent output port of all neighboring routers. In a *Normal-Full-Duplex-Link*, there are 5 inputs: East, West, North, South, and another from the PE. However, considering the presence of the *2X-Link* the *Input decoder* may have upto 9 inputs: 5 similar to the *Normal-Full-Duplex-Links* and new 4 from the *2X-Links*. All flits entering the router go first to the *Input decoder*. If the flit is of *header* type, the header information is read from the flit. The decoder determines the *transaction identification number*, *required bandwidth slots*, *the packet destination*, *TTL*, and *priority* of the flit. If the flit is a different type, i.e *payload* or *tail flit*, only the *transaction identification number* is required. As the *transaction identification number* is not stored in the flits, it needs to be inferred from a previous *header flit*. To accomplish this, the *Input decoder* must be aware of the link arbitration. It needs to store all transactions entering from each neighboring routers and performs a reverse arbitration to attain the *transaction identification number*.

The decoded information is sent to the *wXY-routing* inside the router to determine the route of the transaction. Apart from choosing the direction in which to route the incoming transaction, this part of the router is also responsible for the *VCB* assignment. For each new transaction, a free *VCB* is assigned. The state of all of the *VCBs* in the router are stored in a *LUT*. Once

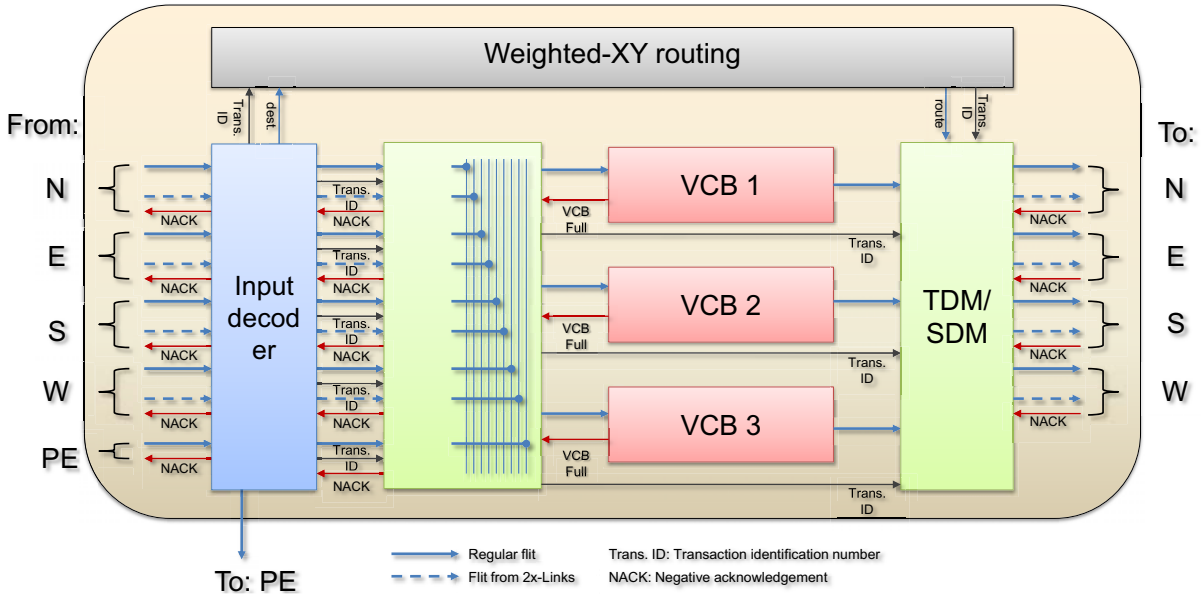


Figure 7.9: An overview of the AdNoC architecture-level part in each router

a transaction has been completed, it is removed from the *LUT*, thus freeing the *VCB* for new transactions. Along with the *VCB* index the assigned bandwidth slots are also stored. These are determined by keeping track of the available slots in four binary arrays (one for each output direction). New transactions requesting n bandwidth slots are assigned the next n slots flagged as available which are then marked as unavailable. Similarly, upon completion of a transaction the reserved slots are again marked as available. Therefore, this implements the *bounded-arbitration-algorithm* [59] proposed in the scope of this thesis.

The *VCA* is responsible for sending flits from the *Input decoder* to the appropriate free *VCB*. To do this, it must first request the index of the right *VCB* from the buffer assignment upon arrival of a *header flit*. This is done by sending the *transaction identification number* to the buffer assignment part which then informs the *VCA* of the appropriate *VCB* index (see Figure 3.4 in Chapter 3 for more details). The *VCA* then sends the flit to the corresponding buffer through a crossbar. The *transaction identification number* and the index are stored locally in the *VCA* allowing the following data flits to be arbitrated without the need to perform routing.

Since, there is a central pool of *VCBs*, output arbitration needs to be done both temporally (*Time Division Multiplexing*, TDM) as well as spatially (*Space Division Multiplexing*, SDM). This may lead to complications as slot arbitration cannot be performed in parallel for multiple transactions, i.e. one *VCB* containing a transaction being routed towards North, another *VCB* containing a transaction being routed towards South, and a third towards South: all transaction are assigned n bandwidth slots. Arbitration of two of these transactions cannot be done without considering the destinations of all other transactions. Therefore, this is solved by determining the slot arbitration in the position where buffer assignment is done. This allows arbitration to be performed in the two domains sequentially. First the *TDM* is informed of the slots assigned to a

particular *VCB* from the buffer assignment part. The *TDM* then retrieves a flit from the *VCB*, if it is assigned the current bandwidth slot then passes it to the *SDM*. The *SDM* retrieves the output direction from the *wXY-routing* for the current *VCB* index. All the values retrieved are stored locally to reduce the overhead for subsequent flits. Due to the bi-directional communication characteristic of the proposed *2X-Link*, the link control part (*TDM/SDM*) also has 9 outputs.

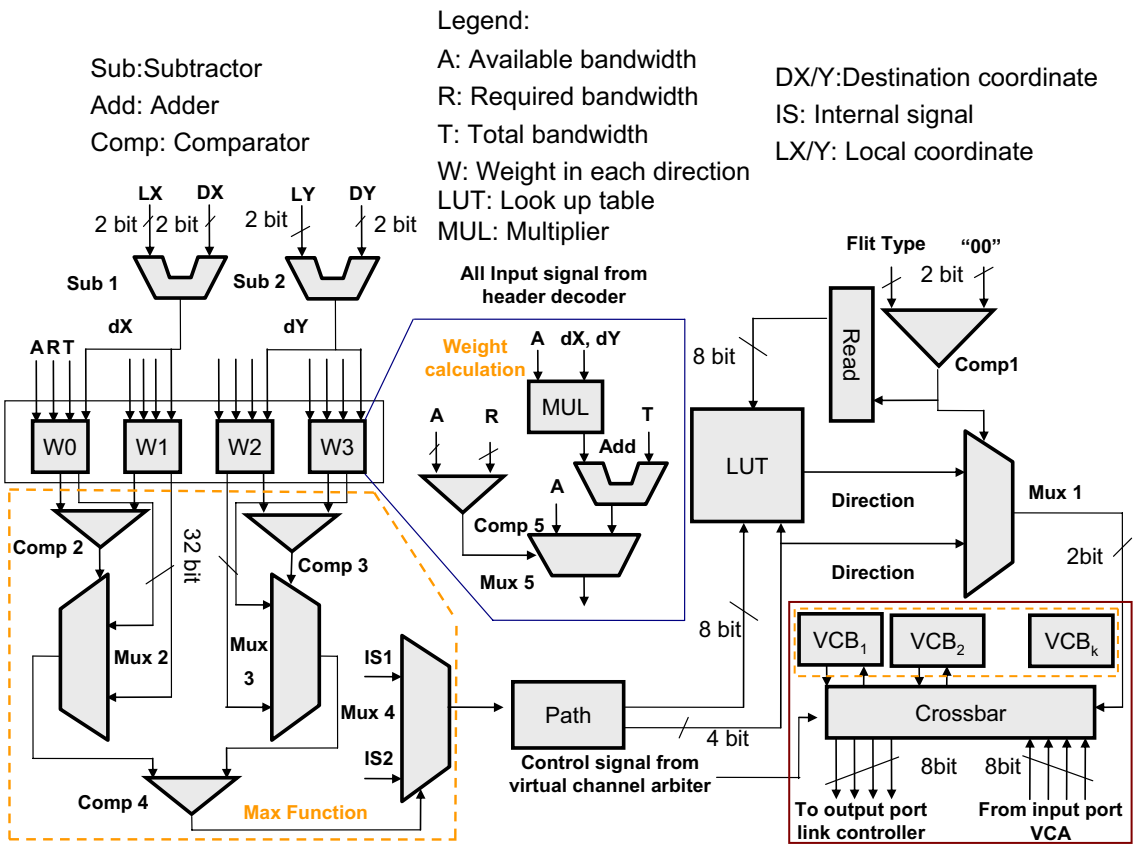


Figure 7.10: Adaptive hardware for the output port of a router

7.4 Hardware Implementation of the AdNoC Architecture

The circuit implementation of different parts of the router for the proposed *AdNoC* architecture that are not part of the router used for the QoS-supported NoC in Chapter 3 are explained in this section. The area requirements of all these parts after synthesizing it for Virtex II FPGA [177] are also explained in the scope of this section. The adaptive parts of the router implementation for the *AdNoC* architecture are as follows: (1) the *wXY-routing* algorithm, (2) the on-demand *VCB* assignment, (3) the *2X-Link* realization, and (4) the *ROAdNoC* infrastructure (the monitoring component). Each part is explained separately with their own circuit implementation

then they are synthesized for downloading onto the Virtex II FPGA [177] and finally the performance and the hardware requirement for each part is explained and the overhead is justified accordingly.

Hardware block	Area requirement [ASIC/FPGA]
Routing logic	2877 gates / 129 slices
VCB_crossbar	1680 gates / 122 slices
Look-up-table	(4 × # of transactions) bits / BlockRAM
VCB size	(flit size × (2m+n)) bits / BlockRAM (m and n are the dimensions of the NoC)
Pointer-table	(# of VCBs × 4) bits / BlockRAM
Registers	(4 × 4) bits / registers

Table 7.3: Hardware requirement for the adaptive scheme

7.4.1 The wXY-routing and the On-demand Buffer Assignment Components

The implementation of the *wXY-routing* and the on-demand *VCB* assignment components of the router for the *AdNoC* architecture are illustrated in Figure 7.10. The route allocation part either decides based on the *LUT* or by calculating the type of the flit. This part is separated by the comparator, *Comp1* and the multiplexer, *Mux1*. If the incoming flit is a *header flit* then the weight of a possible route is calculated. It is shown as W_n. The maximum weight is evaluated using *Comp2*, *Comp3*, *Comp4*, *Mux2*, *Mux3*, and *Mux4*. The direction of the transaction is also stored in the *LUT*. All control bits of the route allocation stem from the header decoder of the router. Each weight calculation is evaluated using a *Multiplier*, an *Adder*, a *Multiplexer*, and a *Comparator*. In the right-bottom part of the Figure 7.10, a crossbar to select an appropriate *VCB* is implemented.

The hardware overhead for the *wXY-routing* and the on-demand *VCB* assignment components is given in Table 7.3. The crossbar to implement the on-demand buffer assignment and the small *LUT* to keep the *VCB* pointers contribute to the area. The implementation of the *VCB* crossbar for a flit size of 8 bits having 4 *VCBs* shared by the 4 output ports costs 122 slices in an FPGA.

The implementation of the adaptive routing algorithm *wXY-routing* also contributes to the overall area. In fact, it takes 129 slices for each router. The entire area overhead may be traded-off to the achievement of the flexibility during runtime. The flexibility to select an available route and the on-demand buffer assignment to that route ensures the QoS (guaranteed bandwidth) for the *AdNoC*.

7.4.2 Configurable Links Components

In order to realize the bi-directional communication on a single link two *tri-state logic gates* are used. *Tri-state logic gates* have three states of the output: *high (H)*, *low (L)*, and *high-impedance (Z)*. The high-impedance state play no role in the logic, which remains strictly binary. These devices have previously been used on buses to allow multiple sources to send data in parallel. A

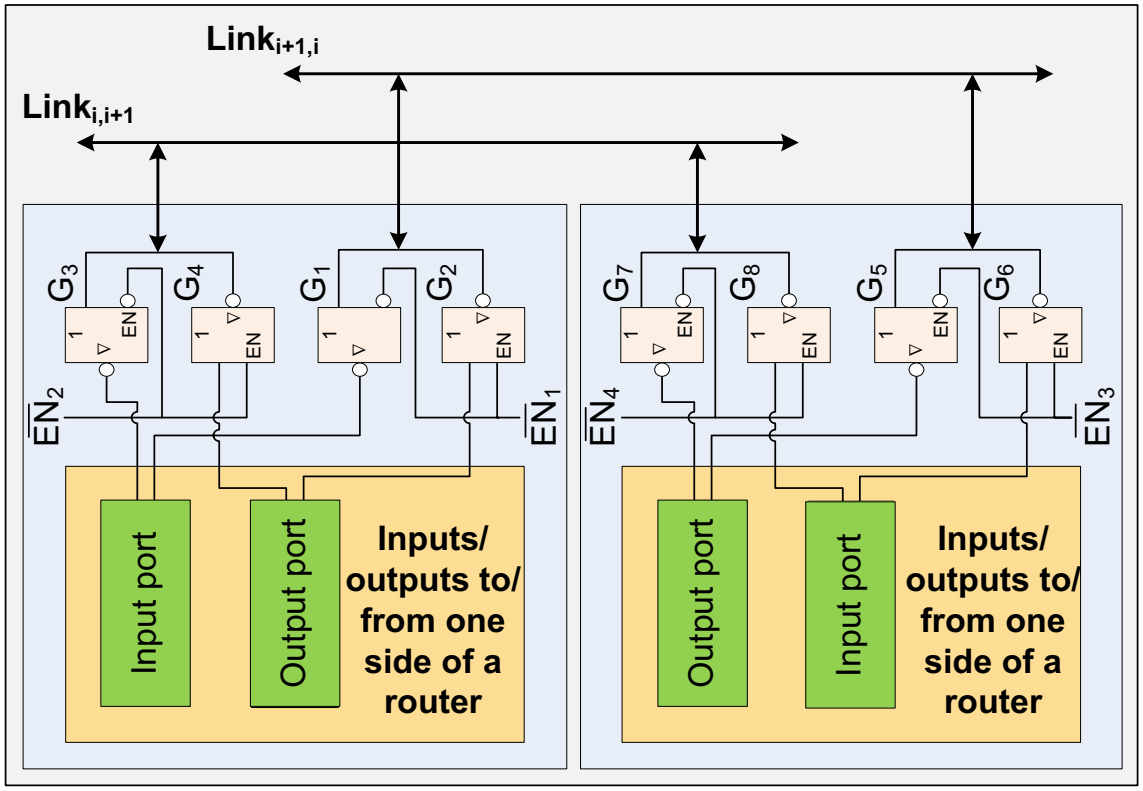


Figure 7.11: Hardware implementation of the 2X-Link

group of *tri-state logic gates* driving a line with a suitable control circuit is basically equivalent to a multiplexer [163].

The architecture of the bi-directional link control using *half-duplex links* for the proposed *2X-Link* is presented in Figure 7.11. Two tri-state logic gates, e.g. G_1 and G_2 work as a group. They are controlled by the same control signal $E\bar{N}_1$. When the value of $E\bar{N}_1$ is *0* then the tri-state gate G_1 is active and G_2 is stalled. Therefore, data may be transmitted from the neighboring *VCBs* through $Link_{i,i+1}$ to the component *Input decoder*. On the other hand, if the value of $E\bar{N}_1$ is *1* then the tri-state gate G_1 is stalled and data may be sent from the *VCB* to the adjacent router's *Input decoder* via $Link_{i,i+1}$. Therefore, the half-duplex communication on one link is realized. In this architecture, the signal $E\bar{N}_1$ and $E\bar{N}_3$ control the data transmission via $Link_{i,i+1}$ while signal $E\bar{N}_2$ and $E\bar{N}_4$ control the data transmission via $Link_{i+1,i}$. They are assigned different values to avoid conflicts on the link and an unpredictable situation.

The extra hardware that is needed to implement either a single *2X-Link* or a single *TDD-Link* on top of the *AdNoC* router is 74 slices. Each *TDD-Links* additionally requires one control unit to control the data transmission direction and the time slot allocation. The extra hardware needed to implement this control unit is another 49 slices. Figure 7.12 shows the additional slice requirements for the *TDD-Links* compared to the *2X-Links*. With the increasing size of the NoC, a NoC with the *TDD-Links* requires more slices compared to the NoC with the *2X-Links*, e.g. in a 7×7 NoC, the *TDD-Links* require *4116* more slices than the *2X-Links*.

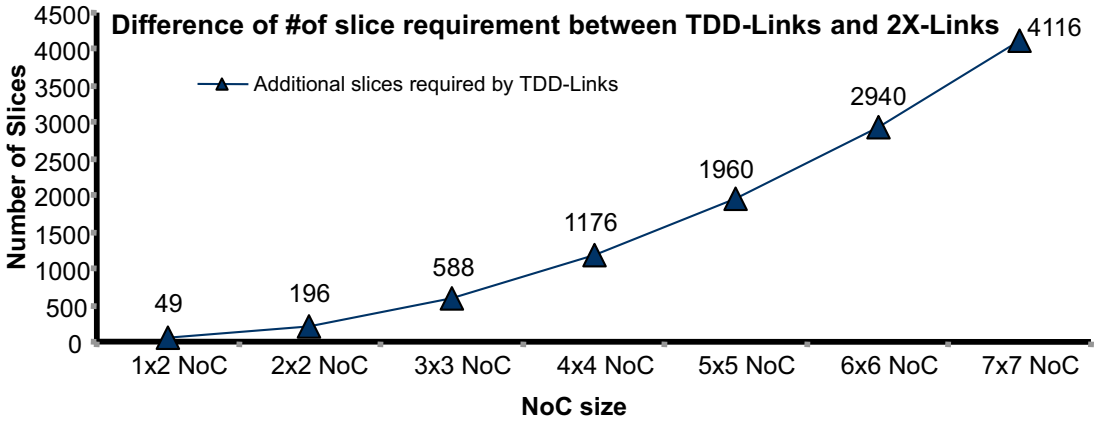


Figure 7.12: Overhead of the TDD-Links compared to the 2X-Links

7.4.3 Monitoring Component

The monitoring component of the *ROAdNoC* infrastructure is event-based and the event-counter values are stored in a *LUT*. One entry in the *LUT* ties the *transaction identification number* to the source of the transaction and to its associated counters. Given the values for a 4×4 NoC (see Table 7.1 and Table 7.2), one entry in the *LUT* is thus 8 bits for the transaction source plus 4 bits for each of the four counters: 24 bits in total. If there are n *VCBs* and k inputs, then $(n + k)$ entries are needed. In other words, more entries are needed than *VCBs* since transactions may cause events even (especially) if all *VCBs* are already occupied. In general, k is equal to the amount of inputs. For example, if a router has 4 *VCBs* and 5 inputs then 9 entries are required.

Transaction numbers are added to the *LUT* when a *header flit* of the first packet for the transaction arrives at a router (Figure 7.13(b)). The arrival causes the *set and configure flags* to be triggered, initiating a write to the *LUT* and setting the counter values to zero. Similarly, a *tail flit* arriving at the router causes the *configure flag* to be triggered while the *set flag* remains zero. This causes the monitoring component to remove the transaction from the *LUT*. Once an event occurs it initiates a *read* from the *LUT* using the event *transaction identification number* (Figure 7.13(a)). It then compares the counter value returned from the *LUT* of the event type corresponding to the event type that arrived. If the counter value has reached its threshold, the *NI* part of the monitoring component is informed and the counter value is set to zero in the *LUT*. If not, the incremented value is written to the *LUT*.

The *NI* part of the monitoring component, upon receiving an event, first compares the transaction source with its own address. If it is not the sender then a packet is sent to the remote sender. Otherwise, the transaction send count is examined to find out if there are previous send attempts by comparing the *transaction-send* counter stored in a register with a given re-send threshold. Based upon this, the *NI* is either told to re-send the packet if the *transaction-send* threshold has not been met or a (re-)mapping if it has. If the packet is re-sent, the *transaction-send* counter is incremented. A (re-)mapping causes the counter to be reset to zero.

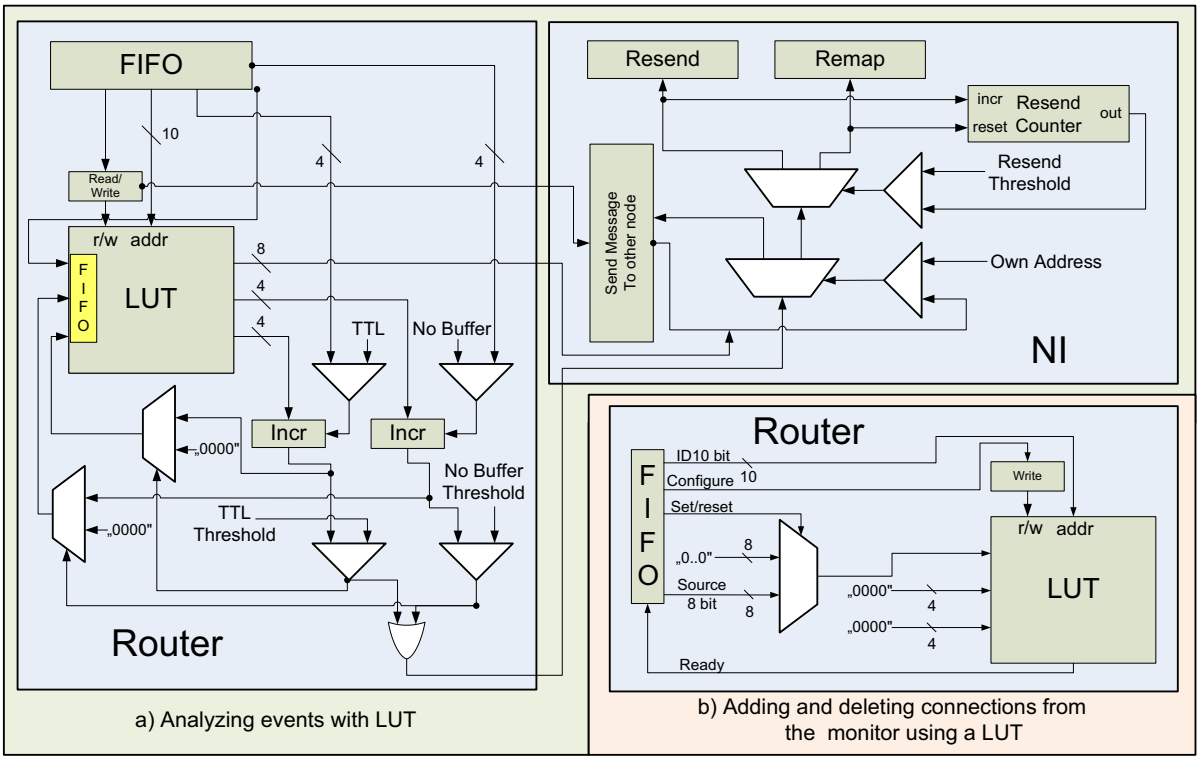


Figure 7.13: Hardware for adding and analyzing monitoring events

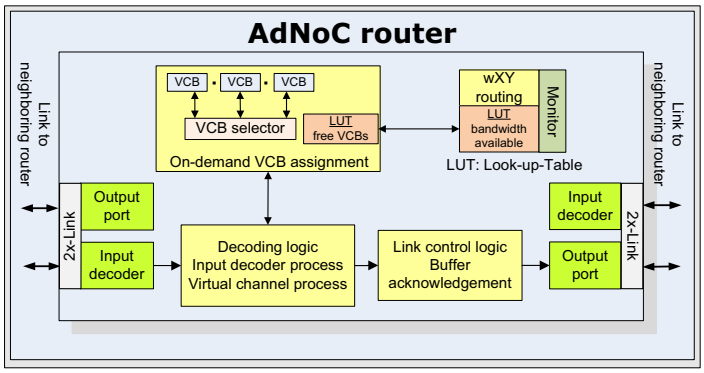
Each router has 5 input ports resulting in 5 possible simultaneous transactions needing to be set up in a monitoring component. Also, using a *LUT* entails a few cycles delay in which new transactions/events cannot be processed. To allow each tile to function using only one monitoring component, FIFOs are added to buffer its inputs.

7.4.4 Hardware Evaluation

The implementation on a XILINX Virtex2 XC2V-6000 FPGA is shown in Figure 8.7(a). The *AdNoC* architecture comes at a hardware overhead, roughly 41% area increase compared to a design-time parameterized router not considering the *VCB* reduction due to resource multiplexing but 59% less area compared to a complete duplication of the basic QoS-supported NoC presented in Chapter 3 for the purpose of reliable system design [11] using the *AdNoC* architecture.

The entire area overhead may be counterbalanced by the level of flexibility that is achieved, the scope of assigning the hardware components where some parts are already faulty to other functional units, finding an alternative route to avoid a faulty path, and a larger number of successful transactions (for details on these results see Chapter 9).

The reusability of the *VCBs* also gives a higher chance to lower the total number of *VCBs*, which may balance a fraction of the overall overhead introduced by the *wXY-routing*, the on-demand *VCB* assignment crossbar, the *2X-Link*, and the monitoring hardware components, though this is



(a) Router architecture of the AdNoC which is synthesized for Virtex II FPGA board

Adaptivecomponents	Hardware requirements [FPGA]
Link reversal hardware	74 slices
wXY- routing algorithm	129 slices
VCB selector	122 slices
Monitoring hardware	46 slices

(c) Additional hardware needed to achieve adaptivity in a 4-input/4-output AdNoC

Figure (a) shows the block diagram of the AdNoC router and table in (b) shows the area requirements for a basic QoS-supported router presented in Chapter 3 and the table in (c) shows the extra hardware needed for the AdNoC built on top of the basic QoS-supported router

Simplerouter components	No of slices
Virtual channel arbiter	87
Crossbar	348
Header decoderprocess	39
Virtual channelprocess	17
Bufferack	23
Virtual channel	19
Header decoder	49
Link control	12
Output	117
Router (4 inout 4 output)	1085

(b) Hardware requirements of a 4-input/4-output router implementation (without adaptive part)

Figure 7.14: Details of the hardware prototype of the proposed AdNoC architecture

not the main motivation of this thesis. Simulations show when using the scheme of *wXY-routing* and the on-demand buffer assignment strategy one can experience a 42% reduction of the *VCBs* compared to a fixed buffer assignment and a fixed routing scheme. For the buffer savings, some simple assumptions are taken (in Table 7.3 the formula for the *VCB* size is given): each *VCB* is a FIFO of depth 20 and width of the flit size. Therefore, it requires 160 bits to implement each *VCB*. Therefore, the area overhead may be further justified by the degree of freedom to achieve higher performance in the *architecture-level* during runtime adaptation together with considering the reliability issues and the possibilities to reduce the number of *VCBs* at design time due to the reusability compared to a fixed assignment approach.

7.5 Conclusion

In summary, the *architecture-level* part of the proposed *AdNoC* architecture is detailed in this chapter. Different novel techniques, i.e. sharing the *VCBs* among different output ports, changing the routing at runtime, and changing the supported bandwidth between adjacent links using configurable links at runtime are explained in details. A runtime observability infrastructure that is employed for the successful *architecture-level* adaptation is also described here. The hardware overhead of the *AdNoC* router is justified by the advantages of the runtime adaptation and the possible reduction of the *VCBs* which takes the premier share of the router implementation. Several experimental results based on *AdNoC* simulation of the proposed *architecture-level* are presented in Chapter 9 in details.

Chapter 8

Simulation and Hardware Prototyping Environment

Several state-of-the-art tools, i.e. OMNeT++ [125] and some inhouse simulation tools which are based on SystemC [161] and C++ programming languages (compiled with GNU gcc [63]) have been used to simulate the proposed adaptive system-on-chip communication architecture. The inhouse simulation tools have been developed during the scope of the dissertation work. VirtexII FPGA has been targeted for the hardware prototyping and therefore, existing prototyping boards CHIPit [36] and HW-AFX-FF1152-200 [177] have been used. The hardware has been described using the VHDL [171] hardware description language and simulated using ModelSim simulator from Mentor graphic [108] and synthesized using Xilinx tools from Xilinx, Inc [177]. In the following, a short description of the tools that are used in this thesis are discussed.

8.1 Simulation Environment

In this section, the simulation tools which are used in the scope of this dissertation are explained.

8.1.1 OMNeT++ Simulator

OMNeT++ [125] which is used to simulate the proposed adaptive system-on-chip communication architecture is a component-based modular network simulation environment. It can be used to build simulations for numerous types of communication networks. It can be executed using either a GUI front-end or as a command-line program. The GUI allows the visualization of network behavior and traffic flows whereas the command-line interface allows faster simulation for the collection of statistics. An OMNeT++ simulation environment is made up of three types of files. These files separately describe the architecture and behavior of the simulation environment and the configuration of individual simulation runs.

Architecture: The simulation architecture is described in *".ned"* files (an example is shown in Figure 8.1). These files define a hierarchy of modules. A module can either be a simple module or a compound module. For each module inputs and outputs are given.

```
module Router
        parameters:
                myid: numeric,
                r_x_dim: numeric,
                r_y_dim: numeric,
                r_processing_delay;
        gates:
                in: in_n, in_ack_n;
                out: out_n, out_ack_n;
        submodules:
                n: OutPort;
                        parameters: routerid = myid,
                                    op_x_dim = r_x_dim,
                                    op_y_dim = r_y_dim;
                r: RouteInput;
                        parameters: routerid = myid,
                                    ri_x_dim = r_x_dim,
                                    ri_y_dim = r_y_dim,
                            processing_delay = r_processing_delay;
transactions:
                in_n --> r.in_n;
        r.out_n --> n.in;
        n.out --> out_n;
        r.ack_n --> out_ack_n;
        in_ack_n --> n.in_ack;
endmodule
```

Figure 8.1: Exemplary router module: ".ned" file with one output port

Simple modules are processing entities whose behavior needs to be implemented. Compound modules are made up of one or more modules and the inter-connections between these modules. For instance, a compound module consisting of one simple module would contain connections linking the relevant inputs of the compound module to those of the simple module as well as connecting the outputs. *"ned"* files also include parameter definitions which may be supplied to the configuration files and accessed by the behavioral implementation.

Behavior: The behavior is implemented in C++. Each simple module has its own implementation with one global header file (*noc.h*) supplying global parameters. The OMNeT++ framework supplies objects such as messages (a communication entity which can be used to implement either packets or flits) and queues (message FIFOs). It also supplies all necessary functions such as inserting and removing messages to and from the queues and sending messages to an output port. Messages entering a simple module from an input in-

voke an event handling routine within the C++ file. The core functionality of the module is implemented in this routine.

This framework also provides mechanisms for gathering and recording statistics. The statistics can be either a scalar type or a vector type which are written to scalar/vector files given in the configuration file. Scalar type values are simply counters which keep track of certain events and are written at the end of the simulation run. For example, router elements keep track of the number of discarded packets and this value is stored as packet-loss. The vector types are used to follow changing values over time. In particular these are used to keep track of bandwidth usage. In this simulation, they are collected on the number of bandwidth slots used at each output port. Vector statistics can also be visualized when using the graphical front-end (see Figure 8.3).

Configuration: Simulation runs are configured in *".ini"* files (an example is shown in Figure 8.2). In these files, parameter values are set for both the global simulation (i.e. simulation length) as well as for the parameters defined in the *".ned"* files. This allows multiple simulations to be run for a given environment: each with different simulation characteristics such as the traffic patterns, routing algorithms, NoC dimensions, etc.

```
[General]
network = noc
sim-time-limit = 4000us

[Parameters]
*.x_dim = 4
*.y_dim = 4
*.processing_delay = 0.000000008
# routing algorithm ("wXY-routing" or "XY-routing")
*.*.*.algorithm = "wXY-routing"
*.*.inter_packet_delay = normal(0.000001, 0.0000001)
*.*.dest_x = intuniform(0,3)
*.*.dest_y = intuniform(0,3)
*.*.flits = 200
*.*.bw = 1
```

Figure 8.2: Exemplary configuration ".ini" file

8.1.1.1 Implementation

The implementation of the adaptive system-on-chip communication architecture for the simulation is made up of the following modules:

- The *processing element* is a simple module which acts as a producer/consumer of the network traffic. The traffic distribution, bandwidth slots requirements, and destination coordinates are provided as parameters.

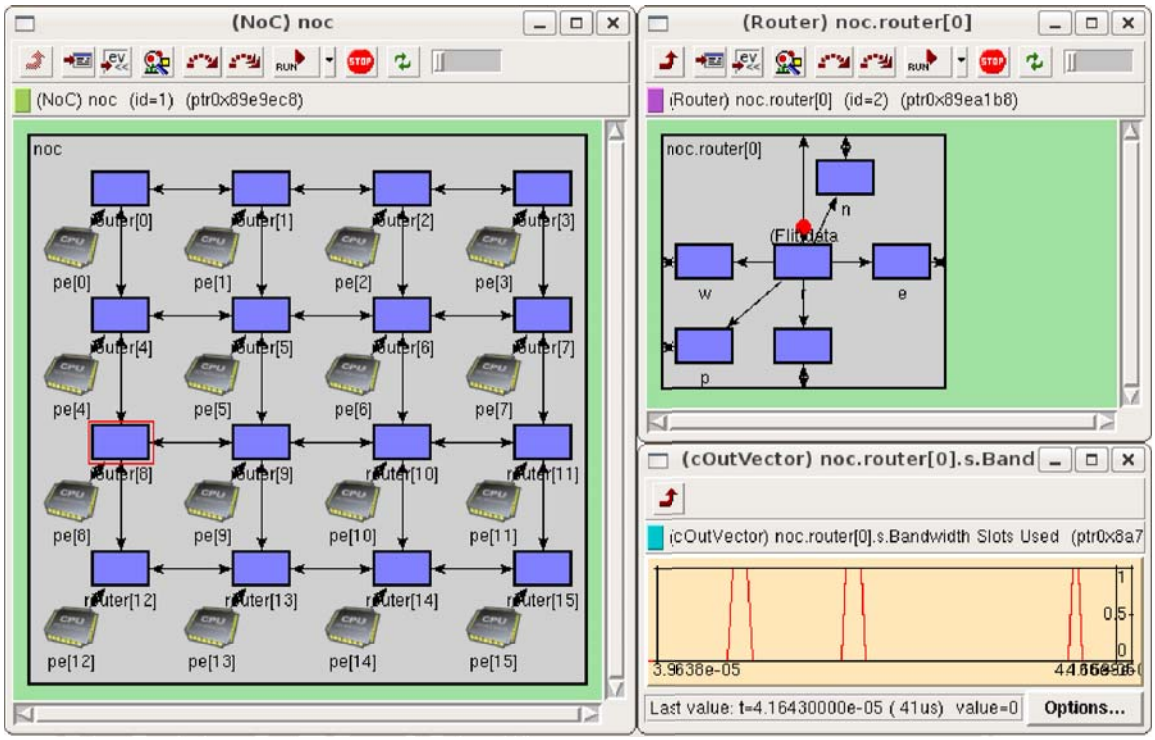


Figure 8.3: Graphical OMNeT++ NoC simulation environment

- The *routing unit* is responsible for choosing the route that flits will traverse. All flits entering a router are sent directly to the routing unit. This unit then sends the flit to the output port of the direction chosen by the routing algorithm. Both the *wXY-routing* and the *XY-routing* algorithms are implemented and may be configured from the configuration file. Since the simulator does not simulate computation, there is no direct processing delay. Therefore, to make the simulation accurate, there is an added sending delay of flits to the output ports of the routing unit. This delay simulates the processing delay and is specified in the configuration file.

- The *output port* is a simple module consisting of a series of queues. These queues are objects provided by the OMNeT++ framework to hold OMNeT++ messages. For the simulation, these messages represent flits and include all necessary fields (flit type, destination, source, transaction ID, required bandwidth slots, and priority). The objective of the module is the delegation of queues to incoming transactions and the arbitration of flits to the corresponding router output and its connected link.

- The *router* is a compound module and consists of a routing unit and 5 output ports – one to each neighboring router and one to the *Processing Element* (PE) connected with the router.

- The *Networks-on-Chip* (NoC) is a compound module consisting of several routers and their respective PEs. These are organized in a 2D Mesh whose dimensions are given as a parameter.

An example simulation setup is shown in Figure 8.3 using the graphical output of the OMNeT++ simulator. Here the simulator is running a 4×4 NoC. The figure also shows the internal connections of one of the routers (top right).

Since a router has 5 output ports, the centralized buffer pool used for the on-demand *Virtual Channel Buffer* (VCB) assignment is not implemented directly in the architecture. However, queue usage is limited by a per router global counter. This counter is decremented each time any of the routers output ports' queues are occupied and incremented upon their release. If the counter reaches zero, no further queues are used in any of the output ports. This indirectly implements the centralized buffer pool needed for the on-demand *VCB* assignment. The value of this counter is not provided as a parameter and must be changed in the header file.

Link arbitration to provide *transaction-level* guarantee is performed by the output ports using the *bounded-arbitration-algorithm* presented in Chapter 4. Each output port can send one flit in every cycle. Sending is rotated through the non-empty message queues giving each queue as many cycles as bandwidth slots allocated to the transaction in each queue. Once all slots are used up the procedure is repeated considering the total number of slots that is specified in the header file. There are never more bandwidth slots allocated then the total available slots. This is insured through the *wXY-routing* algorithm and by discarding transactions those are exceeding the limit at the output port when using the *XY-routing* algorithm.

8.1.1.2 Traffic Model Implementation

The OMNeT++ framework directly provides probabilistic distributions for the network traffic. Therefore, these may be used to define a wide range of traffic characteristics in the configuration files.

Uniform Traffic: A simple traffic scenario is uniform traffic. Here all PEs communicate with all other PEs. Uniform traffic is not very realistic in a NoC scenario. It does however provide a worst-case scenario. Although this kind of traffic distribution is unlikely in a specific application, it reveals some of the rudimentary aspects of the used routing algorithms. It shows the effect of the higher bandwidth traffic on a NoC environment. For simplicity, all transactions are of the same class specification. Therefore, they all have the same packet length in flits and the same number of required bandwidth slots which they may allocate. In the simulation, this is implemented by setting a random destination as default in the simulation configuration file. An exemplary specification is given below:

```
*.*.dest_x = intuniform(0,3)
*.*.dest_y = intuniform(0,3)
```

This parameter is read by the behavioral implementation for each new transaction and provides a different random value between 0 and 3 at each time, i.e. for a 4×4 NoC.

Task-based Traffic: Task based traffic uses traffic from real representative applications suitable for NoC design such as real-time multimedia applications shown in Chapter 9. Com-

munication volume among the computing modules and the characteristics of the application are collected during application exploration. This information is usually given as a task graph where the nodes represent individual tasks and the edges represent communication links weighted by the required communication bandwidth.

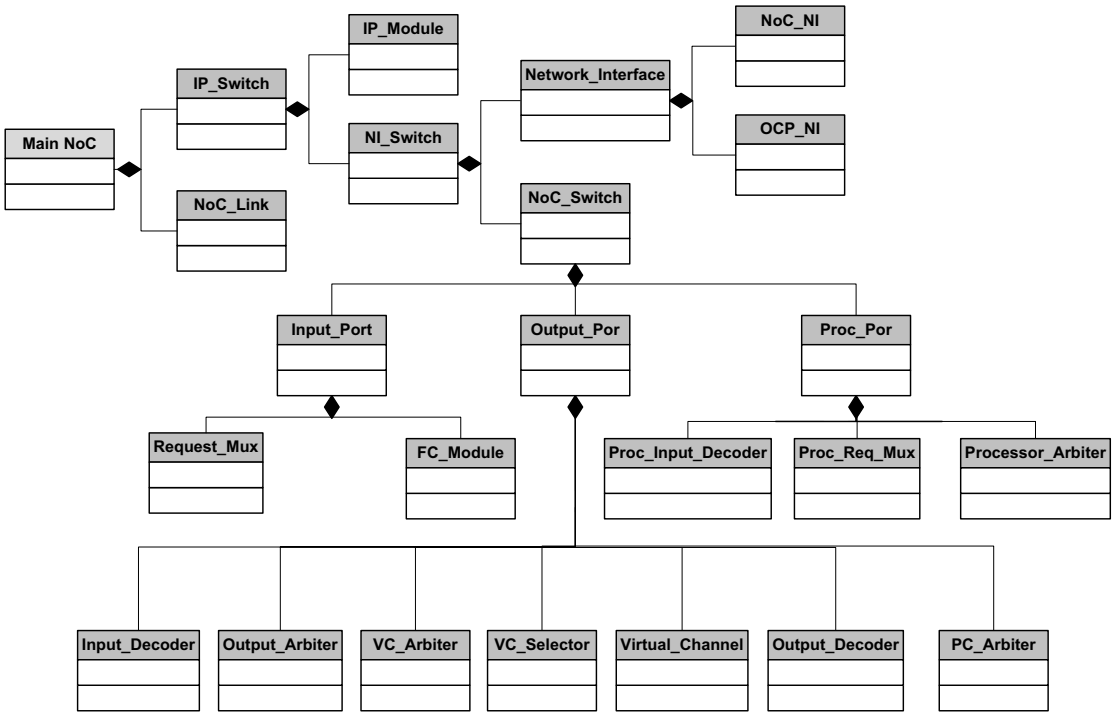


Figure 8.4: The class structure of the configurable NoC (a library structure)

The simulator allows two transactions originating at each PE. Therefore, each PE can send data to two separate destinations. However, each PE only has one packet queue. Packets to different destinations are inserted sequentially. For task based traffic the destination of each transaction must by provided for each PE. This is done as follows:

```
*.pe[1].dests = 2
*.pe[1].dest_x = 1
*.pe[1].dest_y = 2
*.pe[1].dest_x2 = 0
*.pe[1].dest_y2 = 1
*.pe[2].dests = 1
*.pe[2].dest_x = 2
*.pe[2].dest_y = 2
```

Here, the number of different destinations is specified for each PE. The x, y coordinates (in a 2D Mesh topology) for each destination are then specified. The originating PE for each transaction is invariant. Different mappings must be calculated before the simulation starts with possibly a different tool. Therefore, a separate simulation run is needed for each mapping and each with different traffic configurations.

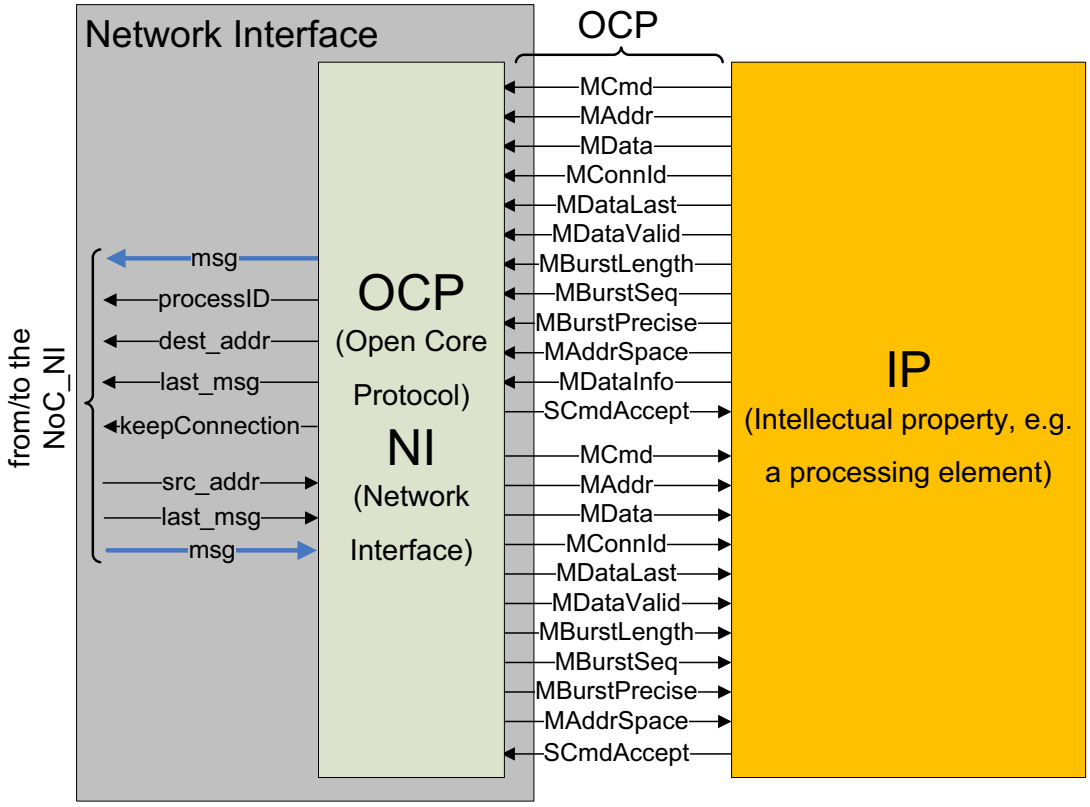


Figure 8.5: An overview of the OCP implementation with its signals

8.1.2 SystemC-based NoC Simulator

Besides the OMNeT++ based NoC simulator a configurable NoC simulation tool for a cycle-accurate simulation at *system-level* is also implemented using SystemC in the scope of this thesis work [60]. Configurability of the NoC is achieved by a library-based approach. *Generic Modules* are used as template components to build topology-dependent parts. Most of the class names include the postfix *"generic"* as they can be configured as concrete modules. Figure 8.4 shows an overview of the class structure of the configurable NoC implementation (a library structure). The tiles are represented by the class `IP_Switch` (see Figure 8.4). Links are implemented as `NoC_Link`. A tile is composed of a switch with a NI (`NI_Switch`) and a specific *Intellectual Property* (IP) (`IP_Module`). The NI switch consists of a router (`NoC_Switch`) and a NI (`Network_Interface`). A router instantiates its ports which are the input ports (`Input_Port`), the output ports (`Output_Port`), and the processor port (`Proc_Port`). A NI holds the inner modules: a NoC NI (`NoC_NI`) and an IP-dependent NI (`OCP_NI`).

Two modules are located in an input port as can be seen on Figure 8.4: a request multiplexer (`request_mux`) and a flow control module (`FC_Module`). They are created, initialized, and connected in the constructor of the input port. The functionality of an input port module is low as it just holds and connects its inner components. But the implementation of the output port modules needs special attention and needs to be configurable. There-

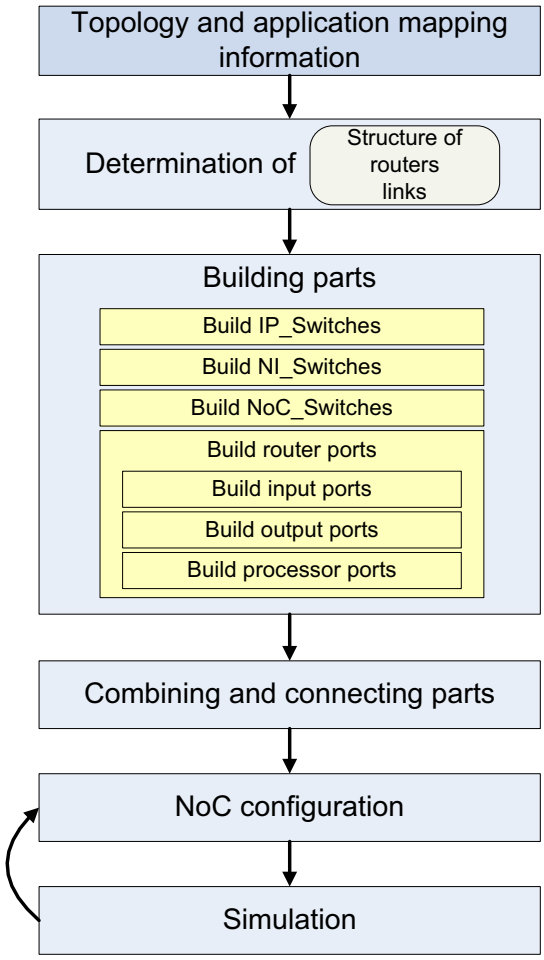


Figure 8.6: Design flow of the NoC architectures having different dimensions

fore, the number of *Virtual Channels* (VCs) is designed to be configurable by the constant `CONFIG_VC_NO`. Every module is instantiated and connected in the constructor of an output port. The components of the output port are shown in Figure 8.4: the Input Decoder (`inp_dec`), the Output Arbiter (`out_arbit`), the Virtual Channel Arbiter (`vc_arbit`), the Virtual Channel Selector (`vc_select`), the Virtual Channels (`vir_ch[CONFIG_VC_NO]`), the Output Decoder (`out_dec`), and the Physical Channel Arbiter (`pc_arbit`). The processor port module (`Proc_Port`) connects the router with the attached NI. It consists of the processor input decoder (`Proc_Input_Decoder`), the processor arbiter (`Processor_Arbiter`), and the processor request multiplexer (`Proc_Req_Mux`) (see Figure 8.4). They are instantiated and connected in the constructor `SC_CTOR(proc_port)`. All parts of the NoC implementation are clocked.

An inter-core protocol is used for the communication with other IPs, i.e. a PE. It utilizes a standardized *Open Core Protocol* (OCP) [121] interface to communicate with the attached NI. The implementation of the OCP with its necessary signals and methods is shown in Figure 8.5. This implementation is used by the `OCP_NI` module presented in Figure 8.4.

8.1.2.1 Configuration for the Application-specific NoC

The presented simulation tool is used to build application-specific NoCs and enhancing the tool for the adaptive system-on-chip communication architecture is on the future working list. The application-specific NoC is customizable therefore, the configuration of the network parameters is done utilizing the presented library-based approach. Most of the network parameters are configurable through configuration files which include definitions of constants. Similar to the *Xpipes Lite* [157] architecture, replacements of constants in the source code are done with the preprocessor commands. The *Xpipes lite* [157] architecture utilizes this strategy to build synthesizable NoCs in SystemC. Global constants for the whole NoC may be found in the `noc_configuration.h` header file. The file `switch_configuration.h` includes configuration constants for a single router. Configuration of the NI is set in `ni_configuration.h`. OCP-dependent parameterization is done in `ocp_configuration.h`. IP Modules are configured in `ip_configuration.h`. An example for setting the parameter flit size is as follows:

```
#define CONFIG_FLIT_SIZE 32
```

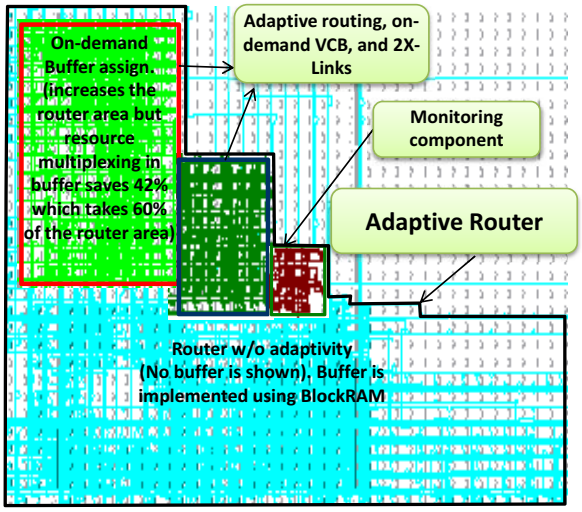
Here, the size of a flit and thus the channel-width of the NoC is set to 32 bits. The complete list of configuration constants are not explained in detail in the scope of this thesis. Most of the modules in a NoC depend on the topology of the network. Therefore, this work introduces *generic* types of modules which may be instantiated as concrete modules through scripts. The introduced components of a NoC in Figure 8.4 are generic modules. A *positional router* in the network has to be specific in such a way that its input and output direction have to be defined. Additionally, to save links for the direction bits the output directions of the neighboring routers are also important. These factors need to be considered when implementing a distinct NoC.

Scripts are written in shell scripting language or C++ programming language to enable the building of concrete modules. Here, concrete means that the input, output directions, and the output ports of the neighboring routers are fixed in the implementation of the modules. Figure 8.6 shows the process of creating an application-specific NoC for simulation purposes. From the topology, the router and link structure is derived. That determines the number of input and output ports of each router and the output ports of their neighboring routers. With that information the single parts can be built out of the library. The specific types can be created invoking the scripts. For a complete NoC implementation all the tiles have to be connected manually in the *main()* function of the simulation. Therefore, the tiles and the links are connected to a whole network in the *main()* function. Topology-independent configuration settings can be made by assigning distinct values to the NoC parameters located in the configuration files. Finally, a simulation of the NoC can be executed. A possible reconfiguration of parameters is denoted by the arrow back to the NoC configuration in the Figure 8.6.

8.1.3 Application Mapping Tool

An application mapping tool is implemented in this thesis based on C++ programming language. The mapping tool is kept configurable by plugging in several state-of-the-art application

mapping algorithms. The agent-based distributed application mapping (ADAM) presented in Chapter 6 is plugged onto this application mapping tool for the simulation purpose. Other mapping algorithms that are studied to design an application-specific NoC, e.g. the *ant colony optimization algorithm*, *genetic algorithm*, and *branch-and-bound* based application mapping algorithm presented in Chapter 4 are also plugged onto this tool. Details of the tool are not explained here.



(a) HW implementation with *Xilinx PlanAhead,* visualized with *FPGA Editor*

Component	Area
wXY-routting	122 slices
On-demand VCB	129 slices
2X-Links	74 slices
Monitoring component	46 slices

(b) Hardware requirement for AdNoC components

(c) HW implementation on *FPGA board*

Figure 8.7: Hardware prototyping of the NoC architectures

8.2 Hardware Prototyping

A configurable NoC library using the hardware description language VHDL is developed during the thesis work. Different hardware parts, their functionalities, and hardware result (in number of slices and look-up-tables) are already discussed in Chapter 7. Here, only the prototyping environment is presented. Hardware results are generated synthesizing these VHDL implementations. The VHDL implementation of the NoC architecture is simulated using the ModelSim simulator from Mentor graphic [108] and synthesized using Xilinx tools from Xilinx, Inc [177]. Different tools from *Xilinx ISE*, e.g. *FPGA Editor*, *XPower*, *CORE Generator*, *PlanAhead design analysis tool* are used for hardware analysis [177]. The prototype of the NoC architecture is evaluated on the prototyping boards the CHIPit [36] and the HW-AFX-FF1152-200 [177]. The hardware synthesis of the adaptive system-on-chip architecture is shown in Figure 8.7. Figure 8.7 (a) shows the hardware usage of different parts of the adaptive system-on-chip architecture placed by *PlanAhead design analysis tool* and visualized by *FPGA Editor*. Figure 8.7 (b) shows the hardware results for the parts that are necessary to implement for achieving *architecture-level* adaptation on top of the basic QoS-supported NoC presented in Chapter 3. Finally, Figure 8.7 (c) shows the targeted prototyping board.

8.3 Conclusion

Different simulation tools and the prototyping environment are explained in short in this chapter. The evaluations of the proposed adaptive system-on-chip communication architecture presented in Chapter 5, Chapter 6, and Chapter 7 are done using the presented simulation tools. The router architecture presented in Chapter 7 is prototyped using the environment described in Section 8.2. The evaluation results are given in Chapter 9.

Chapter 9

Results and Case-Study Analysis

In this chapter, a detailed case-study analysis to show the benefits of the proposed adaptive system-on-chip communication architecture is shown. The result includes both the analysis for the *system-level* adaptation as well as the *architecture-level* adaptation of the *Adaptive Networks-on-Chip* (AdNoC) architecture. Different parts of the *architecture-level* adaptation are evaluated in this chapter to show their flexibility, performance gain, and resource utilization.

The simulation environment and tools that are used to show the case-study analysis of the proposed adaptive system-on-chip communication architecture are presented in Chapter 8.

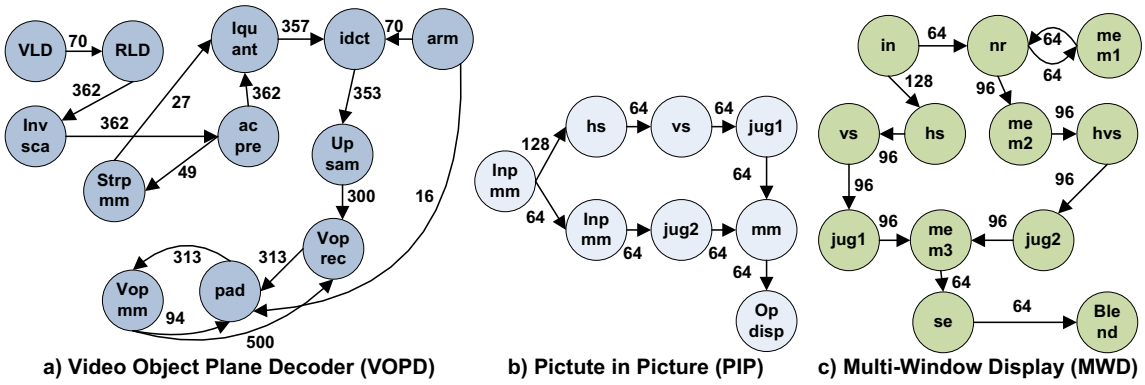


Figure 9.1: Multi-media applications used for the case study analysis

Several embedded systems applications from different application domains, i.e. multi-media applications with real-time constraints, robotic applications having both a control and image processing part, the embedded E3S benchmark suite [50], and applications from *Task Graph for Free* (TGFF) [46] have been used to show the gain in terms of performance, increased resource utilization, etc. for the proposed *AdNoC* architecture compared to the state-of-the-arts. The multi-media applications that are used are: *Video Object Plane Decoder* (VOPD), *Picture in Picture* (PIP), and *Multi-Window Display* (MWD) (see Figure 9.1). The robotic application that is used in the *AdNoC* exploration is the *Image Processing Line* (IPL) application (see Figure 9.2). From the E3S benchmark suite the applications that are used are: the *Automotive/Industry* application, the *Consumer* application, and the *Telecom* application (see Figure 9.3

for a subset of the E3S benchmark suite applications). Several other applications have been generated using the state-of-the-art application task graph generation tool *TGFF* [46].

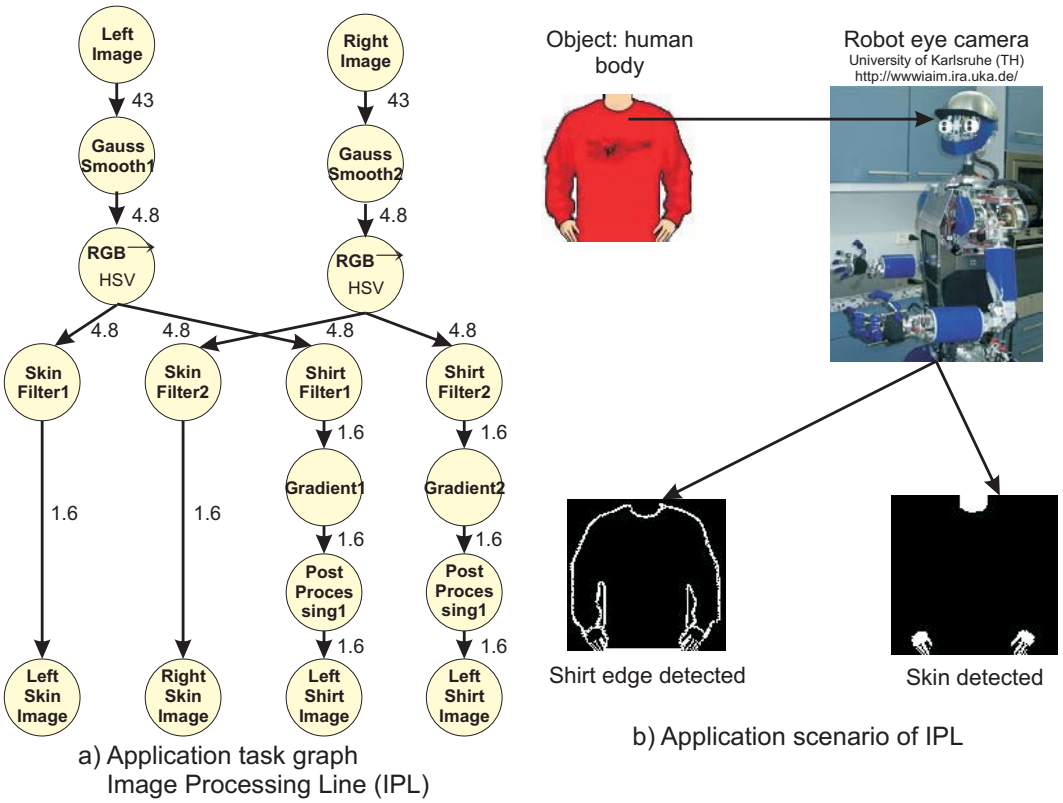


Figure 9.2: Robotic application (IPL) used for the case study analysis

9.1 Evaluation of the Proposed AdNoC Architecture

The adaptive system-on-chip communication architecture (*AdNoC*) is evaluated with different case-study analysis here. The proposed *AdNoC* architecture has mainly two parts: the *system-level* presented in Chapter 6 and the *architecture-level* presented in Chapter 7. Different features and benefits related to both the *system-level* adaptation and the *architecture-level* are described in the following.

9.1.1 ADAM Provides Flexibility and Reduces Computational Cost

The *ADAM* algorithm is evaluated using different evaluation matrices. The performance is shown in terms of execution time and the volume of the generated monitoring traffic and the results are compared to the state-of-the-art centralized approaches [31, 37, 147]. In addition,

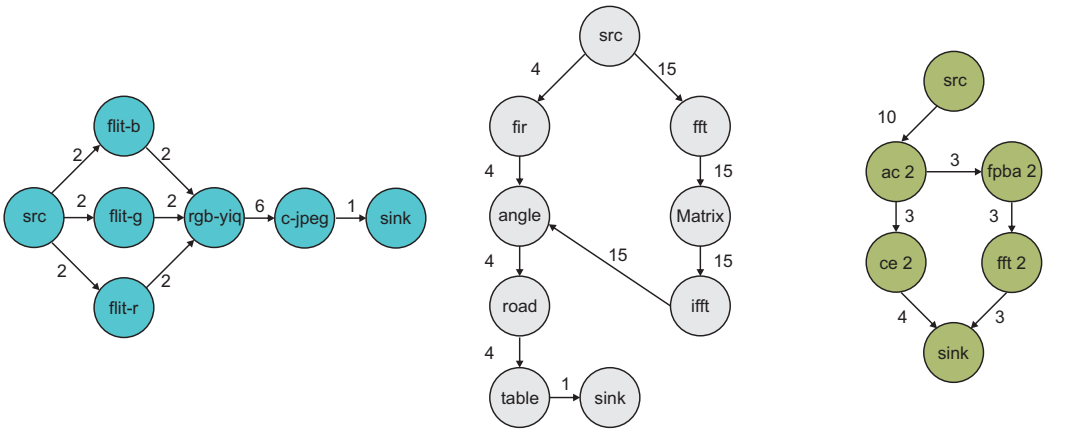


Figure 9.3: Applications form the E3S benchmark used for the case study analysis

the cluster-level mapping algorithm is compared to an exhaustive offline mapping algorithm in order to see how far the *ADAM* is off from an optimum solution.

In Figure 9.4, the *ADAM* algorithm is compared to the centralized one [31]. The mapping computation of the proposed *ADAM* algorithm is partitioned into several steps shown in Figure 9.5. The configuration parameters for this experiment are as follows: the average cluster size is 64 and the number of tasks is 48. In this experiment, the number of cycles to check whether a task can be mapped onto a tile is represented by *"X"* (it may differ depending on the instruction set of the processing element). The computational complexity of different algorithms and their parts are considered to calculate the value of the needed cycles. The computational complexity of the *ADAM* algorithm is given in Chapter 6 and the state-of-the-art algorithms in [31]. It is considered that each task has to be checked for a possible assignment to each tile inside a *virtual cluster* while in the non-clustered approach the tiles of the whole NoC have to be considered. Therefore, the *ADAM* algorithm can reduce the mapping computation complexity, e.g. on a $32x64$ system it is estimated an approximate 7.1 times lower computational effort compared to the simple *Nearest-Neighbor* (NN) heuristics proposed in [31]. Figure 9.6 shows that *ADAM* algorithm scales in the same way as the non-clustered architecture when no clustering approach is considered in the proposed algorithm.

Figure 9.7 demonstrates the advantage of the proposed *ADAM* algorithm when the communication volume generated by the NoC needed for the mapping algorithm is considered. The cluster-based distributed approach is compared to the centralized approaches [37, 147] and a fully distributed approach (each tile acts as an individual cluster). The experimental setup is as following (for details of the parameters see Chapter 6): the number of classes and PE types are 16, the resource requirement encoding requires 1 Byte, the task *identification number* encoding requires 4 Bytes, the number of tasks encoding requires 4 Bytes, and the bandwidth encoding requires 1 Byte of memory space. To calculate the mapping traffic produced by the proposed approach it is needed to break down the communication into the following parts: (1) transmission of the task histogram $thist[]$ to the *GA*, (2) transmission of the task graph to the *CA* of the suitable cluster, (3) reporting the cluster state to the *CA*, and (4) transmission of the cluster

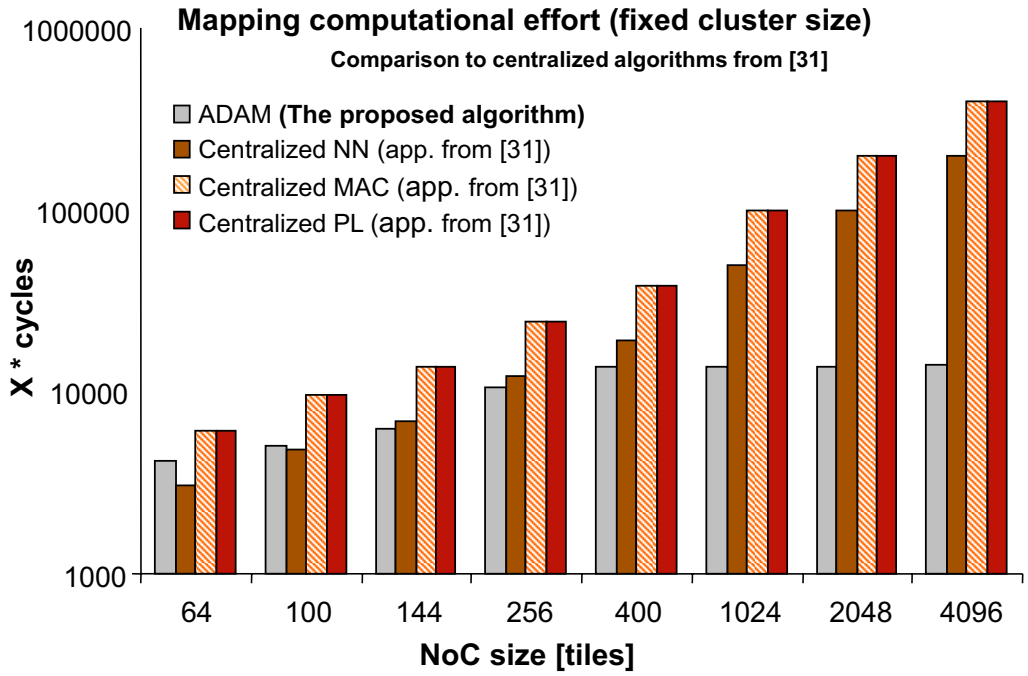


Figure 9.4: Mapping computational effort (fixed cluster size)

state to the *GA*. The experiment shows that the proposed approach has noticeable advantages in reducing the amount of communication volume (10.7 times lower on a 64×64 NoC) caused by the mapping when the heterogeneous MPSoC has many tiles.

In Figure 9.8, a comparison of the suitability of the proposed cluster-level mapping algorithm is presented. It shows that the *ADAM* algorithm does not produce optimum results as they can be produced by the offline exhaustive algorithm which requires a far higher computational effort. But relative to the consumed computation effort the *ADAM* algorithm provides a reasonable near-optimal solution. The communication volume serves as the optimization criteria for the mapping algorithm (it reduces the communication related energy consumption [77]), and on an average a deviation of a mere 13.3% compared to the exhaustive mapping algorithm is observed. For simplicity reasons, to make the comparison to the offline exhaustive mapping algorithm, a homogeneous tile has been considered. The near-optimal result can be used for the runtime task mapping as this result may be traded-off with the adaptivity and the lower computational effort.

The *ADAM* algorithm is further evaluated by means of a robotic application presented in [8]. It is found that in the proposed algorithm the near optimal communication volume to be 120.1 MB/s whereas, in the exhaustive offline mapping algorithm it can be reduced to 106.9 MB/s. The result is acceptable as it is done at runtime using a heuristic algorithm and consuming 2 times lower execution cycles compared to *NN* heuristics. Mapping the *IPL* application takes only $11241 \times X$ cycles using the proposed *ADAM* algorithm orthogonal to any instruction set processor compared to the *NN* heuristics (takes $20480 \times X$ cycles) proposed in [31] on a 32×64 NoC. Therefore, it is observed that the proposed runtime agent-based distributed application

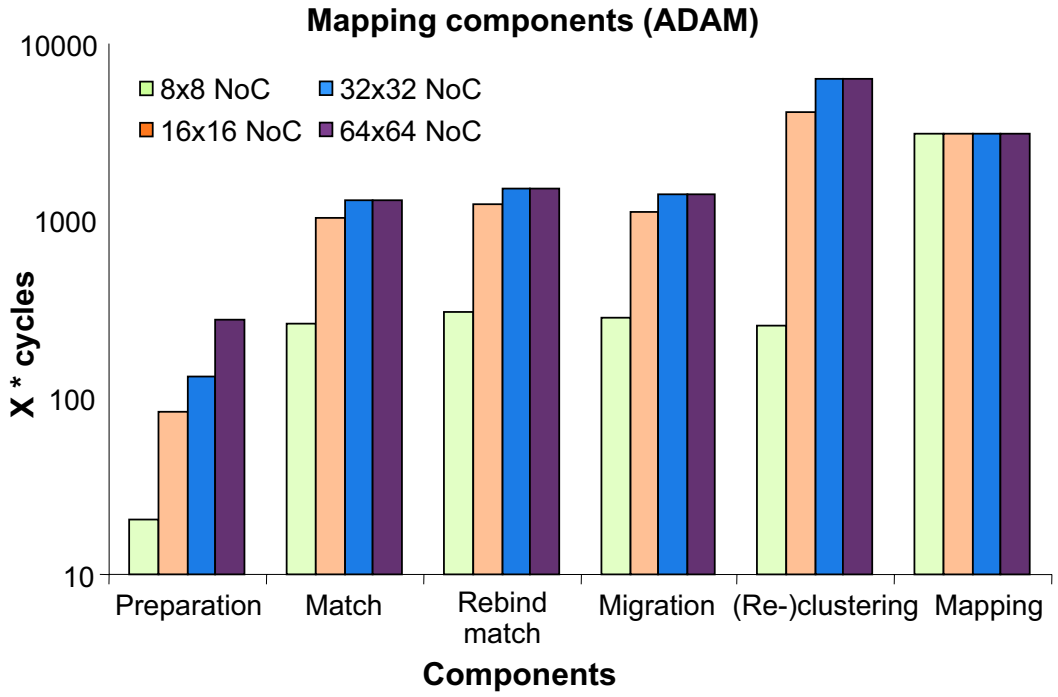


Figure 9.5: Computation complexity of the components of the ADAM algorithm

mapping approach reduces the overall monitoring-traffic compared to a centralized mapping scheme and requires less execution cycles compared to a non-clustered centralized approach.

9.1.2 On-demand VCB Assignment Increases Buffer Utilization

For the evaluation of the on-demand *VCB* assignment and the *wXY-routing* algorithm of the *AdNoC* architecture, the proposed scheme is compared with two representative state-of-the-art static architectures, namely *QNoC* [23] and *Xpipe* [14]. The proposed architecture and these architectures share the same flow control mechanism during transmission except the proposed scheme supports adaptive transactions. Both of these architectures exploit packet-based system-on-chip communication and use wormhole routing with *VC* implementations. *QNoC* provides *QoS* implementing different service classes. It uses a fixed number of *VCBs* in each output port and deploys a distributed deterministic *XY-routing* algorithm. The *Xpipe* architecture also uses fixed number of *VCBs* in each output port and a *source-based* deterministic routing. For a fair comparison, the *source-based* deterministic routing of *Xpipe* is replaced with a *wXY-routing* algorithm in the evaluation. These parameters in the design are the most important for the flow control mechanism and support for the *QoS*. The fixed buffer assignment strategy in the *QNoC* and the *Xpipe* allows us to compare the buffer utilization between the fixed approach and the proposed on-demand buffer assignment approach. Therefore, considering these characteristics/parameters, the proposed approach is compared to the static *QNoC* [23] and *Xpipe* [14] architectures.

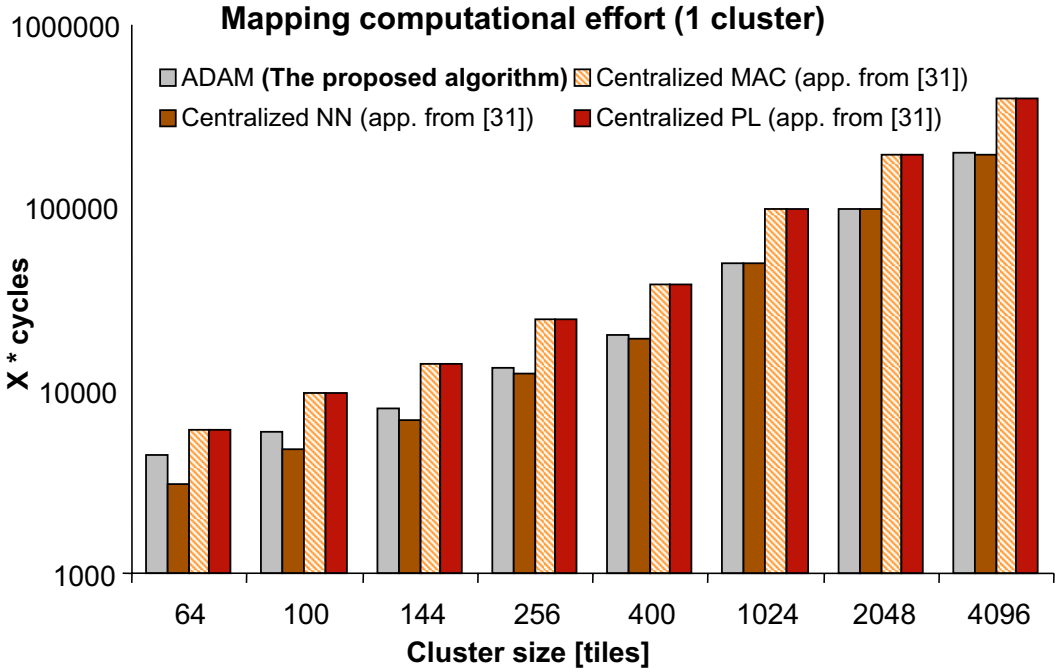


Figure 9.6: Comparison of the computation complexity of the ADAM algorithm

Application	#of tasks	Link capacity (MB/s)	NoC
VOPD	12	845	4×3
PIP	8	200	3×3
MWD	12	221	4×3

Table 9.1: Application and corresponding NoC specification

Some assumptions are taken for the experimental setup. The link capacity for each application is taken as:

$$LC = BW_k^{max} + Avg(B_i)[MB/s] \tag{9.1}$$

where, $i, k = task_s \rightarrow task_d$, LC stands for the link capacity, BW_k^{max} for the maximum bandwidth, and $Avg(B_i)$ for the average bandwidth. For this exploration, three applications shown in Figure 9.1 have been used. The specification of the applications and the dimensions of the NoC architectures are given in Table 9.1. For the evaluation, 25 representative mapping samples for each application have been investigated . The experiments with different subsets of 25 samples from a large mapping space have been then conducted. Experiments revealed that for different subsets of 25 samples, the result remains similar. Therefore, in this thesis only a subset of 25 exemplary results are shown.

The runtime distributed application mapping algorithm (*ADAM*) in *AdNoC* changes the mapping of the running tasks. Therefore, it is needed to configure the underlying communication structure for the current instance of the application. Here, in both the figures *XY-n* stands for the *XY-routing* algorithm having a number of *n VCBs* in each output port, comparable to the *QNoC*

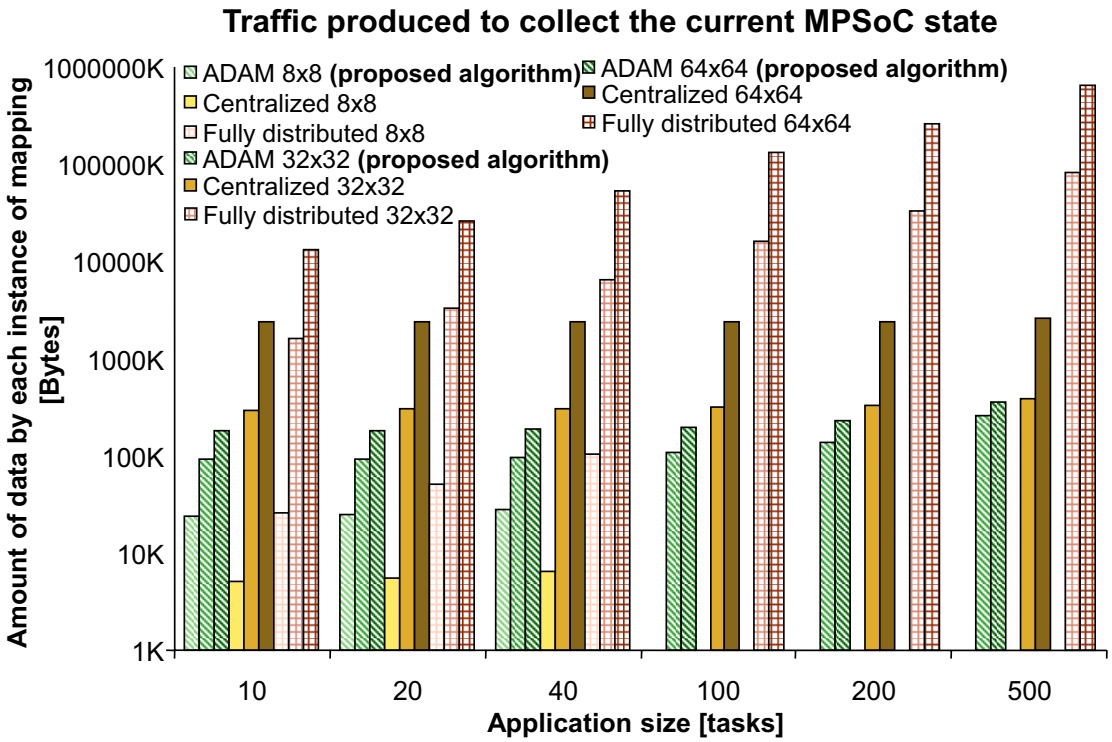


Figure 9.7: Traffic produced by the ADAM compared to the algorithms [31, 37, 147]

architecture, *fB-wXY-n* stands for the *wXY-routing* algorithm having *n* number of fixed *VCBs* in each output port, comparable to the *Xpipe* architecture and *dB-wXY-n* stands for the proposed *wXY-routing* algorithm having *n* number of on-demand *VCBs* in each router, the proposed on-demand VCB assignment scheme in the *AdNoC* architecture.

Figure 9.9 shows the behavior of the different routing and buffer assignment schemes when the *ADAM* algorithm changes the task mapping. It is found that if the *PIP* application uses the *XY-routing* algorithm and in total 24 *VCBs*, then the success rate to provide bandwidth guarantee (QoS) is 11%. If the routing algorithm is just changed to the *wXY-routing* algorithm having the same buffer assignment strategy the success rate increases to 80%. 100% success rate has been found in the transmission using 48 *VCBs* in both the routing schemes. This implies that even if the buffer assignment strategy is fixed using the *wXY-routing* algorithm the proposed scheme can improve the successful transaction rate. Now as it may be seen that in the proposed scheme using the *dB-wXY-2* strategy (requiring only 18 *VCBs*) a success rate of 43% is gained and using only 27 *VCBs* a 100% success rate may be achieved. Therefore, the number of *VCBs* in the *AdNoC* architecture may be reduced by 44% while keeping the success rate 100% like the other schemes. Similar results have also been observed for the application *VOPD* and *MWD*. Therefore, besides the degree of freedom in the adaptation during runtime the proposed on-demand buffer assignment strategy together with the runtime *wXY-routing* algorithm gains a higher *successful transaction rate to the VCB unit* compared to the static approaches *QNoC* and *Xpipe* (shown in Figure 9.10).

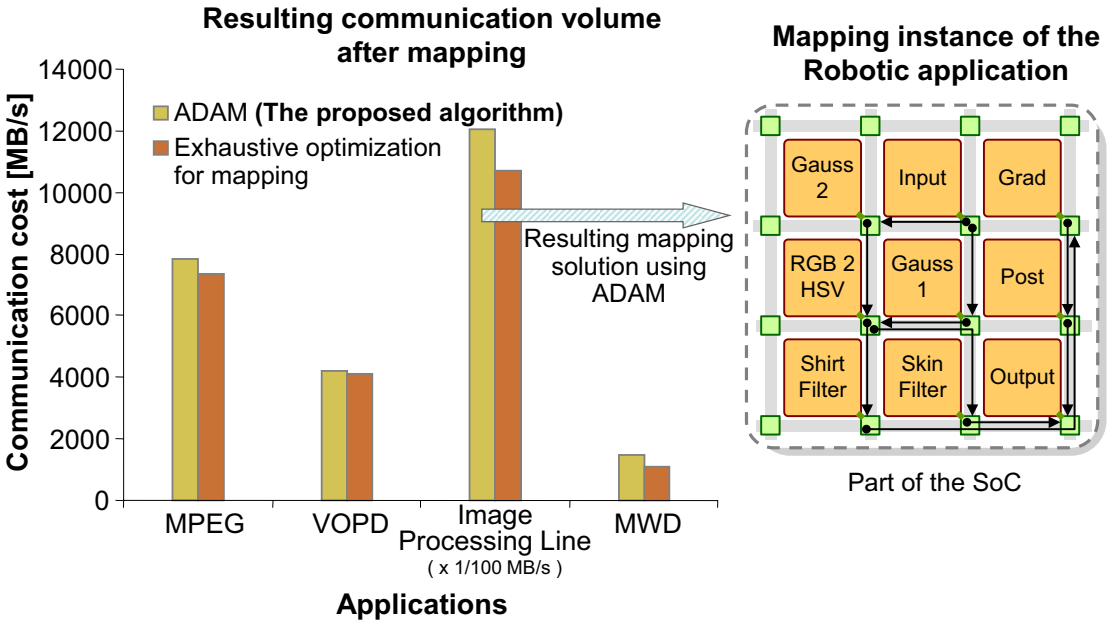


Figure 9.8: Comparing ADAM algorithm to exhaustive offline mapping algorithm

The *XY-routing* algorithm is forced to take deterministic routes where the *wXY-routing* algorithm takes bandwidth usage into consideration, choosing routes with low loads. This focuses the traffic onto specific links increasing their required capacity while other links may remain idle. Figure 9.11 demonstrates this for the *VOPD* application. The result shows that for a given specific mapping and link capacity, the *wXY-routing* algorithm is more likely to find a successful routing than the static *XY-routing* algorithm. On an average this likelihood is around 20% for the *VOPD* application.

It can be concluded that the *AdNoC* architecture enjoys the freedom to adapt during runtime and may very well increase the rate of successful transactions for a changing network compared to static NoCs using the on-demand *VCB* assignment and the *wXY-routing* algorithm. Static NoCs are fixed as far as *VCB* assignment is concerned and they cannot change their configuration and placement of virtual transactions. Besides this unique feature, *AdNoC* also reduces the total number of *VCBs* by 42%. All these unique properties puts *AdNoC* in a favorable position compared to the existing static architectures for the next generation adaptive MPSoC design.

9.1.3 Configurable Links Increase Resource Utilization

In order to evaluate the proposed runtime configurable *2X-Link* different simulations are performed using the multi-media applications (see Figure 9.1) and the E3S benchmark [50] suite (see Figure 9.3).

In Figure 9.12, the *average throughput* for different types of links for the *Automotive* application which is mapped onto a 5×5 NoC having the link capacity of 15 MB/s for the *Normal-Full-*

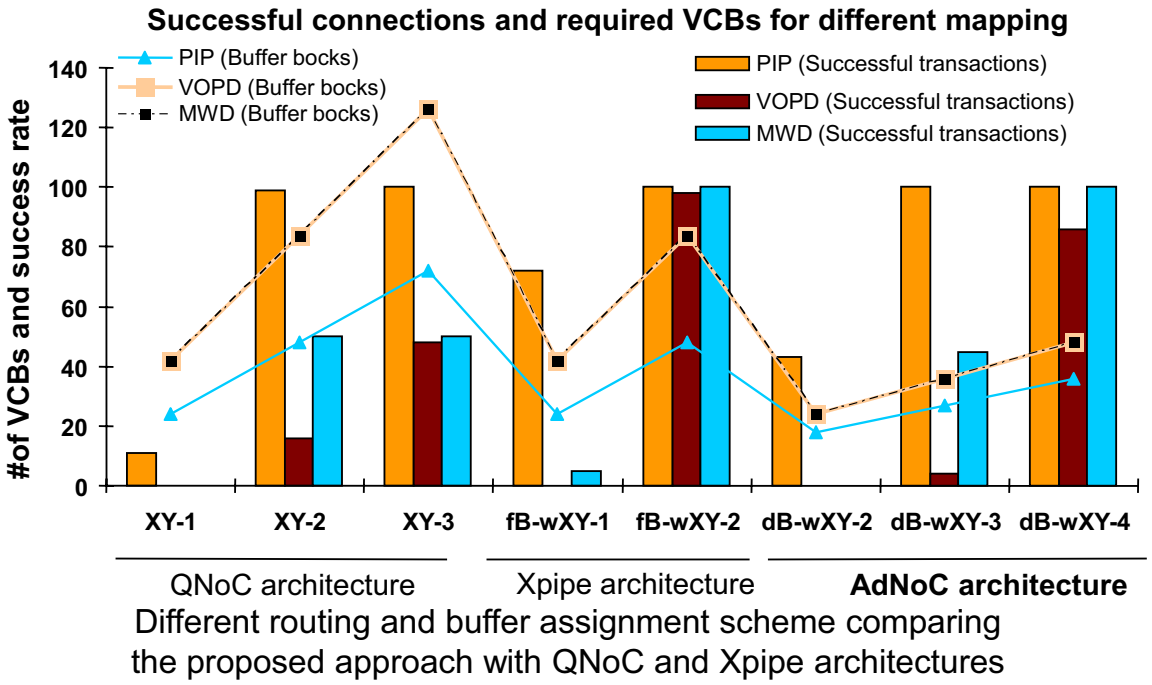


Figure 9.9: Successful transactions and corresponding buffer requirement

Duplex-Links and for each half-duplex component of the *2X-Links*, and 30MB/s for the *TDD-Links*. The *average throughput* is calculated as follows:

$$Th_{aver} = Th_{tot}/(2 \times ((M-1) \times N + (N-1) \times M)) \tag{9.2}$$

In Equation 9.2, $(M \times N)$ is the dimension of the Mesh NoC and Th_{tot} the total throughput. In real-time applications, the higher the data injection rate, the earlier the deadline will arrive. Therefore, to keep the deadline with a higher data injection rate, the bandwidth must be increased to enhance the transmission ability ($BW_{new} = BW_{unit} \times injection\ rate$). It may be observed from the figure that the *average throughput* of the NoC with the *Normal-Full-Duplex-Links* is much lower than the *average throughput* of the NoC with the *2X-Links* or the *TDD-Links* at relatively high data injection rates. In scenarios, where there is only asymmetric traffic on the NoC, there is no difference between the *2X-Link* and the *TDD-Link*. On the other hand, if bidirectional communication exists, the *average throughput* of the NoC with the *2X-Links* is slightly lower than the NoC with the *TDD-Links*.

Figure 9.13 indicates the *timeliness* of the *Telecom* application (mapped onto a 5×5 NoC having the link capacity of 20 MB/s for the *Normal-Full-Duplex-Links* and for each half of the *2X-Links* and 40MB/s for the *TDD-Links*) with different data injection rate. The *timeliness* is the ratio of the number of deadlines met to the total number of deadlines of a given application:

$$timeliness\ =\ \#\ of\ deadlines\ met\ /\ \#\ total\ deadlines \tag{9.3}$$

From this figure, it can be seen that most of the time the *timeliness* of the *Normal-Full-Duplex-Links* , is lower than the other two types of links.

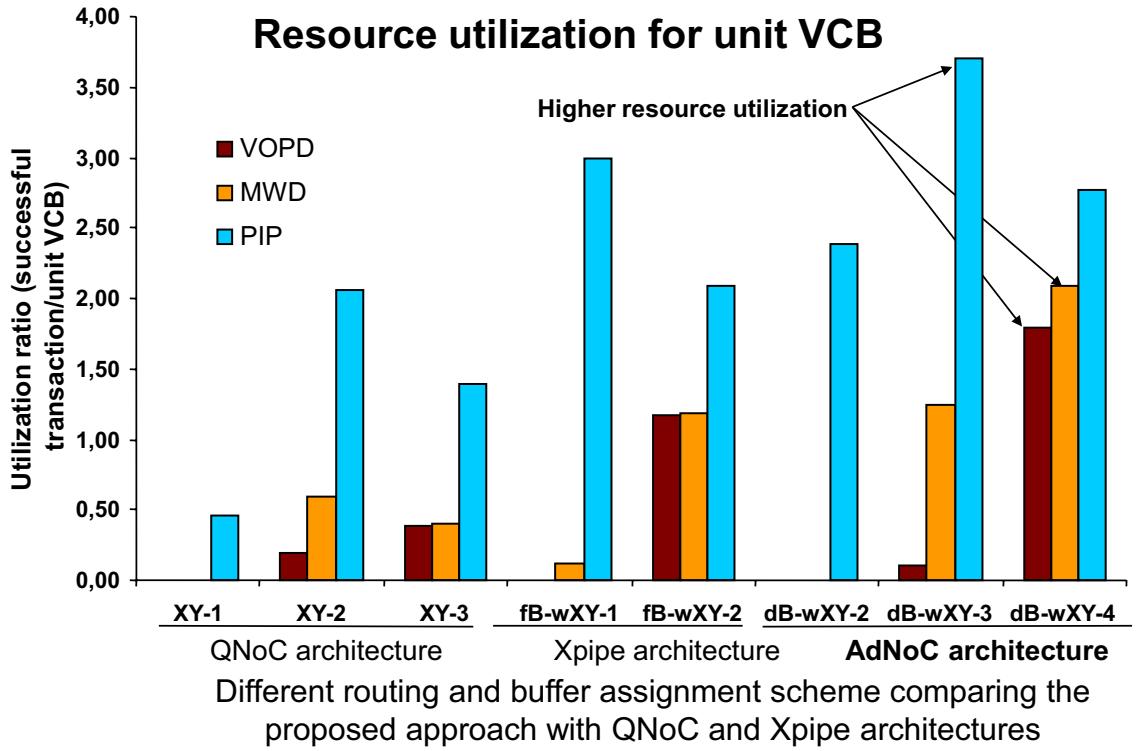


Figure 9.10: Resource utilization for unit virtual channel buffer block

Figure 9.14 depicts the *required link capacity* of the NoC with three different types of links when the multi-media application *VOPD* is running on the NoC (mapped onto a 5×5 NoC having the link capacity of 500 MB/s for the *Normal-Full-Duplex-Links*, two times 500 MB/s for the *2X-Links*, and 1000MB/s for the *TDD-Links*). Obviously, the NoC with the *Normal-Full-Duplex-Links* requires much more link capacity than the *2X-Links* and the *TDD-Links* at design-time. Since the *2X-Links* can dynamically adapt their data-transmitting direction to meet the bandwidth requirement and the *TDD-Links* support bidirectional communication, both of them can guarantee the required bandwidth with lower link capacity compared to the *Normal-Full-Duplex-Links*.

Fault-tolerance ability is important to the MPSoC design. In this simulation, it is focused on the *average timeliness* of the NoC with different types of links when some links fail randomly. The metric *"average timeliness"* is different from *"timeliness"* shown in Figure 9.13 and it is used to calculate the *fault-tolerance ability*. All the possible combinations of the failed-links are enumerated and their *average timeliness* are calculated. The $T_{k,aver}$ is defined as the average ratio of the number of deadlines met to the total number of deadlines of a given application when k links fail randomly. The equation is shown below. There, k represents the number of failed-links, while $\#Comb$ is the un-ordered collection of k failed links among all the links.

$$T_{k,aver} = (\sum_{i=1}^{\#Comb} timeliness) / (\#Comb) \tag{9.4}$$

The state-of-the-art *Normal-Full-Duplex-Links* are static. If one link fails, the NoC will lose

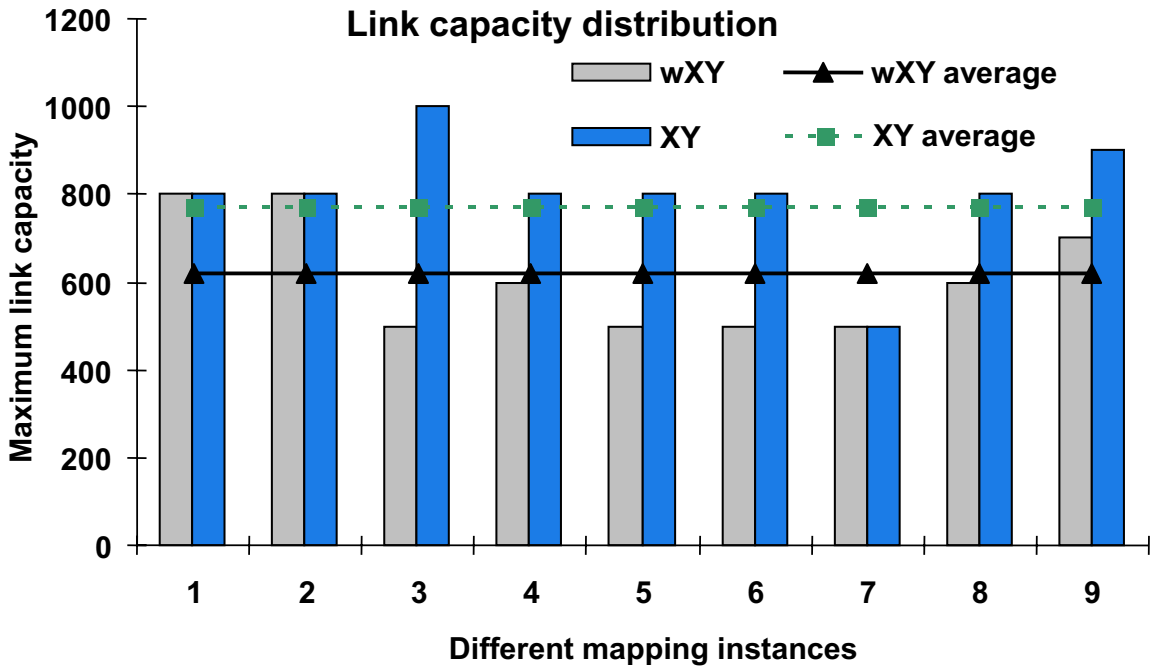


Figure 9.11: Utilization of the link capacity (VOPD application)

the ability to transmit data from the port which the link is tied to. Similar to the *Normal-Full-Duplex-Links*, the *TDD-Links* also do not have the *fault-tolerance ability*. On the contrary, the *2X-Links* can adapt their data-transmitting direction to meet the performance-related guarantees. From the Figure 9.15 it can be seen that the *average timeliness* of the NoC with the *2X-Links* is much higher than the NoC with the *Normal-Full-Duplex-Links* (for the Robotic application which is mapped onto 3×3 NoC having the link capacity of 30.7 MB/s for the *Normal-Full-Duplex-Links*, two times 30.7 MB/s for the *2X-Links*, and 61.4 MB/s for the *TDD-Links*). The *fault-tolerance ability* (ft_k) is calculated according to the following equation (NFDL represent the *Normal-Full-Duplex-Link*):

$$ft_k = \frac{T_{k,aver,\mathrm{2X-Links}} - T_{k,aver,\mathrm{NFDL}}}{1 - T_{k,aver,\mathrm{NFDL}}} \tag{9.5}$$

For a simple example, if the number of failed-links is 1, the *fault-tolerance ability* of the *2X-Link* is $(97.50\% - 94.17\%)/(1 - 94.17\%) = 57.12\%$. This result means that the NoC with the *2X-Links* can still work properly with the probability of 57.12%, when only one link fails randomly.

In the scope of this work, *load balancing* means to spread communication over all the links to avoid producing communication bottlenecks and to relieve the link resource congestion. The advantage of applying load balancing techniques is that hardware resource utilization will be optimized and therefore, it may make the system-on-chip communication more able to meet the requirements of unforeseen traffic circumstance. In the simulation, the *bandwidth occupancy factor* of each port is measured through $BW\_occupancy\_factor = BW_{utilized}/BW$. The lower the factor is, the higher the potential available link capacity is. Therefore, the NoC is more flexible to support unforeseen traffic (e.g. new transactions etc.).

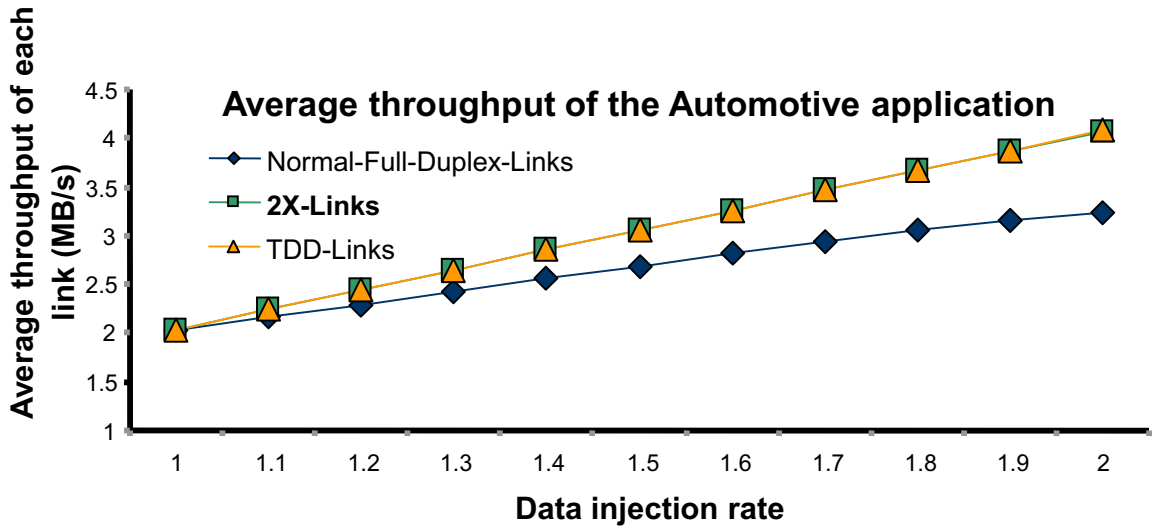


Figure 9.12: Average throughput of the Automotive application

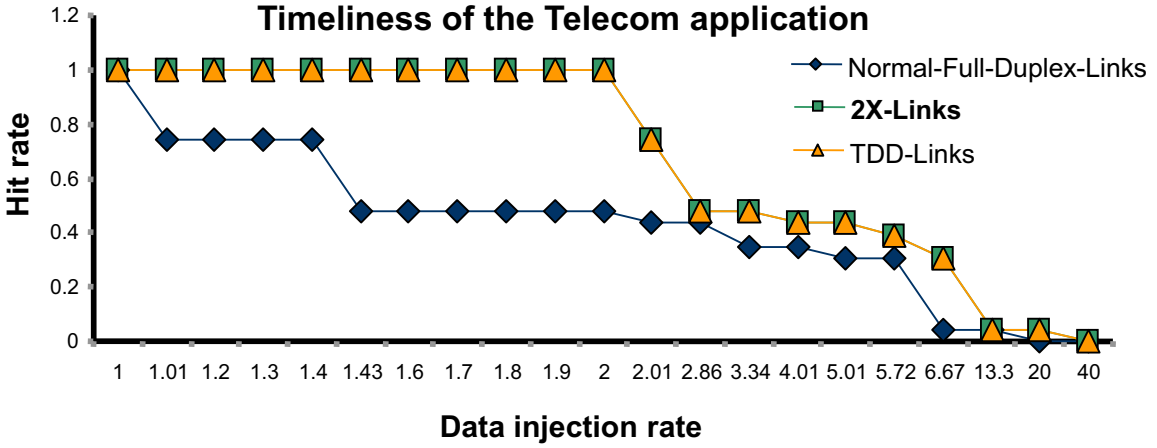


Figure 9.13: Timeliness of the Telecom application

Figure 9.16 (*Consumer* application is mapped onto 4×4 NoC having the link capacity of 9 MB/s for the *Normal-Full-Duplex-Links*, two times 9 MB/s for the *2X-Links*, and 18 MB/s for the *TDD-Links*) represents the *bandwidth occupancy factor* of all ports with different types of links. Obviously, the bars of the *2X-Links* (Figure 9.16 (b)) and the *TDD-Links* (Figure 9.16 (c)) are much lower and the communication is spread more homogeneously on the NoC than with the *Normal-Full-Duplex-Links* (Figure 9.16 (a)). This indicates that the NoCs with the *2X-Links* and the *TDD-Links* have more potentially available link capacity than the NoC with the *Normal-Full-Duplex-Links*.

In order to make the system-on-chip communication more adaptive, the *wXY-routing* algorithm is additionally used to determine the communication route instead of the static *XY-routing* algorithm. Figure 9.16 (d,e,f) depict the *bandwidth occupancy factor* using the *wXY-routing* algorithm. According to the new route, bandwidth requirements of several links are changed. Even

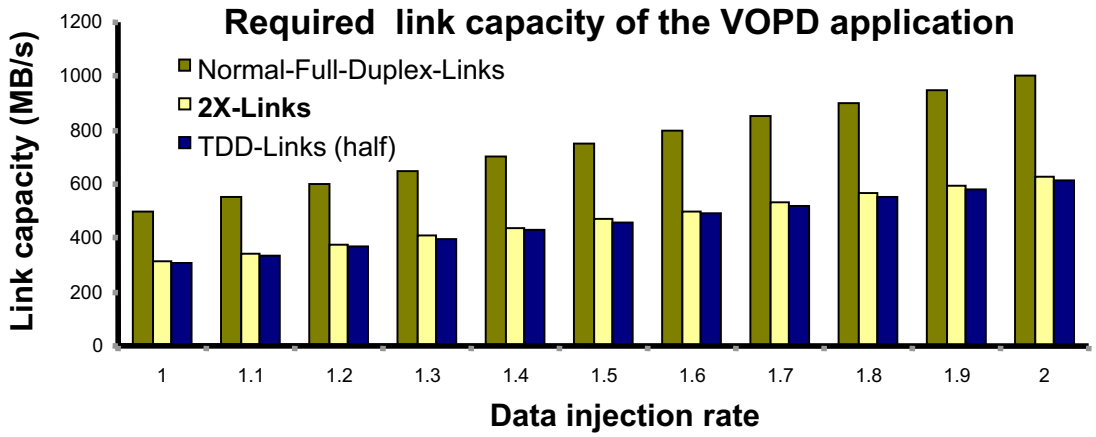


Figure 9.14: Appropriate link capacity of the VOPD application

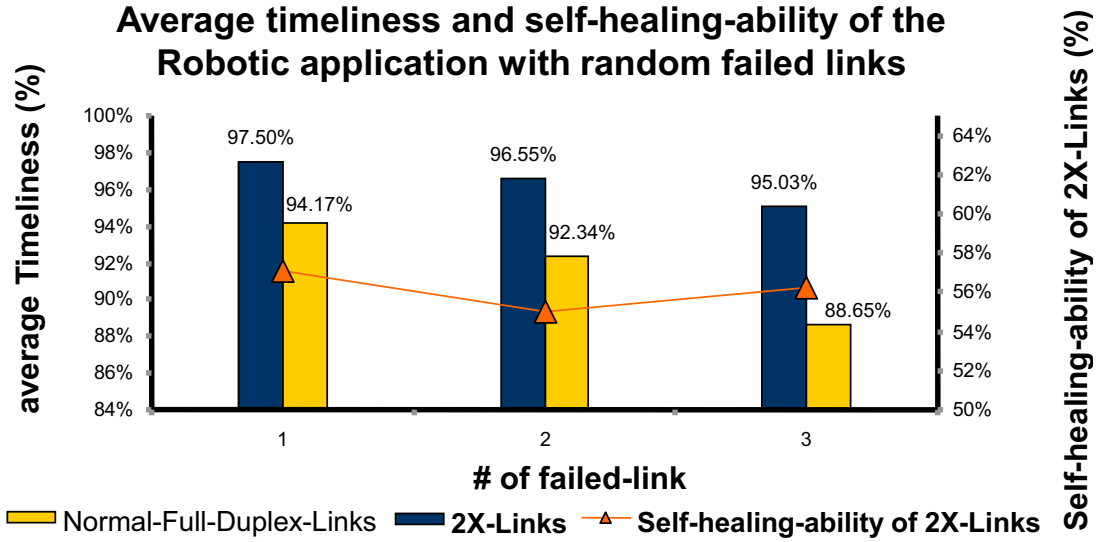


Figure 9.15: Average timeliness and fault-tolerance ability of the Robotic application

when using this adaptive routing algorithm, the result from the *2X-Links* and the *TDD-Links* are far better than using the *Normal-Full-Duplex-Links* as the routing algorithm alone cannot compensate.

In the given experiments, it is observed that both the *2X-Link* and the *TDD-Link* provide better performance results than the *Normal-Full-Duplex-Link* considering the metrics: *average throughput*, *timeliness*, *required link capacity*, and *the traffic load balancing*. The proposed *2X-Link* provide *fault-tolerance ability* unlike both the *Normal-Full-Duplex-Link* and the *TDD-Link*. Therefore, considering the area overhead and the lack of *fault-tolerance ability* of the *TDD-Links* the *2X-Links* have been integrated in the proposed runtime adaptive system-on-chip communication architecture.

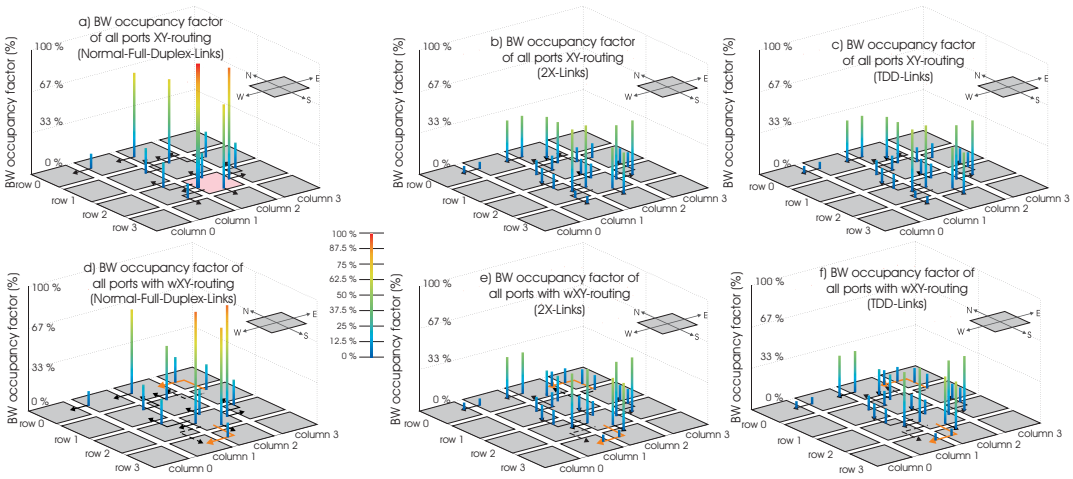


Figure 9.16: Bandwidth occupancy factor of the Consumer application

9.1.4 Light-weight Monitoring Component

The proposed *Runtime Observability infrastructure for the Adaptive Networks-on-Chip architecture* (ROAdNoC) is evaluated with several parameters that directly influence the monitoring traffic and bandwidth usage:

- The *packet injection rate* determines the arrival frequency of the new packets to the network in each router.
- The *packet flit size* is responsible for the duration of traffic (along with the allocated bandwidth slots).
- The allocated link *bandwidth slots* per transaction influence the number of simultaneous transactions per link.
- The number of *VCBs* limits the number of simultaneous transactions per router.

A number of assumptions are made to determine the simulation parameters: the traffic distribution used is uniform, the data packet size is 200 flits, the monitoring packet size is 2 flits, and the bandwidth is 20 slots. These parameters have been chosen to observe the effects of both *no-buffer-event* and *no-route-found-event*.

For the first simulation the data packet injection rates are based on allocated bandwidth slots. They are chosen as to supply a (near) continuous stream of data by using the highest possible rates. For instance, a transaction allocated 1 slot out of 20 can at most send one flit every 20ns (20 cycles). Hence, the highest accommodatable data packet (200 flits) injection rate is one packet every 4us. The traffic is streaming with packet injection being normally distributed with some variance. This traffic is the worst case for *VCB* usage as continuous traffic also requires constant *VCB* assignment. In the simulation, each router has 8 *VCBs*. For the adaptive routing algorithm, the worst case is any slot value greater than 10 as each link can only transmit one transaction of this type. The simulation results (Figure 9.17) show a gradually increasing

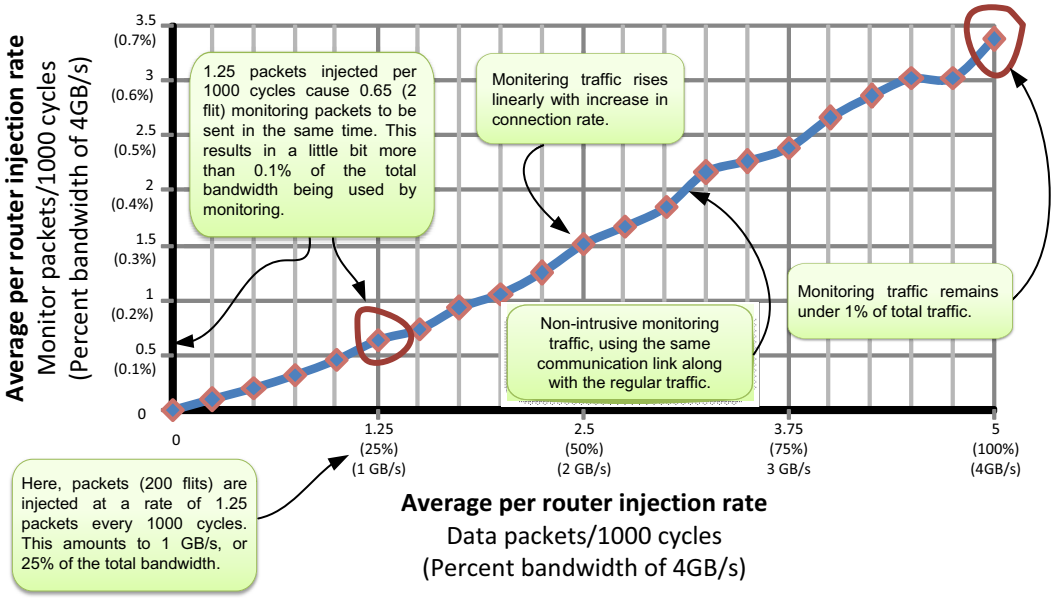


Figure 9.17: Monitoring packets injection and traffic density

monitoring packet injection rate for increasing traffic density. However, the monitoring traffic remains low considering the overall link bandwidth – less than 0.7%.

Figure 9.18 shows the effect of different number of *VCBs* per router and allocated bandwidth slots on the monitoring packet injection rate. There is a clear distinction between the low bandwidth/continuous traffic on the left and the high bandwidth/burst traffic on the right. This is due to the on-demand *VCB* assignment being the dominant cause of monitoring events in the left part and the adaptive *wXY-routing* algorithm in the right part. The traffic generated by *ROAdNoC* infrastructure cannot directly be compared to that of the *Æthereal* monitoring component. The occurrence of events in *Æthereal* monitoring component is different to *ROAdNoC* infrastructure as they are mainly managing events necessary for debugging whereas only types of events that are needed to adapt the system-on-chip communication architecture are managed here. Using only *connection-opened-events* and *connection-closed-events* to calculate the resulting data rate assumes that all transactions are set up successfully and specifically no *alert-events* occur. Under such circumstances the proposed *ROAdNoC* infrastructure would generate no traffic. For comparison *Æthereal* is assumed to have a comparable routing algorithm which is able to choose alternative routes. It is assumed to produce *Æthereal* NoC *alert-events* when no route is found. Furthermore, it is assumed that any failed transaction attempts are resolved through re-routing or (re-)mapping by *ADAM* algorithm if needed.

Taking the assumptions from [39] but expanding the traffic model by the number of successful transactions setup by the initial attempt to set up 200 transactions, the two approaches may be compared. To calculate the total monitoring traffic t_M requires the number of unsuccessful transactions u per second, the number of total transactions c per second (200), the monitoring traffic for an unsuccessful transaction t_u, and the monitoring traffic for a successful transaction

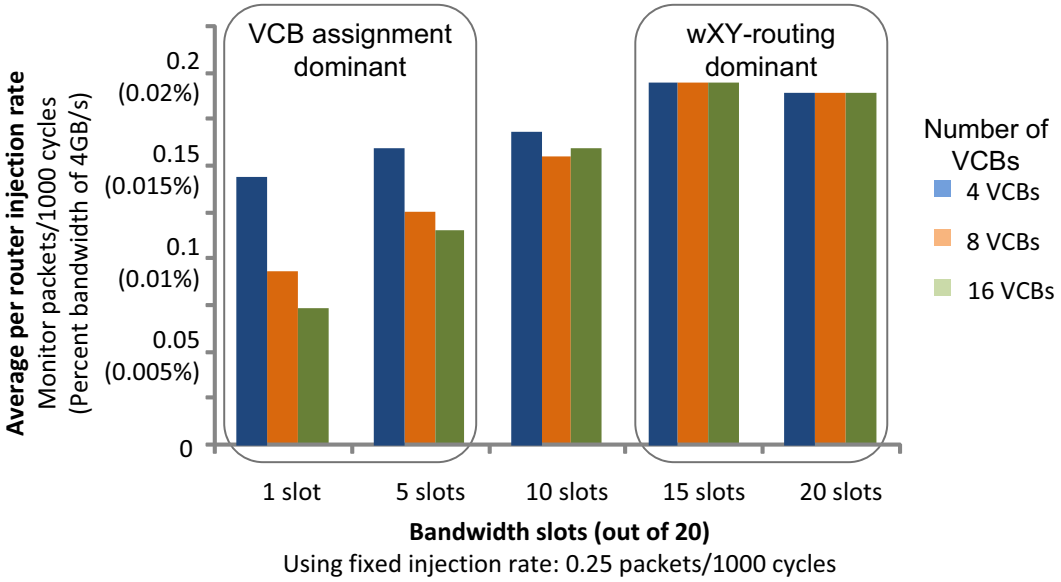


Figure 9.18: Causes of monitoring events

t_s. For the first attempt the monitoring traffic t_{M1} is calculated:

$$t_{M1} = u \cdot t_u + (c - u) \cdot t_s \tag{9.6}$$

Unsuccessful transactions are assumed to be successful after they are re-routed causing the additional monitoring traffic, t_{M2} to be $u \cdot t_s$. By adding t_{M1} and t_{M2} the total traffic is obtained shown in Table 9.2. For the first comparison, the *Æthereal* monitoring component produces both *connection-opened/closed-events* for successful transactions and *alert-events* for unsuccessful ones. The unsuccessful ones then also produce *connection-opened/closed-events* as they are established successfully after routing. However, since *Æthereal* can switch the monitoring of specific events on and off, a more direct comparison is given by limiting the *Æthereal* monitoring to only *alert-events*. The results show that *ROAdNoC* infrastructure generates less traffic even with the simple profile of the *Æthereal* monitoring. In conclusion, both monitoring

Successful transactions	Monitoring traffic **(ROAdNoC)**	*Æthereal* (Alert & Config.-events)	*Æthereal* (Alert-events) only)
50	1.2KB/s	6.6KB/s	1.8KB/s
100	0.8KB/s	6KB/s	1.2KB/s
150	0.4KB/s	5.4KB/s	0.6KB/s
200	0KB/s	4.8KB/s	0KB/s

Table 9.2: Traffic comparison using 200 transactions/second

components are designed with entirely different goals in mind. The *ROAdNoC* infrastructure

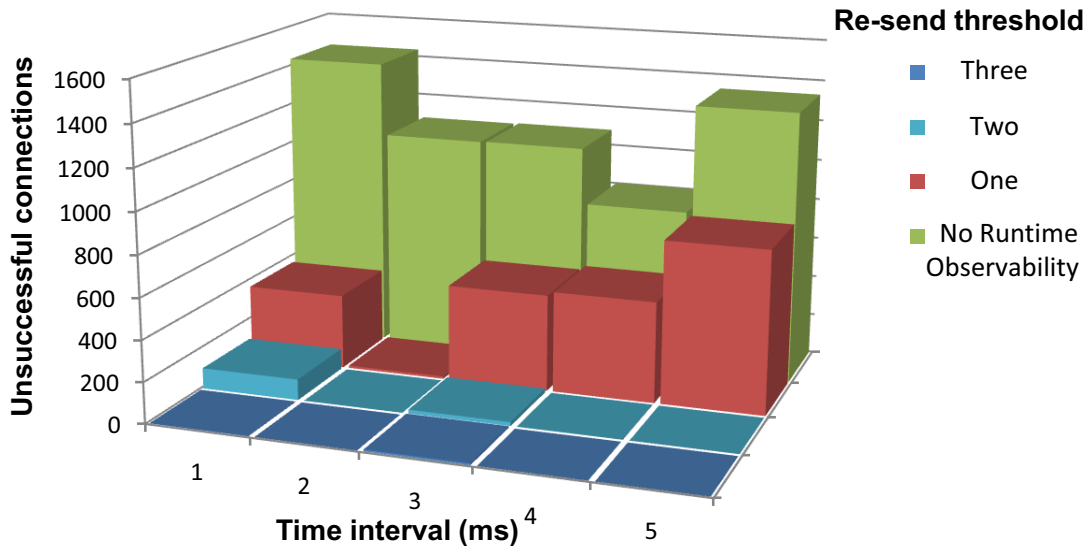


Figure 9.19: Unsuccessful transactions for various re-sending thresholds

is designed specifically to facilitate the adaptivity of the NoC and thus only monitors events required to control the NoC configuration.

Figure 9.19 shows the effect of re-sending packets on the number of packets which are able to be successfully transmitted for the E3S benchmark suite [50]. On an average, a re-sending threshold of 1 is able to increase the success rate by 62% compared to the case without runtime observability. For higher threshold values this value increases even further, allowing the ROAdNoC infrastructure to avoid a costly (re-)mapping.

9.2 Summary of the Evaluation

In this chapter, both the *system-level* and the *architecture-level* adaptation are evaluated using various benchmarks and the simulation tools presented in Chapter 8. Exploration for the *system-level* adaptation shows, the proposed runtime application mapping algorithm *ADAM* obtains 10.7 times lower monitoring traffic compared to the centralized mapping schemes proposed in the state-of-the-art works for a 64×64 NoC [37, 147]. The algorithm also requires less execution cycles compared to a non-clustered centralized approach. It achieves for the experiment on an average 7.1 times lower computational effort of the mapping algorithm compared to the simple *NN* heuristics [31] in a 64×32 NoC.

Exploration for the *architecture-level* adaptation shows the following results: the *2X-Link*, a novel link for the *architecture-level*, provides a higher throughput of up to 36%, with an *average throughput* increase of 21.3%, compared to the *Normal-Full-Duplex-Link* and keeps performance-related guarantees with as low as 50% of the *Normal-Full-Duplex-Link* capac-

ity. Simulation shows when some links fail randomly, the NoC with the *2X-Links* can recover from these faults with an average probability of 82.2% whereas these faults would be fatal for the *Normal-Full-Duplex-Links*. It can be seen that up until now, the number of *VCBs* at one port has typically been fixed at design-time. With on-demand *VCB* assignment, the *VCBs* are not tied to ports, but only to the router itself. The router may distribute the *VCBs* to any route as needed by assigning a transaction to the *VCBs* through the virtual channel arbiter and then assigning the *VCB* to an output port. This adaptive buffer assignment increases the buffer utilization and decreases the overall buffer use on an average of 42% in the case study analysis compared to a fixed buffer assignment. It is shown that a runtime observability scheme (*ROAdNoC*) is required for the runtime system in the *architecture-level*. *ROAdNoC* infrastructure is hardly intrusive, i.e. in worst case it may require a mere 0.7% of the total link capacity. It analyzes the communication architecture during runtime and self-adapts depending on the monitoring traffic on when and how a certain router should be configured for a certain transaction. The runtime observability scheme on an average increases the transaction success rate by 62% compared to having no runtime observability for the E3S benchmark suite. Because of all the advantages of the *AdNoC* architecture discussed above, the area overhead introduced by the *architecture-level* adaptation (the area overhead stems from the *wXY-routing* algorithm, the *2X-Link*, and the monitoring component) may be traded-off against the flexibility to select an available route and on-demand buffer assignment to that route for ensuring the QoS.

Chapter 10

Conclusion

This chapter summarizes the thesis and discusses possible future work.

10.1 Thesis Summary

In this thesis, the novel approach of an adaptive system-on-chip communication architecture is presented. The *AdNoC* is implemented as a solution for the adaptive system-on-chip communication architecture. Adaptivity is employed both in the *system-level* as well as in the *architecture-level*. A runtime agent-based distributed application mapping is provided as a part of the *system-level* adaptation for the next generation self-adaptive heterogeneous MPSoCs. The *architecture-level* part of the runtime adaptive system-on-chip communication architecture adapts the underlying interconnections on-demand in response to changing communication requirements imposed by an application, i.e. runtime application mapping request from the *system-level* due to reliability issues or user behavior.

The first algorithm for a runtime application mapping (as a part of the *system-level*) in a distributed manner using an agent-based approach is introduced in this thesis. The proposed *ADAM* algorithm generates 10.7 times lower monitoring traffic compared to a centralized mapping algorithms in a 64×64 NoC. *ADAM* also has a smaller number of execution cycles compared to a non-clustered centralized approach. During the experiment, on an average 7.1 times lower computational effort for the runtime mapping algorithm compared to the simple *Nearest-Neighbor* (NN) heuristics on a 64×32 NoC is achieved. It is shown that the flexibility of a runtime adaptive mapping, a 7.1 times lower computational effort and a 10.7 times lower monitoring traffic counterbalance the optimization result (a mere 13.3% deviation) compared to the off-line exhaustive and the runtime centralized application mapping algorithms.

To provide on-demand interconnections, a novel adaptive routing algorithm that meets *Quality-of-Service* (QoS) requirements (bandwidth) is included in the *architecture-level* part. The routing algorithm makes decisions locally at each router depending on the available bandwidth in each direction to the neighboring router. Dynamic connections are realized by re-assigning a certain number of buffer blocks to different output ports of a router on-demand. It also increases the resource utilization, especially buffer utilization, through the on-demand buffer block assignment. Experiments show that the on-demand buffer assignment presented as a part

of the *architecture-level* of the adaptive system increases the buffer utilization and decreases the overall buffer use, on an average of 42% compared to a fixed buffer assignment where a fixed number of buffer blocks is tied to the output port.

Furthermore, a runtime configurable link (the *2X-Link*) is integrated in the *architecture-level* to compliment the adaptive system-on-chip communication architecture. An increase in throughput of up to 36% (21.3% on average) using either the *2X-Link* or the *TDD-Link* compared to the *Normal-Full-Duplex-Link* is achieved. The proposed *2X-Link* (the *TDD-Link* also provide similar result) can assure the performance-related guarantees with nearly 50% of the *Normal-Full-Duplex-Link* capacity. The *TDD-Link* has no *fault-tolerance ability* and utilize more hardware than the *2X-Link* e.g. in a 7×7 NoC, *TDD-Links* require an additional 4116 slices more than the *2X-Links*. Observation shows when some links fail randomly, the NoC with the *2X-Links* can recover from faults (at most 3 faults are considered) which would cause *Normal-Full-Duplex-Links* to fail with an average probability of 82.2% for the E3S benchmark, the VOPD, and the Robotic applications (the *Normal-Full-Duplex-Links* and the *TDD-Links* have no capability to recover from unforeseen link faults).

It is demonstrated that in order to achieve successful runtime adaptation, runtime observability is a must, as it provides necessary system information gathered on-the-fly. Therefore, a comprehensive runtime observability infrastructure is included in the *architecture-level* part of the proposed *AdNoC* architecture (*ROAdNoC*). It is hardly intrusive, i.e. in worst case it may require a mere 0.7% of the total link capacity. Besides the main objective of achieving flexibility in the communication architecture for higher resource utilization, the hardware overhead at *architecture-level* due to runtime observation is rather small (46 slices per router). As a result, *ROAdNoC* increases the connection success rate by 62% in average compared to state-of-the-art approaches.

The area overhead in the *architecture-level* is mainly contributed by the following parts: (1) selector required for the on-demand VCB assignment (122 slices), (2) the *wXY-routing* algorithm (129 slices), (3) the *2X-Link* management (74 slices), and (4) runtime observability infrastructure (46 slices). These area overhead those stem from the *architecture-level* adaptation can be traded-off against the flexibility to select an available route using the *2X-Links* and the on-demand buffer assignment (42% buffer saving) to that route for ensuring the QoS.

The proposed *AdNoC* architecture is capable of supporting deadlock-free data transmission and meets required bandwidth guarantees for parallel transactions. Therefore, it is built on top of the basic QoS-supported system-on-chip communication architecture. It is demonstrated that the proposed link-arbitration algorithm, *BAA*, that manages the flow-control, is actually able to provide a 100% guarantee of bandwidth with an average waste of only 3% (i.e. 97% utilization), a value that has not been achieved by others so far. The QoS-supported system-on-chip communication architecture is also used for a case-study analysis for designing an application-specific NoC. The buffer minimization methodology during application-specific NoC design shows that, on an average, a 90.2% reduction in the number of *VCBs* compared to a fixed assignment for the E3S embedded application benchmark suite may be achieved.

10.2 Future Work

The work of this thesis may be further enhanced to build a complete reliable system from forthcoming unreliable components in the late silicon era. This thesis shows the advantages of using an adaptive system-on-chip communication architecture to tackle the upcoming challenges. A statistical fault-model considering the communication behavior of the application may be developed to further facilitate the runtime adaption. An online fault detection scheme may also be attached to enhance the functionality of the *AdNoC* architecture.

Different design-related issues for optimizing the router area, such as using a decomposed strategy for VCB assignment, a faster and parallel routing hardware, etc. may be researched further to optimize the current implementation of the *AdNoC* routers. In the *system-level* task-migration issues, as well as (re-)clustering in a computationally inexpensive way may also be investigated. In the current implementation, the initial design and resource allocation for the *AdNoC* architecture have not been addressed. Therefore, an offline mechanism for characterizing the behavior of the users and then *clustering by behavior* may be explored. The clustering of user behavior may further reduce the initial hardware budget for the *AdNoC* architecture.

The NoC parameters, e.g. *topology*, *floorplanning*, and *switching strategy*, which are not customized due to the targeted technology in the current *AdNoC* architecture, may be further explored by deploying reconfigurable hardware blocks (e.g. FPGA) in a distributed way in the future MPSoC architecture.

Bibliography

[1] T. Ahonen, D. A. Sigüenza-Tortosa, H. Bin, and J. Nurmi. "Topology optimization for application-specific networks-on-chip". *SLIP'04: Proceedings of the 2004 International Workshop on System level interconnect prediction*, pages 53–60, 2004.

[2] AMBA:. More information on AMBA AXI from ARM corporation is available at:. *http://www.arm.com/products/solutions/AMBAHomePage.html*.

[3] AMBA:. Specification. *ARM Inc.*, 1999.

[4] Arteris:. More information on Arteris at:. *http://www.arteris.com/company.htm*.

[5] G. Ascia, V. Catania, and M. Palesi. "Multi-objective mapping for mesh-based NoC architectures". *CODES+ISSS'04: Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/software codesign and system synthesis*, pages 182–187, 2004.

[6] T. Austin, V. Bertacco, S. Mahlke, and Y. Cao. "Reliable Systems on Unreliable Fabrics". *IEEE Design & Test of Computers*, 25(4):322–332, 2008.

[7] Y. Aydogan, C. B. Stunkel, C. Aykanat, and B. Abali. "Adaptive Source Routing in Multistage Interconnection Networks". *IPPS'96: Proceedings of the 10th International Parallel Processing Symposium*, pages 258–267, 1996.

[8] P. Azad, A. Ude, T. Asfour, G. Cheng, and R. Dillmann. "Image-based Markerless 3D Human Motion Capture using Multiple Cues". *International Workshop on Vision Based Human-Robot Interaction*, 2006.

[9] O. Babaoglu, G. Canright, A. Deutsch, G. A. D. Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor, and T. Urnes. "Design patterns from biology for distributed computing". *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(1):26–66, 2006.

[10] J. Bainbridge and S. Furber. "Chain: A Delay-Insensitive Chip Area Interconnect". *IEEE Micro*, 22(5):16–23, 2002.

[11] A. E. Barbour and A. S. Wojcik. "A General Constructive Approach to Fault-Tolerant Design Using Redundancy". *IEEE Transaction on Computers*, 38(1):15–29, 1989.

[12] J. Becker, K. Brändle, U. Brinkschulte, J. Henkel, W. Karl, T. Köster, M. Wenz, and H. Wörn. "Digital On-Demand Computing Organism for Real-Time Systems". *ARCS'06 Workshops*, pages 230–245, 2006.

[13] L. Benini and G. D. Micheli. "Powering networks on chips: energy-efficient and reliable interconnect design for SoCs". *ISSS'01: Proceedings of the 14th International symposium on Systems synthesis*, pages 33–38, 2001.

[14] L. Benini and G. D. Micheli. "Networks on Chips: A New SoC Paradigm". *Computer*, 35(1):70–78, 2002.

[15] D. Bertozzi and L. Benini. "×pipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip". *IEEE Circuits and Systems Magazine*, 4(2):18–31, 2004.

[16] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli. "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip". *IEEE Transactions on Parallel and Distributed Systems*, pages 113–129, 2005.

[17] S. Bertozzi, A. Acquaviva, D. Bertozzi, and A. Poggiali. "Supporting task migration in multi-processor systems-on-chip: a feasibility study". *DATE'06: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 15–20, 2006.

[18] A. Bieszczad, B. Pagurek, and T. White. "Mobile Agents for Network Management". *IEEE Communications surveys and tutorials*, 1(1):2–9, 1998.

[19] T. Bjerregaard and S. Mahadevan. "A survey of research and practices of Network-on-Chip". *ACM Computing Surveys*, 38(1):1, 2006.

[20] T. Bjerregaard and J. Sparsø. "A Router Architecture for Connection-Oriented Service Guarantees in the MANGO Clockless Network-on-Chip". *DATE'05: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1226–1231, 2005.

[21] C. Bobda and A. Ahmadinia. "Dynamic Interconnection of Reconfigurable Modules on Reconfigurable Devices". *IEEE Design & Test of Computers*, 22(5):443–451, 2005.

[22] C. Bobda, A. Ahmadinia, M. Majer, J. Teich, S. P. Fekete, and J. van der Veen. "DyNoC: A Dynamic Infrastructure for Communication in Dynamically Reconfigurable Devices". *FPL'05: Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 153–158, 2005.

[23] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. "QNoC: QoS architecture and design process for network on chip". *Journal of Systems Architecture*, 50(2-3):105–128, 2004.

[24] E. Bolotin, A. Morgenshtein, I. Cidon, R. Ginosar, and A. Kolodny. "Automatic hardware-efficient SoC integration by QoS network on chip". *ICECS'04: Proceedings of the 11th IEEE International Conference on Electronics, Circuits and Systems*, pages 479–482, 2004.

[25] S. Borkar. "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation". *IEEE Micro*, 25(6):10–16, 2005.

[26] S. Borkar. "Thousand core chips: a technology perspective". *DAC'07: Proceedings of the 44th Conference on Design Automation*, pages 746–749, 2007.

[27] L. Braun, M. Hübner, J. Becker, T. Perschke, V. Schatz, and S. Bach. "Circuit Switched Run-Time Adaptive Network-on-Chip for Image Processing Applications". *FPL'07: Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 688–691, 2007.

[28] H. Broersma, D. Paulusma, G. J. M. Smit, F. Vlaardingerbroek, and G. J. Woeginger. "The Computational Complexity of the Minimum Weight Processor Assignment Problem". *WG'04: Proceedings of the 30th International Workshop on Graph-theoretic concepts in computer science*, pages 189–200, 2004.

[29] C. Busch, S. Surapaneni, and S. Tirthapura. "Analysis of link reversal routing algorithms for mobile ad hoc networks". *SPAA'03: Proceedings of the 15th annual ACM symposium on Parallel algorithms and architectures*, pages 210–219, 2003.

[30] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. "Theory of latency-insensitive design". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[31] E. Carvalho, N. Calazans, and F. Moraes. "Heuristics for Dynamic Task Mapping in NoC-based Heterogeneous MPSoCs". *RSP'07: Proceedings of the 18th IEEE International Workshop on Rapid System Prototyping*, pages 34–40, 2007.

[32] J. Chan and S. Parameswaran. "NoCGEN: A Template Based Reuse Methodology for Networks on Chip Architecture". *VLSID'04: Proceedings of the 17th International Conference on VLSI Design*, pages 717–720, 2004.

[33] J. Chan and S. Parameswaran. "NoCOUT: NoC topology generation with mixed packet-switched and point-to-point networks". *ASP-DAC'08: Proceedings of the 2008 Conference on Asia and South Pacific Design Automation*, pages 265–270, 2008.

[34] V. Chandra, A. Xu, H. Schmit, and L. Pileggi. "An Interconnect Channel Design Methodology for High Performance Integrated Circuits". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, page 21138, 2004.

[35] X. Chen and L.-S. Peh. "Leakage power modeling and optimization in interconnection networks". *ISLPED'03: Proceedings of the 2003 International symposium on Low power electronics and design*, pages 90–95, 2003.

[36] CHIPit:. More information on CHIPit is available at:. *http://www.prodesign-europe.com/*.

[37] C.-L. Chou and R. Marculescu. "Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels". *CODES+ISSS'07: Proceedings of the 5th IEEE/ACM International Conference on Hardware/software Codesign and System Synthesis*, pages 161–166, 2007.

[38] C.-L. Chou and R. Marculescu. "User-Aware Dynamic Task Allocation in Networks-on-Chip". *DATE'08: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1232–1237, 2008.

[39] C. Ciordas, T. Basten, A. Rădulescu, K. Goossens, and J. V. Meerbergen. "An event-

based monitoring service for networks on chip". *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(4):702–723, 2005.

[40] C. Ciordas, A. Hansson, K. Goossens, and T. Basten. "A monitoring-aware network-on-chip design flow". *Journal of Systems Architecture*, 54(3-4):397–410, 2008.

[41] M. Coenen, S. Murali, A. Ruadulescu, K. Goossens, and G. De Micheli. "A buffer-sizing algorithm for networks on chip using TDMA and credit-based end-to-end flow control". *CODES+ISSS'06: Proceedings of the 4th International Conference on Hardware/software codesign and system synthesis*, pages 130–135, 2006.

[42] CoreConnect:. More information on CoreConnect from IBM is available at:. *http://www.ibm.com/chips/products/coreconnect*.

[43] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. "xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs". *ICCD'03: Proceedings of the 21st International Conference on Computer Design*, page 536, 2003.

[44] W. J. Dally and B. Towles. "Route packets, not wires: on-chip interconnection networks". *DAC'01: Proceedings of the 38th Conference on Design Automation*, pages 684–689, 2001.

[45] M. Daneshtalab, A. Sobhani, A. Afzali-Kusha, O. Fatemi, and Z. Navabi. "NoC Hot Spot minimization Using AntNet Dynamic Routing Algorithm". *ASAP'06: Proceedings of the IEEE 17th International Conference on Application-specific Systems, Architectures and Processors*, pages 33–38, 2006.

[46] R. P. Dick, D. L. Rhodes, and W. Wolf. "TGFF: Task graphs for free". *CODES/CASHE'98: Proceedings of the 6th International Workshop on Hardware/software Codesign*, pages 97–101, 1998.

[47] J. Duato, S. Yalamanchili, and L. M. Ni. "Interconnection Networks: An Engineering Approach". *Morgan Kaufmann*, 2003.

[48] T. Dumitras and R. Marculescu. "On-Chip Stochastic Communication". *DATE'03: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 10790–10795, 2003.

[49] T. Dumitras and R. Marculescu. "Fault tolerant algorithms for network-on-chip interconnect". *Proceedings of the IEEE Computer society Annual Symposium on VLSI*, pages 46– 51, 2004.

[50] E3S:. More information on E3S at:. *http://ziyang.eecs.northwestern.edu/ dickrp/e3s/*.

[51] R. Esmailzade, M. Nakagawa, and E. A. Sourour. "Time-division Duplex CDMA Communications". *IEEE Wireless Communications*, 4(2):51–56, 1997.

[52] M. A. A. Faruque, T. Ebi, and J. Henkel. "Run-time Adaptive on-chip Communication Scheme". *ICCAD'07: Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, pages 26–31, 2007.

[53] M. A. A. Faruque, T. Ebi, and J. Henkel. "ROAdNoC: Runtime Observability for an Adaptive Network on Chip Architecture". *ICCAD'08: Proceedings of the 2008 IEEE/ACM International Conference on Computer-aided Design*, pages 543–548, 2008.

[54] M. A. A. Faruque, T. Ebi, and J. Henkel. "Configurable Links for Runtime Adaptive On-chip Communication". *DATE'09: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 543–548, 2009.

[55] M. A. A. Faruque and J. Henkel. "Transaction Specific Virtual Channel Allocation in QoS Supported On-chip Communication". *ASAP'07: Proceedings of the 18th International Conference on Application-specific Systems, Architectures and Processors*, pages 76–81, 2007.

[56] M. A. A. Faruque and J. Henkel. "Minimizing Virtual Channel Buffer for Routers in on-chip Communication Architectures". *DATE'08: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1238–1243, 2008.

[57] M. A. A. Faruque and J. Henkel. "QoS-Supported On-chip Communication for Multi-Processors". *IJPP'08: Proceedings of the International Journal of Parallel Programming*, pages 114–139, 2008.

[58] M. A. A. Faruque, R. Krist, and J. Henkel. "ADAM: Run-time Agent-based Distributed Application Mapping for on-chip Communication". *DAC'08: Proceedings of the 45th Conference on Design Automation*, pages 760–765, 2008.

[59] M. A. A. Faruque, G. Weiss, and J. Henkel. "Bounded arbitration algorithm for QoS-supported on-chip communication". *CODES+ISSS'06: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis*, pages 76–81, 2006.

[60] M. A. A. Faruque, X. Ye, G. Weiss, and J. Henkel. "QoS-Supported Configurable Networks on Chip". *NOCS'06: Proceedings of the Workshop in Future Interconnects and Network on Chip*, page 42, 2006.

[61] C. J. Glass and L. M. Ni. "Maximally Fully Adaptive Routing in 2D Meshes". *Proceedings of the 1992 International Conference on Parallel Processing*, I, Architecture:101–104, 1992.

[62] C. J. Glass and L. M. Ni. "The turn model for adaptive routing". *SIGARCH Computer Architecture News*, 20(2):278–287, 1992.

[63] GNU-GCC:. More information on GNU GCC is available at:. *http://gcc.gnu.org/*.

[64] G. Goldszmidt and Y. Yemini. "Delegated Agents for Network Management". *IEEE Communications Magazine*, pages 66–70, 1998.

[65] K. Goossens, J. Dielissen, and A. Radulescu. "æthereal network on chip: concepts, architectures, and implementations". *IEEE Design & Test of Computers*, 22(5):414–421, 2005.

[66] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger. "On-Chip Interconnection Networks of the TRIPS Chip". *IEEE Micro*, 27(5):41–50, 2007.

[67] C. Grecu and M. Jones. "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures". *IEEE Transaction on Computers*, 54(8):1025–1040, 2005.

[68] R. Guérin and V. Peris. "Quality-of-service in packet networks: basic mechanisms and directions". *Computer Networks*, 31(3):169–189, 1999.

[69] P. Guerrier and A. Greiner. "A generic architecture for on-chip packet-switched interconnections". *DATE'00: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 250–256, 2000.

[70] A. Hansson, K. Goossens, and A. Rădulescu. "A unified approach to constrained mapping and routing on network-on-chip architectures". *CODES+ISSS'05: Proceedings of the 3rd IEEE/ACM International Conference on Hardware/Software Codesign and System Synthesis*, pages 75–80, 2005.

[71] M. D. Harmanci, N. P. Escudero, Y. Leblebici, and P. Ienne. "Providing QoS to Connection-less Packet-switched NoC by Implementing DiffServ Functionalities". *International Symposium on System-on-Chip*, pages 37–40, 2004.

[72] M. D. Harmanci, N. P. Escudero, Y. Leblebici, and P. Ienne. "Quantitative modelling and comparison of communication schemes to guarantee quality-of-service in networks-on-chip". *ISCAS'05 (2): Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 1782–1785, 2005.

[73] J. Henkel, W. Wolf, and S. Chakradhar. "On-chip networks: A scalable, communication-centric embedded system design paradigm". *VLSID'04: Proceedings of the 17th International Conference on VLSI Design*, pages 845–851, 2004.

[74] R. Ho, K. Mai, and M. Horowitz. "The Future of Wires". *Proceedings of the IEEE*, pages 490–504, 2001.

[75] P. Horn. "Autonomic Computing: IBM's Perspective on the State of Information Technology". *IBM Corporation*, 2001.

[76] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. "A 5-GHz Mesh Interconnect for a Teraflops Processor". *IEEE Micro*, 27(5):51–61, 2007.

[77] J. Hu and R. Marculescu. "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures". *DATE'03: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 10688–10693, 2003.

[78] J. Hu and R. Marculescu. "Application-specific buffer space allocation for networks-on-chip router design". *ICCAD'04: Proceedings of the 2004 IEEE/ACM International Conference on Computer-aided Design*, pages 354–361, 2004.

[79] J. Hu and R. Marculescu. "DyAD: smart routing for networks-on-chip". *DAC'04: Proceedings of the 41st Conference on Design Automation*, pages 260–263, 2004.

[80] J. Hu and R. Marculescu. "Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 10234–10239, 2004.

[81] Y. Hu, H. Chen, Y. Zhu, A. A. Chien, and C.-K. Cheng. "Physical Synthesis of Energy-Efficient Networks-on-Chip Through Topology Exploration and Wire Style Optimizationz". *ICCD'05: Proceedings of the 2005 International Conference on Computer Design*, pages 111–118, 2005.

[82] M. Hübner, L. Braun, D. Göhringer, and J. Becker. "Run-time reconfigurable adaptive multilayer network-on-chip for FPGA-based systems". *IPDPS'08: Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing*, pages 1–6, 2008.

[83] M. Hübner, M. Ullmann, L. Braun, A. Klausmann, and J. Becker. "Scalable Application-Dependent Network on Chip Adaptivity for Dynamical Reconfigurable Real-Time Systems". *FPL'04: Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 1037–1041, 2004.

[84] iNoCs:. More information on iNoCs is available at:. *http://inocs.com/index.php?id=1*.

[85] "International Technology Roadmap for Semiconductors". http://www.itrs.net. 2007 Edition.

[86] A. Jalabert, S. Murali, L. Benini, and G. D. Micheli. "×pipesCompiler: A Tool for Instantiating Application Specific Networks on Chip". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 884–889, 2004.

[87] A. Jantsch and H. T. (Eds). "Networks-on-chip". *Kluwer*, 2003.

[88] S. Jovanovic, C. Tanougast, C. Bobda, and S. Weber. "CuNoC: A Scalable Dynamic NoC for Dynamically Reconfigurable FPGAs". *FPL'07: Proceedings of the International Conference on Field Programmable Logic and Applications*, pages 753–756, 2007.

[89] P. Kalla, X. S. Hu, and J. Henkel. "A flexible framework for communication evaluation in SoC design". *International Journal of Parallel Programming*, 36(5):457–477, 2008.

[90] W. Karl, M. Leberecht, and M. Oberhuber. "SCI Monitoring Hardware and Software: Supporting Performance Evaluation and Debugging". *SCI: Scalable Coherent Interface, Architecture and Software for High-Performance Compute Clusters*, pages 417–432, 1999.

[91] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen. "A Virtual Channel Router for On-chip Networks". *Proceedings of the IEEE International SOC Conference*, pages 289–293, 2004.

[92] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte. "A Virtual Channel Network-on-Chip for GT and BE traffic". *ISVLSI'06: Proceedings of the IEEE Computer*

Society Annual Symposium on Emerging VLSI Technologies and Architectures, pages 211–216, 2006.

[93] M. E. Kreutz, L. Carro, C. A. Zeferino, and A. A. Susin. "Communication Architectures for System-on-Chip". *SBCCI'01: Proceedings of the 14th symposium on Integrated circuits and systems design*, pages 14–19, 2001.

[94] S. Kumar, A. Jantsch, M. Millberg, J. Öberg, J.-P. Soininen, M. Forsell, K. Tiensyrjä, and A. Hemani. "A Network on Chip Architecture and Design Methodology". *ISVLSI'02: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, pages 117–124, 2002.

[95] R. Laddaga, M. L. Swinson, and P. Robertson. "Seeing Clearly and Moving Forward". *IEEE Intelligent Systems*, 15(6):46–50, 2000.

[96] H. G. Lee, N. Chang, Ü. Y. Ogras, and R. Marculescu. "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches". *ACM Transaction on Design Automation Electronics System*, 12(3), 2007.

[97] S.-J. Lee, K. Lee, and H.-J. Yoo. "Analysis and Implementation of Practical, Cost-Effective Networks on Chips". *IEEE Design & Test of Computers*, 22(5):422–433, 2005.

[98] T. Lei and S. Kumar. "A Two-step Genetic Algorithm for Mapping Task Graphs to a Network on Chip Architecture". *DSD'03: Proceedings of the Euromicro Symposium on Digital Systems Design*, pages 180–189, 2003.

[99] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. "An architecture and compiler for scalable on-chip communication". *IEEE Transaction Very Large Scale Integration (VLSI) Systems*, 12(7):711–726, 2004.

[100] G. Lipsa and A. Herkersdorf. "Towards a Framework and a Design Methodology for Autonomic SoC". *ICAC'05: Proceedings of the Second International Conference on Automatic Computing*, pages 391–392, 2005.

[101] P. Lorenz. "Quality of Service in Communication Architectures". *IMSA*, pages 270–275, 2002.

[102] Z. Lu, B. Yin, and A. Jantsch. "Connection-oriented Multicasting in Wormhole-switched Networks on Chip". *ISVLSI'06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, pages 205–210, 2006.

[103] M. Majer, C. Bobda, A. Ahmadinia, and J. Teich. "Packet Routing in Dynamically Changing Networks on Chip". *IPDPS'05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, pages 154 – 161, 2005.

[104] S. Manolache, P. Eles, and Z. Peng. "Fault and energy-aware communication mapping with guaranteed latency for applications implemented on NoC". *DAC'05: Proceedings of the 42nd Conference on Design Automation*, pages 266–269, 2005.

[105] R. Marculescu, Ü. Y. Ogras, L.-S. Peh, N. D. E. Jerger, and Y. V. Hoskote. "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives". *IEEE Transaction on CAD of Integrated Circuits and Systems*, 28(1):3–21, 2009.

[106] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, page 20890, 2004.

[107] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. "The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip". *International Conference on VLSI Design*, pages 693–696, 2004.

[108] ModelSim:. More information on ModelSim is available at:. *http://www.model.com/*.

[109] A. A. Morgan, H. Elmiligi, M. W. El-Kharashi, and F. Gebali. "Application-specific networks-on-chip topology customization using network partitioning". *IFMT'08: Proceedings of the 1st International forum on Next-generation multicore/manycore technologies*, pages 1–6, 2008.

[110] S. Murali, L. Benini, and G. Micheli. "Mapping and physical planning of networks-on-chip architectures with quality-of-service guarantees". *ASP-DAC'05: Proceedings of the 2005 Conference on Asia South Pacific Design Automation*, pages 27–32, 2005.

[111] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. D. Micheli. "A methodology for mapping multiple use-cases onto networks on chips". *DATE'06: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 118–123, 2006.

[112] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. D. Micheli. "Mapping and configuration methods for multi-use-case networks on chips". *ASP-DAC'06: Proceedings of the 2006 Conference on Asia South Pacific Design Automation*, pages 146–151, 2006.

[113] S. Murali and G. D. Micheli. "Bandwidth-Constrained Mapping of Cores onto NoC Architectures". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 20896–20901, 2004.

[114] S. Murali and G. D. Micheli. "SUNMAP: a tool for automatic topology selection and generation for NoCs". *DAC'04: Proceedings of the 41st Conference on Design Automation*, pages 914–919, 2004.

[115] Nexus:. More information on Nexus from Fulcrum Microsystems is available at:. *http://www.fulcrummicro.com/technology.htm*, 2006.

[116] L. M. Ni and P. K. McKinley. "A Survey of Wormhole Routing Techniques in Direct Networks". *IEEE Computer 26(2)*, pages 62–76, 1993.

[117] C. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers". *MICRO'06: Pro-*

ceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pages 333–346, 2006.

[118] E. Nilsson, M. Millberg, J. Oberg, and A. Jantsch. "Load Distribution with the Proximity Congestion Awareness in a Network on Chip". *DATE'03: Proceedings of the Conference on Design, Automation and Test in Europe*, 01:11126–11131, 2003.

[119] V. Nollet, T. Marescaux, P. Avasare, D. Verkest, and J.-Y. Mignolet. "Centralized Run-Time Resource Management in a Network-on-Chip Containing Reconfigurable Hardware Tiles". *DATE'05: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 234–239, 2005.

[120] V. Nollet, T. Marescaux, D. Verkest, J.-Y. Mignolet, and S. Vernalde. "Operating-system controlled network on chip". *DAC'04: Proceedings of the 41th Conference on Design Automation*, pages 256–259, 2004.

[121] OCP:. More information on OCP at:. *http://www.ocpip.org/*.

[122] U. Y. Ogras, J. Hu, and R. Marculescu. "Key research problems in NoC design: a holistic perspective". *CODES+ISSS'05: Proceedings of the 3rd IEEE/ACM/IFIP internation Conference on Hardware/software codesign and system synthesis*, pages 69–74, 2005.

[123] U. Y. Ogras and R. Marculescu. "Application-specific network-on-chip architecture customization via long-range link insertion". *ICCAD'05: Proceedings of the 2005 IEEE/ACM International Conference on Computer-aided Design*, pages 246–253, 2005.

[124] U. Y. Ogras and R. Marculescu. "Energy- and Performance-Driven NoC Communication Architecture Synthesis Using a Decomposition Approach". *DATE'05: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 352–357, 2005.

[125] OMNeT++:. More information on OMNeT++ at:. *http://www.omnetpp.org/index.php*.

[126] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. "Research Challenges for On-Chip Interconnection Networks". *IEEE Micro*, 27(5):96–108, 2007.

[127] G. Palermo, G. Mariani, C. Silvano, R. Locatelli, and M. Coppola. "A topology design customization approach for STNoC". *Nano-Net'07: Proceedings of the 2nd International Conference on Nano-Networks*, pages 1–5, 2007.

[128] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, and G. D. Micheli. "Design, Synthesis, and Test of Networks on Chips". *IEEE Design & Test of Computers*, 22(5):404–413, 2005.

[129] V. D. Park and M. S. Corson. "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks". *INFOCOM'97: Proceedings of the INFOCOM '97. 16th Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, pages 1405–1413, 1997.

[130] A. Pinto, L. P. Carloni, and A. L. Sangiovanni-Vincentelli. "Efficient Synthesis of Networks On Chip". *ICCD'03: Proceedings of the 21st International Conference on Computer Design*, pages 146–150, 2003.

[131] T. Pionteck, C. Albrecht, R. Koch, E. Maehle, M. Hübner, and J. Becker. "Communication Architectures for Dynamically Reconfigurable FPGA Designs". *IPDPS'07: Proceedings of the 21th IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2007.

[132] V. Rantala, T. Lehtonen, and J. Plosila. "Network on Chip routing Algorithms". *Turku Centre for Computer Science Report No. TUCS 779, August 2006*, 2006.

[133] E. Rijpkema, K. G. W. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip". *DATE'03: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 10350–10355, 2003.

[134] L. G. Roberts. "The Evolution of Packet Switching". *Proceedings of IEEE*, 66:1307–1313, 1978.

[135] D. R. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar. "An Asynchronous Router for Multiple Service Levels Networks on Chip". *ASYNC'05: Proceedings of the 11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 44–53, 2005.

[136] A. Rădulescu, J. Dielissen, K. Goossens, E. Rijpkema, and P. Wielage. "An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration". *DATE'04: Proceedings of the Conference on Design, Automation and Test in Europe*, page 20878, 2004.

[137] S. Russ, J. Robinson, M. Gleeson, and J. Figueroa. "Dynamic Communication Mechanism Switching in Hector". *Mississippi State Technical Report No. MSSU– EIRS–ERC–97–8, September 1997. 17*, 1997.

[138] R. H. S. Kubisch and D. Timmermann. "Adaptive Hardware In Autonomous And Evolvable Embedded Systems". *Proceedings of the embedded world 2006 Conference*, pages 297–306, 2006.

[139] I. Saastamoinen, M. Alho, and J. Nurmi. "Buffer implementation for Proteo network-on-chip". *ISCAS'03: Proceedings of the International Symposium on Circuits and Systems*, pages 113–116, 2003.

[140] F. A. Samman, T. Hollstein, and M. Glesner. "Multicast parallel pipeline router architecture for network-on-chip". *DATE'08: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1396–1401, 2008.

[141] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri. "LiPaR: A light-weight parallel router for FPGA-based networks-on-chip". *GLSVSLI'05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pages 452–457, 2005.

[142] L. Shang, L.-S. Peh, and N. K. Jha. "PowerHerd: dynamic satisfaction of peak power constraints in interconnection networks". *ICS'03: Proceedings of the 17th annual International Conference on Supercomputing*, pages 98–108, 2003.

[143] D. Shin and J. Kim. "Power-aware communication optimization for networks-on-chips with voltage scalable links". *CODES+ISSS'04: Proceedings of the 2nd IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis*, pages 170–175, 2004.

[144] D. Sigüenza. "PROTEO: The Development of a Practical Network-on-Chip". *PhD Thesis: Tampere University of Technology*, 2005.

[145] D. A. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi. "Issues in the development of a practical NoC: the Proteo concept". *Integration*, 38(1):95–105, 2004.

[146] Silistix:. More information on Silistix is available at:. *http://www.silistix.com/*.

[147] L. Smit, G. Smit, J. Hurink, H. Broersma, D. Paulusma, and P. Wolkotte. "Run-time Mapping of Applications to a Heterogeneous Reconfigurable Tiled System on Chip Architecture". *FPL'04: Proceedings of the IEEE International Conference on Field-Programmable Technology*, pages 421–424, 2004.

[148] P. Smith and N. C. Hutchinson. "Heterogeneous Process Migration: The Tui System". *Software – Practice and Experience*, 28(6):611–639, 1998.

[149] SonicsMX:. More information on SonicsMX from Sonics Inc. is available at:. *http://www.sonicsinc.com/sonicsMX.htm*.

[150] V. Soteriou and L.-S. Peh. "Design-Space Exploration of Power-Aware On/Off Interconnection Networks". *ICCD'04: Proceedings of the IEEE International Conference on Computer Design (ICCD'04)*, pages 510–517, 2004.

[151] V. Soteriou, H. Wang, and L.-S. Peh. "A Statistical Traffic Model for On-Chip Interconnection Networks". *MASCOTS'06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 104–116, 2006.

[152] K. Srinivasan and K. S. Chatha. "A low complexity heuristic for design of custom network-on-chip architectures". *DATE'06: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 130–135, 2006.

[153] K. Srinivasan, K. S. Chatha, and G. Konjevod. "Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures". *ICCD'04: Proceedings of the IEEE International Conference on Computer Design*, pages 422–429, 2004.

[154] K. Srinivasan, K. S. Chatha, and G. Konjevod. "Linear-programming-based techniques for synthesis of network-on-chip architectures". *IEEE Transaction on Very Large Scale Integration Systems (TVLSI)*, 14(4):407–420, 2006.

[155] STBus:. More information on STBus from STMicroelectronics is available at:. *http://www.st.com/stonline/products/technologies/soc/stbus.htm*.

[156] M. B. Stensgaard and J. Sparsø. "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology". *NOCS'08: Second International Symposium on Networks-on-Chips*, pages 55–64, 2008.

[157] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G. D. Micheli. "Xpipes lite: A synthesis oriented design library for networks on chips". *DATE'05: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1188–1193, 2005.

[158] STNoC:. "STNoC: Building a New System-on-Chip Paradigm". 2005.

[159] T. Stutzle and M. Dorigo. "ACO algorithms for the quadratic assignment problem". *New Ideas in Optimization*, pages 33–50, 1999.

[160] D. Subramanian, P. Druschel, and J. Chen. "Ants and reinforcement learning: A case study in routing in dynamic networks". *IJCAI'97: Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 832–838, 1997.

[161] SYSTEMC:. More information on SYSTEMC at:. *http://www.systemc.org/home*.

[162] A. S. Tanenbaum. "Computer Networks". *Prentice Hall*, 2003.

[163] R. F. Tinder. "Engineering Digital Design: Revised Second Edition". *Academic Press*, 2000.

[164] D. S. Tortosa and J. Nurmi. "Proteo: A New Approach to Network-on-Chip". *International Conference on Communication Systems and Networks CSN'02)*, pages 9–12, 2002.

[165] J. W. van den Brand, C. Ciordas, K. Goossens, and T. Basten. "Congestion-controlled best-effort communication for networks-on-chip". *DATE'07: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 948–953, 2007.

[166] G. Varatkar and R. Marculescu. "Traffic analysis for on-chip networks design of multimedia applications". *DAC'02: Proceedings of the 39th Conference on Design Automation*, pages 795–800, 2002.

[167] G. V. Varatkar and R. Marculescu. "On-chip traffic modeling and synthesis for MPEG-2 video applications". *IEEE Transaction on Very Large Scale Integration System (TVLSI)*, 12(1):108–119, 2004.

[168] E. G. Varthis and D. I. Fotiadis. "A comparison of Stop-and-Wait and Go-Back-N ARQ Schemes for IEEE 802.11e Wireless Infrared Networks". *Computer Communications*, 29(8):1015–1025, 2006.

[169] S. Vassiliadis and I. Sourdis. "FLUX Networks: Interconnects on Demand". *Proceedings of the Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 160–167, 2006.

[170] P. Vellanki, N. Banerjee, and K. S. Chatha. "Quality-of-service and error control techniques for mesh-based network-on-chip architectures". *Integration, the VLSI Journal*, 38(3):353–382, 2005.

[171] VHDL:. A sample tutorial on VHDL is available at:. *http://www.vhdl-online.de/*.

[172] J. Wang, H. Zeng, K. Huang, G. Zhang, and Y. Tang. "Zero-efficient buffer design for

reliable network-on-chip in tiled chip-multi-processor". *DATE'08: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 792–795, 2008.

[173] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. B. III, and A. Agarwal. "On-Chip Interconnection Architecture of the Tile Processor". *IEEE Micro*, 27(5):15–31, 2007.

[174] P. Wielage and K. Goossens. "Networks on Silicon: Blessing or Nightmare?". *DSD'02: Proceedings of the Euromicro Symposium on Digital Systems Design*, pages 196–200, 2002.

[175] D. Wingard. "Micronetwork-based integration for SOCs". *DAC'01: Proceedings of the 38th Conference on Design Automation*, pages 677–681, 2001.

[176] P. T. Wolkotte, G. J. M. Smit, G. K. Rauwerda, and L. T. Smit. "An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip". *IPDPS'05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 3*, pages 155–162, 2005.

[177] Xilinx:. "More information on Xilinx at". *http://www.xilinx.com/*.

[178] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar. "A design methodology for application-specific networks-on-chip". *Transaction on Embedded Computing Systems (TECS)*, 5(2):263–280, 2006.

[179] T. T. Ye and G. D. Micheli. "Physical Planning for Multiprocessor Networks and Switch Fabrics". *ASAP'03: International Conference on Application-Specific Systems, Architectures and Processors*, pages 97–107, 2003.

[180] K. H. Yum, E. J. Kim, C. R. Das, M. Yousif, and J. Duato. "Integrated Admission and Congestion Control for QoS Support in Clusters". *CLUSTER'02: Proceedings of the IEEE International Conference on Cluster Computing*, pages 325–332, 2002.

[181] H. ZHANG. "Service Disciplines for Guaranteed Performance Service in Packet-Switching Networks". *Proceedings of the IEEE*, pages 1374–1396, 1995.