Faizal Arya Samman

Microarchitecture and Implementation of Networks-on-Chip with a Flexible Concept for Communication Media Sharing

TECHNISCHE UNIVERSITÄT DARMSTADT
FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK

D17
Darmstädter Dissertationen
2010

SAMMAN, Faizal Arya.
"Microarchitecture and Implementation of Networks-on-Chip
with a Flexible Concept for Communication Media Sharing".
Technische Universität Darmstadt, Dissertation (Doctoral Thesis), 2010.

Keywords:

Network-on-Chip,
VLSI Microarchitecture,
Wormhole Cut-Through Switching,
Theory for Deadlock-Free Multicast Routing,
Tree-based Multicast Routing,
Runtime Adaptive Routing Selection Strategy,
Contention- and Bandwidth-Aware Adaptive Routing,
Connection-Oriented Communication,
Guaranteed-Bandwidth Service.

Published in Germany.

Microarchitecture and Implementation of Networks-on-Chip with a Flexible Concept for Communication Media Sharing

Vom Fachbereich 18
Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung der Würde eines
Doktor–Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

M.T.
Faizal Arya Samman
geboren am 05. 06. 1975
in Makassar, Indonesien

Referent:	Prof. Dr. Dr. h. c. mult. Manfred Glesner *Technische Universität Darmstadt*
Korreferent:	Prof. Dr.-Ing. Hans Eveking *Technische Universität Darmstadt*
Tag der Einreichung:	26. 01. 2010
Tag der mündlichen Prüfung:	08. 06. 2010

D17

Darmstadt 2010

for my lovely Mother,

my Father,

and my Family

Acknowledgment

This thesis is based on my work that I have started in October 2006 at *Fachgebiet Mikroelektronische Systeme*, *Institut für Datentechnik*, *Fachbereich Elektrotechnik und Informationstechnik*, *Technische Universtät Darmstadt* as a research assistant with a scholarship from DAAD (*Deutscher Akademischer Austausch-Dienst*). Therefore, I would like to thank the DAAD for awarding me the scholarship to pursue my doctoral degree. Special thanks are due to my advisor Prof. Dr. Dr. h.c. mult. Manfred Glesner for his advices, guidances and nice working environment. The colourful working atmosphere at his research institute reflects his internationally open personality, quality and care, from which I have benefited.

As my co-advisor, I express my acknowledment to Prof. Dr.-Ing. Hans Eveking for his supports and advices. I would like to thank Prof. Dr.-Ing. Gerd Balzer, Prof. Dr.-Ing. habil. Roland Werthschützky and Prof. Dr.rer.nat. Andy Schürr for spending time to be the committee of my oral examination. I would like to acknowledge Prof. Dr. Ir. Nadjamuddin Harun, Ir. Rhiza S. Sadjad, MSEE, PhD and Ir. Eniman Y. Syamsuddin, MSc, PhD for their recommendations to pursue higher educational degree that is useful for my academical career. Furthermore, I would also like to thank Prof. Dr.-Ing. Holger Hanselka and Prof. Dr.-Ing. Thilo Bein as the project coordinator and manager of the AdRIA (Adaptronik–Research, Innovation, Application) Project, in which I can continue my post-doctoral research experience at *Technische Universtät Darmstadt* in collaboration with *LOEWE-Zentrum, Fraunhofer Institut für System-Zuverlässigkeit und Betriebfestigkeit*.

I gratefully acknowledge Dr.-Ing. Thomas Hollstein who has spent much time with me to discuss and share technical knowledge about network-on-chip (NoC) topic with a fruitful discussion until I can really understand the topic which is a new topic for me as I just came to the *Fachgebiet Mikroelektronische Systeme*. I thank Prof. Dr.-Ing. Peter Zipf and Dr.-Ing. Leandro S. Indrusiak for the time to discuss about NoC topic in the particular area of system-level design, as well as to all anonymous reviewers of my journal and conference papers for the positive critics and suggestions.

Many thanks are due to the former staff members at *Fachgebiet Mikroelektronische Systeme*, Dr.-Ing. Tudor Murgan, Dr.-Ing. Oliver Soffke, Dr.-Ing. Oana M. Cobianu, Dr.-Ing. Andre Guntoro, Dr.-Ing. Massoud Momeni, Wang Hao, Heiko Hinkelmann as well as to the current staff members, Petru Bacinschi, Enkhbold Ochirsuren, Christopher Spies, Hans-Peter Keil, Surapong Pongyupinpanich, François Philipp, Ping Zhao, Leandro Möller and Sebastina Pankala for the friendships and cooperations. I would also like

to express my appreciation to the staff members at *Fachgebiet Integrierte Elektronische Systeme*, the head of the institute Prof. Dr.-Ing. Klaus Hofmann and his research and teaching assistant staff. My acknowledgements are granted to Andreas Schmidt and Roland Brand for helping me in many thing about software and hardware matters, and to Silvia Hermann and Iselona Klenk for helping me in many administrative matters.

I express my gratitude to all my supervised students, Youness Sennani, Lufei Shen, Jonatan Antoni, Souhaili Rhazi, Shengtian Le, Björn Dollak and Florian B. Luley, who have made nice cooperations with me in the framework of bachelor/master/diploma thesis. Thank you very much also to Abdul Hakim Hamid for proof-reading and correcting this written-english thesis. My stay in Darmstadt is enhanced by many friends. For the fruitful friendships, I appreciate all my Colleagues from Indonesia, Germany and from all other countries that I could not mention them in this pages.

I deeply acknowledged all my teachers in my primary school, secondary school and my high school in Sungguminasa, Gowa, as well as my lecturers at *Universitas Gadjah Mada* in Yogyakarta and at *Institut Teknologi Bandung* for providing me with basic and advanced knowledge. Many thanks are also given to all teaching, technical and administrative staff members at *Universitas Hasanuddin* in Makassar for their helps and administrative supports.

From the depth of my heart, I am grateful to my lovely mother and my lovely father for their patience to advice and educate me. Their love, care, dedication and their long-life educational supports cannot be expressed with words. I pray for them to be blessed, and their prayer is a strong motivation for me to make them proud. I am also deeply grateful to my lovely wife Wahyuni Sirajuddin for her love, supports and her best care. Special thanks are deeply due to my lovely children Syifa Marabintang, Imam Manggarai and Alya Deapati for their love and patience, for having a busy father. I would also like to thank my brothers Zulfikar Adijana, Tajul Arifin, and my Sisters Azmi Adiarti, Aida Mardiah, Yuyun Zulaena, as well as all my relationships for their supports and for taking care well our lovely mother during her life.

Last but not least, I thank God (*Allah*), All-Mighty, The Most Merciful, the Supreme in knowledge and power, and the Creator of the universe, Whom I have seeked His knowledge and His guidance from and to Whom He has given us life in this world so as to be beneficial to humanity before we go to an eternity life.

Darmstadt, July 2010

Faizal Arya Samman

Abstract

This thesis proposes a concept, VLSI microarchitecture and implementation of a network-on-chip (NoC) supporting a flexible communication media share methodology. The concept and methodology are based on a variable dynamic local identity tag (ID-tag) management technique, where different messages can be interleaved at flit-level on the same communication channel. Each message is multiplexed and allocated to a local ID slot on the shared channel. In order to implement the concept and methodology, a special packet format will be introduced, where additional two control bit fields, i.e. an ID-tag field and a flit-type field, are attached on every flit of the message in line with a data word. The reserved ID slot number, to which the message is allocated, is attributed in the ID-tag field. The flit-type field together with the ID-tag field is used to identify the messages and the type of every message flit, and to control the behavior of certain components in the NoC switch (NoC router) at runtime (during application execution time). The type of the flits is classified into a header used to open the ID-tag reservation, a databody, or a tail flit that is used to terminate the ID-tag reservation.

When entering a new communication channel, the ID-tag of a message is updated. Each message is allocated to a new local ID slot and organized in such a way, that flits belonging to the same message will have the same ID-tag on every communication channel. Therefore, an ID management unit is integrated in a switch multiplexor component at every output port of the NoC router to organize the ID-tag reservation or the ID slot allocation procedure. In order to guarantee a correct routing path configuration at runtime, a routing engine component consisting of a routing state machine and a routing reservation table is implemented on each input port. The routing engine routes the interleaved different messages based on their ID-tag.

The proposed concept and methodology have impacts on the implementation of advantageous and extensive features in the NoC router compared with the existing NoC concepts presented in the literature. The basic advantageous application of the proposed concept and methodology is the ability to implement a new wormhole switching method called *wormhole flit-level cut-through switching method* to overcome the head-of-line blocking problems commonly occur when using traditional wormhole switching method. The problem is solved by allowing the flits of the competing wormhole messages to be interleaved at flit level in the same communication link without using virtual channels.

The proposed concept allows us to implement a new deadlock-free tree-based mul-

ticast routing methodology with static or adaptive routing algorithm, where the routing engines used to route the unicast and multicast messages are the same, resulting in a low-area overhead multicast routing engine. The thesis introduces also a *new theory for deadlock-free multicast routing* suitable for NoCs. The theory is formulated based on a new simple and smart mechanism to handle multicast contentions called *hold-release tagging mechanism*. The multicast deadlock configuration problem in the tree-based multicast routing is solved without the use of virtual channels.

Beside (1) the new wormhole switching method and (2) the new deadlock-free multicast routing method mentioned before, the proposed concept allow us, (3) to develop a new adaptive routing selection strategy (*contention- and bandwidth-aware adaptive routing selection strategy*), (4) to develop a switched virtual circuit configuration method based on the ID-division multiple access technique for implementing a runtime connection-oriented guaranteed-bandwidth service, and (5) to combine the connectionless best-effort and the connection-oriented guaranteed-bandwidth services in a single NoC router prototype.

This doctoral thesis introduces in general a NoC router prototype called **XHiNoC** (e**X**tendable **H**ierarchical **N**etwork-**o**n **C**hip). The VLSI microarchitecture of the XHiNoC routers is flexible and extendable, in which the generic components of the NoC router can be simply replaced by extended components. If needed, a number of new signal paths is added. Hence, a new NoC router prototype with the aforementioned extensive services, such as adaptive routing service, multicast routing service and connection-oriented guaranteed-bandwidth service can be designed from the basic VLSI microarchitecture of the XHiNoC Router.

Kurzfassung

Diese Dissertation stellt ein Konzept für eine VLSI-Mikroarchitektur und Implementierung eines On-Chip Netwerks vor, welches eine flexible Nutzung von Routing-Ressourcen unterstützt. Das Konzept und die Methodologie basieren auf einer variablen lokalen Identität (ID-Tag) von Datenpaketen auf den einzelnen Segmenten des Routing-Netzwerks, wodurch eine gleichzeitige gefaltete Übertragung (Interleaving) mehrerer Datenpakete auf einem Datensegment ermöglicht wird. Hierfür wurde ein spezielles Format für die Flits eines Datenpakets entworfen, welches als zusätzliche Steuerungsinformationen für den Datenfluss Informationen über den Pakettyp und die lokale ID (ID-Tag) des Pakets auf dem aktuell betrachteten Routing-Segment enthält. Mit diesen Zusatzinformationen wird das Verhalten der On-Chip Router des Network-on-Chip (NoC) lokal gesteuert. Datenpakete bestehen aus Datenheader, Payload und einem Deskriptor für das Ende des Pakets.

Wenn ein Router einen Datenheader empfängt, trifft die aus einer kombinierten Routing-Zustandsmaschine und einer Routingtabelle bestehende Routing-Engine am Eingang des Routers eine Entscheidung für die Richtung der Weiterleitung des Pakets und ein eingebauter ID-Manager ordnet dem Paket eine freie ID auf dem ausgehenden Routing-Segment zu. Alle nachfolgenden Flits des Datenpakets werden über deren ID auf dem Dateneingangssegment erkannt, automatisch auf dieses Ausgangs-Routingsegment geschaltet (Switching) und mit derselben lokalen ID versehen, die auch dem Header des Pakets zugeordnet wurde. Passiert ein Paketende einen Router, so wird die für das Paket verwendete lokale ID wieder freigegeben. Die ID-Verwaltung und -Vergabe wird mit Hilfe eines Steuerungsmoduls realisiert, das in den Multiplexern des On-Chip Routers integriert ist.

Im Vergleich zu Konzepten der anderer bekannter On-Chip Netzwerke, haben das vorgeschlagene Konzept und die implementierte Methodologie Vorteile im Hinblick auf eine effiziente Implementierung der On-Chip-Router. Basierend auf dem vorgeschlagenen Konzept können Wormhole-Routingverfahren mit wesentlicher Reduktion der hierbei üblichen Warteschlangen-Blockierungsprobleme implementiert werden. Flits verschiedener Datenpakete können auf dem gleichen Übertragungskanal gemischt werden, wofür keine expliziten virtuellen Datenkanäle benötigt werden, was zu einer erheblichen Reduktion der Größe von Datenpuffern führt.

Das vorgeschlagene Konzept ermöglicht es weiterhin, eine blockierungsfreie Multicast-

Routingsmethode mit einem statischen oder adaptiven Routingsalgorithmus zu implementieren. In der vorliegen Dissertationsschrift wird auch eine neue blockierungsfreie Multicastroutingstheorie vorgestellt, die sich für On-Chip Netzwerke eignet und auf dem beschriebenen Grundkonzept basiert. Die hiervon abgeleitete Methodik ist als sogenannter "Hold-Release-Tagging-Mechanismus" implementiert, und löst das Problem einer möglichen gleichzeitigen Konkurrenz von Datenwörtern um eine bestimmte Ausgangsressource eines Netzwerkrouters. Letzteres stellt insbesondere bei Multicast-Datenkommunikation ein schwerwiegendes Problem dar und wird hier ohne Verwendung virtueller Kanäle gelöst.

Neben (1) der neuen Wormhole-Paketvermittlungsmethode und (2) dem neuen Multicastroutingverfahren, wird unter Verwendung des vorgeschlagenen Konzepts ermöglicht, (3) eine lastabhängige Bandbreitensteuerung lokal in den Routern vorzunehmen, (4) virtuelle Leitungsverbindungen ohne Notwendigkeit einer zentralen Steuereinheit für die Bereitstellung einer verbindungsorientierten Datenübertragung (Quality of Service) bereitzustellen, und (5) kombinierte verbindungslose (Datenpakete) und verbindungsorientierte (Streaming) Datenübertragung in einem on-Chip Router zu implementieren.

In dieser Dissertation wird die NoC-Prototypenarchitektur XHiNoC (eXtendable Hierarchical Network-on Chip) vorgestellt. Die VLSI-Mikroarchitektur des XHiNoC-Routers ist flexibel und erweiterbar, wobei die generische Komponenten des NoC Routers einfach mit anderen erweiterten Komponenten ersetzt werden können. Daher können problemlos neue NoC-Routerprototypen generiert werden, die zusätzliche Dienste, wie die oben genannten Dienste (adaptive Routingsverfahren, Multicast-Routingsverfahren oder verbindungsorientierte Datenübertragung mit Bandbreitengarantie) bereitstellen. Basierend auf der XHiNoC-Architektur können in kürzester Zeit bedarfsangepasste Router erstellt werden, die nur die benötigten Dienste bei minimiertem Overhead abbilden. Da die ganze Architektur auf einen globalen Controller verzichtet, ist sie beliebig skalierbar. Aufgrund der verwendeten Steuerung der Dateninjektion werden die im Netzwerk injizierten Datenmengen der verfügbaren Übertragungskapazität angepasst, womit eine wesentliche Reduktion der Datenpuffer und routerinternen Steuerungsmechanismen einhergeht.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abbreviations

2D	:	2-dimension
3D	:	3-dimension
API	:	Application Programming Interface
Arb	:	Arbiter/Arbitration Unit
ASIC	:	Application-Specific Integrated Circuit
ARM	:	Advanced RISC Machine
BE	:	Best-Effort
BW	:	Bandwidth
BWA	:	Bandwidth-Aware
BWS	:	Buffered Wormhole Switching
CAD	:	Computer-Aided Design
CBWA	:	Contention- and Bandwidth-Aware
CCA	:	Contention- and Congestion-Aware
CDMA	:	Code-Division Multiple Access
CI	:	Congestion Information
CMOS	:	Complementary Metal Oxide Silicon
CMP	:	Chip-Level Multiprocessor
CPU	:	Central Processing Unit
CRC	:	Cyclic Redundancy Check
DSM	:	Distributed Shared Memory
DSP	:	Digital Signal Processor
DVB	:	Digital Video Broadcasting
FIFO	:	First-In First Out
fpc	:	flits per cycle
FPGA	:	Field Programmable Gate Array
GALS	:	Globally-Asynchronous Locally-Synchronous
GB	:	Guaranteed-Bandwidth
GPU	:	Graphics Processing Unit
GS	:	Guaranteed-Service
GT	:	Guaranteed-Throughput

HPF : High Performane Fortran
HPC : High Performane Computing
IC : Integrated Circuit
IEEE : Institute of Electrical and Electronics Engineers
IDMA : Identity-Division Multiple Access
ID-tag : Identity-tag
IP : Intelectual Property
ITRS : International Technology Roadmap for Semiconductors
LC : Link Controller
MAC : Medium Access Control
MCU : Micro-Controller Unit
MIM : Multiplexor with ID Management Unit
MIPS : Microprocessor without Interlocked Pipeline Stages
MPI : Message Passing Interface
MPSoC : Multiprocessor System-on-Chip
NegF : Negative First
NI : Network Interface
NL : North-Last
NoC : Network-on-Chip
OCNI : On-Chip Network Interface
OpenMP : Open Multi Processing
OSI : Open System Interconnection
PCS : Pipelined Circuit Switching
PE : Processing Element
PVM : Parallel Virtual Machine
QoS : Quality of Service
RAM : Random Access Memory
RE : Routing Engine
REB : Routing Engine with Data Buffering
RFIC : Radio Frequency Integrated Circuit
RISC : Reduced Instruction Set Computing
RSM : Routing State Machine
RRT : Routing Reservation Table
RTL : Register Tansfer Level

SAF	:	Store-And-Forward
SEU	:	Single-Event Upset
SMT	:	Symmetric Multi-Threading
SDMA	:	Spatial-Division Multiple Access
SoC	:	System-on-Chip
SVC	:	Switched-Virtual Circuit
TDMA	:	Time-Division Multiple Access
TPG	:	Traffic Pattern Generator
TRE	:	Traffic Response Evaluator
TSV	:	Through Silicon Via
ULSI	:	Ultra Large Scale Integration
VC	:	Virtual Channel
VCT	:	Virtual Cut-Through
VLSI	:	Very Large Scale Integration
WF	:	West-First
WiFi	:	Wireless Fidelity
WiMAX	:	Worldwide Interoperability for Microwave Access
XHiNoC	:	eXtendable Hierarchical Network-on-Chip

Symbols

B_{max}	:	Maximum bandwidth capacity of a link
$F_n(t, i)$	:	A flit with type t and ID-tag i from input port n
b_{type}	:	bit-width for the type field in each flit
b_{tag}	:	bit-width for the ID-tag field in each flit
b_{word}	:	bit-width for the dataword field in each flit
N_{hf}	:	Number of header flits in a multicast message
N_{dest}	:	Number of destination nodes for a multicast message
N_{node}	:	Number of router node in a NoC
$\Re$	:	Set of routers $\{R_1, R_2, \cdots, R_{N_{node}}\}$
R_c	:	Router node c, where $R_c \in \Re$
Λ	:	Set of communication links in a NoC
$L_{i,j}$	:	Link connecting R_i to R_j, where $L_{i,j} \in \Lambda$
N_{inp}	:	Number of input ports in a router
N_{outp}	:	Number of output ports in a router
Φ	:	Set with input port elements $\{1, 2, \cdots, N_{inp}\}$
φ	:	Set with output port elements $\{1, 2, \cdots, N_{outp}\}$
n	:	Input port number, $n \in \Phi$
m	:	Output port number, $m \in \varphi$
$N^{req}_{s,m}$	:	Number of requests to acquire output port m
$N^{req}_{s,n}$	:	Number of requests from input port n
φ^{req}_n	:	Set of active routing requests from an input port n
Φ^{req}_m	:	Set of active routing requests to an output port m
N_{slot}	:	Number of ID Slot on every link
Γ	:	Set of elements $\{0, 1, 2, \cdots, N_{slot} - 1\}$
ε_{type}	:	Set of flit type element $\{header, databody, tail, response\}$
$type$	:	Type of a flit, $type \in \varepsilon_{type}$
Ω	:	Set of ID slots elements $\{0, 1, 2, \cdots, N_{slot} - 1\}$
ID, k	:	Local ID-tag and ID slot, $k \in \Omega$, $\Omega \subseteq \Gamma$

f_{RSM}	:	Routing Function made by a Routing State Machine
f_{IDM}	:	ID-tag Update Function made by an ID-Management Unit
ID_{old}	:	old/previous ID-tag
ID_{new}	:	new ID-tag after update function
N_{usedID}	:	Number of used/reserved ID slot in the ID Slot Table
N_{usedBW}	:	Number of used/reserved bandwidth on an outgoing link
r_{dir}	:	Routing direction (decision)
A_{dest}	:	Address of a destination node
$R(t)$	:	Routing Request Matrix
$r_{n,m}(t)$	:	Matrix element of $R(t)$
$A(t)$	:	Arbitration (Routing Acknowledge) Matrix
$T_{s,m}$	:	Rotating arbitration time at output port m
$a_{n,m}(t)$	:	Matrix element of $A(t)$
$R^*(t)$	:	Tagged Request Matrix
$r^*_{n,m}(t)$	:	Matrix element of $R^*(t)$
S^k	:	State of Id Slot k, where $k \in \Omega$
$S(k)$	:	ID Slot Table
$T(k)$	:	Routing Reservation Table with slot number k
M_n	:	Routing Machine at input port n
T_n	:	Routing Reservation Table at input port n
E_n	:	Routing Engine at input port n
$usedID(m)$	:	Number of reserved ID at output port m
$usedBW(m)$	:	Number of reserved bandwidth at output port m

Chapter 1

Introduction and Overview

Contents

1.1 Background and Motivations

According to the International Technology Roadmap for Semiconductors (ITRS) [105], the transistor feature size will be smaller in submicron (nanometer) scale and integrated circuits operate below one volt. Since the feature size of newer technology is smaller, the integrated circuits using this new technology can then be clocked faster. The smaller transistor feature size also enables the integration of more tansistors on a single die. The challenges related to the progress of the advanced technology are design concepts and design methodologies that can make use of such new technology. The most attractive thing of the new and smaller technology is the reduced cost. As the technology evolves toward the production of larger and larger circuit functions on a single die and unit cost falls as the number of component per circuit rises, then the cost advantage will continue to increase [160].

System-on-chip (SoC) design methodology is one of the potential solutions for system level design. The SoC design method is based on design reuse method which is acceptable in industry and compatible with industrial standard computer-aided design (CAD) tools. As the feature size of a CMOS technology decreases, the working frequency of the SoC system can be increased in order to improve the system performance. However, this popular technique has run out of steam, due to excessive power consumption, heat dissipation and electro-migration reliability issues [54]. Hence, solving a very complex computation by participating more computing elements will be a preferable solution. SoC architecture paradigm will potentially move from single processing element to multiple

processing elements [31], which is called as a multiprocessor SoC (MPSoC).

Traditionally, a SoC or an MPSoC system interconnects intellectual properties (IP) components by using a *bus-based interconnect system*. When the number of participating components is more than ten, then the bus system will have a performance bottleneck problem [106]. In order to solve the performance bottleneck problem, a *fully crossbar interconnect* can be used. However, this approach will implicate a wiring complexity in the circuit, in which wires could be more dominant than the logic parts, especially when the number of the interconnected components is very high. Another problem in the fully crossbar interconnect is the effect of electromagnetic interference that can disturb the interconnect functionality. A *point-to-point interconnect* (*dedicated wires*) is also another alternative solution to the performance bottleneck problem and to the wiring complexity problem. However, this approach is not flexible. Instead of connecting the top-level components by routing the dedicated wires, an *on-chip interconnection network* can be implemented and interconnect the interacting components by routing packets through the network [59].

Since interconnect technology affects more profoundly on chip performance and power usage, improving on-chip communication technology has become increasingly important to researchers and processor manufacturers [78]. A high-throughput communication infrastructure is required to meet the bandwidth requirement of each data communication flows generated due to interacting processors in the MPSoC systems. This issue can be potentially handled by a communication infrastructure based on the *network-on-chip* (NoC), which has better scalability to provide sufficient communication bandwidth.

On-chip network infrastructure also enables advanced intellectual properties (IP) communication concepts for MPSoC. In embedded MPSoC systems, NoCs can provide a flexible communication infrastructure, in which several components such as microprocessor cores, MCU, DSP, GPU, memories and other intellectual property (IP) components can be interconnected by using reusable NoC routers via general modular interfaces. The MPSoC systems can also be reconfigured for a certain embedded computing application and can be customized to improve the communication performance in the application. Hence, the NoC-based systems combine performance with design modularity [176]. The innovation of a flexible NoC communication infrastructure will enable accordingly the IP vendors to sell not only their IP components but also a system architecture [54].

The main component of the NoC system is an *on-chip router* (*switch*). Research in the field of off-chip interconnection network is not a new activity. The off-chip interconnection network has been a mature technology. However, there are some issues that should be addressed regarding the adoption of the "off-chip network" concepts into the "on-chip network" implementations. We are sure that the new innovations related to switching method, adaptive routing algorithm, network flow control and buffering scheme suitable for NoCs are still required. Until now, there is no standard for the NoC architecture similar to that of the internet world. This thesis is motivated to provide a new switching method, new adaptive routing strategies and a new deadlock-free theory and methodol-

ogy for tree-based multicast routing and its VLSI implementation, which are in any case different from the existing methods mentioned in the literature and suitable for NoCs.

1.2 Research Scope and Objectives

The research scope of this thesis are the concept of VLSI architecture and implementation of on-chip routers with advantageous features and characteristics to develop networks-on-chip for multiprocessor systems. Since the main focus of the research is the NoC routers design concept, then this thesis will discuss some issues and aspects of the NoC router architecture and its supporting modular components. Therefore some topics such as switching method, routing algorithm, network flow control, and the internal NoC router pipeline microarchitecture including its pipeline control are the main scopes of this thesis.

The research experiments on the NoC-based multiprocessor systems equipped with a programming model, and application programming interface (API) of the multiprocessor system with distributed memory architecture are part of the research interests conducted in our institute. However, the designs of NoC-based multiprocessor systems, on-Chip Network Interface (OCNI), and parallel programming models are beyond the scope of this thesis.

The general objective of this doctoral thesis is to present a design concept and generic architecture of a NoC prototype with specific features supporting specific services. The specific objectives of this thesis are intended to improve the existing methodology, design concepts and characteristics of NoC routers that have been developed so far in the NoC research area. The specific objectives are:

- to present a new wormhole switching method [223] [229], [237] and to show theoretically the advantageous characteristics compared to traditional wormhole switching, in which the head-of-line-blocking problem is solved without using virtual channels,
- to present a new theory for deadlock-free multicast routing algorithm [234] and to show the advantageous characteristics and VLSI implementations [224], [227], [232] compared to existing deadlock-free multicast routing methods, in which the multicast dependency (contention) problem is solved without the use of virtual channels,
- to present a new approach to design runtime adaptive routing selection strategies based on contention and bandwidth information or combination of both information, and to show their advantageous performance characteristics compared to other adaptive routing selection strategies presented in the literatures [236],
- to present a new and more flexible Switched-Virtual Circuit (SVC) configuration method to design a NoC router with connection-oriented guaranteed-bandwidth

service and to show the advantageous VLSI architecture and methodology to combine the guaranteed-throughput service with the connectionless best-effort service compared to existing methodologies presented so far in the NoC research area [221], and

- to introduce a flexible VLSI microarchitecture of a NoC communication infrastructure that can flexibly support the aforementioned novel theory and methods.

1.3 Thesis Outline

The remaining chapters are generally divided into three chapter groups, i.e. the introductory chapter represented by Chap. 2, the contribution chapters describing the contributions of this thesis (Chap. 3–Chap. 7), and the concluding chapter represented by Chap. 8. The brief descriptions of each chapter are shown in the following.

- *Chap. 2.* This chapter describes the general theory and basic knowledge about interconnection networks such as network topologies, generic switch architecture, switching methods (store-and-forward, virtual cut-through, wormhole, circuit switching, etc.) and routing algorithms (deterministic, adaptive). Research challenges on the design of on-chip interconnection networks as well as some research areas related to NoCs such as multicore systems, parallel programming models and NoC testing methods are also briefly presented in this chapter.

- *Chap. 3.* This chapter describes formally the generic architecture and components of a router prototype called *XHiNoC*, which is developed as a part of this doctoral research. The XHiNoC router prototype consists of generic components and is extendable to include some additional services with small modifications in the generic components. This chapter also presents the main concept of the XHiNoC being flexible in sharing communication media in the NoC. The concept realized is based on a tag-division multiple-access technique, in which the multiplexed messages are assigned to a local identity (ID) slot. When entering a new communication channel, the local ID slot allocation or the ID-tag assignment to the message is update dynamically at runtime. The ID-tag assignment is organized in such a way that each individual message can be identified properly, and each flit (*flow control digit*) of the message can be routed to its routing paths correctly. The features and characteristics of the XHiNoC, which are achieved due to the implementation of the proposed concept, are described. The main issue related to the local ID slot scalability in guaranteeing service availability for all possible considered traffics is also formally described in this chapter.

- *Chap. 4.* This chapter proposes a new wormhole switching method called "wormhole cut-through switching method", in which flits of different wormhole messages

can be interleaved among each other at flit-level in the same link. The realization of the new wormhole switching is based on the main XHiNoC concept described previously in Chap. 3. The performance characteristics of the proposed switching method are evaluated under various commonly used data distribution scenarios. This chapter also shows how the head-of-line blocking problem is solved during saturating and non-saturating condition, and compares it visually with the virtual-channel-based solution. Interesting performance behaviors of the new switching method are presented during non-saturating and saturating conditions. In the non-saturating condition, end-to-end average data rate of each individual communication can be kept constant following the expected average data rate despite the increase in the number of workload. When the expected data rate is increased such that the NoC is saturated, the actual measured injection will follow the average actual acceptance rate of each considered traffic that is reduced to a steady-state point lower than the expected data rate. Hence, because of a link-level data overflow control, all message flits injected to the source nodes can be accepted without any loss at the destination nodes.

- *Chap. 5.* The extended version of the XHiNoC router supporting unicast and multicast services is presented in this chapter. This chapter introduces a new theory for deadlock-free multicast routing, as well as the VLSI microarchitecture of the router implementing the new deadlock-free multicast routing method. State-of-the-art multicast routing methods that have been used in high performance computing arena and in NoC research area is also presented in this chapter. By using the concept presented previously in Chap. 3, combined with a "hold-release multicast tagging mechanism", oblivious multicast dependency in each router that can lead to a permanent deadlock configuration can be solved effectively. Routing algorithm used to route multicast messages is also used for unicast messages resulting in an efficient routing machine implementation. Performance comparisons of the static and adaptive tree-based multicast routing are evaluated in this chapter. This chapter also presents an output selection function to perform efficient spanning trees of the tree-based multicast routing method when using an adaptive routing algorithm.

- *Chap. 6.* This chapter presents new selection strategy for runtime adaptive routing based on bandwidth space reservations and contention information between alternative output directions. State-of-the-art runtime adaptive routing selection strategies is described in this chapter. Five output selection strategies are introduced, i.e. bandwidth-aware (BWA), contention-aware, congestion-aware, as well as combinations of two strategies, i.e. contention- and bandwidth-aware (CBWA), and contention- and congestion-aware (CCA) output selection functions. All output selection strategies are implemented by using the wormhole cut-through switching method and the media share concept that have been presented previously in Chap. 4 and Chap. 3, respectively. Performance evaluation and logic synthesis results from the NoC router prototypes using the adaptive routing selection strategies are also

presented in this chapter.

- *Chap. 7.* The extended version of the XHiNoC router supporting runtime connection-oriented guaranteed-bandwidth service for unicast and multicast messages is presented in this chapter. This chapter introduces an efficient concept for communication media sharing to configure switched virtual circuits. This chapter presents the State-of-the-art switched virtual circuit configuration methods or multiple access techniques that have been implemented so far for NoCs, including the advantages of our proposed local ID-based multiplexing techniques compared to the other techniques. An XHiNoC router prototype combining connectionless best-effort and connection-oriented guaranteed-throughput communication protocols is also introduced in this chapter.

- *Chap. 8.* The new contributions of this thesis are summarized in this chapter. The directions for future works are also briefly described in this chapter.

Chapter 2

On-Chip Interconnection Networks

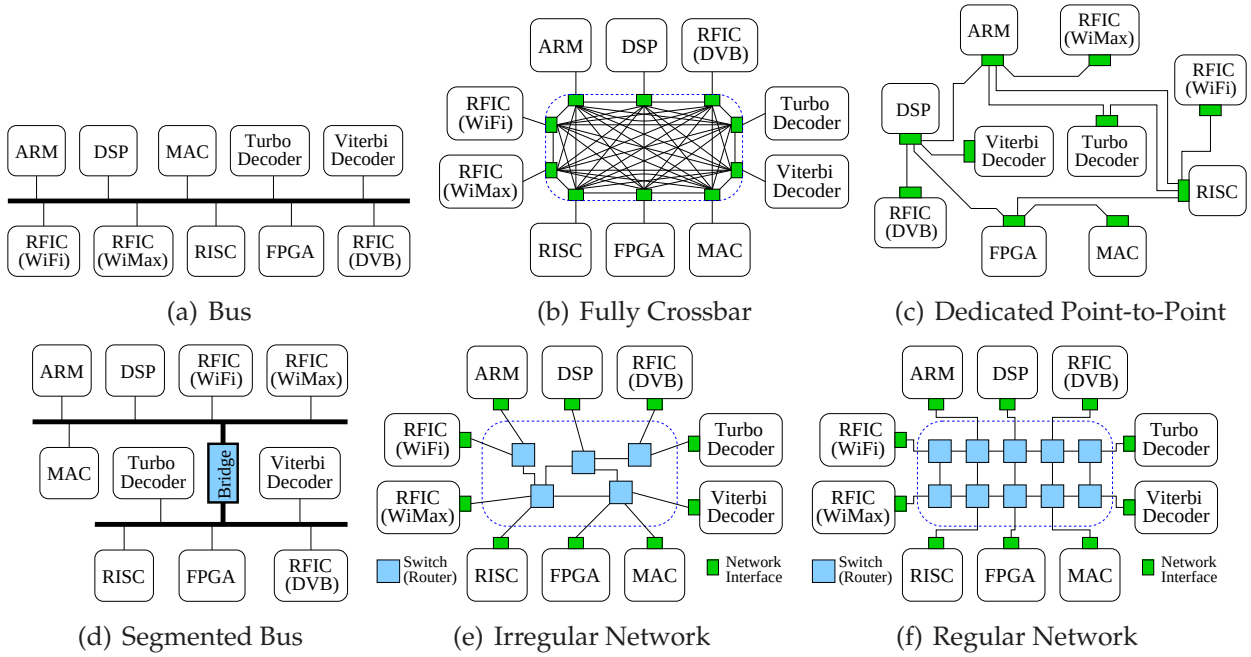
Contents

Networks-on-Chips (NoC) has been a bridge concept of a new design paradigm from Systems-on-Chip (SoCs) into Multiprocessor System-on-Chip (MPSoC). In the new computer era, where the design perspective to increase computing performance moves from increasing working frequency of a single core processor system to increasing the number of working processors in a multicore processor system, the NoC will become a preferred communication infrastructure, when the number of cores will be more than ten cores. A sophisticated communication structure is needed for the inter-processor data exchanges. Rather than using a traditional interconnect infrastructure such as a bus system (Fig. 2.1(a)), fully point-to-point (crossbar) (Fig. 2.1(b)) or dedicated point-to-point interconnect systems (Fig. 2.1(c)), a concept of shared segmented communication infrastructures is proposed to support application-scalability and high-performance inter-task communication.

The main problem using the bus interconnect system is the performance bottleneck due to its bandwidth limitation. The fully crossbar interconnect system leads to high electromagnetic interference and interconnect capacitance problems due to its metal wire domination. The main problem using the dedicated point-to-point interconnect system is the low flexibility. The bandwidth limitation in the bus system can be solved by using a hierarchical (segmented) bus system, in which a bus system is interconnected to other bus systems via a bridge component as presented in Fig. 2.1(d). However, since distributed bus arbitration corresponds to the aggregate actions of multiple arbitration units, computing optimal overall settings will be very complex and time consuming [54].

The NoC is the possible solution for such problems and requirements. Fig. 2.1(e) and Fig. 2.1(f) show examples of NoC topology architecture in irregular and regular structure, respectively. The NoC consists of several switches or routers used to route a packet/message sent by one IP component to another. Therefore, the main philosophy of the NoC is the development of communication infrastructure that enable us to route the packets instead of the wires [59]. The use of NoCs can be classified into two main categories, i.e. in embedded SoC applications domain commonly called *Multiprocessor System-on-Chip (MPSoC)* and in general-purpose microcomputer systems domain commonly called *Chip-Level Multiprocessor (CMP)* systems.

A SoC design approach, which is mainly used to develop application specific to embedded applications, integrates more than one Intellectual Property (IP) components into a single chip. Since the amount of processing element (PE) included in the multicore



(a) Bus (b) Fully Crossbar (c) Dedicated Point-to-Point

(d) Segmented Bus (e) Irregular Network (f) Regular Network

Fig. 2.1: SoC Interconnect Communication Infrastructure.

embedded SoC is more than one, then the SoC is generally called a *Multiprocessor System-on-Chip (MPSoC)*. The PEs in the MPSoC send and receive messages to and from other PEs for interacting computational processes in order to complete parallel tasks in the embedded applications. The main aspect that should be taken into account in the MPSoC systems is the lower power design. The power supply in the embedded applications, which is commonly used in electronic-handhelds and portable electronic appliances, is limited by the battery life. Therefore, the power constraint, which is also directly related to the logic area constraint, is the main issue to design the NoC-based MPSoC systems.

Fig. 2.2 shows a typical MPSoC system which consists of 16 cores in a 2D 4×4 mesh network architecture. The core can be a shared memory, a digital signal processor (DSP), a bus-based microprocessor system (such as ARM, MIPS, or RISC processor system), an ASIC component, FPGA-based configurable block, or any other core types. Each core is connected to one mesh Router (R) via an *On-Chip Network Interface (OCNI)*. The OCNI is the main component used to assemble a data into a packet before the data is sent from one core to another core through the network node, which is then disassembled back to the original data before being sent to the core.

An example of a NoC-based (networked) *chip-level multiprocessor (CMP)* system is presented in Fig. 2.3. The chip consists of 30 tiles interconnected in a 2D 6×5 mesh topology. Each tile consists of a microprocessor system, an on-chip network interface (OCNI) and a router (R). The microprocessor system can comprise of one or more CPU (central processing unit) blocks, a local memory block, a global (shared) memory, a memory controller (MCtrl), an IO interface and other components. The CMP system is typically a homogeneous (symmetric) multiprocessor system. Although in some cases, special-purpose

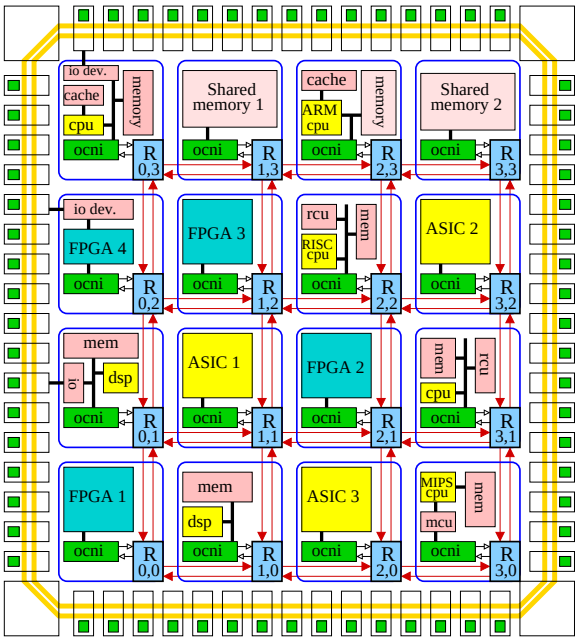


Fig. 2.2: Embedded multiprocessor system-on-chip (MPSoC) on mesh-connected NoC.

ASIC cores can be implemented on certain nodes in the CMP system. This characteristic is achieved due to the application domain of the CMP systems for general purpose microcomputer use, where every user's computer program will be compiled for a single core type target to simplify the program compilation and debugging steps made by the general computer users.

So far, on-chip communication infrastructures have been used in some the MPSoC and CMP applications. Commercial products such as game consoles are one of many potential NoC-based multiprocessor applications. IBM, Sony and Toshiba have jointly developed a *Cell Broadband Engine Processor* known as Cell Processor [116] dedicated for Playstation 3 Game Console. The cell processor consists of a 64-bit power processor element (PPE), eight specialized processors called synergistic processor elements (SPEs) [89], a high-speed memory controller and a high-bandwidth bus interface. All components are integrated on-chip and interconnected in a ring topology architecture. The Xbox 360 game console [10] has also used a CMP system consisting of 3 CPU cores, memory, I/O components and graphics processing unit (GPU). Since the number of PEs is relatively small, the components are interconnected through node crossbar/queuing, not a NoC communication infrastructure. However, this node crossbar/queuing can be interpreted as a single crossbar switch that is commonly used in a NoC router.

In academia, some works have investigated the potential applications of the NoC-based multiprocessor systems. For instance, the work in [74] has develop an adaptive and predictive NoC architecture based on FPGA for vision systems dedicated to image analysis. The work in [114] integrates ten processing elements for task-level parallelism with single-instruction multiple-data (SIMD) programming model. The memory-centric NOC-based processor system is used to compute the key-point localization stage of object

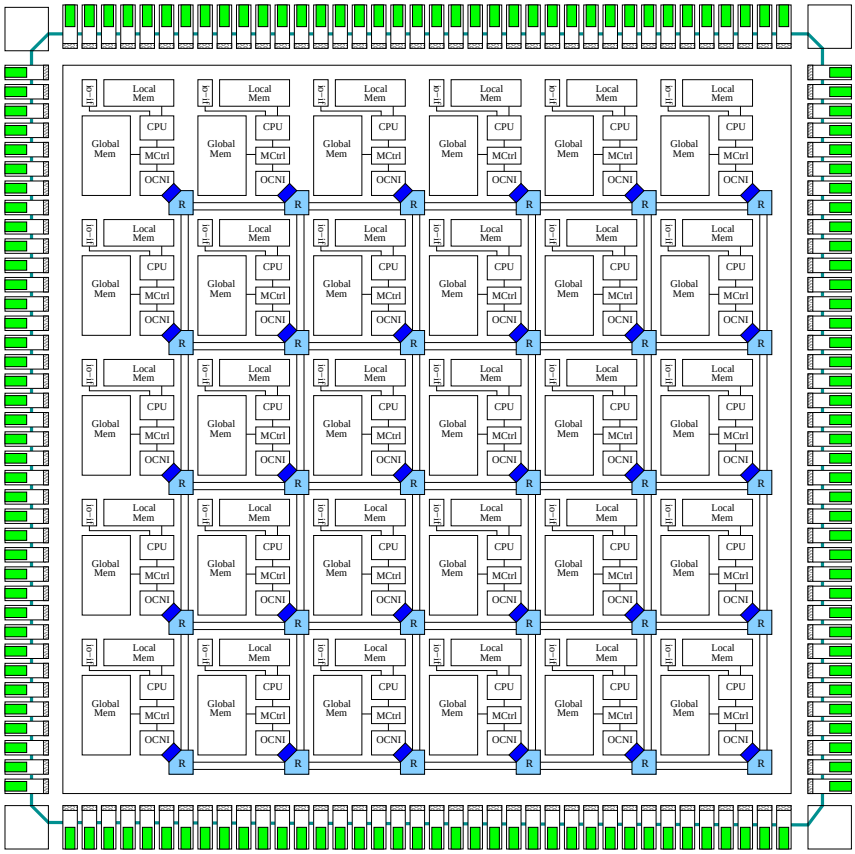


Fig. 2.3: Chip-Level multiprocessor (CMP) system on mesh-connected NoC.

recognition. The work in [115] shows a NoC-based parallel processor with bio-inspired visual attention engine. The NoC topology is custom-made by using two 7×7 crossbar switches and one 6×6 crossbar switch.

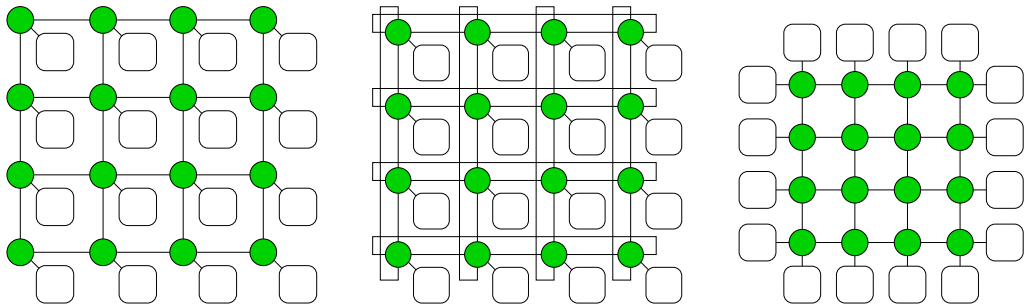
Independent from the targeted application domains mentioned above, the NoC can be implemented using different network topologies, data switching method and inter-switch data synchronization techniques. Table 2.1 represents several NoC prototypes developed in academic world and industries. The table presents some existing NoC prototypes that have been published so far with different communication data synchronization (synchronous, asynchronous or mesochronous), network topologies and switching methods. Most of the NoC proposals presented in the table uses the mesh topology and the synchronous communication with packet switching method.

2.1 Network-on-Chip Topology Architecture

In this section, some network topologies that have been used by some existing NoC architectures are presented. The selection of the network topologies is based on some reasons and backgrounds regarding the fulfilment of the bandwidth requirements for specific applications and parallel computing applications as well. In general, on-chip network topol-

Tab. 2.1: Networks on Chip Prototypes.

NoC Prototypes	NoC Topo.	Synch.	Switch.
SPIN [90]	Fat-tree	Async.	VCT.
MESCAL [196]	Custom	Async.	Pck.
MicroNet [212]	Custom	Sync.	Pck.
CLICHÉ [125]	Mesh	Sync.	Pck.
Proteo [191]	Mesh	Sync.	Pck.
RAW [203]	Mesh	Sync.	Worm.
Octagon [111]	Octagon	Sync.	Circ./Pck.
Chain [16]	Chain	Async.	Pck.
ECLIPSE [72]	Superswitch	Sync.	Pck.
SoCBUS [211]	Mesh	Sync.	Pck.
Æthereal [187]	Custom	Sync.	Circ.
Nostrum [157]	Mesh	Sync.	Pck.
Hermes [161]	Mesh	Sync.	Pck.
Arteris [149]	Custom	Sync.	Pck.
HiNoC [194], [97]	Hi. Mesh	ASync.	Circ.
Xpipes [30]	Custom	Mesoc.	Worm./Pck.
ASPIDA [8]	Chain	Async.	Pck.
IMEC NoC [25]	Irregular	Sync.	VCT.
ANoC [28]	Mesh	Async.	Pck.
DSPIN [181]	Mesh	Mesoc.	Circ.
PNoC [96]	Mesh	Sync.	Circ.
ASNoC [215]	Hi. Custom	Sync.	Pck.
ALPIN NoC [27]	Mesh	Async.	Pck.
KAIST NoC [129]	Hi. Star	Sync.	Pck
STNoC [177]	Custom	Sync.	Pck.
INoC [165], [166]	Irregular	Sync.	Pck.
MANGO [36]	Mesh	Async.	Circ.
IBM Cell EIB [3]	Ring-Star	Sync.	Pck.
Tile64 [210]	Mesh	Sync.	Worm.
Ambric MPPA [41]	Mesh	Sync.	Pck.
TRIPS [87]	Mesh	Sync.	Worm.
Intel Teraflops [98]	Mesh	Mesoc.	Worm.
SCC NoC [103]	3-ary 2-cube	Sync.	Worm.
EVC-NoC [123]	Mesh	Sync.	Pck.
XHiNoC [232], [229]	Mesh	Sync.	WormCT



(a) Mesh (b) Torus (c) 4-side Crossbar

Fig. 2.4: Mesh-like networks.

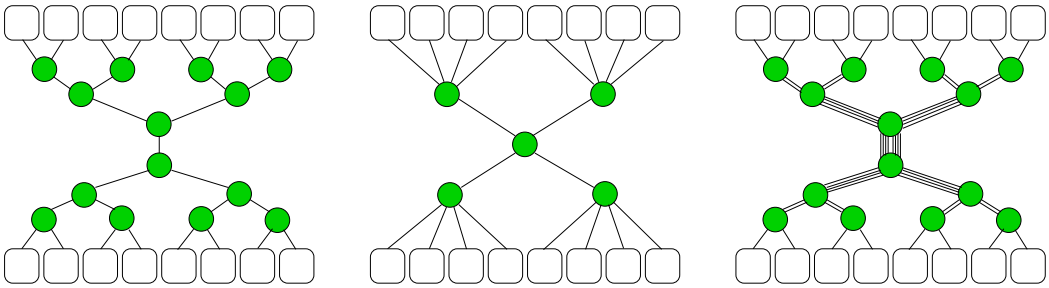
ogy can be divided into regular network architectures and irregular or custom network architectures.

2.1.1 Mesh-Like Network Topology

The most commonly used on-chip network topology as presented in the Table 2.1 is a mesh-based network. Compared with other on-chip network topologies, the mesh topology can achieve better application scalability. The implementation of routing functions in mesh topology is also simpler and can be characterized well. In the on-chip interconnection networks for on-chip multiprocessor systems, the mesh architecture is widely used and preferable. An example on-chip multiprocessor system that uses mesh topology is Intel-Teraflops system [205]. The 80 homogeneous computing elements are interconnected in through NoC routers in the 2D mesh 8×10 network topology.

Fig. 2.4(a) presents three different kinds of network architectures based on mesh structure. The mesh node consists of five ports, i.e. East, North, West, South and Local ports. The Local port of each mesh node is connected directly to one processing element. The other ports are connected with other ports of adjacent routers. Each node in the mesh network can be addressed well and simple. Therefore, routing algorithm for the mesh network architecture can be kept simple and well designed. The total bandwidths provided by the mesh communication links are scalable. For a 2D mesh with $N \times M$ size and the implementation of internode connection is with unidirectional full-duplex channels, then there will be $L_{mesh} = 2N(M-1) + 2M(N-1)$ physical communication channels available in the mesh network connecting $N \times M$ computing resources. The minimum hop to the nearest neighbor in the mesh is $H^{min}_{mesh} = 2$ hops, while the maximum hop to the longest neighbor is $H^{max}_{mesh} = N + M - 1$.

Fig. 2.4(b) shows a mesh-like network architecture called *torus* architecture. The main difference between the mesh and torus topology is the additional communication links connecting a nodes at the edge of the network with another node at the opposite edge in the same vertical or horizontal paths as presented in the figure. Designing a deadlock-free routing algorithm for the torus network will be more complex (especially in the network



(a) Binary-Tree (b) Quad-Tree (c) Fat-Tree

Fig. 2.5: Tree-based networks.

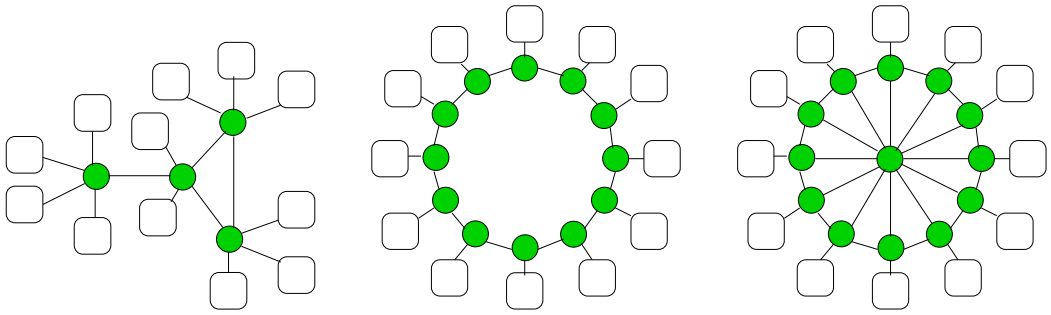
nodes at the network edges) because of these additional communication links. For a 2D $N \times M$ torus network where the communication channels is implemented with unidirectional full-duplex links, then there will be $L_{torus} = 4NM$ communication channels available in the torus network.

Another mesh-like network architecture is a *four-side mesh-based crossbar* network topology as presented in Fig. 2.4(c). The number of available communication links according to the $N \times M$ network size is similar to similar to the mesh-based network, i.e. $L_{4side-xbar} = 2N(M-1) + 2M(N-1)$. However, the number of computing elements that can be connected to the four-side crossbar network is $2(N+M)$. Hence, the ratio between communication resources over computing resources of the four-side crossbar network is higher than that of the mesh-based network.

2.1.2 Tree-based Network Topology

Tree-based NoC topologies are presented in Fig. 2.5. In Fig. 2.5(a), it shows a binary-tree network topology. In the binary-tree network, every router is connected to one up-level router and two down-level routers. At the end of the binary-tree network, two computing elements can be connected to each router. Packet routing in the binary-tree network is very simple. A direct routing method made directly reading the binary address attached in the packet header can be used to route packets from a source to a destination node.

Fig. 2.5(b) shows a quad-tree network topology. In the quad-tree network, every router is connected to one up-level router and four down-level routers. At the end of the quad-tree network, four computing elements can be connected to each router. Similar to the binary-tree network, packet routing in the quad-tree network can be simply implemented by using the direct routing method. Routing in the binary-tree and quad-tree network is static. The minimum hop to the nearest neighbor in the binary-tree and quad-tree network is equal, i.e. $H^{min}_{bin-tree} = H^{min}_{quad-tree} = 1$ hop, while the maximum hop to the longest neighbor depends on the number of switch nodes in the tree network. However, compared to the binary-tree network, the quad-tree network has lower average maximum network hop.



(a) Irregular Network (b) Ring 12 (c) Spidergon 12

Fig. 2.6: Irregular, Ring and Spidergon networks.

Fig. 2.5(c) exhibits a fat-tree network topology. In the fat-tree network, the topology can be similar to the binary-tree network, where at the end of the binary-tree network, two computing resources can be connected to each router. The difference is that the number of communication links connected to the upper-level router is increased. Hence, the link bandwidth capacity of the fat-tree router is higher than the binary-tree and quad-tree NoC router. In the fat-tree network, it is possible to apply adaptive routing algorithm.

2.1.3 Irregular or Custom Network Topology

Irregular or custom network topology is used to design a network architecture in order to optimize the use of communication resources and to save power consumption. By using the irregular custom network topology, the number of switches used to design a network architecture can be optimized. Accordingly, power dissipation and data communication energy in the optimal number of switches can be reduced. The irregular customized networks are also suitable for embedded MPSoC applications, in which the IP components used in the MPSoC devices have different sizes. Fig. 2.6(a) presents a network in irregular topology consisting of four switches. The network switches connect 12 computing resources.

The main drawback of the irregular/custom network topology is the complexity of the routing algorithm. Each routing algorithm of the irregular router must be customized to avoid possible cyclic dependency. In small-size irregular networks, the irregular routing algorithm is relatively not to complex. But, in very large-size irregular networks, the complexity to develop routing algorithms in every irregular network switch will be higher.

2.1.4 Other Network Topologies

Other commonly used network topology architectures are ring, octagon and spidergon topology. Fig. 2.6(b) shows a ring network architecture, which consists of 12 switches, where each switch is connected with one computing resource. The IBM-cell processor

system [116] for instance, uses ring topology to interconnect 8 synergistic processing elements together with a single PowerPC microprocessor. Fig. 2.6(c) exhibits a spidergon network architecture, which also consists of 12 network switches, in which each switch is also connected with one computing resource.

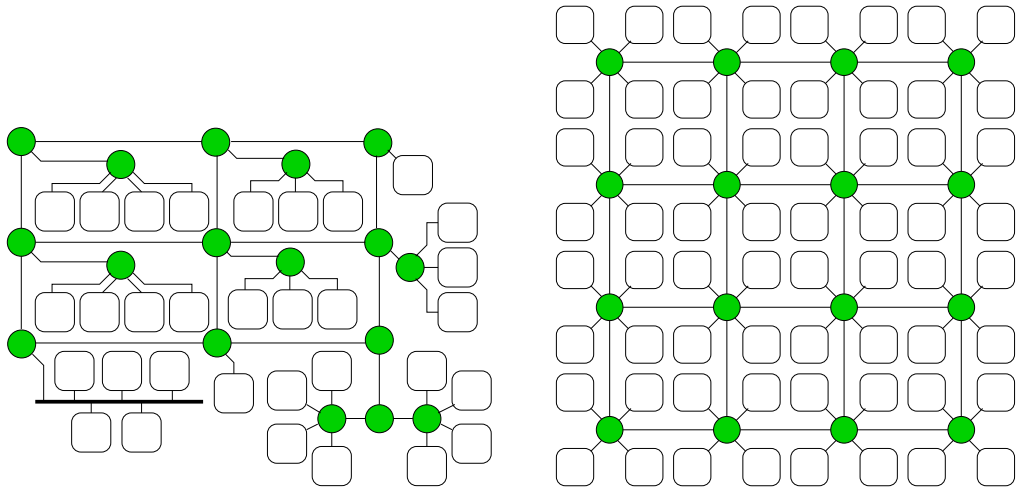
The main difference between the ring and the spidergon topology is an additional switch attached in the center of the spidergon network. The additional center switch is used to cut the network hop latency in the ring topology. As presented in Fig. 2.6(b), the largest network hop in the ring network depends on the number of N switches interconnected in the ring architecture, i.e. $H_{ring}^{max} = \frac{N}{2} + 1$ when N is even, or $H_{ring}^{max} = \frac{N+1}{2}$ when N is odd. The additional central switch acts as an intermediate switch in the spidergon network which reduces the maximum number of network hop delay to $H_{spidergon}^{max} = 3$ hops. Both the ring and the spidergon networks has equal minimal hop latency, i.e $H_{spidergon}^{min} = H_{ring}^{min} = 2$.

2.1.5 Hybrid and Hierarchical Network Topology

The other interesting network topology architecture is a hybrid hierarchical network. The hybrid hierarchical network can be a network that combines two or more network topology in a hierarchical interconnect architecture. For example in Fig. 2.7(a), it presents a regular mesh network, where each mesh node is connected with a lower-level hierarchical network. The lower-level network can a tree-based network, irregular network or a bus-based interconnect system. The hybrid hierarchical network is suitable for a parallel computing system having local computing domains. It will take on the advantages of the scalable bandwidth of the regular mesh topology and the low latency characteristics (low average hop) of the tree-based or irregular networks.

The hybrid hierarchical network presented in the Fig. 2.7(a) needs a 3D $(x, , y, z)$ node addressing. The 2D (x, y) node addressing is used for the mesh network nodes and the (z) is used to address the lower-hierarchical network nodes. NoCs that uses a hierarchical star (H-Star) topology architecture is presented for example in [129]. The H-Star topology consists of four clusters, where each cluster consists of four NoC routers and is connected to a central router. Therefore, the central router has four IO ports, while each cluster router has five IO ports. A mesh-of-trees presented in [17], is also an alternative interconnection networks for single-chip parallelism.

Fig. 2.7(b) presents also another interesting network topology called a mesh butterfly network. Each mesh node connected to four local processing elements (PEs). Originally, this architecture is a hybrid hierarchical network, where each mesh node is connected to a lower-level network interconnecting four PEs. However, instead of interconnecting the four PEs to the lower-hierarchical network node, they are directly connected to the mesh node. This implementation is made to avoid the low bandwidth capacity of the link connecting the mesh node and the lower-hierarchical node, and to use accordingly the maximum bandwidth capacity of the mesh switch. Like the network topology presented



(a) 2D Hybrid Hierarchy (b) Mesh Butterfly

Fig. 2.7: Mesh butterfly and hybrid hierarchical networks.

in Fig. 2.7(a), this network also requires the 3D network node addressing.

2.2 Generic On-Chip Switch Architecture

The microarchitecture of the switch or router for a certain network-on-chip is unique, depending on the implemented switching method, the implemented quality of service, the routing algorithm used and the utilization of an inter-switch communication synchronicity. Fig. 2.8 shows the typical router architecture, which consists of five input-output (I/O) ports, where one I/O port is connected to a local computing resource via network interface. In general, a switch (router) consists of five main components that are explained in the following:

1. *First-In First-Out (FIFO) Buffer.* This component is used to buffer incoming and outgoing data in the router. Some on-chip routers implement FIFO buffers either in input ports or in output ports to cut data buffering cost. In a switch having virtual channels, the FIFO buffers are replicated in the inputs and/or the output ports of the switch. Since adding buffers can significantly increase logic area overhead and power dissipation, the trend in switch architecture design is to move into the design of the switch without virtual channels, except for the use to provide guaranteed packet delivery service.

2. *Routing Engine.* The Routing Engine is utilized to compute routing decision of the incoming packets. In general, there are two different implementation of the routing engine circuit. The first one is Routing State Machine, and the other one is Table-based Routing. The combination of both Routing State Machine and Table-based Routing can also be used as explained later in Chap. 3.

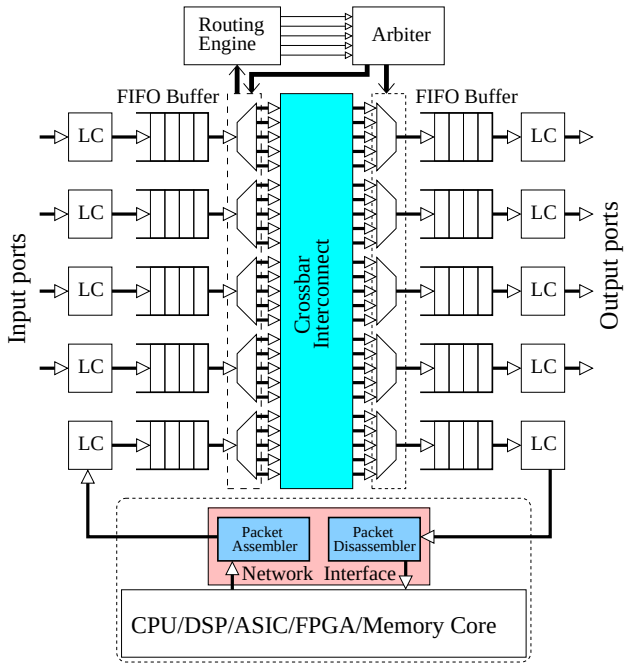


Fig. 2.8: Typical router structure.

3. *Arbiter.* The Arbiter unit is utilized to select a packet from a certain incoming port to access its requested outgoing port. The arbiter plays a role as a referee to control contentions between some packets requiring the same outgoing port in the router. There are some existing methods to implement the arbiter such as first-come first-serve, round-robin, priority-based, contention-aware and flit-by-flit rotating arbitration as presented later in Chap. 3.

4. *Crossbar Multiplexor-Demultiplexor.* These components form crossbar interconnects between input and output ports of the router. In some cases, the demultiplexor units can be neglected to optimize the crossbar area. All possible input data lines will be connected to the input ports of the crossbar multiplexors, and then the output data from the input data lines is controlled by the arbiter unit.

5. *Link Controller.* The link controller unit is used to control data transmission between input and output ports of adjacent routers. Data control is used to avoid data overflows and incorrect data replications. Some existing control mechanisms such as credit-based method can be implemented in this unit. Data synchronization interfaces are also implemented in this unit to synchronize correct data transmission from one switch to another one. Some data synchronization methods that can be implemented in the NoC are e.g. source-synchronous, mesochronous, asynchronous queue-based, pipelined repeater-based and the most well-known handshake mechanism.

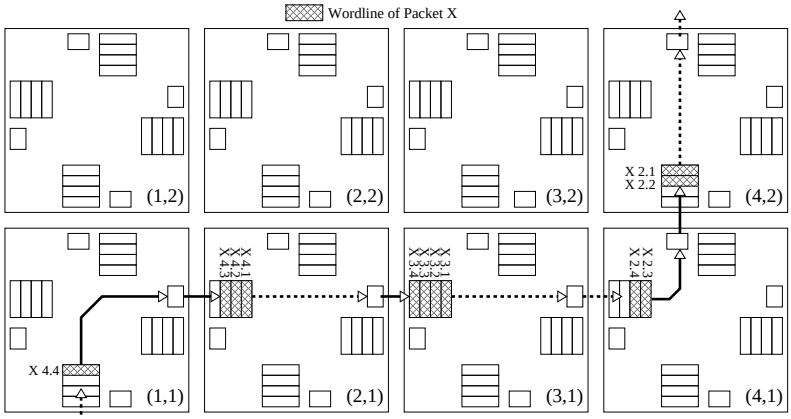


Fig. 2.9: Store-and-Forward Switching.

2.3 Switching Methodology

Nowadays, there have been many innovations to switch data in interconnection networks. The selection of switching methodology will determine the architecture of an on-chip router, and may also determine service that can be provided by the network. Among many existing methods, basic switching methods such as store-and-forward, wormhole, virtual cut-through and circuit switching method are described in the following subsections. There are still a few hybrid methods for data switching that have been proposed in the interconnection network community such as pipelined circuit switching (PCS) [7], [77] that combines the characteristics of the wormhole and circuit switching method, and buffered wormhole switching (BWS) which is a variant of the wormhole switching that combines the store-and-forward packet switching characteristics. The BWS was firstly introduced in IBM Power Parallel SP2 [94]. The other alternative switching methods are *Mad Postman Switching* [108] and *Scouting Switching* [64], [60]. The scouting switching is proposed to improve the performance and the capability of the PCS methods to tolerate faulty links. The work in [65] has also summarized well the mechanisms and the history of the switching methodologies used so far in the existing interconnection networks and high performance computing (HPC) arena.

2.3.1 Packet Switching (Store-and-Forward)

Packet Switching method is commonly called also as *Store-and-Forward (SAF)* switching. This switching method is implemented by dividing data messages into a number of packets. Each packet is completely stored in a FIFO buffer before it is forwarded into the next router. Therefore, the size (depth) of FIFO buffers in the router is set similar to the size of the packet in order to be able to completely store the packet. Fig. 2.9 shows the visual diagram of the store-and-forward switching method. As presented in the figure, message X consists of packets depicted with $Xn.m$, where n is the packet number and m is the dataword (wordline) number in the packet. Each packet $Xn.m$ as shown in the figure

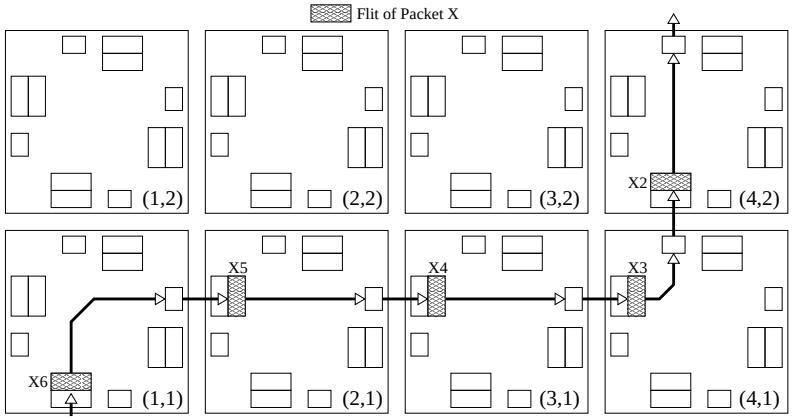


Fig. 2.10: Wormhole Switching.

consists of four wordlines. The first wordline ($Xn.1$) is the header and the last wordline ($Xn.4$) is the tail. The wordlines of the third packet for example, i.e. $X3.1$, $X3.2$, $X3.3$ and $X3.4$, are completely stored in the West input buffer of the router node (3,1). The packet $X3.m$ can then be forwarded to the next router. If the routing has been made then the West input buffer of the router node (4,1) is free from data.

The packet switching method is the the first switching method that has been used in many parallel machines. The early parallel machines that use the packet switching are for example the Denelcor HEP machine, which is well introduced in [67], the MIT Tagged Token Dataflow machine [13] and the Manchester Dynamic Dataflow computer [92]. Like in the off-chip networks area, most of the early NoC concepts and prototypes use also the packet switching method such as Proteo [191], [190], Nostrum [157], MESCAL [196], MicroNet [212], CLICHÉ [125] and Arteris [149] (See also Table 2.1).

2.3.2 Wormhole Switching

In the wormhole switching method, messages are divided into a number of *flow control digit* or commonly called as *flit*. Every flit may bring a data word. The main advantage of the wormhole switching is that the buffer size can be set as small as possible to reduce the buffering area cost. Fig. 2.10 shows the visual diagram of the wormhole switching method. The message in the network flows like a worm through holes in the ground. The main drawback of the wormhole switching method is the problem of *head-of-line blocking*. As presented in Fig. 2.10, flits of the message X occupy (reserve) some buffers in the input ports of the network routers. Every flit of the message X is symbolized with Xn, where n is the flit number. Other messages cannot acquire the reserved buffers until the flits of message X have released the buffer reservation. The tail flit or the end flit of the message will terminate the buffer reservation.

The wormhole switching method was firstly introduced in [57]. The work in [56] has presented also the performance of the wormhole switching in k-ary n-cube interconnec-

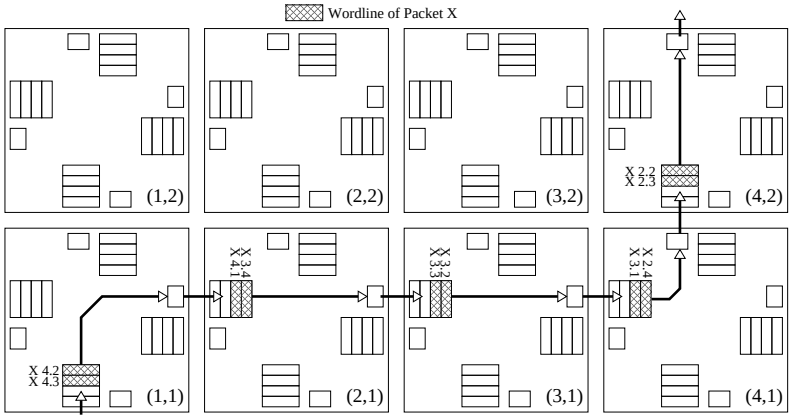


Fig. 2.11: Virtual Cut-Through Switching.

tion networks. Some parallel machines in the HPC area that use wormhole switching are for example, Intel Paragon XP/S [104], Cray T3D (Toroidal 3D) [173], IBM Power Parallel SP1 [201] and Meiko CS-2 (Computing Surface) [26]. In the NoC area, the wormhole switching is also preferred and has been used in some of the latest NoC-based CMP systems prototypes such as Tile64 [210], TRIPS [87], Teraflops [98] and SCC NoC [103] (See also Table 2.1).

2.3.3 Virtual Cut-Through Switching

In the store-and-forward packet switching method, the packet is completely stored before it is forwarded to the next router. The delay to wait for the complete packet storing can be reduced by forwarding the first lines of the packet to the next router soon after routing has been made for the packet and when there is enough space in the required FIFO buffer in the next router to store the first wordlines of the packet. This switching technique is known as *Virtual Cut-Through (VCT)* switching and was firstly introduced in [113]. On-chip router of Alpha 21364 [162] is one of the multiprocessor system that uses VCT switching method. The work in [120] presents Chaos Router, which is one of the best VCT switching implementations. A few NoC prototypes such as SPIN [90] and IMEC NoC [25] use this VCT switching method.

Fig. 2.11 presents the visual diagram of the virtual cut-through switching method. As presented in the figure, the header of the third packet ($X3.1$) has been forwarded to the West input buffer in the router node (4,1), because routing has been made and there is already a free space to store the packet header. Meanwhile, the tail of the third packet ($X3.4$) is still behind in the West input buffer of the router node (2,1). The router in node (3,1) does not need to store the entire packet wordlines to forward the first wordlines of the packet. Hence, as shown in Fig. 2.11, every packet can virtually cut-through in the network nodes.

2.3.4 Circuit Switching

The circuit switching method is commonly used in a connection-oriented communication protocol. The circuit switching method is performed by establishing connection and reserving some communication resources. When a virtual circuit from a source to a destination node has been configured and the successful connection has been informed by the destination node by sending a response packet to the source node, then the message can be transmitted through the network in a pipeline manner. At the end of the data transmission, a control packet is sent to the network to terminate the connection circuit. The circuit switching method is commonly used to provide guaranteed-bandwidth or guaranteed-throughput communication protocol for quality of service.

The circuit switching method is originally used in telephone networks. In HPC area, some parallel machines that have used the circuit switching method are Intel iPSC/2 [171] that uses a Direct Connect Communications Technology and Motorola-based BBN GP 1000 [37], which uses multistage interconnection network with butterfly interconnect structure. In the NoC area, the circuit switching method is used to provide guaranteed-throughput service. Some NoCs that uses the circuit switching method are DSPIN [181], PNoC [96], MANGO [36] and Æthereal [187].

2.4 Routing Algorithms

This section will present some basic backgrounds and concept about routing algorithms. In general, the selected routing algorithm for a network is topology dependent. This section will give only a brief description about *deadlock-free routing algorithm* suitable for mesh-based network.

2.4.1 Deadlock and Livelock Configuration

The main issue related to routing algorithm selection is *deadlock configuration*. Cyclic dependency between packets in the network leads to the deadlock configuration. Fig. 2.12 shows an example of a deadlock configuration. The deadlock configuration is formed due to cyclic dependency between four packets, i.e. Packet A, B, C and D. Packet A from router node (1,1) is routed to node (1,2), but it cannot be further routed to node (2,2) because the East output port has been acquired by packet B. Meanwhile, Packet B from node (2,2) cannot be further routed to node (2,1) because its required South output port of node (2,2) has also been acquired by Packet C. The same situations also occurs at the West output port of the router node (2,1) between Packet C and D, and at the North output port of the router node (1,2) between Packet A and D. All packets form a cyclic dependency and cannot move further.

Deadlock configuration caused by the cyclic dependency between four packets as pre-

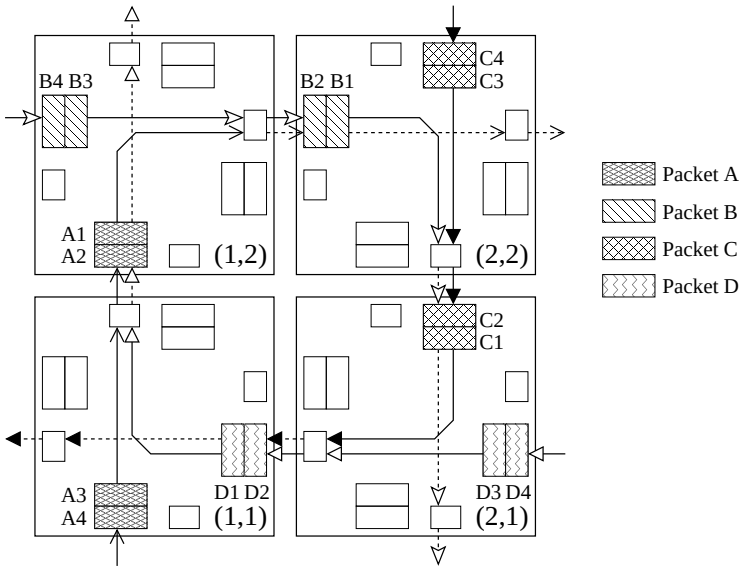


Fig. 2.12: Deadlock configuration.

sented in Fig. 2.12 can occur because packets are allowed to make all turns in clock-wise and counter clock-wise turn directions. Deadlock configuration can be avoided by applying allowed turns and prohibiting minimal one turn in every clock-wise and counter clock-wise turn direction. The prohibited turns will avoid cyclic dependency between packets in the network. Routing algorithms derived from turn models will be explained later in Section 2.4.3. Deadlock configuration can be also avoided by introducing virtual channels which will be explained in Section 2.4.4. The works in [58] and [62] have presented theoretically how to design deadlock-free message routing algorithms.

If the packets are allowed to make *non-minimal* adaptive routing (*misrouting*), then a problem called *livelock configuration* may occur. The livelock configuration is a situation where a packet moves around a destination node but it never reaches the destination node. The livelock configuration can be avoided by only allowing the packets to make *profitable* (*minimal*) routing. However, if the misrouting is allowed, then the mechanism to detect livelock configuration must be implemented. The definition of the profitable (minimal) and the misrouting (non-minimal) adaptive routing will be explained in the following subsection (Section 2.4.2).

2.4.2 Taxonomy of Routing Algorithms

This section presents a taxonomy of routing protocols that is classified according to several criteria [65].

- *Number of destinations.* According to the number of destination nodes, to which packets will be routed, routing algorithms can be classified into *unicast routing* and *multicast routing*. The unicast routing sends the packets from single source node to

single destination node. The multicast routing sends the packets from single node to multiple destination nodes. The multicast routing algorithm can be divided further into *Tree-based multicast routing* and *Path-based multicast routing*.

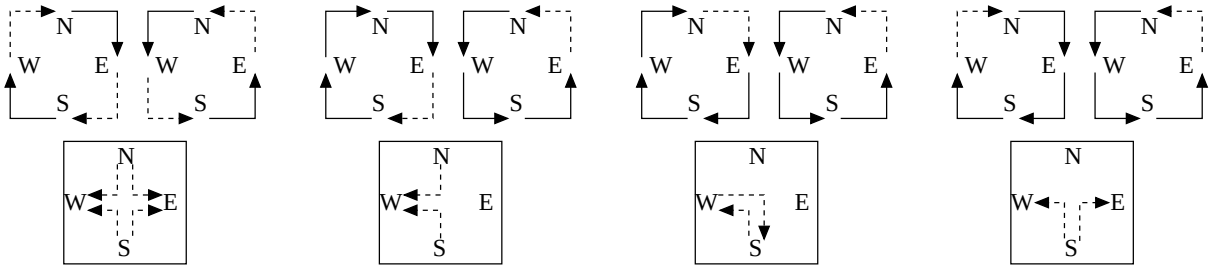
- *Routing Decision Locality.* According to the place where the routing decisions are made, routing algorithms (unicast or multicast routing) can be classified into *source routing* and *distributed routing*. In the source routing, routing paths are computed at source node. The pre-computed routing information for every intermediate node, to where a message will travel, will be written in a routing probe. All routing probes that represent the routing paths from the source to destination node will then be assembled as packet headers for the message. In the distributed routing, there will be one header probe (for unicast routing case) containing the address of the destination node (probably also the source node). The routing information is locally computed each time the header probe enters a switch node.

- *Implementation.* According to the way the routing information are computed, routing algorithm can be implemented into *table-lookup* and *finite-state machine* . In the table-lookup implementation, routing slot tables store the routing information that are implemented on each router. The routing direction on each router is computed by reading the slot number attached in the header probe of the packet and finding the appropriate slot number in the routing table, in which the routing information is stored. In the finite-state machine, the routing information on each router is computed by a routing algorithm according to the destination address attached in the header probe and the current address of the router node.

- *Adaptivity.* In both case of the routing implementation, the routing algorithm can be either *deterministic* or *adaptive*. In the deterministic routing algorithm, the computed paths from source to destination node will always be similar. In the adaptive routing algorithms, the paths from source to destination can be different because the adaptive routing select adaptively the alternative output ports. An output channel is selected based on the congestion information or the channel status of the alternative output ports. The adaptive routing algorithms generally guide messages away from congested or faulty regions in the network.

- *Progressiveness.* According to the progressiveness of the message movement, the adaptive routing algorithms can be applied as *progressive* or *backtrace-enabled* routing algorithm. In the progressive approach, the message headers will always move forward or move towards a progressive direction. In the backtrace method, the message headers can track back to a previous path and release the previously reserved channels. The adaptive backtracking algorithms are mainly used for fault-tolerant routing algorithm.

- *Minimality.* According to the minimality of the routing path, the adaptive routing algorithms can be classified into *profitable* (*minimal*) or *misrouting* (*non-minimal*) algo-



(a) X-First (XY) (b) West-First (WF) (c) Neg.-First (NegF) (d) North-Last (NL)

Fig. 2.13: Turn models that can avoid deadlock configuration.

rithm. The profitable adaptive routing algorithm will not allow a message to move away from its destination node. In other words, the message will always be routed closer to its destination node. The adaptive routing algorithm will route the message through the minimal paths that can be selected adaptively. In the adaptive misrouting algorithm which is also called as the detour routing algorithm, the message can be routed away from its destination node. This adaptive routing algorithm must be designed carefully, because it can lead to the *livelock configuration* that has been explained in Section 2.4.1.

- *Number of paths.* The adaptive routing algorithm can be classified according to the number of alternative adaptive turns as *fully adaptive* and *partially adaptive* routing algorithm.

According to the work in [84], the degree of adaptiveness A_{alg} of a minimal adaptive routing algorithm can be determined based on the number of the shortest paths that can be used to route a packet from a source node to a destination node. Equ. 2.1 presents the degree of adaptiveness of a minimal fully adaptive routing algorithm for a 2D mesh network topology to a packet from (X_{source}, Y_{source}) node to the (X_{target}, Y_{target}) node, where $\Delta x = X_{offset} = |X_{target} - X_{source}|$, and $\Delta y = Y_{offset} = |Y_{target} - Y_{source}|$.

$$A_{fully} = \frac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!} \tag{2.1}$$

2.4.3 Routing Algorithms based on Turn Models

In order to avoid cyclic dependency leading to deadlock configuration, a turn model represented as turn directions in clock-wise and counter clock-wise can be applied to the design of a deadlock-free routing algorithm. Design of adaptive routing algorithms based on turn models has been introduced in [83]. The work has presented examples of turn models for adaptive routing algorithms in 2D mesh-based interconnetion network. In the mesh network, there will be four available turns at each clock-wise and counter clock-wise turns.

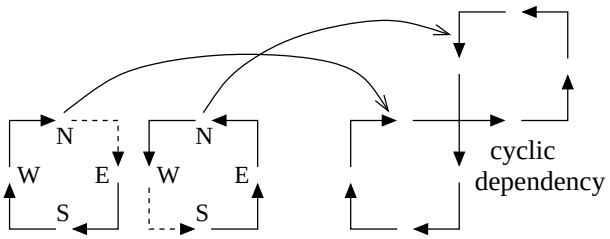


Fig. 2.14: Turn model that cannot avoid deadlock configuration.

Fig. 2.13 presents four selected turn models that can be used to avoid deadlock configuration. The solid lines in the figure represent the allowed turns, and the dashed lines represent the prohibited turns. In the turn models, allowed and prohibited turns are introduced to avoid possible deadlock configuration. One of the well-known static routing algorithm based on the turn model is *dimension-order routing algorithm*. Routing algorithms based on the turn models can be classified into static and adaptive routing algorithms.

The selection of the prohibited and allowed turns in every clock-wise and counter clock-wise turn direction must be selected correctly in such a way that deadlock configuration can be avoided. Fig. 2.14 shows an example of an incorrect turn model that cannot avoid a deadlock configuration. The turn model is shown in the left-side of the figure. If the turn model are combined together as presented in the right-side of the Fig. 2.14, then it looks like cyclic dependency can still occur in the network.

In 2D mesh network, a dimension-order *X-First (XY) routing algorithm* or *Y-First (YX) routing algorithm* can be used. The turn model of the *static XY routing algorithm* or *static X-First routing algorithm* is presented in Fig. 2.13(a). As shown in the figure, four turns are prohibited to avoid a deadlock configuration, i.e. North–East, South–East, North–West and South–West turns. Because of the applied prohibited turns, routing algorithm is static whereby packets are always routed firstly to X-direction, then to Y-direction. Alg. 1 describes the static XY routing algorithm.

As a counterpart to the X-First routing algorithm, an alternative static routing algorithm can be implemented by firstly routing packets to the Y-direction before they are routed to the X-direction. This alternative static routing algorithm is called *static Y-First routing algorithm* or *static YX routing algorithm*. In this case, four turns are prohibited to avoid cyclic dependency, i.e. East–North, East–South, West–North and West–South turns. For all source-destination node communications, the degree of adaptiveness of the static routing algorithm is always 1 ($A_{static} = 1$).

The turn model of *adaptive West-First (WF) routing algorithm* is presented in Fig. 2.13(b). As shown in the figure, two turns are prohibited to avoid deadlock configuration, i.e. North–West and South–West turns. Because of the applied prohibited turns, routing algorithm can be made adaptively, when source–destination offsets, i.e. $X_{offset} > 0$ and $Y_{offset} > 0$ as well as when $X_{offset} > 0$ and $Y_{offset} < 0$, where $X_{offset} = X_{target} - X_{source}$ and $Y_{offset} = Y_{target} - Y_{source}$.

Alg. 1 Static X-First (XY) Routing Algorithm

1: $X_{offset} = X_{target} - X_{source}$
2: $Y_{offset} = Y_{target} - Y_{source}$
3: **if** $X_{offset} = 0$ and $Y_{offset} = 0$ **then**
4: $Routing = LOCAL$
5: **else if** $X_{offset} > 0$ **then**
6: $Routing = EAST$
7: **else if** $X_{offset} < 0$ **then**
8: $Routing = WEST$
9: **else if** $X_{offset} = 0$ and $Y_{offset} > 0$ **then**
10: $Routing = NORTH$
11: **else if** $X_{offset} = 0$ and $Y_{offset} < 0$ **then**
12: $Routing = SOUTH$
13: **end if**

Alg. 2 describes the minimal adaptive West-First routing algorithm. In the West-First routing algorithm, packets will always be routed firstly to the West direction before they are routed to the North direction when $X_{offset} < 0$ and $Y_{offset} > 0$, as well as firstly to the West direction before they are routed to the South direction when $X_{offset} < 0$ and $Y_{offset} < 0$. Therefore, this adaptive routing algorithm is called *Adaptive West-First Routing Algorithm*. The degree of adaptiveness of the West-First adaptive routing algorithm is presented in Equ. 2.2.

Alg. 2 Minimal Adaptive West-First (WF) Routing Algorithm

1: $X_{offset} = X_{target} - X_{source}$
2: $Y_{offset} = Y_{target} - Y_{source}$
3: **if** $X_{offset} = 0$ and $Y_{offset} = 0$ **then**
4: $Routing = LOCAL$
5: **else if** $X_{offset} < 0$ **then**
6: $Routing = WEST$
7: **else if** $X_{offset} > 0$ and $Y_{offset} > 0$ **then**
8: $Routing$=**Select**$(EAST, NORTH)$
9: **else if** $X_{offset} > 0$ and $Y_{offset} < 0$ **then**
10: $Routing$=**Select**$(EAST, SOUTH)$
11: **else if** $X_{offset} > 0$ and $Y_{offset} = 0$ **then**
12: $Routing = EAST$
13: **else if** $X_{offset} = 0$ and $Y_{offset} > 0$ **then**
14: $Routing = NORTH$
15: **else if** $X_{offset} = 0$ and $Y_{offset} < 0$ **then**
16: $Routing = SOUTH$
17: **end if**

$$A_{west-first} = \begin{cases} \dfrac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!}, & X_{target} \geq X_{source} \\ 1, & otherwise \end{cases} \tag{2.2}$$

Fig. 2.13(c) presents the turn model of *adaptive Negative-First (NegF) routing algorithm*.

As shown in the figure, two turns are prohibited to avoid deadlock configuration, i.e. West–South and South–West turns. Because of the applied prohibited turns, routing algorithm can be made adaptively, when source–destination offsets, i.e. $X_{offset} > 0$ and $Y_{offset} > 0$ as well as when $X_{offset} < 0$ and $Y_{offset} < 0$, where $X_{offset} = X_{target} - X_{source}$ and $Y_{offset} = Y_{target} - Y_{source}$.

Alg. 3 describes the minimal adaptive Negative-First routing algorithm. In the Negative-First routing algorithm, packets will always be routed firstly to Negative directions, i.e. South or $Y-$ direction before East direction when $X_{offset} > 0$ and $Y_{offset} < 0$, as well as West or $X-$ directions before North direction when $X_{offset} < 0$ and $Y_{offset} > 0$. Therefore, this adaptive routing algorithm is called *Adaptive Negative-First Routing Algorithm*. The degree of adaptiveness of the Negative-First adaptive routing algorithm is presented in Equ. 2.3.

Alg. 3 Minimal Adaptive Negative-First (NegF) Routing Algorithm

1: $X_{offset} = X_{target} - X_{source}$
2: $Y_{offset} = Y_{target} - Y_{source}$
3: **if** $X_{offset} = 0$ and $Y_{offset} = 0$ **then**
4: $Routing = LOCAL$
5: **else if** $X_{offset} > 0$ and $Y_{offset} > 0$ **then**
6: $Routing$=**Select**$(EAST, NORTH)$
7: **else if** $X_{offset} \geq 0$ and $Y_{offset} < 0$ **then**
8: $Routing = SOUTH$
9: **else if** $X_{offset} < 0$ and $Y_{offset} < 0$ **then**
10: $Routing$=**Select**$(WEST, SOUTH)$
11: **else if** $X_{offset} < 0$ and $Y_{offset} \geq 0$ **then**
12: $Routing = WEST$
13: **else if** $X_{offset} = 0$ and $Y_{offset} > 0$ **then**
14: $Routing = NORTH$
15: **else if** $X_{offset} > 0$ and $Y_{offset} = 0$ **then**
16: $Routing = EAST$
17: **end if**

$$A_{neg-first} = \begin{cases} \dfrac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!}, & (X_{target} \leq X_{source} \; and \; Y_{target} \leq Y_{source}) \; or \\ & (X_{target} \geq X_{source} \; and \; Y_{target} \geq Y_{source}) \\ 1, & otherwise \end{cases} \tag{2.3}$$

The turn model of *adaptive North-Last (NL) routing algorithm* is presented in Fig. 2.13(d). As shown in the figure, two turns are prohibited to avoid deadlock configuration, i.e. South–East and South–West turns. Because of the applied prohibited turns, routing algorithm can be made adaptively, when source–destination offsets, i.e. $X_{offset} > 0$ and $Y_{offset} < 0$ as well as when $X_{offset} < 0$ and $Y_{offset} < 0$, where $X_{offset} = X_{target} - X_{source}$ and $Y_{offset} = Y_{target} - Y_{source}$.

Alg. 4 describes the minimal adaptive North-Last routing algorithm. In the North-Last routing algorithm, packets will be routed at last to North direction after East direction when $X_{offset} > 0$ and $Y_{offset} > 0$, as well as at last to North direction after West direction when $X_{offset} < 0$ and $Y_{offset} > 0$. Therefore, this adaptive routing algorithm is called *North-Last Adaptive Routing Algorithm*. The degree of adaptiveness of the North-Last adaptive routing algorithm is presented in Equ. 2.4.

Alg. 4 Minimal Adaptive North-Last (NL) Routing Algorithm

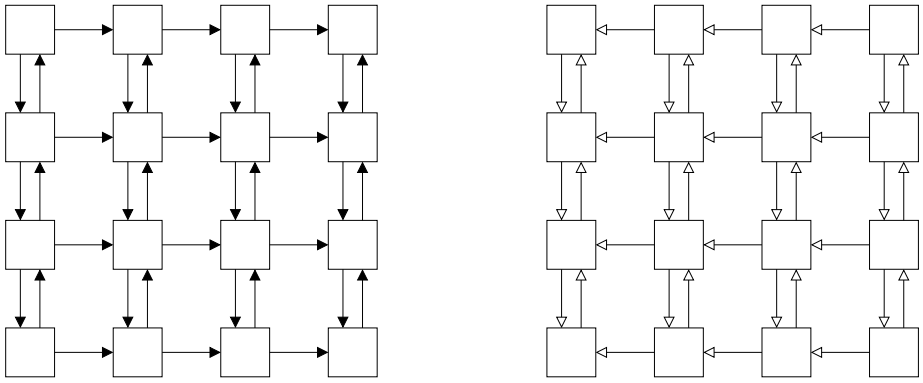
1: $X_{offset} = X_{target} - X_{source}$
2: $Y_{offset} = Y_{target} - Y_{source}$
3: **if** $X_{offset} = 0$ and $Y_{offset} = 0$ **then**
4: $Routing = LOCAL$
5: **else if** $X_{offset} > 0$ and $Y_{offset} < 0$ **then**
6: $Routing$=**Select**($EAST, SOUTH$)
7: **else if** $X_{offset} > 0$ and $Y_{offset} \geq 0$ **then**
8: $Routing = EAST$
9: **else if** $X_{offset} < 0$ and $Y_{offset} < 0$ **then**
10: $Routing$=**Select**($WEST, SOUTH$)
11: **else if** $X_{offset} < 0$ and $Y_{offset} \geq 0$ **then**
12: $Routing = WEST$
13: **else if** $X_{offset} = 0$ and $Y_{offset} > 0$ **then**
14: $Routing = NORTH$
15: **else if** $X_{offset} = 0$ and $Y_{offset} < 0$ **then**
16: $Routing = SOUTH$
17: **end if**

$$A_{north-last} = \begin{cases} \dfrac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!}, & Y_{target} \leq Y_{source} \\ 1, & otherwise \end{cases} \tag{2.4}$$

The other well-known adaptive routing algorithm based on the turn models is the adaptive routing based on *Odd-Even Turn Model* that was firstly introduced in [48]. In this approach, the allowed and prohibited turns in the odd and in the even column of the mesh-based network can be different, but must be still able to guarantee free from deadlock configuration.

2.4.4 Routing Algorithms with Virtual Channels

Adaptive routing algorithms can be used in a network by implementing virtual channels. Virtual channels are used to provide alternative channels virtually to avoid cyclic dependency. One of many techniques to implement adaptive routing algorithms with virtual channels is by introducing virtual networks or sub networks. Fig. 2.15 presents two virtual networks on a 2D 4×4 mesh network.



(a) X^+ Subnet (Virtual Net 0) (b) X^- Subnet (Virtual Net 1)

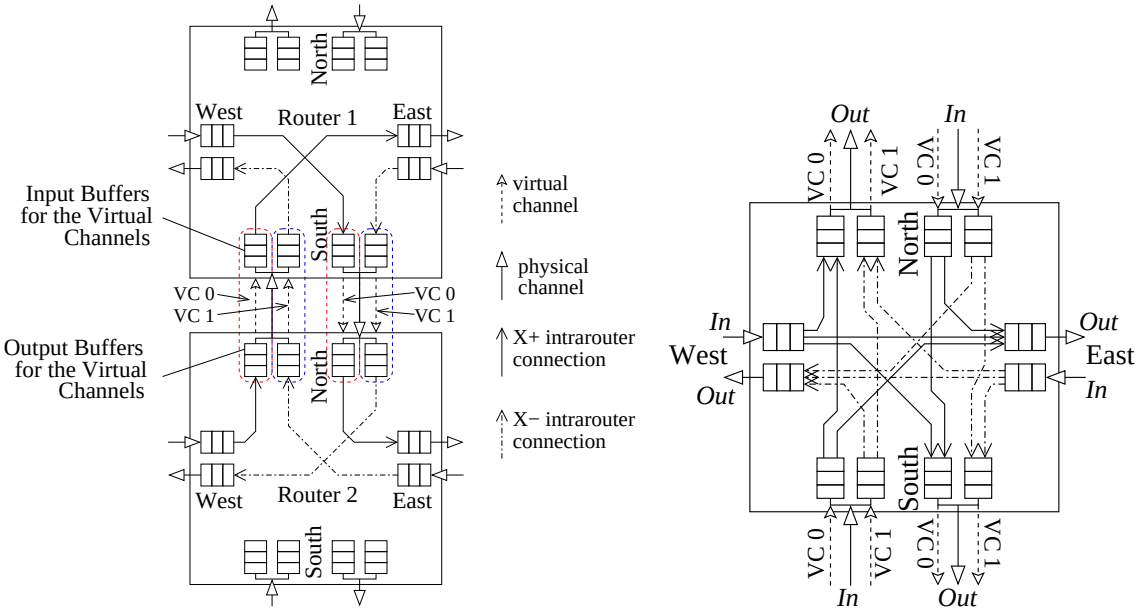
Fig. 2.15: Mesh Network separated into two virtual networks.

As presented in figure, the virtual networks are called X^+ Subnet or Virtual Network 0 as shown in Fig. 2.15(a) and X^- Subnet or Virtual Network 1 as presented in Fig. 2.15(b). In the X^+ virtual network, packets can be routed adaptively from West to North and from North to East, as well as from West to South and from South to East. While in the X^- virtual network, routing from East to North and from North to West, as well as routing from East to South and from South to West are allowed. The proposed network partitioning has been introduced in [47] as a *2D planar adaptive routing algorithm* with 2 virtual channels for the two virtual networks. The degree of adaptiveness of the 2D Planar Adaptive Routing Algorithm is shown in Equ. 2.5.

$$A_{neg-first} = \begin{cases} 1, & X_{target} = X_{source} \; or \; Y_{target} = Y_{source} \\ \dfrac{(\Delta x + \Delta y)!}{\Delta x! \Delta y!}, & otherwise \end{cases} \tag{2.5}$$

If the offset between target and source node is greather than zero, i.e. $X_{offset} = X_{target} - X_{source} \geq 0$, then packets will be routed through X^+ Virtual Network. While if the offset between target and source node is less than zero, i.e. $X_{offset} = X_{target} - X_{source} \leq 0$, then packets will be routed through X^- Virtual Network. If the $X_{offset} = 0$, then packets can be routed either through X^+ or X^- Virtual Network. Once packets have been routed in the X^+ Virtual Network, they will not be routed in the X^- Virtual Network. In contrast, once packets have been routed in the X^- Virtual Network, they will not be routed in the X^+ Virtual Network. By implementing such routing rules, deadlock configurations can be avoided.

A deadlock-free routing algorithm can be also obtained by allowing packets being routed from the X^+ Virtual Network to the X^- Virtual Network. But once the packets migrate from the X^+ Virtual Network into the X^- Virtual Network, they will not be routed back to the X^+ Virtual Network. As an alternative, once the packets migrate from the X^- Virtual Network into the X^+ Virtual Network, they will not be routed back to the X^- Virtual Network.



(a) 2 switches with VCs (b) Intra IO paths

Fig. 2.16: Two switches connected with virtual channels and the intra-IO interconnect paths of the switch.

When both virtual networks are combined in a mesh network with full-duplex interconnection, then two virtual channels connecting North and South ports must be implemented. In the full-duplex interconnection between two adjacent nodes, two single direction physical channels are used to connect the input and output ports between two adjacent port lines of the adjacent nodes. Fig. 2.16(a) presents two network routers, i.e. Router 1 and Router 2, connected with virtual channels through their North and South Input-Output ports.

As presented in Fig. 2.16(a), the *South-to-North physical link* connecting the South output port of the Router 1 with the North input port of the Router 2 consists of two virtual channels. Each virtual channel consists of two physical buffers, one is placed at the input port and the other is located at the output port. Similarly, *North-to-South physical link* connecting the North output port of the Router 2 with the South input port of the Router 1 consists of two virtual channels. The virtual channels are assigned with virtual channel ID 0 ($VC\ 0$) and virtual channel ID 1 ($VC\ 1$), respectively.

Packets routed through X^+ subnet will be routed through the $VC\ 0$, and packets routed through X^- subnet will be routed through the $VC\ 1$. The arrow lines presented in Fig. 2.16(a) are the physical and the virtual channels of the router. The figure also shows possible turns of routing paths that can be made by the packets through $VC\ 0$ and $VC\ 1$. For example, a packet routed from the West input port of the Router 1 can be routed to the North output port through $VC\ 0$, and a packet routed from the East input port of the Router 1 can be routed to the North output port through $VC\ 1$. After the packets routed from Router 1 through $VC\ 0$ has entered Router 2, then this packet can be routed

adaptively to either the $VC\ 0$ at the South output port or to the East output port. Meanwhile, after the packets routed from Router 1 through $VC\ 1$ has entered Router 2, then this packet can be routed adaptively to either the $VC\ 1$ at the South output port or to the West output port.

Alg. 5 shows the deadlock-free minimal adaptive routing algorithm with VCs for the 2-subnetwork interconnection presented in the Fig. 2.15 and the switch structure depicted in Fig. 2.16(b). In the Fig. 2.16, the Local IO port connecting the mesh router with a compute element is not presented for the sake of simplicity. As presented in the Alg. 5 and in the Fig. 2.16(b), routing paths through the $X+$ Sub-Net is done through the virtual channel ID 0 ($VC\ 0$), while routing paths through the $X-$ Sub-Net is done through the virtual channel ID 1 ($VC\ 1$). The all allowable intra IO crossbar interconnects in the switch with the VCs for both Sub-Nets is presented in the Fig. 2.16(b).

2.5 Performance Evaluation

Two main aspects that are important to test and evaluate the performance of a network are performance measurement metrics and workload models. Both aspects are explained in the following subsections.

2.5.1 Performance Measurement Metrics

Performance measurement metric enable us to compare the evaluation results of two or more network architecture. The performance of an on-chip network is architecture-dependent and technology-dependent. Two different network router architectures implemented on the same CMOS technology (similar transistor feature size) may have different performance. In contrast, two similar network router architectures will have different performance when they are synthesized in different CMOS technology. The smaller the size of the CMOS technology used to synthesize a network router, the higher the performance of the network router.

- *Packet Latency.* Packet latency can be defined as packet transfer delay, i.e. the amount of time needed by the packet to travel through the network from a node from where the packet is injected until its destination node. If t_{inject} is defined as the time at which the packet is injected from a source node, t_{eject} is defined as the time at which the packet arrives its destination node, then the packet delay Δt is formulated as $\Delta t = t_{inject} - t_{eject}$. The packet latency can be measured in *clock cycle period*, which is technology-independent metric or in *second*, which are technology dependent. The clock cycle period is ideal performance metric for latency measurement, because the architectures of the evaluated networks can be compared fairly, independent from the selected technology.

Alg. 5 Minimal Adaptive Routing Algorithm with VCs for 2 Sub-Networks

$X_{offs} = X_{target} - X_{source}$.
$Y_{offs} = Y_{target} - Y_{source}$.
Networks in divided into $SubNet\ X^+$ and $SubNet\ X^-$.
$North(VC\ 0)$ Virtual Channel 0 at $North$ output port.
$North(VC\ 1)$ Virtual Channel 1 at $North$ output port.
$South(VC\ 0)$ Virtual Channel 0 at $South$ output port.
$South(VC\ 1)$ Virtual Channel 1 at $South$ output port.
while $Packet$ is in $SubNet\ X^+$ i.e.($X_{offs} \geq 0$) **do**
  if $X_{offs} = 0$ and $Y_{offs} = 0$ **then**
    $Routing = LOCAL$
  else if $X_{offs} = 0$ and $Y_{offs} > 0$ **then**
    $Routing = NORTH(VC\ 0)$
  else if $X_{offs} = 0$ and $Y_{offs} < 0$ **then**
    $Routing = SOUTH(VC\ 0)$
  else if $X_{offs} > 0$ and $Y_{offs} = 0$ **then**
    $Routing = EAST$
  else if $X_{offs} > 0$ and $Y_{offs} > 0$ **then**
    $Routing$=**Select**($NORTH(VC\ 0), EAST$)
  else if $X_{offs} > 0$ and $Y_{offs} < 0$ **then**
    $Routing$=**Select**($SOUTH(VC\ 0), EAST$)
  end if
end while
while $Packet$ is in $SubNet\ X^-$ i.e.($X_{offs} \leq 0$) **do**
  if $X_{offs} = 0$ and $Y_{offs} = 0$ **then**
    $Routing = LOCAL$
  else if $X_{offs} = 0$ and $Y_{offs} > 0$ **then**
    $Routing = NORTH(VC\ 1)$
  else if $X_{offs} = 0$ and $Y_{offs} < 0$ **then**
    $Routing = SOUTH(VC\ 1)$
  else if $X_{offs} < 0$ and $Y_{offs} = 0$ **then**
    $Routing = WEST$
  else if $X_{offs} < 0$ and $Y_{offs} > 0$ **then**
    $Routing$=**Select**($NORTH(VC\ 1), WEST$)
  else if $X_{offs} < 0$ and $Y_{offs} < 0$ **then**
    $Routing$=**Select**($SOUTH(VC\ 1), WEST$)
  end if
end while

- *Communication Bandwidth.* Communication bandwidth can be defined also as data throughput, i.e. the amount of dataword (N_{word}) accepted during a certain time period (N_{time}). Hence the data-throughput/bandwidth (B_{comm}) is formulated as $B_{comm} = \frac{N_{word}}{N_{time}}$. The communication bandwidth can be measured in *number of accepted words per cycle* or *number of accepted flits per cycle*. The maximum value of the data throughput is 1 words/flits per cycle, which means that in every one clock cycle period, one data word/flit is accepted at destination node. When the evaluated on-chip network with a certain width of dataword have been synthesized using certain technology library and the maximum data (working) frequency is known, then network link and router bandwidth capacity can be measured in *kilobytes/Megabytes per second*.

2.5.2 Workload Models

The workload models to test the performance of networks are characterized by three main aspects, i.e. *Traffic Scenario*, *Injection Rate* and *Message Size*. The three aspects are generally explained in the following items.

- *Traffic Scenarios.* Traffic scenarios or traffic patterns can be called also as the *distribution of source-target communication pairs*. There are many traffic scenarios that can be used to evaluate the network performance behaviors. In [65], some data distribution scenarios are presented such as bit-reversal, bit-complement, butterfly, matrix transpose and perfect shuffle traffic scenario. The following items will present four examples of data distribution scenarios, where each network node with certain binary address will inject data to a destination node having the bit permutation address of the binary address of the data injecting node. Formally, the binary node-to-node data communication for a network node having n-bit binary address is described as

$$a_{n-1}a_{n-2}\cdots a_1a_0 \Leftrightarrow f_{permutation}(a_{n-1}a_{n-2}\cdots a_1a_0). \tag{2.6}$$

  1. *Bit Reversal.* The bit permutation of the bit reversal data distribution scenario is presented in the following.

$$f_{bit-reversal}(a_{n-1}a_{n-2}\cdots a_1a_0) = a_0a_1\cdots a_{n-2}a_{n-1} \tag{2.7}$$

  2. *Bit Complement.* The bit permutation of the bit complement data distribution scenario is presented in the following.

$$f_{bit-complement}(a_{n-1}a_{n-2}\cdots a_1a_0) = \neg a_{n-1}\neg a_{n-2}\cdots \neg a_1\neg a_0 \tag{2.8}$$

  3. *Perfect Shuffle.* The bit permutation of the perfect shuffle data distribution scenario is presented in the following.

$$f_{perfect-shuffle}(a_{n-1}a_{n-2}\cdots a_1a_0) = a_{n-2}a_{n-3}\cdots a_0a_{n-1} \tag{2.9}$$

The perfect shuffle permutation can be interpreted as a cyclical one-bit left-wise rotation function.

4. *Matrix Transpose.* The bit permutation of the matrix transpose data distribution scenario is presented in the following.

$$f_{transpose}(a_{n-1}a_{n-2}\cdots a_{\frac{n}{2}}a_{\frac{n}{2}-1}\cdots a_1a_0) = a_{\frac{n}{2}-1}\cdots a_1a_0a_{n-1}a_{n-2}\cdots a_{\frac{n}{2}} \quad (2.10)$$

- *Injection Rates.* The number of words/flits injected in every certain period of time is called injection rate. Injection rate can be measured in *number of injected words/flits per cycle* metric, or in *number of bytes per second* metric. The injection rates of messages from data producer nodes can affect the packet latency and communication bandwidth of the evaluated networks. Therefore, the bandwidth and latency responses of the network can be evaluated by varying the rates of data injection at the source nodes.

- *Message Sizes.* The number of words/bytes data injected from data producer nodes can affect the packet latency of the evaluated networks. The more workloads injected to the network, then the more traffics flow in the network, where in general the packet latency will also increases.

2.6 Research Fields Related to Networks-on-Chip

According to *Open System Interconnection (OSI)*, communication protocols in an interconnection network can be divided into 7 layers, i.e. *Application*, *Presentation*, *Session*, *Transport*, *Network*, *Data Link* and *Physical* layers as shown in Fig. 2.17. Further descriptions of each layer in the OSI model can be found in [50]. In the NoC communication protocols, the protocol layers can be shortened into five layers, i.e. application layer at the top layer, network switch layer and network link layer at the bottom layers, and in the middle layer, there are two interfaces to guarantee correct data interchanges between the top and the bottom layers. They are *on-chip network interface* ($OCNI$) layer and programming interface layer (software driver) which can be called as an *application programming interface* (API). The programming interface is a middeware (software) containing library or drivers to enable a processing element (PE) core passing messages to other cores via the network interface. The network interface (NI) is commonly implemented in hardware to assemble and disassemble a message into packets in such a way that the message can be routed correctly from a source PE to one or more target PEs through the network routers and network communication links.

The survey paper in [35] has presented research fields in the NoC area. The survey paper presents the NoC research area that is classified into four level, i.e. system level, network adapter (network interface) level, network switch level and network link level. Based on NoC layers presented in the Fig. 2.17 and the NoC research area classification

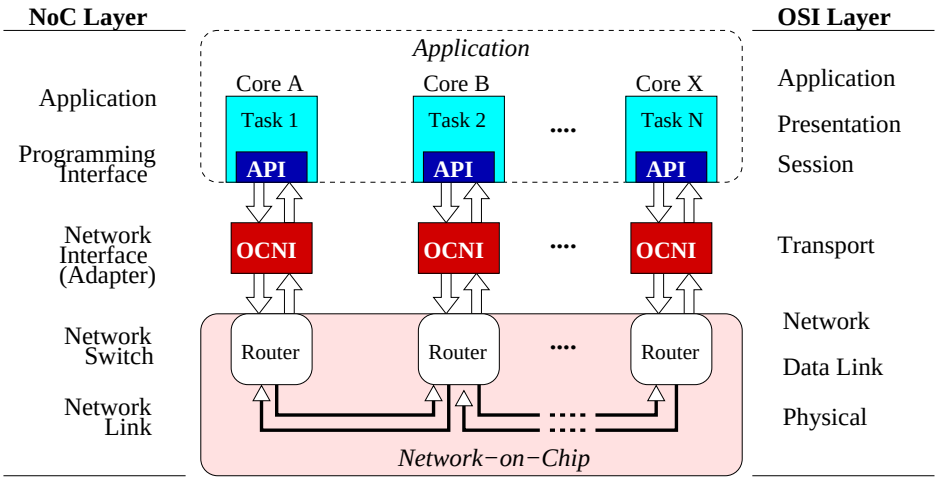


Fig. 2.17: NoC and OSI Model for interconnect protocol layers and the related NoC research areas.

described in [35], we will also present a NoC research area classification described in the following items.

1. *Application and System Level.* NoC research fields classified into this group are: hardware/software parallel task partitioning, task-Level parallelism, application mapping, NoC-based multiprocessor architecture exploration, traffic characterization and benchmarking, design methodology and system-level abstraction, etc.

2. *Application Interface Level.* NoC research fields classified into this group are: development of easy-to-use and comprehensive application programming interface library, development of drivers for embedded IP components, etc.

3. *Network Interface Level.* NoC research fields classified into this group are: service management for connectionless and connection-oriented data communication, end-to-end error correction and data protection, design reuse of network adapters, reconfigurable network adapters, etc.

4. *Network Switch Level.* Innovative switching methodologies, deadlock-free and fault-tolerance adaptive routing algorithms, routing methods for collective communication service, performance evaluation of network topologies, quality-of-service with data type classification, error protection encoding, switch circuit layout, etc.

5. *Network Link Level.* NoC research fields classified into this group are: data synchronization, link-level data flow control, link-level error correction and data protection, low-power data encoding, new data transmission media technology such as fiber optic and wireless media, etc.

Since the NoC research areas are very wide, we will briefly discussed about a few of them in the next subsections.

2.6.1 NoC Quality-of-Service

The NoC research area classification has been mentioned above. One of the interesting NoC research area is the *Quality-of-Service* (QoS) for NoC. The QoS for NoCs can be implemented almost in all NoC communication protocol layers. According to the paper chapter in [51], quality-of-service in end-to-end level can be implemented into three ways, i.e. *best-effort service*, *differentiated service* ($soft\ QoS$) and *guaranteed service* ($hard\ QoS$). The best-effort service does not provide guarantee of the data communication, where the messages are sent with connectionless protocol. The differentiated service classify the messages into several types with different levels of priority. The guaranteed service requires an absolute reservation of network resources.

The guaranteed service can be enabled by implementing a *switched virtual circuit configuration* method. This method is basically known as a data multiplexing technique. Some data multiple access techniques for NoCs have been introduced in the literature such as *Time-Division Multiple Access (TDMA)* [187], *Code-Division Multiple Access (CDMA)* [209], *Spatial-Division Multiple Access (SDMA)* [131] and *Identity-Division Multiple Access (IDMA)* [229]. Comparisons of the data multiplexing methodology will be explored later in Chap. 7.

The QoS for NoCs can also be implemented in the link-level and switch-level to guarantee the correctness of the data transmission. The work in [220] for instance presents a Generic and Extensible Spidergon NoC (GEX-Spidergon) that supports a CRC (Cyclic Redundancy Code) calculation for the link-level error transmission correction. The work in [202] presents also a NoC design methodology to tolerate timing errors due to overclocking operation mode without substantially affecting the latency for data communication. A low power and error protection coding for NoC is also presented in [206]. A bus-invert encoding method is used to reduce switching activity that can lead to high dynamic power dissipation. The work in [75] proposes a joint crosstalk avoidance and triple-error-correction/quadruple-error-detection codes as an error control coding schemes along the interconnects of NOC architectures. The works presented in [220], [202] and [206] have presented examples of the QoS for NoCs in the network switch and network link layers. However, the work in [68] has mentioned that the use of error-control schemes in on-chip networks results in degradable systems. Therefore the works introduces an "Interconnect Performability", i.e. a jointly unified performance and reliability measurement to consider the trade-off between performance and energy consumption of the error-control scheme.

2.6.2 NoC in Globally-Asynchronous Locally-Synchronous Context

The context of globally-asynchronous and locally-synchronous (GALS) should be considered in networked-multiprocessor systems. Skewed delay of the long distributed clock-tree is the key issue on why we need to apply the GALS context for the NoC-based multi-

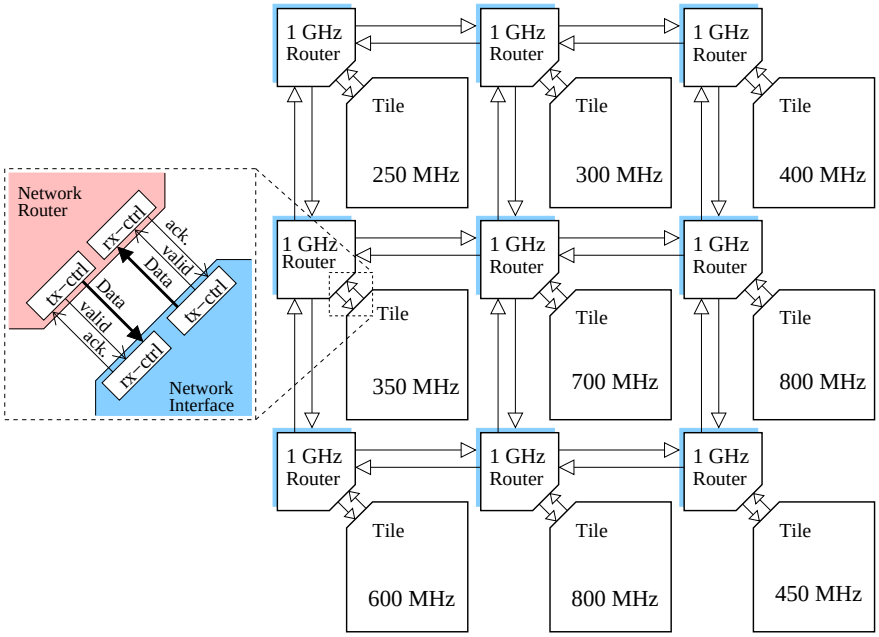


Fig. 2.18: Network-on-Chip-based multiprocessor system in GALS context.

processor systems. There are several techniques that can be used to implement a link-level data synchronization for GALS-oriented NoC. Examples of techniques that have been implemented so far in the NoCs are based on a *handshaking mechanism*, *source-synchronous technique*, *dual-clock buffer implementation*, *mesochronous technique* and so on.

Fig. 2.18 shows a multiprocessor system interconnected in 3×3 mesh network architecture in GALS context. The on-chip routers acting as the communication resource are clocked in 1 GHz, while computing resources are clocked with different clock cycle frequencies. The figure presents an example of GALS concept implementation by using data handshaking mechanism between each NoC router in the mesh network and each tile of the multiprocessor system. As presented in the figure, the handshaking asynchronous communication is made by introducing a pair of a data *validation* signal and an *acknowledge* signal in line with the data wire. The work in [141] for example presents a handshaking mechanism to implement asynchronous interconnects for SoC design. The work presents clock domain converters consisting of synchronous-to-asynchronous (S2A) converter for outbound data and asynchronous-to-synchronous (A2S) converter for inbound data.

In the source-synchronous method, the inter switch data communication is synchronized by using an additional clock signal line from the source (data transmitter) switch to the data receiver switch in line with the data lines. A NoC proposal that uses the source-synchronous technique is presented in [82]. The work proposes an interconnect scheme called a variable-tolerant low-power *source-synchronous multicycle* (SSMC). The work in [195] also presents a low power high-speed source synchronous transceiver.

The work in [198] has presented a multisynchronous and fully asynchronous NoCs

for GALS architecture. The asynchronous data communication between NoC switches is synchronized by using dual-clock FIFO buffers as a multisynchronous domain. The bisynchronous FIFO buffer is placed on each direction of the NoC link between data transmitter and data receiver switches. The bisynchronous FIFO buffer is then clocked by the clock signal from the data transmitter side for FIFO-write operation mode, and by clock signal from the data receiver side for FIFO-read operation mode. Since metastability issue (synchronization failure) probably occurs by using the multisynchronous approach, the work in [198] proposes also the fully asynchronous approach by providing synchronous interfaces to each local subsystem.

The work in [27] presents a power aware GALS-oriented NoC called *ALPIN* (*An Asynchronous Lower Power Innovative NoC*). Each tile connected to the ALPIN NoC is considered to have independent clock domain and voltage domain. The data communication synchronization is done by using a *pausable clock mechanism* which is implemented by using *Synchronous-to-Asynchronous and Asynchronous-to-Synchronous* Interface (SAS). A local clock generator is programmable and implemented within each unit to generate a variable clock frequency in a predefined and programmable tuning range. In order to optimize the power consumption, an adaptive power reduction technique is introduced to control the local power supply in the internal core.

The work in [127] presents a reconfigurable baseband platform (*FAUST chip*) by using the asynchronous GALS-implemented NoC presented in [27]. Another work that considers a NoC as partitioned voltage-frequency island is presented in [174]. Theoretically the work has presented the methodology to assign the frequency and voltage level for partitioned NoC. However, the work in [174] has not verified the impact of the voltage level assignment on the energy consumption, since the work implements the NoC on an FPGA device that does not support voltage level conversion.

In the mesochronous technique, the clock signal distributed for IP components is the same. The mesochronous approach considers each component to have an arbitrary amount of skew, i.e. time-invariant phase clock offset, depending on the distance of the component floorplan position from the clock source. The work in [207] proposes a *SIM-L* architecture (*Skew-Insensitive Mesochronous Link*) to solve the wire delay problem in the asynchronous design style.

Although the asynchronous on-chip interconnects are interesting topics for future NoC developments, not all the asynchronous methods are compatible with industrial CAD design tools. The work in [185] for instance has presented an effort to integrate the standard CAD design flow dedicated to design synchronous integrated circuit. The synchronous NoCs is still an interesting topic. The innovation of a low skew clock-tree for pipeline stages implementation in the synchronous NoCs will still be a challenging issue.

2.6.3 NoC Application Mapping

In the embedded MPSoC applications, the processing elements can be a software-based processor component such as CPU, DSP, microcontroller, and can be a hardware-dedicated component such as an ASIC device. In order to run an application into the NoC-based MPSoC systems, the application must be partitioned into several tasks. The work in [24] for instance has presented the partitioning result of an MPEG4 decoder core into five tasks run concurrently on an 2D 3×2 mesh architecture. In general the application is depicted in a *task communication graph*.

Some algorithms used to solve combinatorial problems [193] have been used so far to map an application onto a NoC platform, since the application mapping problems is related to the combinatorial problems. Based on the NoC topology architecture to which the application will be mapped, the NoC application problem can be in general divided into application mapping on regular and non-regular NoC topologies. The work in [101] uses a *Branch and Bound Algorithm* to map application onto a regular mesh architecture. The objective function of the mapping algorithm is to optimize performance and communication energy. The extended work has been presented in [49], in which the work considers the problem of mapping multiple applications onto regular mesh architecture for NoCs with multiple voltage levels. In order to obtain an efficient mapping result, the methodology is based on an incremental mapping approach by using a *Near Convex Selection Technique*.

The other works presenting mapping problems on the regular networks are presented in [192], [132] and [148]. The work in [192] presents a tool called *SMAP* that uses a *Spiral Algorithm* to map application onto 2D mesh platform and compared the result with *Genetic Algorithm*. As mentioned in the paper, the tool can also be used to map an application onto other NoC topologies with different network sizes. The work in [132] also uses a *Genetic Algorithm* to optimize computational and communication energy by creating concurrently a voltage island partitioning and assignment for NoC with multiple voltage levels. The work in [148] compares and proposes algorithms to obtain a low-energy mappings onto NoCs. The compared and proposed algorithms are *Exhaustive Search (ES) algorithm*, two stochastic search algorithm, i.e. *Simulated-Annealing (SA) Algorithm* and *Tabu-Search (TA) Algorithm*, a greedy heuristic algorithm called *Largest Communication First (LCF)* and another *Greedy Incremental (GI) Heuristic Algorithm*, as well as a mixed LCF-SA and mixed LCF-TS algorithms.

The application mapping onto non regular or custom-made network topologies are more complex than the mapping onto regular topologies because of an additional case to generate NoC topology architecture before the task application mapping. The work in [43] presents a *power-aware topology construction algorithm* to map applications onto a custom NoC topology in order to optimize communication power for inter-task communications. The work in [45] also presents an automated technique for the synthesis of application-specific NoC. The other versions of the work is presented in [199] and [128]

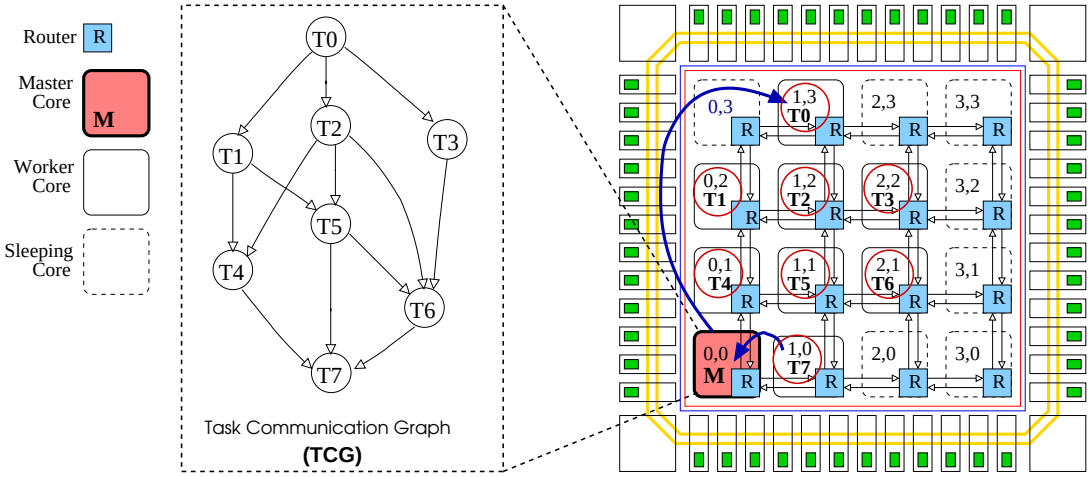
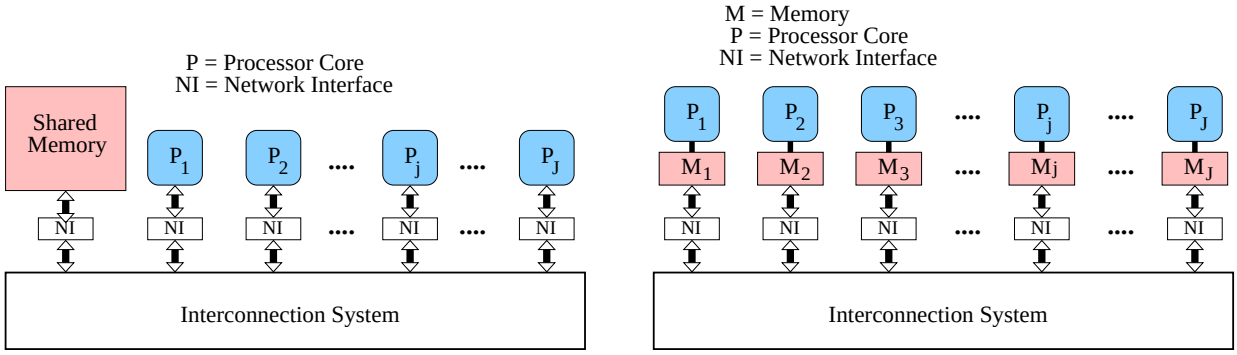


Fig. 2.19: Parallel task-based application mapping on the CMP System.

that uses a simple linear programming technique and genetic algorithm, respectively, to generate and to map applications onto NoC with non-regular topology. The work in [32] shows a NoC synthesis flow to map applications onto NoC with different topologies such as mesh, torus, hypercube, 3-stage clos and butterfly topology. The framework uses *SUNMAP* tools to perform *topology mapping* and *selection function* to select the best topology from a library of NoC topologies.

The application mapping topics can be divided generallay into *Pre-Chip Manufacture* and *Post-Chip Manufacture* application mappings. Most of the current embedded MPSoCs make the application mappings at design time, where applications are known before the chip is fabricated. This approach is static and limited to a specific application, and task allocation (mapping) can be better optimized. However, this approach is not valid when the applications are not known at design time [54]. Therefore, the application mappings on the post fabricated chip is an interesting approach to provide an open solution for several target derivatives (various applications) on the same silicon [54]. The post-chip manufacture application mapping is the special case of the task allocation issue for chip-level multiprocessor (CMP) systems. However, this approach could also be an interesting issue for the embedded MPSoC application.

Fig. 2.19 shows the mapping of a parallel task-based application on a CMP system. Every task is assigned into a single networked computing core (worker core). The mapping algorithm is done in the master core of the CMP system to optimize the communication energy of the application. Afterwards, the master core will deploy the executable code of each task to the local instruction memory of each computing resource to which the task is assigned. We can see for example, the task number 0 and 7 (*T0* and *T7*) is assigned to the worker core at node (1,3) and node (1,0), respectively.



(a) Shared Memory (b) Distributed Memory

Fig. 2.20: Shared-memory and distributed-memory multiprocessor architecture.

2.6.4 NoC-based Multiprocessor Systems and Parallel Programming

In general, multiprocessor architecture can be divided into two classification, i.e. *Shared-Memory Architecture* and *Distributed-Memory Architecture*. Fig. 2.20 shows the shared-memory and distributed memory architectures for multiprocessing systems. The selection of parallel programming model can determine the architecture for the multiprocessor system. The shared-memory architecture as presented in Fig. 2.20(a) is suitable when using a shared memory programming model. The parallel programming models for multiprocessor system will be explained later towards the end of this subsection. However, the shared-memory programming model can also be run on the distributed memory architecture as shown in Fig. 2.20(b). This approach is commonly called a *Distributed Shared-Memory (DSM)* system architecture.

The DSM system architecture has been an issue in all kinds of multiprocessor systems in recent years. Especially in supercomputing, memory access topologies and memory bandwidth are the crucial points for gaining the targeted overall system performance. In [66], a performance evaluation for the Cray X1 DSM architecture is presented. In X1 multistreaming processors (MSPs), memory access is performed via a cache, which is shared by four single stream processors (SSPs). Four MSPs share 16 memory banks, having 16 individual memory controllers. This allows local memory access in parallel to global data communication, accessing some of the 16 memory banks.

Principles of DSM architectures have already been presented in [169], where structure, granularity and coherence issues are described. The work in [42] gives a clear description and evaluation of producer-consumer mechanisms in shared memory multiprocessors. Comparing producer-initiated and consumer-initiated data communication schemes, producer-initiated mechanisms (as data forwarding and user-level message passing) provide the highest efficiency, being comparatively insensitive to network parameters (latency, bandwidth) [42]. In [4], a dynamic approach for balancing memory access and avoiding access contention is presented, which applies memory page migration in consumer-initiated DSM systems.

Interesting DSM reference architectures are represented also by the MIT Alewife Machine architecture [2] and the Stanford DASH (Directory Architecture for Shared Memory) multiprocessor [130]. Bhuyan et al. [34] presented a multistage bus-based architecture for the realization of a DSM system. Most of the aforementioned multiprocessor system architectures are dedicated for "off-chip" or "on-board" multiprocessor system. The work in [158] for instance, presents a crossbar NoC architecture as a platform for a shared-memory architecture, where several processing elements, several shared memory units and a main memory controller are connected to a central crossbar. This approach also follows the NUMA paradigm.

According to the online tutorial presented in [20], parallel programming models can be classified into *Shared Memory Programming*, *Message Passing Programming*, *Thread-based Programming* and *Data Parallel Programming* models. According to Flynn's Taxonomy as shown in the same source, the programming models can be traditionally classified based on the way the data and instructions are parallelized, i.e. *Single-Instruction Single-Data (SISD)*, *Single-Instruction Multiple-Data (SIMD)*, *Multiple-Instruction Single-Data (MISD)* and *Multiple-Instruction Multiple-Data (MIMD)* programming models. The MIMD programming model is an interesting model for industries and academia that have been applied to model parallel computations. The work in [139] for example, have demonstrated the used of the MIMD programming models in conducting "A Large Ion Collider Experiment" (ALICE) at CERN, the European Nuclear Research Center in Geneva. The work has presented the development of *Multi-Chip Module (MCM)* for transition radiation detector (TRD) that requires 70,848 chips, where each chip consists of four RISC processors resulting in a total number of 283,392 processors. GeForce 6800 [159] as media processor system for instance, is programmed by using the SIMD programming models. The GeForce contains 6 vertex processors and 16 fragment processors, where each of which is a VLIW and SIMD parallel processing engine.

In the shared memory programming model, a common address space in a global memory is shared by several parallel tasks. Data ownership in the shared-memory program cannot be well viewed by programmers, and data communications between the tasks are very implicit. Hence, data locality cannot be controlled by programmers. In contrast, in the message passing programming model, data locality and communications are explicit in the programmers point of view, because the programmers are responsible to determine parallelism and data exchanges between the tasks. Application programming interface (API) library that are commonly used as subroutines to develop a message passing parallel program are "MPI" [156] and "PVM" [79]. Both libraries are available for C/C++ and Fortran computer languages. Tutorial for the message passing programming model by using the MPI library can be found in [21].

The thread-based parallel programming model can be implemented in the shared-memory multiprocessor architecture. Routine or procedure in a computer that can be run concurrently with other ones is best described as a thread. Multiple independent instruction streams (threads) running simultaneously must be scheduled by computer

operating system. Two common application programming interface (API) libraries used to a build thread-based (multi-threaded) shared-memory parallel application are POSIX-Threads [23] and OpenMP [22]. The POSIX-Threads API has been an IEEE Standard (IEEE Std 1003.1) to develop portable thread-based parallel computer programs. A joint group of major computer hardware and software vendors has defined the specification of the OpenMP (Open Multi-Processing) API library [175]. The work in [15] presents the paradigm shift in the OpenMP view from thread-centric to task-centric. The task-centric OpenMP was developed to express the task-level parallelism and to improve the limitation of the existing OpenMP standard presented before. The enhancement of the OpenMP such that it can be implemented to program multicore systems is presented in [44].

Some existing multiprocessor systems such as Montecito [151], AMD Opteron [121] and Niagara [119] are multiprocessor systems that can be programmed by using the thread-based parallel programming model. Multiprocessor systems capable of running multithread parallel applications are commonly called *Symmetric Multi-Threading (SMT) Machine*.

The report in [53] presents the evaluation of the capability and limits of current scientific simulation development tools and technologies with specific focus on their suitability for use with the next generation of scientific parallel applications and High Performance Computing (HPC) platforms. The report presents also an interesting spreadsheet outlining current capabilities and characteristics of leading and emerging tools in the high performance computing arena. Probably, not all specifications dedicated for HPC platforms could be adopted to the embedded MPSoC or general-purpose CMP platforms, but the idea of such concept can be implemented for future on-chip multicore processors generation. The work in [40] has proposed an aggressive framework, including a full-featured compiler infrastructure for thread-extraction, that can produce scalable parallelism from a sequential program without changes to the sequential programming model.

2.6.5 Testing Methods for NoC-based Multiprocessor Systems

A complete test for a NoC-based CMP system is grouped into two main tests: network and tile tests. The network test can be divided into switch and link test, while the tile test can be divided generally into memory and processor core test. A complete strategy for a multiprocessor system has been proposed in [5]. Each resource is equipped with a Test Wrapper in accordance with a standard wrapper presented in [186] to enable a local BIST (Built-in Self Test). In general, the main objective of testing methods are to minimize test application time and energy, to find fault-models such that faults can be found and localized, and to accurately verify the fabricated chips, whether they can be released in markets or not.

Testing methodology for network switch and link have been presented so far in literature. A test method for crosstalk-induced delay and glitch faults in NoCs with an

asynchronous communication protocol is presented in [29]. An efficient NoC switch test methodology has been presented in [99], where test vectors are broadcasted using a minimum spanning tree technique. The work in [88] has also proposed a multicast scheduling algorithm to optimize the test time for NoC communication fabrics. A cost-effective test sequence for the testing of data, control and handshake mechanisms in a mesh NoC is exhibited in [55] and [69]. The work in [33] has presented a robust concurrent methodology to make an online NoC testing.

Testing the functionality of the tiles interconnected in a NoC communication system is also a challenging research area. Since the tiles can be an ASIC (IP component) or a microprocessor system containing cache (on-chip memory) and CPU core, then testing methodology can be done with different approaches. The works in [9] and [122] use a NoC as test access mechanism (TAM) to test embedded cores. However, in this approach, the NoC switches and links must have been tested previously to let us know the NoC condition. Totally fault switch can lead to the isolation of the core connected directly to the fault switch from the NoC-based multiprocessor system.

2.6.6 ASIC and FPGA Implementation Issue

In general, NoC VLSI architectures can be implemented on an *Application-Specific Integrated Circuit* (ASIC) or on a *Field Programmable Logic Device* (FPGA). The work in [180] has presented and analyzed the challenge to implement a NoC router on ASIC technology. The work in [184] has presented also potential problems and challenge to implement a NoC router on ASIC device by using 65-nm CMOS standard-cell technology. Layout-aware analysis of NoCs for multiprocessor systems has been described in in [11].

The NoC architecture, in a special case where the size of the overall system architecture is not too large, can also be implemented on the FPGA device. Current FPGA devices that have been released in market thus far have a limited number of programmable logic slices. Based on such situation, the implementation of a large size (massive) NoC-based multiprocessor system on an FPGA device is still limited, and will be implementable until FPGA devices having massive logic slices exist. The FPGA implementation of the NoC architecture on the FPGA device will be preferably made for system emulation purpose [81].

The work in [164] for example has presented a platform for MPSoC emulation, where the limitation of the FPGA logic block enforces the authors to intoduce two solutions. When, the number of the computing resources is four or fewer, then a single FPGA device with more than 1,000 IO pins is used to design the MPSoC system. For more complex MPSoC platforms, a board based on an array of FPGAs that is oriented to a NoC-based communication resource is used to design the platform.

2.6.7 Advanced NoC Research Issues

Advanced issues related to future NoC research consist of a novel integration technology and architecture for massively-parallel NoC-based multiprocessor systems and new media technology for low-power data communications. A stacked 3D NoC integration is one of the possible solutions to integrate IP components, memory components and communication resources on a single chip. The main issue of the new concept is the development of new computer-aided design (CAD) tools for automated NoC design, synthesis and floorplanning. Since the power consumption is one of important aspects to design extremely ultra-large scale integration (ULSI) systems, the innovation of an ultra low-power medium is also one of the top future topics.

The main feature of the 3D integration is the vertical interconnects between the stacked layers in the 3D ICs. The work in [143] has presented a low-overhead fault tolerance scheme for 3D NoC links based on the Through Silicon Via (TSV). The work in [183] has presented a multi-layered on-chip interconnect for 3D NoC Router architecture. The work tries to optimize and reduce the overall area requirements and power consumption by using a cycle-accurate 3D NoC simulator. The research about 3D NoC integration is not yet mature and still requires further investigations to cover some issues such as CAD supports, potential chip defects, power dissipations, inter-layer data synchronization, etc.

The innovations of a new medium technology to provide low-power data communication will be a challenging issue in the future. Among many ideas, two potential new media that have been investigated so far as presented in the literature are fibre optics and wireless (air) media. In the fibre optics media, data or information are transported via light (photonic) signals. The main issue related to the photonic technology is that the light cannot be stored in storage components such as FIFO buffers in NOC router switches. CMOS photonics for high-speed interconnects has been developed in [91]. The work in [197] has analyzed a promising approach to use fibre optics as a new medium for NoC data communications. Meanwhile, the work in [218] has presented a wireless ultra-wideband NoC using SD-MAC (synchronous and distributed medium access control) protocol with collision-free on-air data routing to provide a quality-of-service. The work takes advantages from the recent Radio-Frequency CMOS technology for wireless physical channel design. However, like the 3D integration, the photonic and wireless interconnect technologies for NoCs are also not yet mature.

2.7 Summary

This chapter has presented some fundamental aspects related to networks-on-chip. The VLSI microarchitecture and implementation of a NoC router depends on the selection of the data transmission service (with unicast, with or without multicast service), routing implementation (state machine or routing table or combination of both), routing algorithm (static or adaptive), switching methodology (packet, wormhole or circuit switch-

ing), data synchronization mode (synchronous or asynchronous) and the data integrity requirement (with or without error correction check). The routing algorithm of a NoC is strongly dependent on the topology architecture of the NoC. Deadlock configuration problem is an important issue to design a routing algorithm for a NoC router. There are many techniques to handle the deadlock configuration problem. This chapter has presented only two techniques, i.e. deadlock avoidance based on turn models and virtual-channels, which are in general the most preferable solutions.

This chapter has also presented some issues related to NoC research area and other topics strongly related to the NoCs such multiprocessor systems, network interface and parallel programming models. However, this thesis will only focus on four main aspects, i.e. switching method, multicast routing implementation, routing adaptivity design and QoS implementation for guaranteed message delivery service in the network protocol layer. Prior to discussions of all these aspect in the next chapters, a concept and VLSI microarchitecture of a NoC called **XHiNoC** (e**X**tendable **H**ierarchical **N**etwork-**o**n-**C**hip) that flexibly supports the aforementioned aspects as well as the formal descriptions of the proposed NoC router will be presented next in Chap. 3.

Chapter 3

Overview of the On-Chip Router

Contents

This chapter will give a general overview of the concept, special features, characteristics and generic modular VLSI architecture of a NoC router prototype called *XHiNoC*

(e**X**tendable **H**ierarchical **N**etwork-**o**n-**C**hip) developed at Institute of Microelectronic Systems, Darmstadt University of Technology, as part of this thesis. A preliminary introduction to the concept of the sharing of communication media (NoC links and routers) for networked multiprocessor systems is described in Section 3.1.

The generic components and microarchitecture of the XHiNoC router supporting the development of the flexible communication media are explained in Section 3.2. This chapter will also present some specific features and characteristics of the XHiNoC communication system as described in Section 3.3. The simulator equipment (testbench modules) for performance evaluation of the XHiNoC as well as the special feature of the simulator equipment is explained in Section 3.4. Section 3.5 will summarize the discussion about the XHiNoC communication infrastructure including several issues related to the design concept.

3.1 Design Concept

The XHiNoC is developed not only to provide a communication infrastructure for embedded multiprocessor systems-on-chip (MPSoC) devices, which are generally implemented for consumer electronic products, but also for further implementation of chip-level multiprocessor (CMP) systems, which are designed for general-purpose multicore computer systems. The main purpose of the XHiNoC is to give a flexible shared communication media of the on-chip interconnection network. In order to provide such flexible communication media sharing, a concept of locally organized packet identity (ID) division multiple access (IDMA) method suitable for NoCs is introduced. Local ID slots are distributed over every communication link, which can be attached to every flit of a packet or data stream as its local ID-tag.

The use of the locally organized message ID, where variable local ID-tag in line with additional flit type control bits attached on each flit of messages, will:

1. enable us to apply a runtime routing reservation table programming when the table-based routing method is used to route packets,

2. enable us to reduce the size of the routing reservation table on every NoC router input port when the NoC will be designed for logic area optimization purpose, and would be applied for embedded MPSoCs having predictable traffics characteristic.

3. allow us to implement a wormhole switching method, in which flits belonging to different messages can be interleaved at flit level in the same communication link,

4. allow us to apply a very flexible concept for communication media share methodology supporting unicast and multicast message routing for collective communication service, and

5. enable us to implement a flexible runtime connection-oriented communication that is commonly required in multimedia applications with streaming-based data (audio/video) communications, i.e. the virtual circuit configuration can be made autonomously by packet or data stream headers at runtime during application execution time.

3.1.1 Media Sharing with Local ID Management

Fig. 3.1 presents an example of the flexible communication media sharing by using the local ID-tag organization and management. The local ID-tag reconfiguration and management in the XHiNoC are made autonomously by packet headers at runtime during application execution time. The figure shows one example of many possible local ID slot configurations and shows a small 2D 3×2 mesh NoC topology (6 NoC routers i.e. R_k, where $k \in \{1, 2, 3, 4, 5, 6\}$). Seven messages, i.e. message A, B, C, D, E, F and G are routed in the networks and share the communication resources. Message B for instance is injected from router node R_4 with local ID-tag 1 and is accepted or ejected in the router node R_3 with local ID-tag 0. On the next remaining intermediate downstream links, the message B reserves local ID-tag 1 and 0, successively, until it reaches its destination node (R_3). During transmission in the intermediate links, the local ID-tag of each message is updated and mapped properly to allow resource communication sharing with other flits of different messages in interleaving manner. For example, in the link connecting East port of R_4 and West port of R_5, the local ID-tag of the message B is 2. While the others messages, i.e. message A and message D reserve local ID-tag 0 and local ID-tag 1 respectively. Thus, three messages share the link with three different local ID tags. There is no restriction such that every message must be allocated to a certain local ID slot. The local ID slot reservation made by every message depends on the current status of the local ID slots, they are free or have been reserved by other messages.

In order to support such flexible method to share the communication media, a policy must be applied to guarantee correct routing path and proper flits forwarding of each message. A few rule that must implemented in the proposed methodology is explained in the following.

1. Flits belonging to the same message or data stream will always have the same local ID-tag. In oder words, different flits belonging to different message will have different local ID-tags on every local communication link.

2. The ID-tag of each message will be updated each time it enters a new communication resource (link). Two parameters, i.e. previous ID-tag in the previous link and input port number from which a message comes can be used to index a reserved local ID tag.

3. A local ID-tag that has been reserved by a message cannot be used by other messages until the message has terminated the reservation of the local ID-tag.

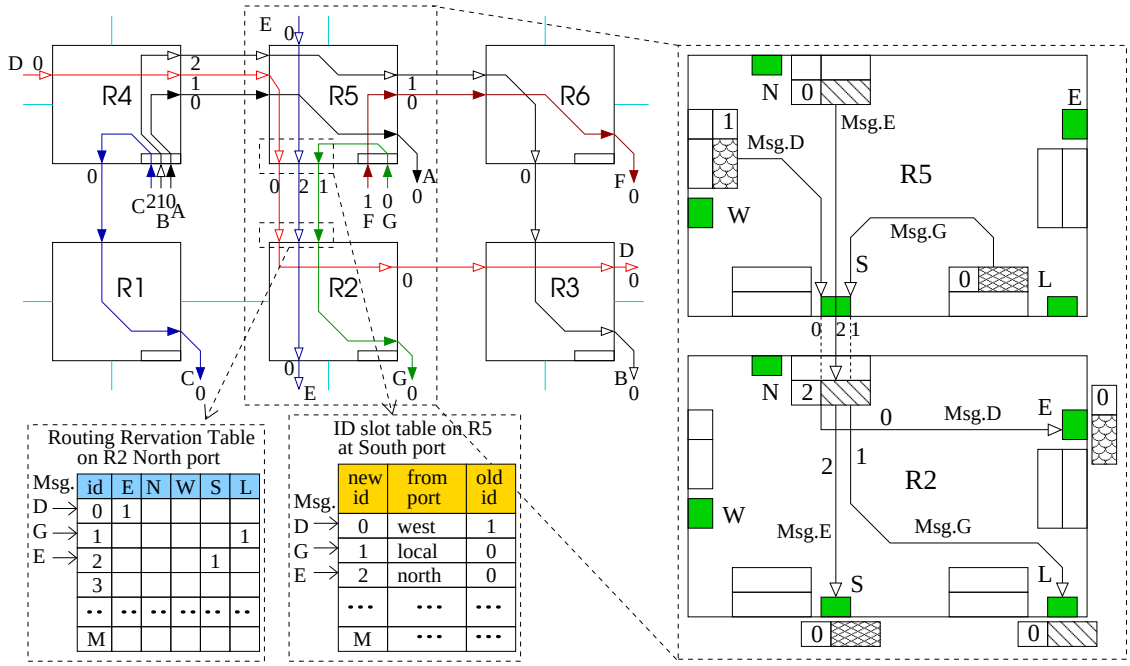


Fig. 3.1: Flexible concept view of the communication media share with local ID-tag management.

Based on the policy and rules to implement the proposed flexible communication media share methodology, few implementation consequence must be fulfilled to support the concept. Firstly, the need for a specific packet format to transport message through the network, and secondly, the need for local ID slots, which must be implemented over all local communication link, where a message allocated to a local ID slot k on a current link will use the slot k as its ID-tag on the link.

Fig. 3.2(a) presents the generic specific packet format of the XHiNoC. A message or a streaming data is divided into *flow control digits* called *flit*. The definitions of the flits are described in Def. 3.1, Def. 3.2 and Def. 3.3, respectively. The total bit-width of each flit of the message or streaming data is $b_{total} = b_{type} + b_{tag} + b_{word}$, where b_{type} is the bit-width of the flit type field, b_{tag} is the bit-width of the id-tag field and b_{word} is the bit width of the data word.

Unicast message/stream is single packet that consists of one header flit, databody flits, and one tail flit, while a multicast message/stream consists of more than one header flits, where the number of header flits depends on (equals to) the number of the multicast destination nodes. Even if the size of the message/stream is extremely large, it has only one tail flit. In other words, a message or data stream is assembled in a single packet.

Definition 3.1 (Data Flit) *A data flit (flow control digit) coming from input port n is represented as $F_n(type, ID)$. It consists of a data word with additional flit type field ($type$) and local ID-tag field. Each flit will always bring a data word together with its type and its local (See Fig. 3.2(a)).*

Definition 3.2 (Flit type) *The $type$ field represents the type of each flit. The flit $type$ can be a header, a databody or a tail flit ($type \in \varepsilon_{type} | \varepsilon_{type} = \{header, databody, tail, response\}$).*

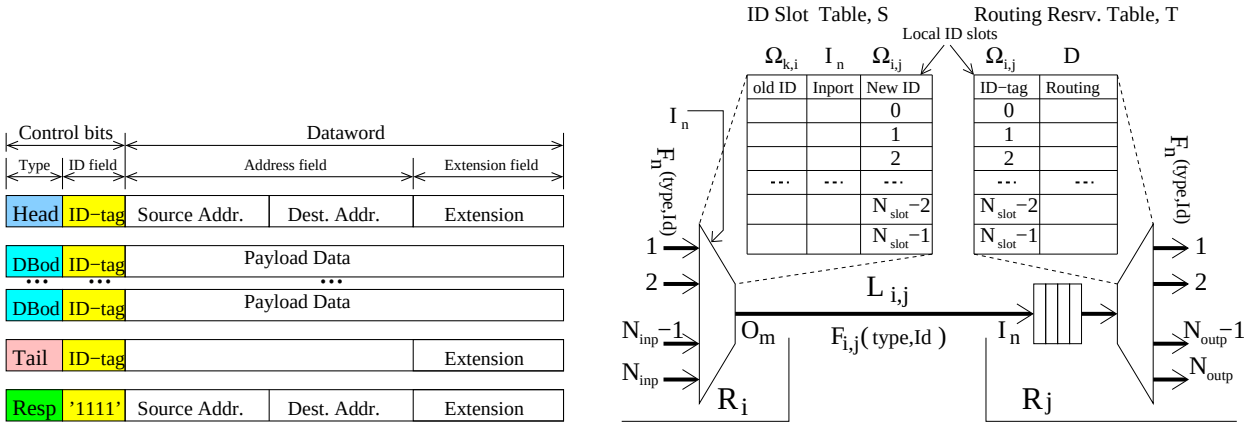
Based on the Def. 3.2, the types of flits can be identified into four types that are described in the following.

- *Header* flit is a flit that is attached in the probe of a message. As the leading flit, the address of the destination node is written on the bit-field of the flit. The header flit initiates the ID slot reservation including routing table slot reservation and ID tag updating function. The header flit is used basically to establish routing path or to configure connection.
- *Databody* flit is identified as the payload data of a message/stream. Hence, the substantial word of the message is injected into the NoC as databody flits.
- *Tail* flit is used to mark the end of a message/stream. The tail flit can bring a substantial word of the message or a special information or non-substantive word. The tail flit is basically used to close the routing path or to terminate the connection.
- *Response* flit is used when a connection-oriented guaranteed-service communication protocol is implemented in the NoC router, where the response flit is sent by a destination node to inform the source node about the status of the connection, i.e. successful or fail. When a best-effort data communication protocol is implemented, the response flit is used to initiate data retransmission when data drop mechanism is allowed, because one free ID tag in the acquired link cannot be allocated for the message, since all available ID slots have been reserved by other messages (runout of ID slots).

Definition 3.3 (Flit ID-tag) *ID-tag field present on each flit is a local label (ID-tag) to indicate and differentiate the flit from different flits. Flits belonging to the same message or streaming data will always have the same local ID-tag on each communication link $L_{i,j} \in \Lambda$. The value of the local ID-tag is defined as $ID \in \Gamma | \Gamma = \{0, 1, 2, \cdots, N_{slot} - 1\}$ where N_{slot} is number of available ID slot on communication link $L_{i,j} \in \Lambda$. See also later the definition of the local ID slot in Def. 3.4.*

Definition 3.4 (Local ID Slots) *Each communication link $L_{i,j} \in \Lambda$ has N_{slot} number of available local ID slots, which is defined as a set $\Omega_{i,j} \subseteqq \Gamma$ (See Def. 3.3). If an assumption is made such that all communication links has the same number of available ID slots, then we will have $\Omega_{i,j} \subseteqq \Omega$. We define a single local ID slot $k \in \Omega$, where $k = N_{slot} - 1$ is reserved for packet flow control purpose, and the usable ID slots are $\forall k \in \Omega \cap k \neq N_{slot} - 1$.*

Fig. 3.2(b) presents the concept view of a communication link $L_{i,j}$ of the XHiNoC connecting router R_i and R_j. A flit flowing through the link $L_{i,j}$ is exhibited as $F_{ij}(type, ID)$. Based on Def. 3.3 and Def. 3.4, a message allocated to an ID slot $k \in \Omega$ will then use the slot number k as its local ID-tag ($ID = k$). Fig. 3.2(b) give us insight that the flows of message flits on the link $L_{i,j}$ is controlled by configuring the ID slot table at the output port of the router R_i and the routing reservation table at the input port of the router R_j.



(a) Generic Packet Format (b) ID slots of Local Link

Fig. 3.2: The specific packet format and local ID slots.

3.1.2 Main Issue Related to Local ID Slots Availability

The main issue related to the local ID-based method for flexible communication media share is a *runout of local ID problem*. In a certain NoC topology, there will be a fact that there are some links that are never used by packets sent from some computing element cores to some destination cores. Let us examine the case of a NoC in 2D $N \times M$ mesh topology, and let us also set the address and port names of each node as (x, y, O_p), where x is the horizontal address such that $0 \leq x \leq N-1$, y is the vertical adddress such that $0 \leq y \leq M-1$, and $O_p \in \{East, North, West, South, Local\}$. Thus, when each communication edge of the 2D $N \times M$ mesh architecture is implemented with full-duplex link, then, except for the local port connected directly to a computing element core, the number of available local ID slots in the North, South, East and West port can be set to a minimum number, i.e. less than $N \times M$ number of local ID slots.

When a *minimal fully adaptive* routing algorithm is used to route packets, then the minimum number of the available ID slots that can be set to each output port of every NoC router is shown in Equ. 3.1.

$$N_{Min.Slot}^{Full.Adap}(x, y, O_p) = \begin{cases} M(x+1), & 0_p = East \quad and \quad 0 \leq x < N-1 \\ N(y+1), & 0_p = North \quad and \quad 0 \leq y < M-1 \\ M(N-x), & 0_p = West \quad and \quad 0 < x \leq N-1 \\ N(M-y), & 0_p = South \quad and \quad 0 < y \leq M-1 \\ NM-1, & 0_p = Local \\ 0, & otherwise \end{cases} \tag{3.1}$$

When a *XY static* routing algorithm is used to route packets, then the minimum number of the available ID slots that can be set to each output port of every NoC router is shown in Equ. 3.2.

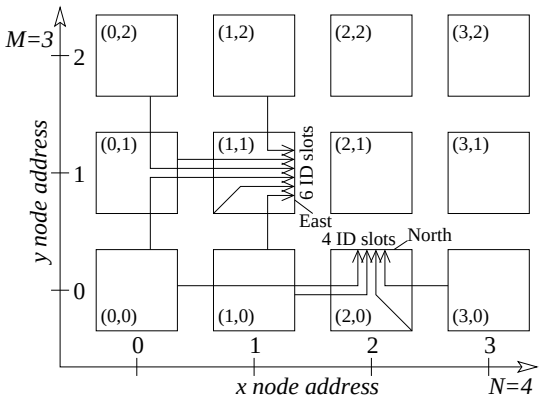


Fig. 3.3: An example of the minimum number of available ID slots at two selected output ports when using minimal fully adaptive routing.

$$N_{Min.Slot}^{XY\ Static}(x,y,O_p) = \begin{cases} x+1, & 0_p = East \quad and \quad 0 \leq x < N-1 \\ N(y+1), & 0_p = North \quad and \quad 0 \leq y < M-1 \\ N-x, & 0_p = West \quad and \quad 0 < x \leq N-1 \\ N(M-y), & 0_p = South \quad and \quad 0 < y \leq M-1 \\ NM-1, & 0_p = Local \\ 0, & otherwise \end{cases} \tag{3.2}$$

The Equ. 3.1 and Equ. 3.2 are derived with the assumption that a router node will not send packets to itself. The minimum number of ID slots at the Local output port is set to $(NM-1)$ to anticipate an *all-to-one* communication, a kind of collective communication mode, where all nodes send a message to the one target node. When the number of ID slots at each output port is set according to Equ. 3.1 and Equ. 3.2, then we can guarantee that the ID slot runout problem in the 2D $N \times M$ mesh architecture can be avoided. Therefore, a packet dropping mechanism in the data link layer and retransmission protocol in the transport layer can be neglected, because both mechanisms may end up in unfairness for some traffic flows

Fig. 3.3 shows an example of the minimum number of acceptable available ID slots at the East output port in the mesh node $(1,1)$ and at the North output port in the mesh node $(2,0)$, when we use a minimal fully adaptive routing algorithm for the 2D 4×3 $(N=4, M=3)$ mesh network architecture. By using Equ. 3.1, the minimum number of ID slots at the East output port at the node $(1,1)$ should be 6 ID slots. As presented in the Fig. 3.3, the East outpot port at node $(1,1)$ can be acquired by 6 messages from 6 other mesh nodes, i.e. from mesh nodes $(0,0)$, $(1,0)$, $(0,1)$, $(1,1)$, $(0,2)$ and $(1,2)$. By using the static routing algorithm, then according to Equ. 3.2, the East output port will be only acquired by 2 messages, i.e. messages from mesh nodes $(0,1)$ and $(1,1)$. Meanwhile, the North output port in the mesh node $(2,0)$ can be accessed by 4 messages according to Equ. 3.1, i.e. the messages injected from mesh nodes $(0,0)$, $(1,0)$, $(2,0)$ and $(3,0)$ as

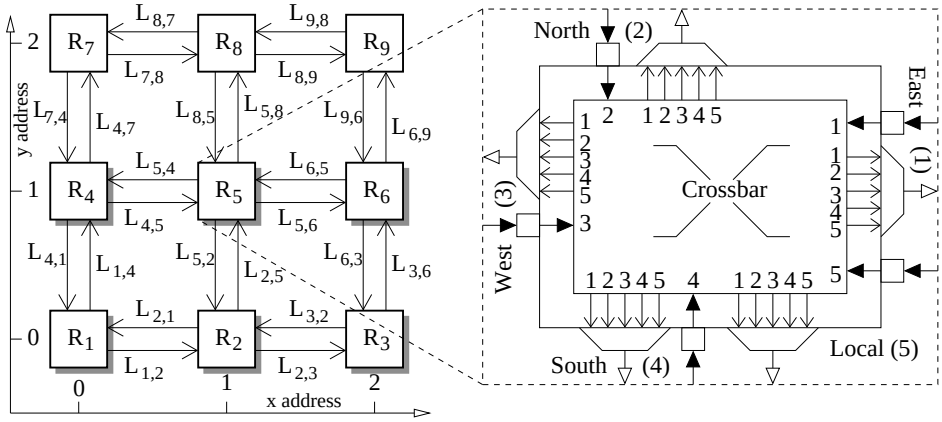


Fig. 3.4: An example of 2D 3×3 mesh network and a typical mesh router.

presented in Fig. 3.3.

3.2 Generic VLSI Architecture

The XHiNoC microarchitecture and components library are packaged in a generic modular design and synthesis flow to develop NoC communication infrastructure for the multiprocessor systems. Before the generic microarchitecture and generic components of the XHiNoC router are introduced, some basic definitions related to on-chip interconnection network and on-chip router are defined in the following.

Definition 3.5 (Network on Chip) *A network on chip (NoC) can be represented as a directed graph $G(\Re, \Lambda)$, where Λ is represented as a set of edges (communication links) and $\Re$ is represented as a set of vertices (router nodes).*

Definition 3.6 (Router) *NoC consisting of N_{node} number of node will have a set of NoC Router $\Re = \{R_1, R_2, \cdots, R_{N_{node}}\}$ or $R_c \in \Re | c = \{1, 2, \cdots, N_{node}\}$.*

Definition 3.7 (Communication Link) *Communication link $L_{i,j} \in \Lambda$ is a communication resource connecting router node R_i and R_j where $R_i, R_j \in \Re$, and $i, j = \{1, 2, \cdots, N_{node}\}$.*

The number of link components in the set $\Re$ depends on network-on-chip topology. We can describe that $\neg\forall i, j$ such that $L_{i,j}$ exists. The left part of the Fig. 3.4 shows a NoC in 3×3 mesh network topology with full duplex links connection. Based on Def. 3.5, it looks that the router set is $\Re = \{R_1, R_2, R_3, R_4, R_5, R_6, R_7, R_8, R_9\}$, while the set of the communication resources is defined as $\Lambda = \{L_{i,j}, i \neq j, \forall i, j : R_i, R_j \; are \; connected\}$. For instance, there are two links connecting R_1 and R_2, i.e. $L_{1,2}$ and $L_{2,1}$. For a 2D mesh with grid size of $N \times M$, then there will be a number of $2N(M-1) + 2M(N-1)$ links in the mesh network.

Definition 3.8 (Router IO Port) *For a number of N_{inp} input ports and a number of N_{outp} output ports of a router $R \in \Re$, the set of input ports as is described as $\rho_I = \{I_1, I_2 \cdots, I_{N_{inp}}\}$ and the set of output ports as $\rho_O = \{O_1, O_2 \cdots, O_{N_{outp}}\}$. Hence, if $\Phi = \{1, 2, \cdots, N_{inp}\}$ and $\varphi = \{1, 2, \cdots, N_{outp}\}$, then an input port of router is defined as $I_n \in \rho_I | n \in \Phi$, and an output port of a router is defined as $O_m \in \rho_O | m \in \varphi$. Furthermore, $\forall n \colon I_n = n$ and $\forall m \colon O_m = m$.*

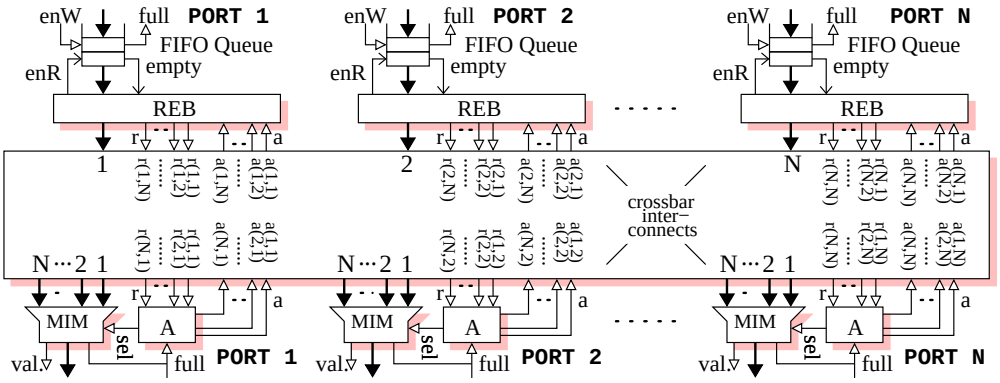
The right part of the Fig. 3.4 shows an example of a NoC router for the 2D mesh network architecture. The router consists of five IO ports, which are labeled with numerical alphabets, i.e. East (1), North (2), West (3), South (4), and Local (5) ports. Based on Def. 3.8, the sets of the router input and output ports are $\Phi = \{1, 2, 3, 4, 5\}$ and $\varphi = \{1, 2, 3, 4, 5\}$. In this case, the number of input and output ports are the same ($N_{inp} = N_{outp}$).

The generic microarchitecture of our NoC router (switch) is presented in Fig. 3.5(a). In general, the router consists of two main component groups, i.e. component groups at input and output ports. At every input port, there are a *First-In First-Out (FIFO) buffer/queue* and a *Routing Engine with Data Buffering* (REB) components. The REB components consists of three modules, i.e. a *Route Buffer*, a *Routing Engine (RE)* and a *Grant Controller (GC)*. The RE module consists of a *Routing State Machine (RSM)* unit and a *Routing Reservation Table (RRT)*. The following items describes briefly the functionality of the input port component group.

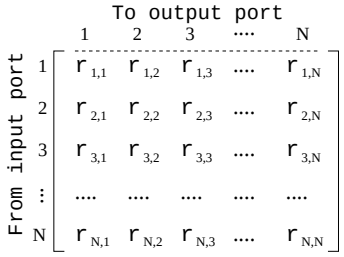
- The *FIFO Bufer* is used to buffer data coming from a neighbor to the input of the NoC router. The depth of the FIFO buffer can be set to only 2 registers in the proposed VLSI microarchitecture. The detail explanation of this component will be explored in Section 3.2.1.
- The *RE* module is used to make a routing decision such that a message can be routed from an input port to an output port. The detail and formal explanation of this component will be explored in Section 3.2.2.
- The *Route Buffer* is used to buffer a message flit soon after the routing decision has been made for the flit. This single buffer module is introduced to insert an additional pipeline stage in the router such that the performance of the router can be improved.
- The *GC* module is a combinatorial logic used to control the data read operation of the FIFO buffer, i.e. to control the data pipeline stage from the FIFO buffer to the register of the Route Buffer.

At each output port, there are two main modules, i.e. an (*Arbiter (A)* unit and a *Crossbar Multiplexor with ID Management (IDM) Unit (MIM)*. In each MIM modules at the output port, there is an *ID Slot Table*. The functionality of the output port component group is described briefly in the following items.

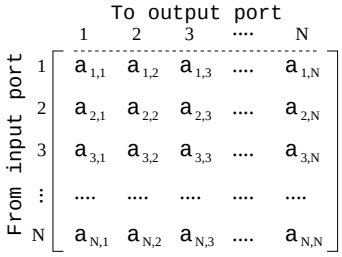
- The *Arbiter* or *Arbitration* unit at an output is used to select a message flit from an input port that will be switched out to the output port. The detail and formal explanation of this component will be explored in Section 3.2.3.



(a) Generic microarchitecture



(b) Routing Path Matrix



(c) Arbitration Path Matrix

Fig. 3.5: Generic microarchitecture of the XHiNoC and the 2D array (matrix) representation of its routing and arbitration control paths.

- The *MIM* module is used to multiplex message flits from input ports and concurrently used to update the ID-tag of each message and manages the *ID Slot Table* in such a away that flits belonging to the same message will have the same ID-tag. Section 3.2.4 will explore in detail the functionality of this component.

In the crossbar wire area as presented in Fig. 3.5(a), there are two signal paths, i.e. data paths and control paths, where each of them can be formally described in a matrix as shown in Fig. 3.5(b) and Fig. 3.5(c), respectively. The data paths are the data wires connecting the data output ports of the REB unit and the data input ports of the MIM modules. The control paths are divided into *routing paths* and *arbitration paths*. Both control paths are the wire sets of the routing request signals and the routing acknowledge or arbitration signals.

Fig. 3.5(b) and Fig. 3.5(c) show the 2D array formation of the routing signals and arbitration signals, respectively. The presentations of the routing signals $r(n,m)$ and arbitration signals $a(n,m)$ as shown in Fig. 3.5(a) are similar to the presentations of the routing signals $r_{n,m}$ shown in Fig. 3.5(b) and the arbitration signals $a_{n,m}$ shown in Fig. 3.5(c) ($r(n,m) \cong r_{n,m}$ and $a(n,m) \cong a_{n,m}$). In Fig. 3.5, it is defined that $N_{inp} = N_{outp} = N$ (See Def. 3.8).

In a regular switch structure with a number of N IO ports, the REB component at input port $n \in \Phi$ sends N-bit routing request signal ($r(n,m)$) to every $m \in \varphi$ arbiter units at each output port. As the responses to the routing request signals, the REB component will

receive N-bit routing acknowledge (arbitration) signal ($a(n,m)$) from the $m \in \varphi$ arbiter units. If a data flit is buffered in the FIFO, it will be then routed and buffered in the REB unit. The routing signal is sent to the arbiter unit at the requested output port. If the REB unit receives a routing acknowledge signal from the arbiter unit, it will be switched out to the outgoing link in the next cycle.

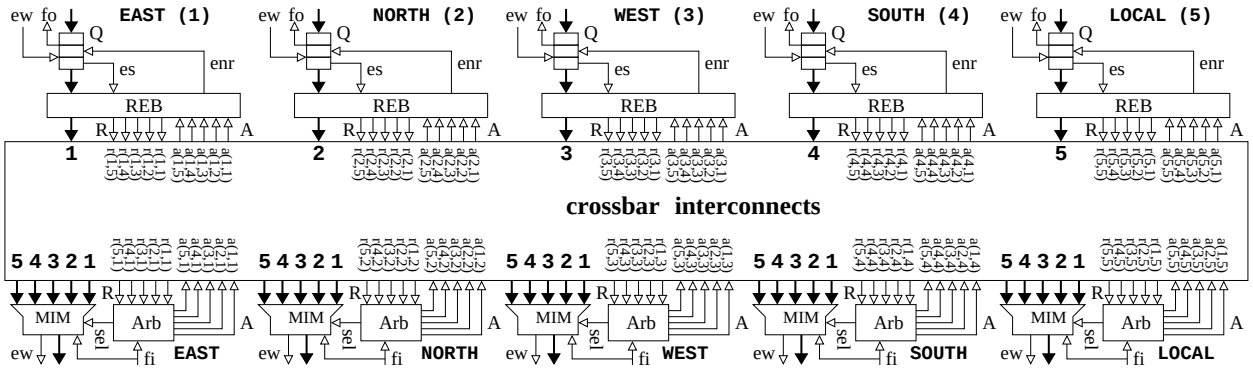
The routing request signals $r_{n,m}$ and the arbitration signals $a_{n,m}$ are implemented in the digital VLSI hardware as binary-encoded signals. By considering the mesh router structure presented in the right part of Fig. 3.4 consisting of 5 output ports, then based on Def. 3.8, $n \in \Phi = \{1,2,3,4,5\}$ and $m \in \varphi = \{1,2,3,4,5\}$. Every RE module at an input port theoretically can make five output routing directions, i.e. output port direction 1 (East), 2 (North), 3 (West), 4 (South) and 5 (Local). Hence, the binary-encoding of the output routing direction will be represented in Equ. 3.3.

$$\begin{array}{lcccccc} To\ port\ 1 & \cong & 1 & 0 & 0 & 0 & 0 \\ To\ port\ 2 & \cong & 0 & 1 & 0 & 0 & 0 \\ To\ port\ 3 & \cong & 0 & 0 & 1 & 0 & 0 \\ To\ port\ 4 & \cong & 0 & 0 & 0 & 1 & 0 \\ To\ port\ 5 & \cong & 0 & 0 & 0 & 0 & 1 \end{array} \tag{3.3}$$

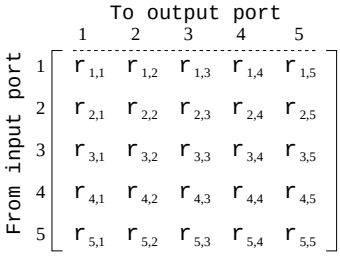
Every single digit of the encoding signal is sent in-order to every single output port. Therefore, in general, the width of the binary encoding signals is set equal to the number of the output port of the router. For instance, when the routing direction from any input port n is to output $port\ 2$, then the encoded-routing signal is $r_{n,1:N} = [0\ 0\ 0\ 0\ 0]$, where $r_{n,1} = [0]$ sent to output $port\ 1$, $r_{n,2} = [1]$ sent to output $port\ 2$, $r_{n,3} = [0]$ sent to output $port\ 3$, $r_{n,4} = [0]$ sent to output $port\ 4$ and $r_{n,5} = [0]$ sent to output $port\ 5$. By using such encoding signals, it is easy to implement a multicast routing. For example if a flit from input port n will be routed to more than one output port, e.g. to $port\ 1$, $port\ 4$ and to $port\ 5$, then the encoded-binary signals generated from the routing engine at input port n is $r_{n,1:N} = [1\ 0\ 0\ 1\ 1]$. Fig. 3.6(a) presents a typical microarchitecture with the data and control path structure of the XHiNoC mesh router. Fig. 3.6(d) shows the detail IO components of a NoC mesh router. For the sake of simplicity, only router components in the West Input port and in the East Output port are exhibited in the figure.

3.2.1 First-In First-Out Buffers

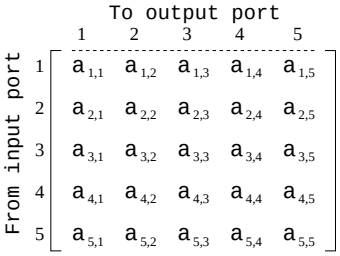
This section will give a brief description about the behavioral model of a First-In First-Out (FIFO) buffer and operations applied in the component. The block diagram of the Register Transfer Level (RTL) model of the FIFO buffer is exhibited in Fig. 3.7. In general, FIFO buffer consists of FIFO registers and a FIFO controller. The FIFO controller controls modes of FIFO operation by reading control-path input signals and providing control-path output signals and read-write operation applied to the FIFO registers. The read-



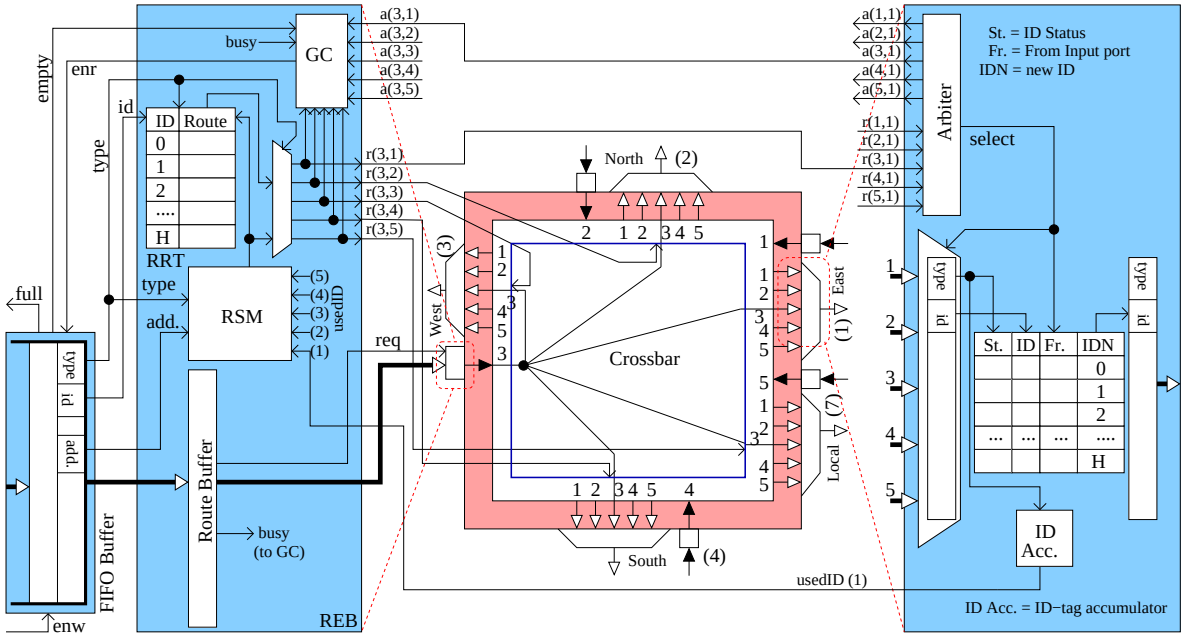
(a) Mesh crossbar switch



(b) Routing request matrix



(c) Arbitration matrix



(d) Detailed components and architecture

Fig. 3.6: Typical microarchitecture, routing request matrix, arbitration matrix and detail IO components of XHiNoC mesh router (5 IO ports).

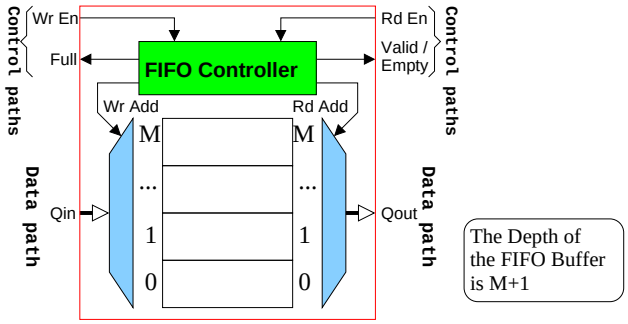


Fig. 3.7: The typical structure of the FIFO buffer.

address ($RdAdd$) and write-address ($WrAdd$) are the internal signals of the FIFO buffer that determine the addresses where the incoming data will be stored in the FIFO registers and from which register the data will be read out from the FIFO buffer, respectively.

The FIFO buffer comprises *data paths* and *control paths*. The data paths are the data input (Qin) and the data output ($Qout$). The control paths as the input control signals are write-enable ($WrEn$) and read-enable ($RdEn$) signals. While the output control signals are $Full$ and $Empty/Valid$ signals. If there is no more free space in the FIFO register, then the $Full$ signal will be set to $'1'$. While, if there is no data stored in the FIFO register, then the $Empty/Valid$ signal is reset back to $'0'$.

The first-in first-out operation in the FIFO buffer is presented in Alg. 6. The function *incr()* is a circulating incremental function. When the address pointers ($WrAdd$ and $RdAdd$) have pointed to the largest register address, then the address pointer will point back to the lowest register address of the buffer after undertaking a considered FIFO operation. When the data queues of the FIFO buffer are full, then the $full$ flag signal is set, while in the absence of data in the FIFO buffer, then the $empty$ signal is set.

There are three modes of operation applicable to the FIFO buffer, i.e. read operation, write operation and simultaneous read-write operation. Fig. 3.8 presents the three modes of operation. When the FIFO queue is in an empty state, then only write operation is applicable, and when the FIFO is in full state, then only read operation is applicable to the FIFO buffer. The simultaneous read-write operation could be applied to the FIFO buffer, when the FIFO is in full state. However, it is difficult to control and avoid drops of data, if the read-write operation is enabled in the full state.

Fig. 3.8 present an example of successive write, read-write and read operations in the FIFO buffer. In Fig. 3.8(a), flit A appears in the input port of the queue and set the $WrEn$ signal to $'1'$. Initially, the FIFO is empty and the $WrAdd$ as well as the $RdAdd$ are set to register number "0". Thus in the next cycle, the data flit A is stored in the Register "0" (write operation), and the $WrAdd$ is incremented to register number '"1" and the $Valid/Empty$ signal is set to $'1'$ as presented in Fig. 3.8(b). The flit A appears now at the output port of the FIFO buffer. In the same period, a new data flit B appears in the input port of the queue, and the $RdEn$ signal is set to $'1'$. Thus in the cycle, a simultaneous

read-write operation occurs as presented in Fig. 3.8(c). Both the $WrAdd$ and the $RdAdd$ are incremented to "2" and "1", respectively. The flit B (in register number "1") appears at the output port because the $RdAdd$ signal is now set to "1". The $RdEn$ signal is still set to '1', and the $WrEn$ signal is now set to '0'. Thus in the next cycle, the read operation is applied as presented in Fig. 3.8(d). The content of the register number "1" is removed, and the $Valid/Empty$ signal is then reset to '0'.

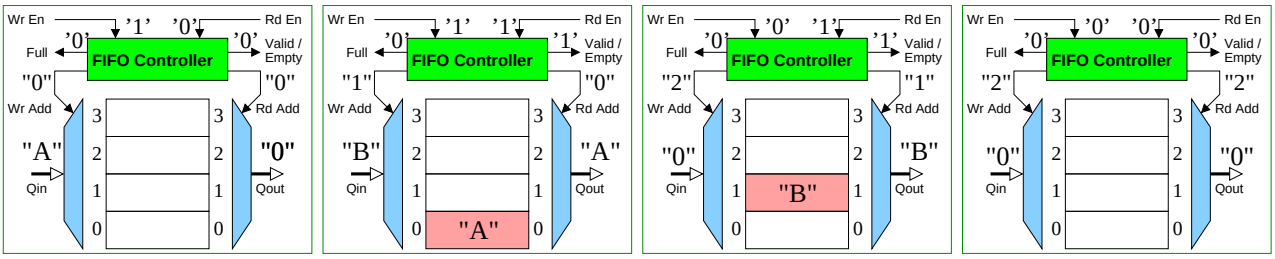
Alg. 6 First-In First-Out Queue

$WrEn$, $RdEn$: Write and Read enable
$WrAdd$, $RdAdd$: Write and Read address pointer
$Reg(k)$: Queue register
Qin, $Qout$: Input and output data queue
$Full$, $Empty$: Buffer full and empty signals
$MaxBuff$: Maximum number of buffers

1: BEGIN $FIFO\ queue$
2: **if** $WrEn$ is $True$ and $RdEn$ is $True$ **then**
3: $Reg(WrAdd) \Leftarrow Qin$; incr($WrAdd$)
4: $Reg(RdAdd) \Leftarrow \emptyset$; incr($RdAdd$)
5: **else if** $WrEn$ is True and $RdEn$ is False **then**
6: $Reg(WrAdd) \Leftarrow Qin$; incr($WrAdd$)
7: $Empty$ is False
8: **if** incr($WrAdd$)=$RdAdd$ **then**
9: $Full$ is True
10: **end if**
11: **else if** $WrEn$ is False and $RdEn$ is True **then**
12: $Reg(RdAdd) \Leftarrow \emptyset$; incr($RdAdd$)
13: $Full$ is False
14: **if** incr($RdAdd$)=$WrAdd$ **then**
15: $Empty$ is True
16: **end if**
17: **end if**
18: $Qout \Leftarrow Reg(RdAdd)$
19: END $FIFO\ queue$

Some works regarding the impact of the FIFO buffer implementation on several aspects have been shown in many literatures. The work in [219] has explored router area implication based on the buffer allocation in the router implemented on an FPGA device. The experiment has presented that, when the overall amount of buffers in the output and in the middle-buffer architecture is constant or equal, then the middle-buffer architecture provide a slightly smaller logic block area. However, if the sizes of the FIFO buffer are set to the minimum size (e.g. 2 register), then the input or the output-buffer architecture will have much smaller logic gate area consumption compared with the middle-buffer architecture.

The depth of a buffer or the number of register space in a buffer of the router can have a significant impact on the gate area of the router. The work in [59] uses conventional virtual channel flow control with a large amount of of buffer size, i.e. about 10K bits in each



(a) Snapshot 1 (b) Snapshot 2 (c) Snapshot 3 (d) Snapshot 4

Fig. 3.8: Examples of successive mode of operations in the FIFO buffer.

input controller. An alternative flow control method can substantially reduce the buffer storage requirements at the expenses of reduced performance. Some methodologies can be used to reduce the buffer size. For instance, by using packet dropping or misrouting when the packets encounter contention [59].

The misrouting approach used to limit the size of the FIFO buffer and to reduce packet dropping possibility has been introduce in [85]. The work proposes some technique to reduce packet dropping and the resulting consequence, i.e. misrouting, external misrouting and extra loopback channels. However, the proposed misrouting and external misrouting methodologies still allow packet to drop in any circumstance, e.g. the number of misrouting has reached its maximum value, since the number of misrouting must be limited to avoid livelocks. The proposed extra loopback channels method allows additional buffering resources at each switch to prevent packet drops. However, this extra buffer can increase logic area of the router and still enable packet dropping (the methodology can only reduce the packet dropping possibility). Packet dropping protocol reduces the NoC performance, while misrouting protocol increases wire loading and hence power dissipation [59].

The work in [102] proposes an efficient algorithm that optimizes the allocation of buffering resources across different router channel while matching the communication characteristics of the target application. The work in [38] also suggests a non-uniform buffer allocation. A buffer-sizing algorithm for NoCs using TDMA and credit-based end-to-end flow control is also proposed in [52], in which, due to the usage of credit-based end-to-end flow control that places additional requirements on the buffer sizes, the flow control delays need to be taken into account. The aforementioned algorithms are designed to find the minimal decoupling buffer sizes for a NoC, subject to the performance constraints of the applications running on the SoC. However, we are sure that even if the proposed algorithms mentioned above can consider multiple-use cases, where the SoC can be dedicated to run some applications, the number of implementable applications is limited. It is certainly clear that the proposed methods are only suitable for pre-fabricated MPSoC systems and cannot be used in post-fabricated MPSoC and general-purpose chip-level multiprocessor (CMP) systems. This is because after the post-manufacturing step, there is no more chance to add or reallocate buffers in the NoCs.

A variation-aware low-power buffer design is proposed in [167]. The work in [117] proposes adaptive channel buffers as storage elements in addition to existing router buffers when NoC are working at high load. However, both methodologies take into account the usage of virtual channels that can lead to very large area overhead. The work in [152] has a decoupled control and datapath to design a NoC router. The approach is also used in our NoC router architecture, where the control and data paths are design separately. The XHiNoC concept is intended to implement FIFO buffers, which can give similar performance even when the depth of FIFO is increased by applying the link-level flit flow control and flit-level data multiplexing based on the local ID management concept.

3.2.2 Routing Engines

The Routing Engines used in the NoC router architecture are a combination of a routing state machine (see Def. 3.10) and a routing reservation table (see Def. 3.11). The combination is aimed at supporting a runtime routing organization during application execution time, in which the routing decisions are made locally in each NoC router that is distributed over the NoC (local/distributed routing).

Definition 3.9 (Routing Direction) *A routing direction r_{dir} is a signal made by a routing engine. The routing direction values are the set member of the output port numbers, i.e. $r_{dir} \in D, D = \{1, 2, \cdots, N_{outp}\}$.*

Definition 3.10 (Routing State Machine) *The routing state machine (RSM) provides a routing function f_{RSM} where the output of the function depends on the destination address appearing on the target address field A_{dest} of a header or a response flit. If a header or a response flit is detected by the routing state machine, then the routing direction of the flit is computed with routing function f_{RSM}: $f_{RSM}(A_{dest}) \Rightarrow r_{dir}$, where A_{dest} is the destination address present on the header or the response flit, and $r_{dir} \in D$ according to Def. 3.9.*

Definition 3.11 (Routing Reservation Table) *A Routing Reservation Table (RRT) of the RE unit at an input port is defined as*

$$T(k|k \in \Omega) = r_{dir} \in D = \{1, 2, 3, \cdots, N_{outp}\} \tag{3.4}$$

or $T(k) \in D|k \in \Omega$. Definition of D can be found in Def. 3.9. So, one can define that $\forall k \in \Omega$, the value of the Routing Reservation Table $T(k)$, is a routing direction $r_{dir} \in D$. The routing direction is indexed in the Routing Table based on the ID-tag.

The array structure of the Routing Reservation Table T can be seen in Fig. 3.2(b). The number reservable routing slots is N_{slot}, which is equal to the number of local ID slots in the link $L_{i,j}$.

Definition 3.12 (Routing Engine) *Routing Engine (RE) is a component to make a routing direction $r_{dir} \in D$. For N_{inp} number of router input ports, then the number of RE per router is $N_{RE} = N_{inp}$. The RE at input port $I_n \in \rho_I$ is $E_n, n \in \rho_I$. The E_n provides a routing function f_{RE} giving routing direction D, or the function is defined as f_{RE}: $r_{dir} = f_{RE}\,(type, ID, A_{dest})$, where A_{dest} is the destination address field present in the header flit.*

Alg. 7 Runtime ID-based Routing Mechanism

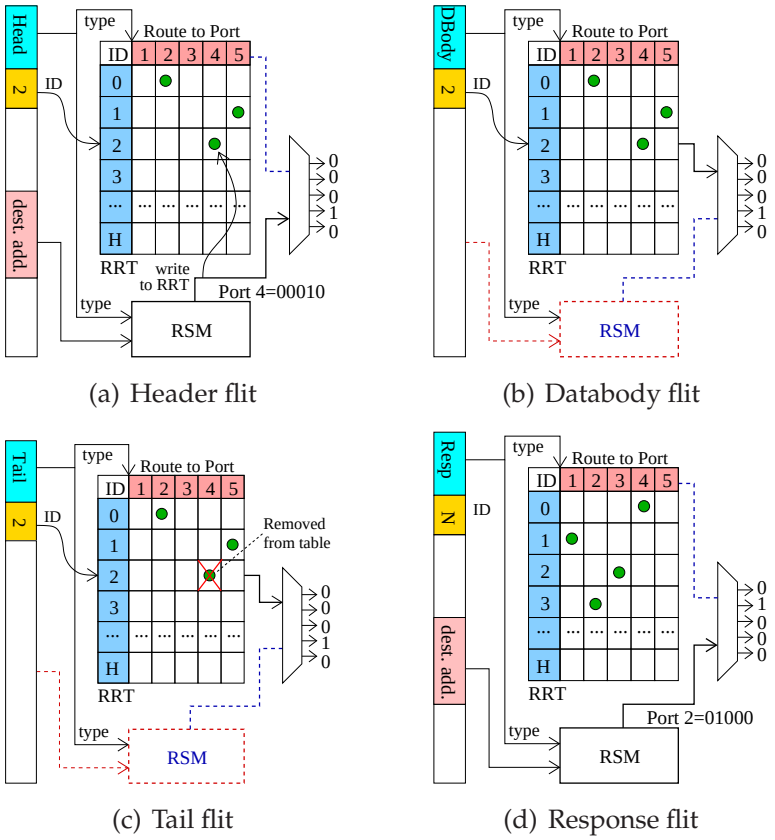
Read Data Flit from Queue : $F_n\,(type, ID)$

1: $F_{type} \Leftarrow type$
2: A_{dest} is obtained from Header flits
3: BEGIN $Routing$
4: **if** F_{type} is $Header$ **then**
5: $\quad r_{dir} \Leftarrow f_{RSM}\,(A_{dest})$
6: $\quad T(ID) \Leftarrow r_{dir}$
7: **else if** F_{type} is $Databody$ **then**
8: $\quad r_{dir} \Leftarrow T(ID)$
9: **else if** F_{type} is $Tail$ **then**
10: $\quad r_{dir} \Leftarrow T(ID)$
11: $\quad T(ID) \Leftarrow \emptyset$
12: **else if** F_{type} is $Response$ **then**
13: $\quad r_{dir} \Leftarrow f_{RSM}\,(A_{dest})$
14: **end if**
15: END $Routing$

The RE (E_n) at input port $n \in \Phi$ consists of a combination of *routing state machine* M and *routing reservation table* T. Accordingly, there will be pairs of $(M_n, T_n)\,|\forall n \in \Phi$. Both components are allocated at each input port. If a data flit $F_{n,m}\,(type, ID)$ is coming from an input port n to an output port m of the router R_i, then the Routing Engine will compute and organize routing directions of the flit by using a routing organization algorithm as shown in Alg. 7.

As shown in Alg. 7, the routing operation in the Routing Engine depends on the type of the data flit $F_n(type, ID)$. The type of flit will determine which component (M or T) that will give a routing direction. When a header flit $F_{n,m}\,(header, ID)$ is coming from an input port n, a routing direction r_{dir} is computed by the RSM, and it is concurrently written (copied) in the slot number k in the RRT, where $k = ID$ (equal to the ID-tag of a header flit) such that $T(ID) = r_{dir}$. When a databody $F_{n,m}\,(header, ID)$ or tail flit $F_{n,m}\,(header, ID)$ is coming to the input port n, then the r_{dir} is fetched directly from the slot number $k = ID$ in the RRT. The operation $r_{dir} \cup T(ID)$ in the algorithm is the union operation between the current content of the $T(ID)$ in the slot number ID and the current value of the routing direction r_{dir}. When the flit type is a $response$ flit, then the routing direction is made by the RSM without making further access to the RRT.

Fig. 3.9 presents the architectural concept view of the routing reservation and organization based on local ID-tag for 4 different types of flits. Fig. 3.9(a) shows the routing operation when a header flit is read from the data buffer of the REB component. In this



(a) Header flit (b) Databody flit

(c) Tail flit (d) Response flit

Fig. 3.9: Local ID-based routing reservation and organization.

case, the RSM unit will compute the routing direction r_{dir}. The RE will write or assign the encoded r_{dir} ($r = [0\ 0\ 0\ 1\ 0]$) into the programmable register/slot of the RRT (i.e. slot number 2) according to the ID-tag of the header. Fig. 3.9(b) shows the routing operation when a databody flit is routed to the output port 4. The RE will read the ID-tag of the flit and find the routing direction from the slot number 2 in the RRT unit according to the ID-tag of the databody flit. The same situation is presented in Fig. 3.9(c) when a tail flit is routed to the output port. But, at the end of the switching phase, i.e when the tail flit will be switched out to the output port 4, the content of the slot number 2 is removed. The previous flits mentioned above (i.e. the header, databody and tail flit) have the same ID-tag. It means that they belong to the same message/data stream, and is routed to the same output port.

Fig. 3.9(d) exhibits the routing operation made by the RE components when a response flit is routed to an outgoing port. The RSM will compute the routing direction for the response flit, but the routing direction will not be assigned in the routing reservation table (RRT). The response flit is a single flit sent by a node another node to inform the connection status made by a guaranteed-service header. Hence, the response flit is used only when the XHiNoC router implements the connection-oriented guaranteed-service.

3.2.3 Arbitration Unit

Every communication link is fairly shared by contenting packets. This is allowed by applying a rotating (circulating) flit-by-flit arbitration as defined in Def. 3.16.

Definition 3.13 (Time-Varying Binary Request) *A time-varying input-output request from an input port $n \in \Phi$ to an output port $m \in \varphi$ is defined as $r_{n,m}(t)\colon \forall n, m | r_{n,m}(t) \in \{0,1\}$. Hence, the* time-varying input n binary request *can be defined as array of binary or $r_{n,m=1\ to\ N_{outp}}(t)$ or $r_{n,m=1:N_{outp}}(t)$ or $r_{n,1:N_{outp}}(t)$, and the* time-varying output m binary request *is defined also as $r_{n=1\ to\ N_{inp},m}(t)$ or $r_{n=1:N_{inp},m}(t)$ or $r_{1:N_{inp},m}(t)$.*

Definition 3.14 (Set of Requests from an Input Port) *A set of requests from an input port n to output ports is defined as φ_n^{req}, thus the number of set members $N_{s,n}^{req}$ is defined as the number of requests from an input port n to output ports at time stage t_s. Further, we can define the definitions in the following equation.*

$$\begin{aligned}
\varphi_n^{req} \subset \varphi &\Leftrightarrow N_{s,n}^{req} = \textstyle\sum_{m=1}^{N_{outp}} r_{n,m}(t_s) < N_{outp} \\
\varphi_n^{req} \subseteq \varphi &\Leftrightarrow N_{s,n}^{req} = \textstyle\sum_{m=1}^{N_{outp}} r_{n,m}(t_s) = N_{outp} \\
\varphi_n^{req} = \emptyset &\Leftrightarrow N_{s,n}^{req} = \textstyle\sum_{m=1}^{N_{outp}} r_{n,m}(t_s) = 0 \\
h \in \varphi_n^{req} &\Leftrightarrow r_{n,h}(t) = 1
\end{aligned} \tag{3.5}$$

According to Def. 3.14 for example, if $r_{n,1:5}(t_s) = [1\ 0\ 1\ 1\ 0]$ or $r_{n,1}(t_s) = [1]$, $r_{n,2}(t_s) = [0]$, $r_{n,3}(t_s) = [1]$, $r_{n,4}(t_s) = [1]$ and $r_{n,5}(t_s) = [0]$, then $N_{s,n}^{req} = 3$ and $\varphi_n^{req} = \{1, 3, 4\}$.

Definition 3.15 (Set of Requests to an Output Port) *A set of requests to an output port m from input ports is defined as Φ_n^{req}, thus the number of set members $N_{s,m}^{req}$ is defined as the number of requests to an output port m from input ports at time stage t_s. Furthermore, we can define the definitions in the following equation.*

$$\begin{aligned}
\Phi_m^{req} \subset \Phi &\Leftrightarrow N_{s,m}^{req} = \textstyle\sum_{n=1}^{N_{inp}} r_{n,m}(t_s) < N_{inp} \\
\Phi_m^{req} \subseteq \Phi &\Leftrightarrow N_{s,m}^{req} = \textstyle\sum_{n=1}^{N_{inp}} r_{n,m}(t_s) = N_{inp} \\
\Phi_m^{req} = \emptyset &\Leftrightarrow N_{s,m}^{req} = \textstyle\sum_{n=1}^{N_{inp}} r_{n,m}(t_s) = 0 \\
l \in \Phi_m^{req} &\Leftrightarrow r_{l,m}(t) = 1
\end{aligned} \tag{3.6}$$

According to Def. 3.15 for example, if $r_{1:5,m}(t_s) = [0\ 1\ 1\ 0\ 1]$ or $r_{1,m}(t_s) = [0]$, $r_{2,m}(t_s) = [1]$, $r_{3,m}(t_s) = [1]$, $r_{4,m}(t_s) = [0]$ and $r_{5,m}(t_s) = [1]$, then $N_{s,n}^{req} = 3$ and $\Phi_m^{req} = \{2, 3, 5\}$.

Definition 3.16 (Rotating Flit-by-Flit Arbitration) *A* rotating flit-by-flit arbitration *is an arbitration strategy at every output port m that circulates its selection among input ports $l \in \Phi_m^{req}$ according to Def. 3.15 and Equ. 3.6 in flit-by-flit manner.*

Alg. 8 shows the rotating/circulating flit-by-flit arbitration at every output port m. At initial time (no request at all), the arbiter will firstly serve, the first incoming flit. When many flits need arbitration service at the same cycle, the arbiter will rotate its selection like a round-robin arbiter, but the arbitration is done in a flit-by-flit and active-port-by-active-port manner. Active-port refers to an input port having request for routing. It means that the arbiter circulates a selection but will not select input ports having no data flit.

Definition 3.17 (Rotating Arbitration Time) *Input port selection in every output port m will be recirculated in every $T_{s,m}+1$, where $T_{s,m}=N_{s,m}^{req}$ is a* Rotating Arbitration Time.

Definition 3.18 (Binary Output Acknowledgement) *We have defined the time-varying output m binary routing request in Def. 3.13. Thus, we can now define the* Request-Dependent Binary Output Acknowledgment *as $a_{1:N_{inp},m}(t)$. For $\forall n|a_{n,m}(t)\in\{0,1\}$ where only one element of $a_{1:N_{inp},m}(t)$ can be set to '1' because of natural hardware constraint of an arbiter unit. When $N_{s,m}^{req}>1$, then physically it mean that there is contention between $N_{s,m}^{req}$ number of flits between input ports in the set Φ_m^{req} to access the same output port m.*

For example, if the binary request $r_{1:5,m}(t_s)=[0\ 1\ 0\ 1\ 1]^T$, then according to Def. 3.17 and Equ. 3.6, $T_{s,m}=3$. The set of possible order of the rotating arbitration at output port m is $a_{1:5,m}(1)=[0\ 0\ 0\ 0\ 1]^T$, $a_{1:5,m}(2)=[0\ 0\ 0\ 1\ 0]^T$ and $a_{1:5,m}(3)=[0\ 1\ 0\ 0\ 0]^T$. According to Def. 3.15, $\Phi_m^{req}=\{2,4,5\}$. We can see that $r_{1:5,m}(t_s)=\bigcup_{t=1}^{t=3}a_{1:5,m}(t)$.

Alg. 8 Rotating Flit-by-Flit Arbitration

```
Binary Request  :  r_n ∈ {0,1} : n = 1,2,··· ,N_inp
Initial Value   :  Rot_Start = N_inp, Rot_Stop = 1
BEGIN Arbitration
while ∃n: r_n ≠ 0 & the next queue is ¬Full do
    for n = Rot_Start downto n = Rot_Stop do
        if r_n = 1 then
            select ⇐ n
            Rot_Start = n − 1
            Rot_Stop = n
        end if
    end for
end while
Result: select ⇒ I_n
END Arbitration
```

3.2.4 Crossbar Multiplexor with ID Management Unit

For each communication resource (link/channel) $L_{i,j}$ connecting an outport port ($O_m\in\rho_O$) of an on-chip router R_i with an input port ($I_n\in\rho_I$) of an adjacent (neighbor) router

R_j, then an amount of local identity (ID) slots $N_{slot}^{i,j}$ is implemented on the communication link $L_{i,j} \in \Lambda$.

Definition 3.19 (ID Slot Table) *ID Slot Table State is defined as a set of slot state S, where $S = \{S^0, S^1, S^2, \cdots, S^{N_{slot}-1}\}$ and $\forall k \in \Omega\colon S^k \in \{true, false\}$ (See Def. 3.4). If $S^k = true$, it means that the ID Slot state is "free", else if $S^k = false$, then the ID Slot is being "used" by any message. We can define the ID Slot Table as*

$$\forall k \in \Omega\colon S(k) = (ID_{old}, F_{from}) \in (\Omega, \Phi) \tag{3.7}$$

The set $\Phi = \{1, 2, 3, \cdots, N_{inp}\}$ according to Def. 3.8. We define F_{from} as the selected flit from any input port as the arbitration result.

Fig. 3.2(b) can help to comprehend Def. 3.19 and describes the structure of the ID Slot Table S. The figure presents the ID Slot Table in output port of Router R_i and the Routing Table in the input port of Router R_j. A communication link $L_{i,j}$ connecting the output and input port of the R_i and R_j. For the sake of simplicity and for maintaining a design regularity, it is assumed $N_{slot}^{ij} = N_{slot}$. Therefore, the number of available ID tags on each communication link is uniform.

As shown in Alg. 9, when a header flit $F_n(header, ID)$ with ID-tag ID coming from input port n is switched to an output port m then an ID-tag update function $f_{IDM}\colon (ID_{old}) \mapsto ID_{new}$ is made, and a free ID slot is searched for the header. When a free ID slot k is found, the input port n and the ID-tag ID are written in the slot number k in the S, and the header uses this ID slot k as its new ID-tag ($ID_{new} = k$). When the header fails to find a free ID slot, it will be assigned to ID slot $N_{slot} - 1$. When a databody $F_n(databody, ID)$ or tail $F_n(tail, ID)$ flit having the same ID-tag with the previously routed header $F_n(header, ID)$ flit is switched to the output port m, then they will be assigned with the same new ID-tag k. The databody and tail flits will be dropped from the network if its header having the same ID-tag fails to reserve an ID Slot $k \in \Omega \cap k \neq N_{slot} - 1$ on a certain link. Like response flits, header flits having ID tag $N_{slot} - 1$ will always be routed in the NoC with the ID tag $N_{slot} - 1$, in which their paths can be guaranteed correct even when many $F_n(header, N_{slot} - 1)$ and $F_n(response, N_{slot} - 1)$ flits flow in the NoC. Since these flits are only single-flit, destination address that is attached in the their address-field are independent from the routing table contents accordingly.

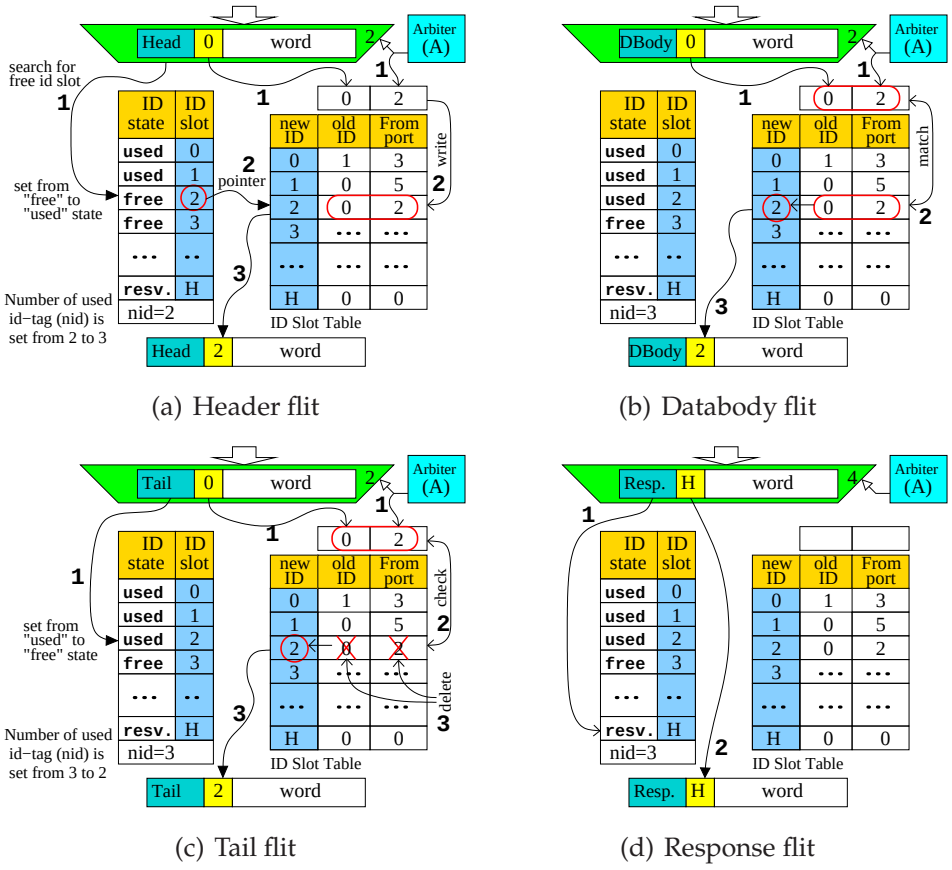
The conceptional view of the local ID-tag update and management mechanism is presented in Fig. 3.10. Fig. 3.10(a) presents a mechanism to update and to organize the ID slot table in the MIM modules when a header flit is switched out to an output port. The figure shows a packet header coming from $port\ 2$ port with ID-tag 1. The header flit is just switched from crossbar switch its ID-tag is updated to reserve an new ID-tag in the slot table of the MIM module. The ID update process can be described into 4 steps.

When the IDM detects a new incoming packet header, then the IDM will look for a free ID slot by checking the ID-state table. According to the figure the ID slot 0 and the ID

Alg. 9 Runtime Local ID-tag Update

Outgoing Data Flit : $F_n\,(type, ID)$
Input Arbitration : $n = \{1, 2, \cdots, N_{inp}\}$
N_{usedID} : number of used/reserved ID slots
$N_{slot} - 1$: Slot reserved for control purpose ($N_{slot} - 1 = H$)

1: $F_{type} \Leftarrow type$; $ID_{old} \Leftarrow ID$; $F_{from} \Leftarrow n$
2: BEGIN $ID\ Update$
3: **if** F_{type} is $Header$ **then**
4: **if** $ID_{old} = N_{slot} - 1$ **then**
5: $ID_{new} \Leftarrow N_{slot} - 1$
6: **else if** $ID_{old} \neq N_{slot} - 1$ **then**
7: **for** $k = 0$ to $k = N_{slot} - 2$ **do**
8: **if** $\exists k$: S^k is true **then**
9: $S(k) \Leftarrow (ID_{old}, F_{from})$
10: $S^k \Leftarrow false$ /* the ID Slot is used now */
11: $ID_{new} \Leftarrow k$; $N_{usedID} \Leftarrow N_{usedID} + 1$
12: **else**
13: $ID_{new} \Leftarrow N_{slot} - 1$
14: **end if**
15: **end for**
16: **end if**
17: **else if** F_{type} is $Databody$ **then**
18: **for** $k = 0$ to $k = N_{slot} - 1$ **do**
19: **if** $\exists k$: $k \neq N_{slot} - 1$: $S(k) = (ID_{old}, F_{from})$ **then**
20: $ID_{new} \Leftarrow k$
21: **else if** $\nexists k$: $k \neq N_{slot} - 1$: $S(k) = (ID_{old}, F_{from})$ **then**
22: $ID_{new} \Leftarrow \emptyset$
23: The $Databody$ flit is dropped
24: **end if**
25: **end for**
26: **else if** F_{type} is $Tail$ **then**
27: $N_{usedID} \Leftarrow N_{usedID} - 1$
28: **for** $k = 0$ to $k = N_{slot} - 1$ **do**
29: **if** $\exists k$: $k \neq N_{slot} - 1$: $S(k) = (ID_{old}, F_{from})$ **then**
30: $ID_{new} \Leftarrow k$; $S(k) \Leftarrow (\emptyset, \emptyset)$
31: $S^k \Leftarrow true$ /* the ID Slot is now free */
32: **else if** $\nexists k$: $k \neq N_{slot} - 1$: $S(k) = (ID_{old}, F_{from})$ **then**
33: $ID_{new} \Leftarrow \emptyset$;
34: The $Tail$ flit is dropped
35: **end if**
36: **end for**
37: **else if** F_{type} is $Response$ **then**
38: $ID_{new} \Leftarrow N_{slot} - 1$
39: **end if**
40: $ID_{new} \Rightarrow ID$
41: END $ID\ Update$



(a) Header flit (b) Databody flit

(c) Tail flit (d) Response flit

Fig. 3.10: Local ID-tag update and mapping management.

slot 1 have been used by messages coming from *port* 3 and *port* 5, respectively. The ID-tag 2 is now detected as a free ID slot, and then this ID is assigned as the new ID-tag for the new packet header. Afterwards, the ID-slot 2 is indexed based on two informations i.e., the old (previous) local ID-tag 0 and data "2" (*port* 2) from which port the header flit comes. In the same cycle, ID-tag 2 state is set from "free" to "used" state, and the number of used ID (NID) is incremented from 2 to 3. When all ID slots have been used, then "empty free ID flag" is set.

In the next time period, when payload (databody) flits come from *port* 2 with ID-tag 0, then they will also be assigned with the new ID-tag 2. Fig. 3.10(b) shows how this ID-tag is assigned automatically by the ID management (IDM) unit. The IDM unit just reads the current ID-tag of the databody flit and from which port it comes as an input references. Afterwards, it searches for the suitable new ID-tag by comparing the contents of the programmable registers of the ID Slot Table with both input references. When the tail flit (the end of databody) of the message with ID-tag 0 from *port* 2 is passing through the outgoing port, then the ID-tag 2 state is set from "used" to "free", and the NID is decremented from 3 to 2. The tail flit is also assigned with the local ID-tag 0 like databody flits. But, at the end of the cycle, the values in slot number 2 of the ID Slot Table are removed from the programmable registers as presented in Fig. 3.10(c).

When a response flit having local ID-tag $H = N_{slot} - 1$ is switched out from an output port as presented in Fig. 3.10(d), then this flit will also assigned with its previous ID-tag. The local ID slot H is reserved for control purpose. The response flit is used to inform the status of connection establishment when the XHiNoC is implemented for connection-oriented guaranteed-service. The response flit will always have ID-tag $H = N_{slot} - 1$ over all communication links in the XHiNoC. When header flits fail to establish connection, i.e. they fail to reserve e.g. a local ID slot on a certain outgoing link, then they will be allocated also in the local ID slot $H = N_{slot} - 1$. Like response flits, header flits having ID tag $H = N_{slot} - 1$ will be always routed in the NoC with the ID tag $H = N_{slot} - 1$, in which their paths can be guaranteed correct even when many $F_n(header, N_{slot} - 1)$ and $F_n(response, N_{slot} - 1)$ flits flow in the NoC, since these flits are only single-flit, in which destination address is attached in the their address-field. Afterwards, the header flits will be always assigned to the local ID slot H until they reach their destination nodes.

3.3 Characteristics and Features

This section will mentioned some characteristics of the XHiNoC as well as the interesting features of the XHiNoC that makes it a unique NoC router compared with other existing NoC routers. The specific features are the consequences of the implementation of the ID-based multiple access technique.

3.3.1 Pipeline Architecture

Compared with Intel Teraflops NoC [98], which has six-stage pipeline, i.e. *Buffer Write*, *Buffer Read*, *Route Compute*, *Port/Lane Arbitration*, *Switch Traversal* and *Link Traversal*, the XHiNoC uses also the same technique with reduced pipeline stages. The XHiNoC uses only 4-stage pipeline, i.e. *Buffer Write*, *Buffer Read+Route Compute*, *Port Arbitration* and *Switch/Link Traversal*. The Buffer Read and Route Compute pipeline line stages are combined to reduce the cycle delay from the input port to the output port of the router, while Intel Teraflops cannot because it implements a double-pumped (dual lane) crossbar switch. When the REB unit reads a flit from the FIFO queue, then the REB will compute the routing direction and store the flit at its buffer in the same step. We combine also the Switch and Link Traversal stages because, once the flit is switched out, then it can be written in the FIFO queue in the next router. Certainly, both pipeline stage reductions must be controlled carefully to avoid flit drops and unnecessary flit replications.

Fig. 3.11 shows the timing diagram of the inter-switch data flow. Flits of message A are switched from the West input link to the East output link of a router. A data flit is transmitted on the input link in every two-cycle. Then every flit is stored in the queue and switched to the outgoing link through two phases, i.e routing request-phase and routing-acknowledge phase. Phase 2 and phase 3 shown in Fig. 3.11 presents the request

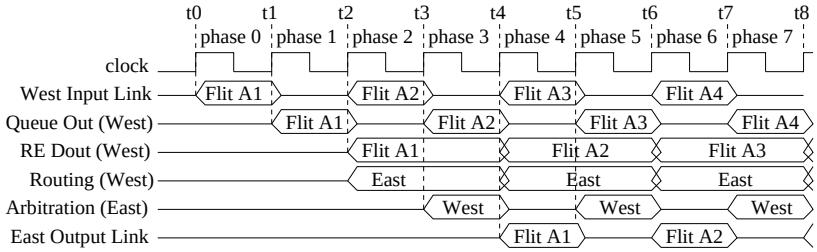


Fig. 3.11: Timing diagram (without contention) of the data switching and control paths.

and the acknowledge (grant) phases for the $Flit\ A1$. While phase 4 and phase 5 presents the request and the grant phase for the $Flit\ A2$.

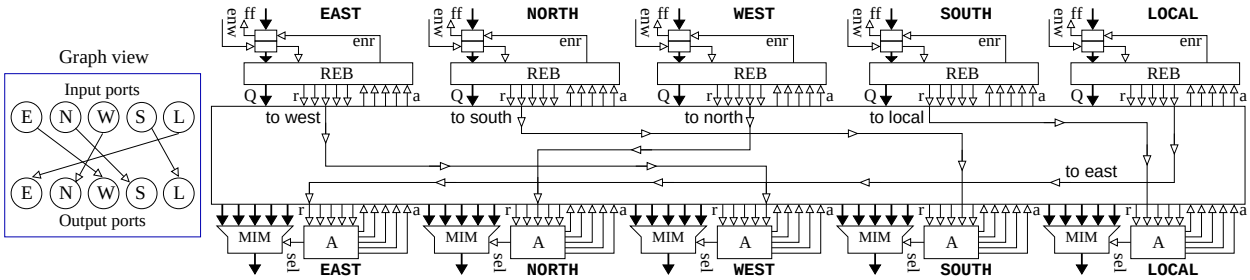
The signal flow in data path of each flit is made pipeline synchronously, and the signal flow in control path is made in two cycle phases. Although the flit flow in the router is delayed for two cycles, the flow rates of the flits in the input link will be as same as the flow rates in output link as long as the flits is transmitted on link with 0.5 flit per cycle or slower and there is no blocking situation in the next routers. Based on the timing diagram shown in Fig. 3.11, the *Buffer Write*, *Buffer Read+Route Compute*, *Port Arbitration* and *Switch/Link Traversal* pipeline stages are represented in the figure in the cycle phase 1, phase 2, phase 3 and phase 4, respectively.

3.3.2 Simultaneous Parallel Data Input-Output Intra-Connection

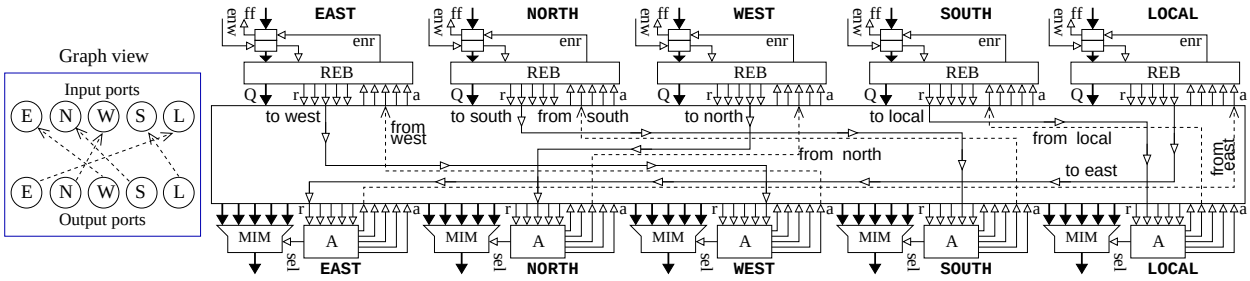
Our NoC can switch maximum N simultaneous crossbar intraconnects in parallel. The N number depends on the number of I/O pairs in the router. This feature is certainly not a new topic in the NoC research area as it has been implemented by some NoC architectures such as Intel Teraflops NoC [98], SCC NoC [103], Æthereal NoC [188], etc. However, we will present in this section, how the NoC performs such advantageous feature of a modern NoC router design with high bandwidth capacity. Fig. 3.12 presents graph views (left side) and structural views (right side) of a five-simultaneous crossbar interconnection in the XHiNoC mesh router for instance, i.e. the data input-output connections are from East (E) to West (W), North (N) to South (S), West (W) to North (N), South (S) to Local (L), and from Local (L) to East (E). The five simultaneous data input-output intra-connections are explained in the following items.

1. *Routing-Request Phase.* After the flit is buffered and appears at the output port of the $Queue$, then in the next cycle phase, the RE module computes routing request bit signals and sends the signals to an $Arbiter$ unit at the requested output port. Fig. 3.12(a) shows the graphical view and structural view of this phase.

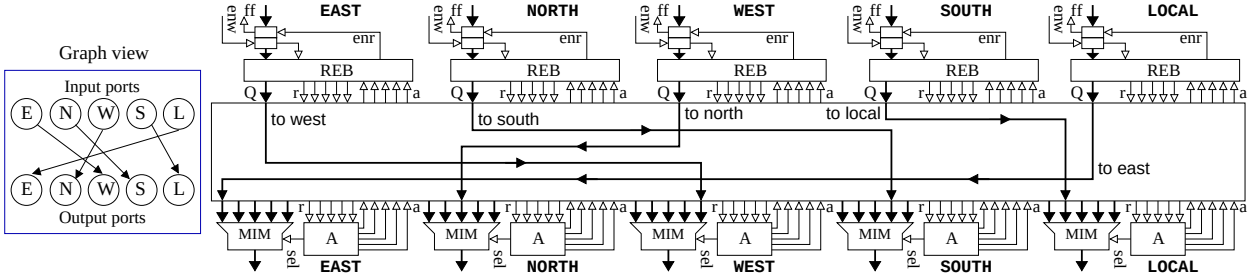
2. *Request-Acknowledge Phase.* When the $Arbiter$ units at the requested output ports detect the routing request signals, then each $Arbiter$ sends back a grant or routing acknowledge signal to the RE component. At the same cycle the $Arbiter$ unit



(a) Routing-Request signals



(b) Request grant or Arbitration signals



(c) Data outgoing

Fig. 3.12: Request-Grant-Accept mechanism to switch data in the XHiNoC router.

sends also a selection signal to the MIM component. This phase is presented by Fig. 3.12(b).

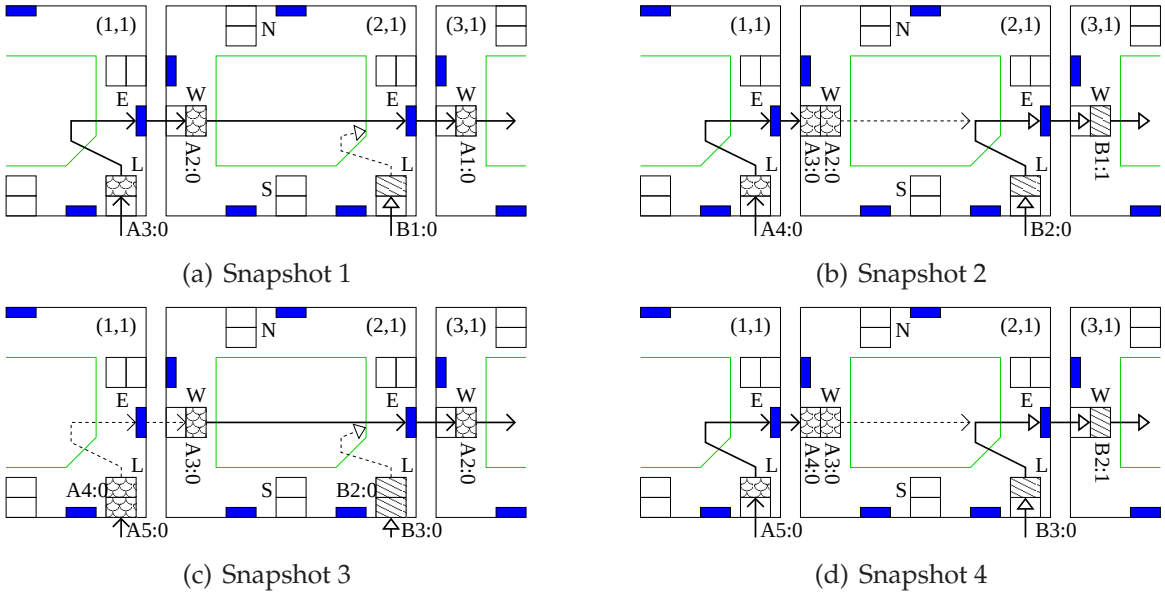
3. *Output-Switching Phase.* The $Arbiter$ responses the routing request signals by sending two signals as mentioned in phase 2. The output selection signal determines a data from an input port that will be switched out to the outgoing port, and the routing acknowledge signal enables concurrently to read the data from the input port. Hence, in the next cycle, the considered data will be switched out from input ports to output ports as presented in Fig. 3.12(c).

3.3.3 Link-Level Flit Flow Control

Data flows in our NoC are controlled using a link-level flit flow control especially when contentions between high-rate data occurs in the XHiNoC. The data flows on every communication link are controlled in XHiNoC and at flit level because of the use of the flit-by-flit rotating arbitration to switch and schedule data flow between contenting flits requesting the same outgoing link. The link-level data flow control is implemented using credit-based method, i.e. when the FIFO buffer in the next input port is full, then a new data flit will not be switched out to the outpot port, until there is a free buffer space in the FIFO buffer.

Fig. 3.13 presents four snapshot of the link-level flit flow control used in XHiNoC. If the contenting flits are injected with a very high data flow rate from the source nodes such that the total communication bandwidth of the contenting flits exceeds the maximum bandwidth capacity of the shared communication media, then the NoC will be saturated. The information of network congestion (i.e. full flag signals) from FIFO buffer will trace back to the injection nodes. The same flow control mechanism is also applied in the data flow between the local port of the router and network interfaces. Hence, even if the NoC is saturated, the injection rates at source nodes are automatically controlled by the full flag signals from FIFO buffers in the Local input ports of the routers. As presented in Snapshot 3 (Fig. 3.13(c)) and Snapshot 4 (Fig. 3.13(d)), we can see that new flits are not injected in the Local input ports of the router nodes (1,1) and (2,1) because the FIFO buffers in the Local input ports are full.

Fig. 3.14 presents a timing diagram of the data flow when contention occurs. The figures present the flit flows of message A transmitted from West input link and message B injected from Local input port. They compete to acquire the same outgoing link (East Output link). We assume that each input port has two-depth FIFO queue. The figure also presents the queues (R0 and R1) in each input port to show the contents and the signal (full flag) states of the queues during contention. The message A is transmitted on the West input link in every two-cycle, while the message B is injected at every one cycle (at maximum injection rate). Because of contention, the FIFO queues will be full (the full flag is set) at any cycle period. In the figure, the full flag of the Local FIFO queue is set in



(a) Snapshot 1 (b) Snapshot 2

(c) Snapshot 3 (d) Snapshot 4

Fig. 3.13: Link-level flit flow control in the XHiNoC.

phase 3, 4 and 5, because the registers R0 and R1 of the Local FIFO queue are used to store the $Flit\ B2$ and $Flit\ B3$. The $Flit\ B1$ itself in the data buffer of the routing engine (RE) must wait for a few cycle because the arbiter in the East outgoing port has selected the flit ($Flit\ A1$) from the West input port in phase 3 as the winner flit to access the outgoing port. In phase 5, the arbiter selects the flit ($Flit\ B1$) from Local input port to be switched in the East output link in the next cycle. Hence, in phase 6, the $Flit\ B1$ is switched out in the East output link, and the full flag of the Local FIFO queue is reset back. Hence, in the next cycle (phase 7), $Flit\ B4$ that has been waiting in the Local input port can be now stored in the FIFO queue.

In general, the characteristic of the link-level and data flit flow control can be seen by observing the data flow in the West input link, in the Local input port and in the East outgoing link. In the East outgoing link, the flit flow rate is about 0.5 flit per cycle (fpc), or one flit per two-clock-cycle (the maximum data rate in XHiNoC). We can also see that the arbiter unit makes a flit-by-flit rotating arbitration. Because the East outgoing link is shared in a fair manner by the flits of message A and message B, then the flit flow rates of both messages in the West and the Local input ports experience slower rates. They share also the maximum bandwidth capacity (0.5 fpc) of the shared outgoing link, i.e. 0.25 fpc (half of the maximum capacity) for each message, or one flit is transmitted in every four-cycle in both West input link and Local input port.

The congestion occurs in the West input link presented in Fig. 3.14 will affect the flow rate of the message A on the upstream links in successive clock cycles. The congestion situation will soon reach the source node from where the message A is injected. The injection rate reduction experienced by the message B will also occur in the source node of the message A. Therefore, globally, the injection rates of the message A and message B in their source nodes will be as same as their acception rates in their destination nodes,

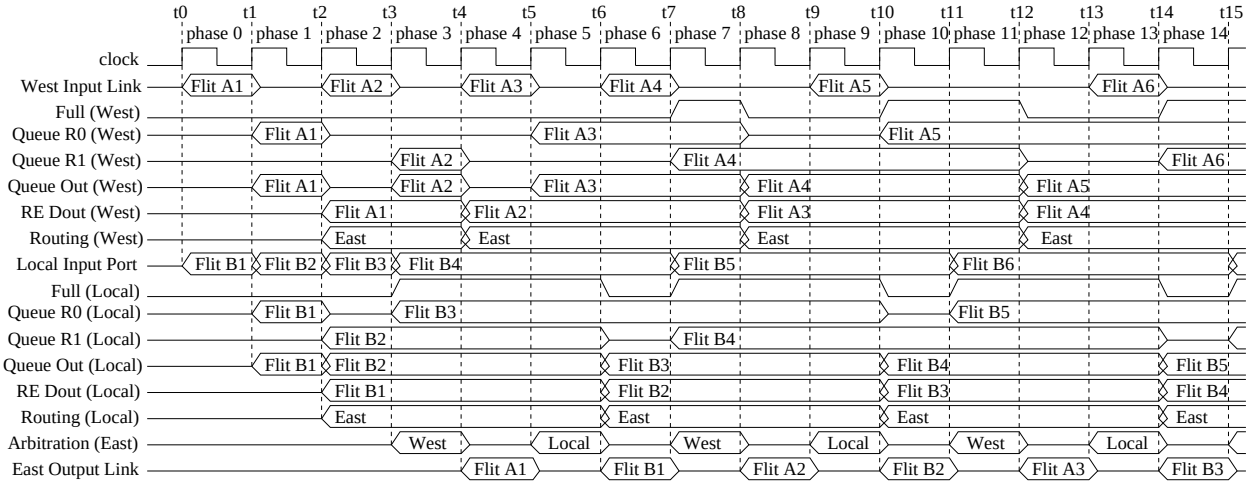


Fig. 3.14: Timing diagram (with contention) of the data switching and control paths.

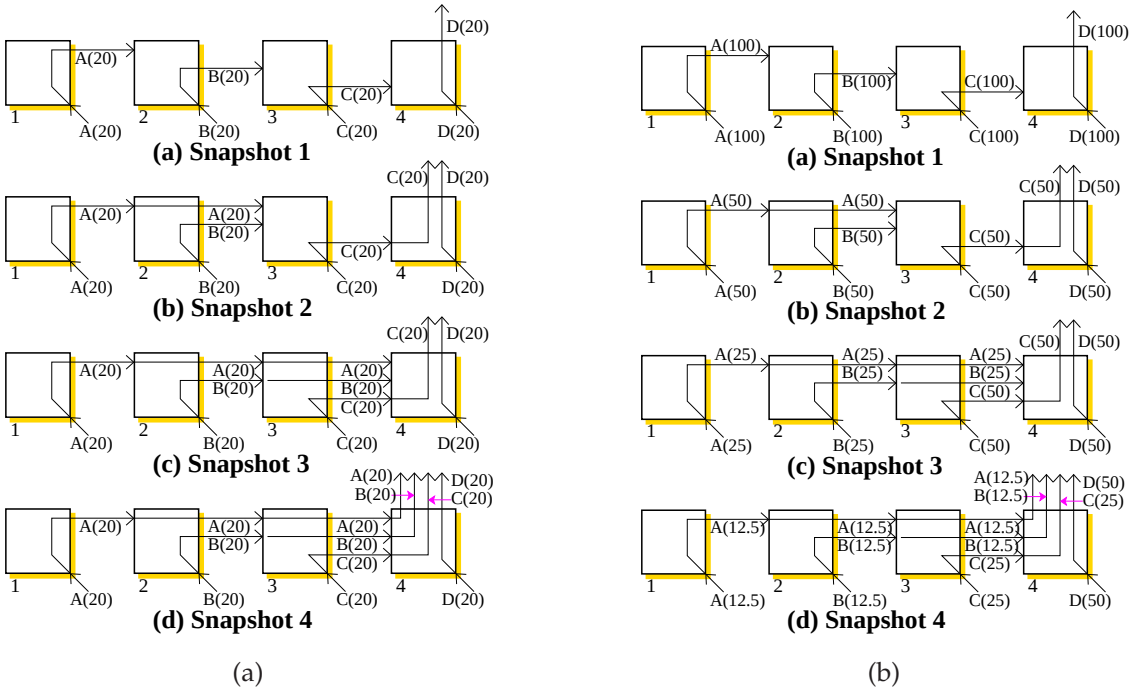
i.e. 0.25 flit per cycle if we assume that there is no other traffics considered in the NoC.

The same mechanism is also valid in the input side of the West input link. By using the flit flow regulation mechanism mentioned before, the data flit flows at link-level can be controlled automatically. This mechanism is also useful not only to enable the reduction of the buffer sizes of the FIFO queue but also can avoid data drops in the NoC. Data dropping in the context of NoC-based multiprocessor computation can degrade the application's performance.

3.3.4 Saturating and Non-Saturating Conditions

When messages are injected in the XHiNoC such that the total bandwidth requirement of considered traffics on every link does not exceed the maximum link bandwidth capacity, then the XHiNoC will not be saturated. Fig. 3.15(a) shows 4 snapshots of link bandwidth sharing situation as well as the local ID slot reservation, where the total bandwidth requirement of 4 messages is less than the maximum bandwidth capacity of the link. The values in the brackets represent the actual percentage message bandwidth over the maximum NoC link capacity and the reserved ID slot ($\%\ of\ Max\ BW\ :\ local\ ID\ slot$). As presented in the figure, Message A, B, C and D are injected to the NoC with local ID-tag 0 and consuming 20% of the maximum bandwidth capacity of the NoC link, resulting in the total bandwidth consumption of 80% of the maximum link capacity when they share a link in the North output port at node 4 as presented in the figure. In this situation, the NoC will be not saturated.

If a link is consumed by a few or some messages, in which the total expected bandwidths of the messages exceeds the maximum capacity of the link, then the message flows will be blocked for a while because of the flits' contention. The blocking situation in the XHiNoC is acceptable. The flow of the data flits in the congested link is constant at its maximum rate. Thus, the contenting flits must share this maximum rate. Therefore, the



(a) (b)

Fig. 3.15: Four snapshots of link bandwidth sharing situation (a) when the NoC is not saturated, and (b) when the NoC is saturated.

flow rates of the contenting flits will be slower than their expected rates. While, the injection rates at their source nodes are still at their expected rate, which are larger than the actual rates on the congested link, then the NoC will be saturated. Network Interface (NI) at source node will then stop injecting new flit when a queue in the Local input is full. Because of the benefit of the flit interleaving and link sharing capability, the data flows are not blocked permanently. After a few cycle, there will be a free space again in the queue and the NI can inject the next flit again. So, in steady-state situation, the actual injection rates at source node should follow the actual acception rates at targeted nodes of each communication edges in the NoC.

Fig. 3.15(b) shows the other 4 successive snapshots of the actual bandwidth consumption and local ID slot reservation, where the messages are expected to be injected to consume 100% of the maximum link bandwidth capacity. Initially, all messages will utilize the maximum link bandwidth capacity. Afterwards, when they start sharing a link, their data rate will be automatically reduced such that the total actual bandwidth of all message is equal to 100%. In this situation, the NoC will be saturated. Because our NoC is facilitated with the link-level flit flow control, no flit will be dropped. This congestion state will trace back to the injection nodes such that the injection rates of all messages will be reduced dynamically following their steady data rate point in the congestion nodes as presented in each snapshot in Fig. 3.15(b). This phenomena will also be presented later by observing the runtime actual injection and acception rates in the simulation results.

3.3.5 Special Features of the XHiNoC

The XHiNoC (described in VHDL model) is the extended version of HiNoC [97], [194], [219] (described in SystemC model). The extensions include the new microarchitecture and some features as the contributions of this thesis that are explained in the following points. More explanations about the points will be presented later in the next chapters.

1. *Specific Wormhole Switching with Flit-Level Packet Interleaving*. One of the special interesting features of the XHiNoC design concept is the implementation of a unique wormhole switching technique where flits of different messages can be interleaved and share the same communication media based on the locally organized message identity. As mentioned earlier, flits belonging to the same message will always have the same local ID-tag when acquiring a communication media (network link). By applying the flit-by-flit circulating arbitration technique, the wormhole messages can be interleaved at flit-level because every flit brings a local ID-tag to differentiate it from other flits of different flits. The local ID tag is updated by an ID management unit implemented in output port. By using this kind of wormhole switching, the head-of-line blocking problem commonly happen in the traditional wormhole switching can be solved partially without implementing virtual channels. More information about the novel wormhole switching method will be explored later in Chap. 4.

2. *Hold-Release Tagging Policy for Deadlock-Free Multicast Routing*. By using the local-ID-based method to switch wormhole packet over the network and by using further the flit-by-flit arbitration technique, a multicast deadlock configuration problem due to a multicast data request dependency can be solved effectively by using a so called *hold/release-tagging-based multicast policy* implemented on every NoC router. The multicast conflicts, which potentially lead to deadlock configuration (multicast dependency), are allowed and well organized by using the the *multicast conflict control and management* resulting in a new deadlock-free multicast routing methodology. The multicast flow control is based on the fact that a multicast data will not be released from FIFO buffer at input port if the set of all multicast routing requests has not been granted to access the multiple output ports. If a subset of the requests is granted, then the granted requests will be reset to avoid improper multicast flit replications. Further exploration about this novel deadlock-free multicast routing will be explained in Chap. 5.

3. *Flexible Runtime Connection-Oriented Guaranteed-Bandwidth Service*. The other interesting consequence of the local-ID-based routing organization is the ability to implement a flexible runtime connection-oriented guaranteed-service either for guaranteed-throughput or for guaranteed-bandwidth. The connection from a processing element to a single partner (unicast communication) or to multiple communication partners (multicast communication) is established at runtime. The connection, local-

ID reservation and bandwidth reservation are made autonomously by a header flit for a single partner or by multiple headers for multicast partners during application execution time. Communication media can be shared very flexible by allocating every data stream/message to a local ID slot. More explanation of this interesting feature will be further explored in Chap. 7.

3.4 RTL Simulator Infrastructure

The main infrastructure of the RTL simulator for XHiNoC performance evaluation is the testbench equipment module designed using VHDL. This module consists of two modules, i.e. a *Traffic Pattern Generator (TPG)* and a *Traffic Response Evaluator (TRE)*). Both components are explained further in the following sub-sections.

3.4.1 Traffic Pattern Generator

The TPG unit is used to generate traffics that will be injected to the local input port of the NoC. One TPG unit is connected to a local input port of one network node. There are some parameters that can be controlled by users to generate the types, the burst sizes and the flow rates of the traffics.

- The TPG units can generate a *Unicast Best-Effort* or *Multicast Best-Effort* packets as well as a *Unicast Guaranteed-Service* or *Multicast Guaranteed-Service* streams. The users can set the parameters in the VHDL testbench programs.
- The burst sizes of the generated packets or data streams can be also controlled by the users by setting the values of burst sizes or communication volumes in the VHDL testbench programs. The burst size values can be set uniform for all network nodes, or unique for every network node.
- The injection rates of the generated packets or data streams can also be set by the users by setting the injection rates values in the VHDL testbench programs. The injection rate values can be set uniform for all network nodes, or unique for every network node. The TPG unit can also give the transient response of the runtime actual injection rate measurement on every active node in a certain duration determined by the users.
- The users can also automatically set the size of the network. This option is valid only when the network structure is regular, such as the 2D $N \times M$ mesh network.
- Each message is encoded by the TPG unit such that every message can be well identified and differentiated from other messages in the network.

- Each flit of the message is numbered in-order by the TPG unit. Thus, it is easy for the TRE unit to check whether any or some flits are either loose in the NoC or are not accepted in the destination node.

3.4.2 Traffic Response Evaluator

The TRE unit is used to analyze the ejected/accepted/incoming data flits from the local output port of the NoC. One TRE unit is connected to a local output port of one network node. The following items will explain in general the functionality of the TRE units in our VHDL-based RTL simulator infrastructure.

- Each TRE unit at destination node will check the header of a packet, and analyzes whether the accepted packet is correct (the packet has attained its destination node correctly). The TRE unit counts how many clock cycles the header needs to attain the destination node. In this simulation, we interpret the latency metric as the number of clock cycles to transfer a flit from its source to its destination node.

- For each accepted flit, the TRE unit will check again one by one the order and the packet code of every accepted flit. Q number of flits or equivalent to $Q \times 4\ Byte = 4Q\ Bytes$ messages are injected from the TPG units at each injector (data producer) node.

- The TRE units will give some information, i.e. how many clock cycles that are required to transfer an amount number of flits, for examples, the 500^{th}, 1000^{th}, 2000^{th}, 3000^{th}, 4000^{th}, 5000^{th}, 6000^{th}, 7000^{th}, 8000^{th}, 9000^{th} and the 10000^{th} flits for each communication partner (source–destination pairs).

- The TRE units will write the simulation results into output text files. The generated text files are e.g. the communication bandwidth measured in Mega-Byte per second (MB/s) for each communication pair (source–destination pair) and the average value of the overall communication bandwidths of the considered communication pairs. The tail flit acceptance latency measured in clock cycle period for each communication pair (source–destination pair) and the average latency of the overall communication latency of the considered communication pairs are also written into output text files. The latency in clock cycle period to accept the header flits, response flits and the first payload flits, as well as the tail flits for different number of the injected flits are also reported in by the TRE units.

- The TRE unit can also give the transient response of the runtime actual acceptance rate measurement on every active node in a certain duration determined by the users.

3.4.3 Performance Evaluation Graphs

Based on the features of the TPG and the TRE units attached on the RTL-Simulator Infrastructure of the XHiNoC, there are some graphs that can be depicted to show the performance evaluation results of the XHiNoC. The graphs are described in the following items.

- *2D graph of the the last flit transfer latency vs injection rate.* This graph will present the acceptance of the last flit of each communication pairs when the expected rate of the data injection at each source node is increased or decreased. In general, the average last flit acceptance delay of all communication pairs over the expected data injection rate changes can be also presented in a graph. This type of graph is commonly used to evaluate NoCs performance as exhibited in [182] and [18].
- *2D graph of the the last flit transfer latency vs workloads.* This graph will present the acceptance of the last flit of each communication pairs when the number of injected data flits at each source node is increased or decreased. In general, the average last flit acceptance delay of all communication pairs over the number of workload changes can be also presented in a graph.
- *2D graph of the communication bandwidth vs injection rate.* This graph will present the actual (real) measured communication bandwidth (throughput) of each communication pairs when the expected rate of the data injection at each source node is increment or decrement. In general, the average real throughput of all communication pairs over the expected data injection rate changes can be also presented in a graph.
- *2D graph of the communication bandwidth vs workloads.* This graph will present the actual (real) measured communication bandwidth (throughput) of each communication pairs when the number of injected data flits at each source node is increased or decreased. In general, the average real throughput of all communication pairs over the number of workload changes can be also presented in a graph.
- *3D graphs of the link and bandwidth occupancy.* These graphs will present the link occupancy represented by the number of reserved ID slots and reserved bandwidth space at each output port of the NoC routers. In general, the total outgoing link occupancy of all output ports can be presented in a graph. This graph is interesting to see hotspots in a 2D network topology.
- *2D graphs of the injection and acceptance rate transient response.* These graphs will shows us the transient responses of the actual injection rate at a source node and the acceptance rate at a destination node measured at runtime during certain time period at certain active nodes, which are determined by the users. From these graph, we can see the time responses of each communication partner and analyze their steady state points compared to the expected data rates of each communication.

3.5 Summary

The main issue related to the implementation of the local ID management technique is the available ID slots on each communication link. If the parameterizable ID field on each flit is set to 4 bits, then a maximum number of 16 packets (2^4) can be in flight on the same link. The number of available ID slots can be increased by increasing the number of ID field bits as presented in the packet format, resulting in an increase of the routing table size and ID slot table size in the ID management unit. The number of required ID slots is application-dependent and cannot be increased anymore if the NoC had been implemented on ASIC. Hence, an optimal post-manufacture application mapping should be made, in order to avoid more than 16 packets interfering with each other across the same link.

In Chip-level Multiprocessor (CMP) systems running a coarse-grain multiprocessing applications, i.e. the ratio between computation to communication is more than one, it seems that 16 ID slots per channel are enough to run several applications. But, if the computation to communication ratio is less than one (fine-grain), then the number of available ID slots per channel must be taken into account. Programmers must ensure that each channel will not be overloaded with excessive communication traffics. This effort can be easily done especially when an explicit parallel programming model is considered. The problem may appear when we use implicit parallel programming models such as shared-memory and multithread programming models. Therefore, it is reasonable to anticipate the ID run out problem in the CMP systems by setting the minimum acceptable number of ID slots per link and setting the number of available ID slots at each local output port equal to the number of the processing element cores. This issue has been well addressed in Section 3.1.2.

Fortunately, in the context of embedded Multiprocessor System-on-Chip (MPSoC), applications traffic patterns are predictable. Hence, it is possible in this case to map the application in the NoC-platform in such a way that every considered traffic will be able to reserve one ID slot per link to perform its data communication with fulfilled bandwidth requirement. Although it would be a rare case that more than 15 messages are in-flight in the same link, it is however a good decision if the packet dropping mechanism is applied in this case to avoid data flow stall. If the number of ID tags per link N_{slot} is set to cover all considered traffics, e.g. equal to Equ. 3.1 when using a minimal adaptive routing algorithm, then the packet dropping mechanism can be neglected. There is a design trade-off in this aspect. By setting the minimum number of ID slots (N_{slot}) per link as discussed in Section 3.1.2, the size of the RRT and ID Slot Table units would be larger, but there is no need for a retransmission protocol. When data dropping is applied and the number of entries in the tables units is reduced, then router size will be smaller, but the retransmission protocol must be applied leading to area overhead in the network interface, and probably time overhead when the data drop occurs.

Chapter 4

Wormhole Cut-Through Switching: Flit-Level Messages Interleaving

Contents

Head-of-line blocking problem is a main issue when messages are routed in a network using traditional wormhole switching. The problem will be theoretically and virtually explained in Section 4.1. This chapter proposes a *wormhole cut-through switching method* that can solve the head-of-line blocking problem by interleaving different messages at flit-level in the same buffer pool without using virtual channels. The novel switching method suitable for NoCs will be described in Section 4.2. Section 4.2 will show also the specific packet format used in the router as the key factor to implement the proposed wormhole switching, describe the formalism to prove the correctness of the routing method, and present the performance characteristics of the router during saturating and non-saturating condition.

Section 4.3 presents the experimental results under several traffic scenarios. The design customization method to optimize the logic area of the router is presented in Section 4.4. Section 4.5 exhibits the synthesis results of the XHiNoC routers with fully IO and custom-made IO interconnects. Section 4.6 will discuss some issues related to the proposed wormhole switching method.

4.1 Blocking Problem in Traditional Wormhole Switching

Wormhole switching has been widely used for NoC routers because of the need for smaller size buffers compared to store-and-forward switching scheme. In the wormhole switching, each packet is divided into a sequence of flow control digits (flits). The header flit of each packet makes a routing direction on each node and reserves a set of routing paths, while the payload flits will follow the path set up by the header flit. The tail flit of each packet at the end will then terminate the reservation. Unfortunately, the main problem of this traditional wormhole switching method is a head-of-line blocking problem. If the header flit is blocked then it will also block the remaining paths that are still used by the wormhole packet.

Fig. 4.1 shows two snapshots of a blocked data flow of the traditional wormhole packet switching. The wormhole packets A, B, and C will be routed to node (3,1) and their data flow rates are very high (probably they consume high bandwidth capacity). If the packet C is not blocked and continuesly moves to the next requested output port, then packet A and B will be blocked until the entire flits of the wormhole packet C have been passed on, and the last flit of the wormhole packet C has removed the link reservation. Each wormhole packet can acquire the link after the other packet finishes forwarding its last flits.

In Snapshot 1 of the figure, if the wormhole packet C is blocked at node (3,1) because

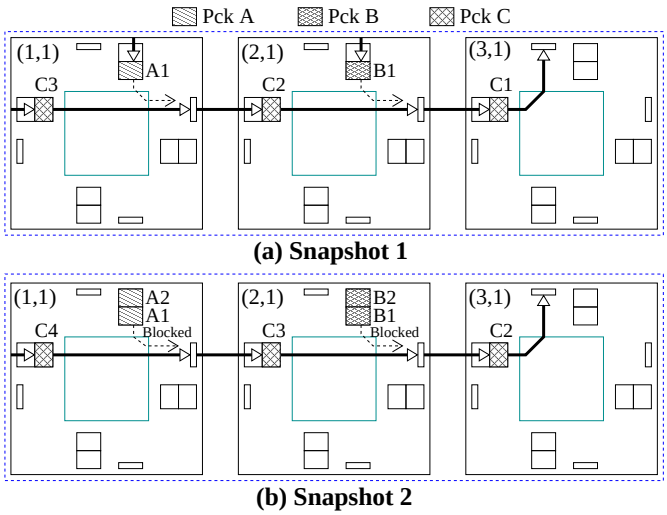


Fig. 4.1: Head-of-line blocking problem in wormhole switching.

the required link is acquired by another packet for instance, then Packets A and B would be also blocked at node (1,1) and (2,1). If the size of a packet, which is blocking the flow of the packet C is very large, then the blocking situation will also occur for long time. If the wormhole packet size is set small, e.g. 4 flits, then we can estimate the situation in the Snapshot 2 of Fig. 4.1 that packet A and B will escape from the blocking situation after a few cycles.

As presented of the Snapshot 2 of the figure, because the depth of the FIFO buffers are two, then in the blocking situation, the last two flits (flit C3 and C4) of the 4-flit wormhole packet C can occupy two FIFO buffers at West input port of the router node (3,1). Thus, packet A can occupy the West port FIFO at node (2,1). If the FIFO depth is set to small number of M registers (e.g. 4,6 or 8), and the size of the wormhole packet is limited to M flits, then a *buffered wormhole packet switching* scheme can be implemented in this context. This switching scheme is actually similar to the virtual cut-through switching. However, this switching technique can only solve any blocking situations partially. The wormhole switching is used to minimize the buffer size in routers. Hence, this technique, which requires larger buffer size, waives the objective of using the wormhole switching.

The head-of-line blocking problem can also be solved by using virtual channels. However, the main criticism of the use of virtual channels in the NoC context is prohibitive area cost in terms of buffering and slower speed of router cycle time. Virtual channels will increase total buffer counts and result in power consumption that would exceed the target constraint for an embedded application [103]. The silicon area of a NoC router is dominated by buffers. Hence, power is mostly dissipated from these buffer components. The design of NoC router with minimum buffer size would be always an important aspect for NoC-based embedded multiprocessor systems-on-chip (MPSoC), whose power supply capacity is limited by the battery life of the system. However, the minimum buffer size would also be an interesting design parameter for chip-level multiprocessor (CMP) systems domain, especially if network size is very large, in which the power dissipa-

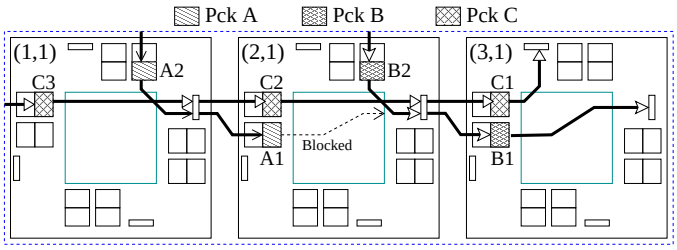


Fig. 4.2: Head-of-line blocking problem solution with 2 virtual channels per input port.

tions are scaled up by the network size. The work in [204] has found that additional VCs do not increase router cycle time because the router complexity slows down the router working period. Virtual channels will also add more arbitration to router's critical path, potentially affecting the cycle time or pipeline depth of the router [87]. The same result is presented in [12], in which additional arbitration and multiplexing circuits for VCs on physical channels introduce delays into the critical path in implementing alternate routing algorithms.

Fig. 4.2 shows the solution of the head-of-line blocking problem by using 2 virtual channels per input port. As presented in the figure, the packet B will not be blocked by packet C, since it uses the other virtual channel buffer at the West input port in the network node (3,1). However, packet A is still blocked in the network node (2,1) because all 2 virtual channels in the network node (3,1) are busy (occupied by packet B and packet C). The problem can be further solved by inserting a new virtual channel buffer in the router input port. It seems that the more virtual channels are inserted to the router, the more wormhole packets can share the same communication link, which can give a very significant impact on the NoC area overhead. The following subsection will show how the link can be shared by the wormhole packets by using local ID slot which replaces the VC-ID functionality and denies the use for virtual channels accordingly.

4.2 The Novel Wormhole Cut-Through Switching Method

The head-of-line blocking problem that mainly occurs when using the traditional wormhole switching method can be solved without the need for virtual channels by using the proposed flit-level interleaving wormhole switching method that is implemented in the XHiNoC router. The use of virtual channel ID is replaced by the use of variable local ID in the context of our proposed method. In general, the purpose of using variable local ID and virtual channel ID is the same, i.e. to solve the head-of-line blocking problem. The functionality of virtual channel controller units used in the context of the virtual channel solution is implicitly replaced by local ID management units implemented on our NoC router.

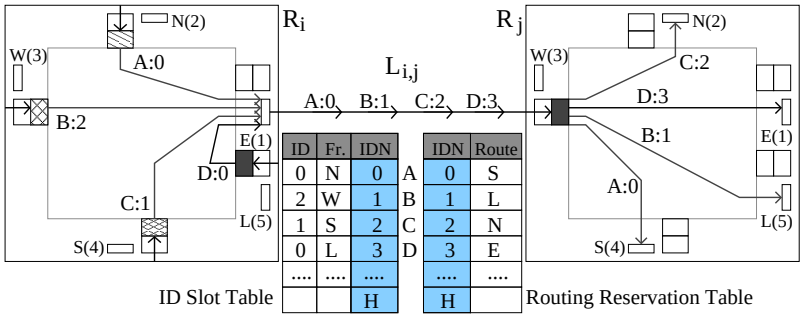
4.2.1 Virtual-Channelles Solution with ID-based Multiple Access Technique

Fig. 4.3 depicts the concept of the novel wormhole switching method, in which flits of different messages/data streams can be interleaved (virtually mixed-cut-through at flit level). The novel method is supported by the existence of an ID slot table and and a routing reservation table on every communication link connecting an input port of a router with an output port of another router. In Fig. 4.3, we can see a communication link $L_{i,j}$ connecting the East output port of router R_i and the West input port of router R_j. At the East output port, there is a local ID slot table with a number of H ID slots, and at the west input port, there is a routing reservation table with a number of H routing direction slots as well.

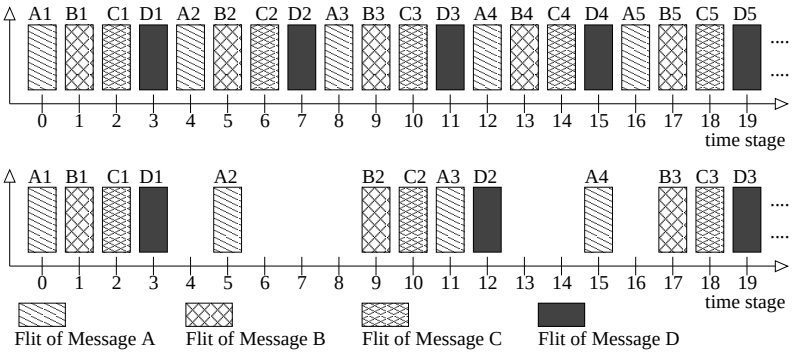
The regulation of the switching method is based on the fact that an ID tag number is attached on each flits of message, where the flit allocated to local ID slot k in the ID slot table will have a local ID tag k, and flits belonging to the same messages will have the same local ID-tag. In other words, a flit brings a data word together with a local ID-tag number. The ID slot table consists of $H + 1$ programmable slot registers. A new incoming flit (header flit) of a message/data stream can reserve one local ID slot, and index the reserved local ID slot number based on two local information, i.e. its previous (old) local ID tag and from which port it comes. Fig. 4.3 shows that messages A, B, C and D coming from North ($N(2)$), West $W(3)$, South $S(4)$ and Local $L(5)$ port, respectively, are allocated to local ID slots 0, 1, 2 and 3, and use the slots as their new ID tags, respectively. The combined alphabetical and numerical symbols *A:0*, *B:2*, *C:1*, *D:0* in the router R_i means that the messages A, B, C and D have ID tag 0, 2, 1 and 0, respectively when they are buffered at input ports N, W, S and L. The ID slot number 0 for example is reserved by message A, and is indexed with tag 0 and port N (North), because the flits of message A come from the North port with the previous ID-tag 0.

By indexing the local ID slots based on the aforementioned two local information, one local ID slot can be allocated for only one message, and different messages can be guaranteed to be allocated to different ID slots. As presented in Fig. 4.3, on the communication link $L_{i,j}$ and at the West input buffer of the router R_j, the flits of messages A, B, C, D can be multiplexed and flow through the link with new local ID-tags (*A:0*, *B:1*, *C:2*, *D:3*). When the flits of the messages are buffered in the input ports, then the flits will be routed to their requested routing direction based on their current local ID-tags. The proof of the correctness of the ID-based routing methodology for the novel wormhole switching technique will be exhibited in Section 4.2.3.

As presented in the figure, four routing registers of the routing reservation table at the West input port have been programmed with different output directions, i.e. the routing register numbers 0, 1, 2 and 3 have been written with the output directions S, L, N and E, respectively. Routing engine at input port can route each flit with an attached ID-tag to an output port direction by reading the ID-tag k of the flit and fetch the output direction



(a) Conceptional view.



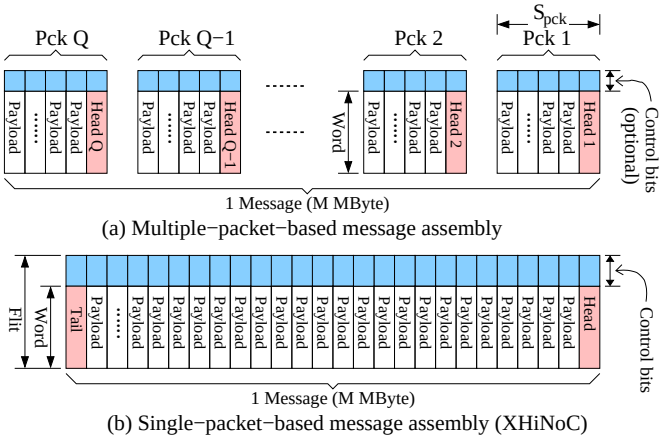
(b) Bandwidth share example.

Fig. 4.3: Local ID-based Data Multiplexing.

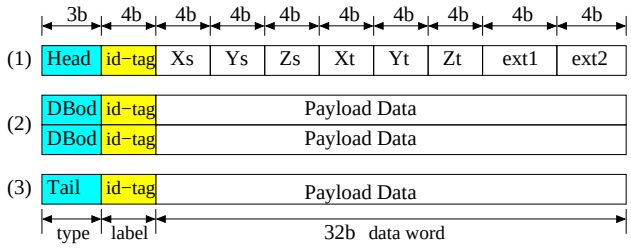
from the register number k of the routing table. Message A for example has local ID-tag 0 at the West input port of the router R_j, thus it fetches the routing direction by reading the register slot number 0 (according to its ID-tag) of the routing reservation table. As presented in Fig. 4.3, the table content in the register number 0 of the routing reservation table is S (South output routing direction). Therefore, the flits of message A are routed to South (S) output port in the router R_j.

As presented in Fig. 4.3(b), two example cases are exhibited. The first case (upper figure) shows a flit interleaving where the total bandwidth consumption of all messages are 100% of the maximum link bandwidth capacity (B_{max}), i.e. each of 4 messages consumes 25% B_{max}. The second case presents that 57.5% of the B_{max} have been consumed by all packets, i.e. packet A is 20%, and packet B, C and D are 12.5% B_{max}, respectively. Thus, there is still 42.5% free BW that can be used by other wormhole packets coming to the link.

The mechanisms to reserve a local ID slot from the ID slot table and to program a routing output direction in the routing reservation table by the wormhole packets are made at runtime during application execution time. Therefore, the XHiNoC uses a special packet format for the wormhole packets by introducing a flit type bit field (beside the ID-tag bit field) in every flit of the wormhole packets to enable such mechanisms as explained later in Section 4.2.2.



(a) Packet format comparison



(b) XHiNoC's packet format

Fig. 4.4: (a) Comparisons of multiple-packet-based and single-packet-based message assembly, and (b) the XHiNoC packet format.

4.2.2 Packet Format

A message in XHiNoC is associated with a single packet. For a unicast message, the packet will have only one header flit, even if the size of the message is very large. Hence, in this paper, the terms "packet" and "message" have similar interpretation. Fig. 4.4(a) presents the difference between a multiple-packet-based message assembly (Fig. 4.4(a)(a)) and a single-packet-based message assembly (Fig. 4.4(a)(b)) that is used by our NoC. In the multiple-packet-based assembly, the message is divided into Q number of packet, where the size of each packet (S_{pck}) can be freely determined. Each packet consists of a header containing routing information or the address of the destination node and a few payload flits. The control bits can be optional, e.g. when the first payload of each packet represents the number of payload flit per packet then the control bit can be neglected. But when all payload flits in each packet represent a data, then the control bits can be used to determine the type of each payload data in each packet.

If adaptive routing algorithms are used to route the large-size multiple-packet-based message containing large number of packets, then an out-of-order problem may occur at each destination node. This is because the headers of each Q number of packets ($Head\ 1$, $Head\ 2$ until $Head\ Q$), which are routed adaptively in the NoC, may arrive at the destination node with different order as they injected from the source node. Because of that

problem, our XHiNoC uses an alternative message assembly, i.e. single-packet-based format shown in Fig. 4.4(a)(b). Moreover, the single-packet-based format, in which additional control bits are attached on each flit, is the key factor to implement the wormhole switching capable of interleaving message at flit-level in order to solve the head-of-line blocking problem.

The detail packet format and the control bits used in the XHiNoC architecture is presented in Fig. 4.4(b). The message is split into several flits and has 39-bit width, 32 bits for dataword plus 7 extra bits i.e., 3-bit field to define the type of flits and 4-bit field to determine the local identity label or ID-tag of the message. The ID-tag field is set 4 bits, resulting in 16 available ID-tag which conforms to the number of available ID slots on every communication link.

For the NoC version with *Best-Effort* (BE) service, the flit type can be (1) a header flit ($Head$), (2) a databody (payload) flit ($DBod$), or (3) a tail (end of payload data) flit ($Tail$). The other possible flit types can be introduced for the NoC version with *Guaranteed-Throughput* (GT) service, or combination of both BE and GT services. Routing direction on each router is made only once by the packet header. Afterwards, the payload flits (probably a very long data stream) will track the routing paths made by the header. By using the packet format shown in Fig. 4.4(b), out-of-order problem can be avoided when we use an adaptive routing algorithm.

The message is classified into three flit types i.e., header flit ($Head$), databody or payload data flit ($DBod$) and tail flit ($Tail$). The source and target addresses of the message are defined into 3D address (x, y, z). The z address is not used in this current 2D mesh topology but it is spared to be used for developing a hierarchical 2D, or stacked 3D networks on chip. The header flits are introduced to autonomously make the local ID slot reservation in the ID slot table and the routing direction reservation in the routing reservation table on every XHiNoC router. Flits belonging to the same message have the same local identity number (ID-tag) to differentiate it from other flits of different messages, when it passes through a communication link of the NoC. The ID-tag of the data flits of one message will vary over different communication links allowing different messages are interleaved each other at flit-level while being routed with wormhole switching.

4.2.3 Correctness of the Routing Path Establishment

Lemma 4.1 *By organizing the ID-tag of each flit of packets using Alg. 9 with local ID Slot Table defined in Def. 3.19, then we can guarantee that flits belonging to the same packet will always have similar local ID-tag $k \in \Omega$ on each communication link $L_{i,j} \in \Lambda$.*

Proof of Lemma 4.1 *Based on Def. 3.4 and Def. 3.19, we can see that the local ID slot $k \in \Omega$ is indexed by using two variables i.e. the ID-tag $ID \in \Omega$ of a message from an input link and from which port $n \in \Phi$ the message come. Further, we define $F_n(type, ID)$ as a flit from input port n with local ID-tag ID.*

If we make a pre-assumption*, that every single Id slot k is allocated for every flit of similar message on every link $L_{i,j}$ connected to an input port n, then $(\forall v, w \in \Omega_{i,j}) \cap (v \neq w)$, a flit $F_n(type, v)$ will not belong to a similar message with $F_n(type, w)$, where $\Omega_{i,j}$ is the set of local ID slots on the link $L_{i,j}$. $\forall p, q \in \Phi \cap p \neq q$, there is a probability that $F_p(type, v)$ and $F_q(type, w)$ have similar local ID-tag ($v = w$), because the sets of local ID Slot on each communication link is the same. But certainly $F_p(type, v)$ and $F_q(type, w)$ are flits of different packets, although $v = w$, $\because p \neq q$. If every single ID slot k is assigned to a new packet by identifying two parameters i.e., from which port the packet come and its current old ID-tag, we can make sure that $(\forall x, y \in \Omega) \cap (x \neq y) \Rightarrow S(x) = (v, p) \neq S(y) = (w, q)$ according to Equ. 3.7, $\because \forall p, q \in \Phi \colon p \neq q$, or if $p = q$ (ports connected to the same link $L_{i,j}$) then $v \neq w$, $\forall v, w \in \Omega_{i,j}$ according to the* pre-assumption*. x and y is the new id-tags for flit $F_p(type, v)$ and $F_q(type, w)$, respectively*

Therefore, by further applying a local ID-tag management described in Al. 9, we can make sure that each different message can be allocated to one ID slot $k \in \Omega$ and guarantee that flits belonging to the same packet can be assigned to a similar local ID-tag $k \in \Omega$ on each communication link. Accordingly, the proof makes also the aforementioned pre-assumption *to be a* valid assumption *continuously on every communication link.*

Lemma 4.2 *By using ID-based routing mechanism as described in Alg. 7 and organizing the ID-tag of each flit of packets as described in Alg. 9, then each flit belonging to the same packet can be routed to a correct routing direction $r_{dir} \in D$ although the flits are interleaved with other flits that belong to other different message by applying the rotating flit-by-flit arbitration described in Def. 3.16.*

Proof of Lemma 4.2 *Based on the **Proof of Lemma** 4.1, if we have proved that different packets can be allocated to different local ID Slot such that flits belonging to the same packet will always have similar local ID-tag, then by implementing ID Slot Table at every outgoing port such that if $\exists\, S^k, S(k) | k \in \Omega$ on each communication link $L_{i,j} \in \Lambda$ connecting routing node $R_i, R_j \in \Re$, then we can also further implementing a Routing Table $T(k)$ (Def. 3.11) at every incoming port, which routes flits of packets based on their ID-tag k in such a way that the interleaved different flits can be correctly routed into their correct paths.*

4.2.4 Switching Behaviors in Saturation and Non-Saturation

Saturating and non-saturating conditions in interconnection networks are two situations that could be used to represent the performance characteristics of a network. The capabilities of a network to handle both situations represents an advantageous characteristic of the network. When any or some packet flows are blocked in the network, then saturation condition will happen. The blocking situation occurs because the data rate of any or some packets exceeds the maximum bandwidth capacity of the used communication resources. In Section 3.3.4 and Section 3.3.3 of Chap. 3, the performance characteristics of the XHiNoC under saturating and non-saturating conditions as well as the overflow control mechanism during blocking situation have been described well.

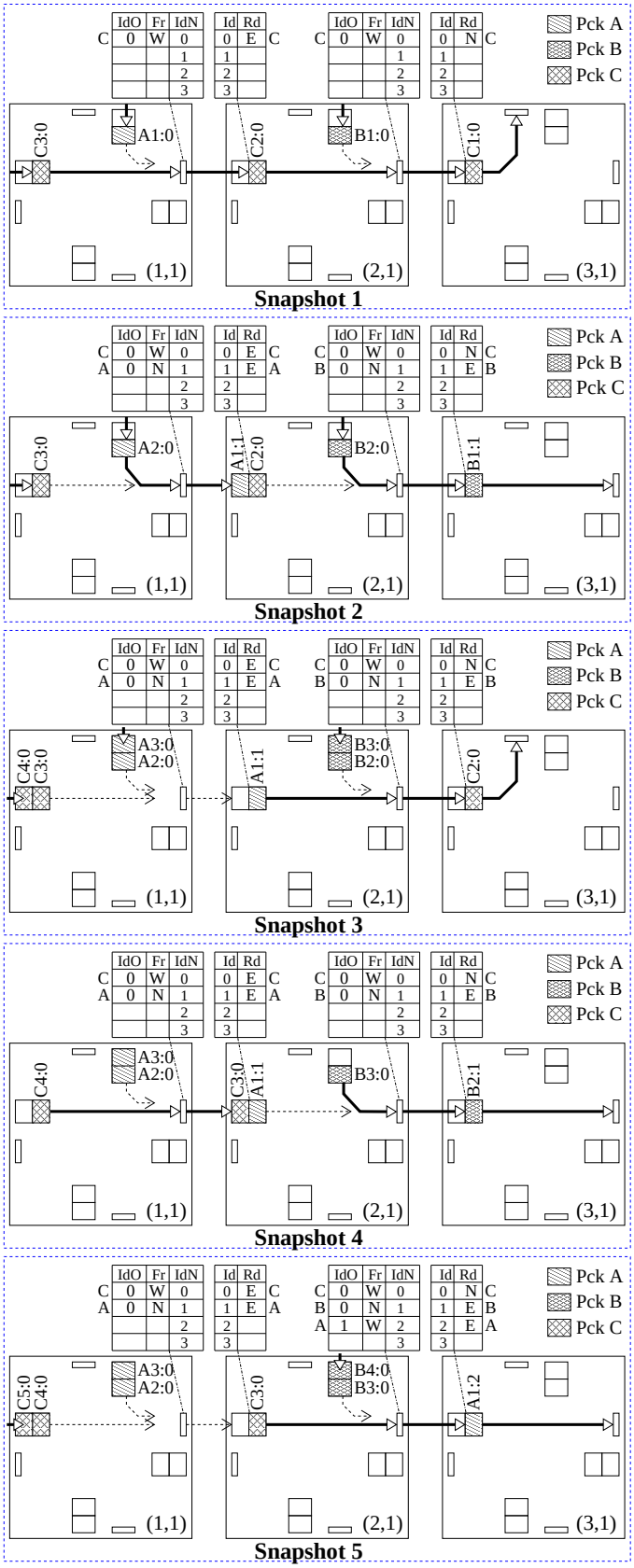


Fig. 4.5: Switching behavior in saturation.

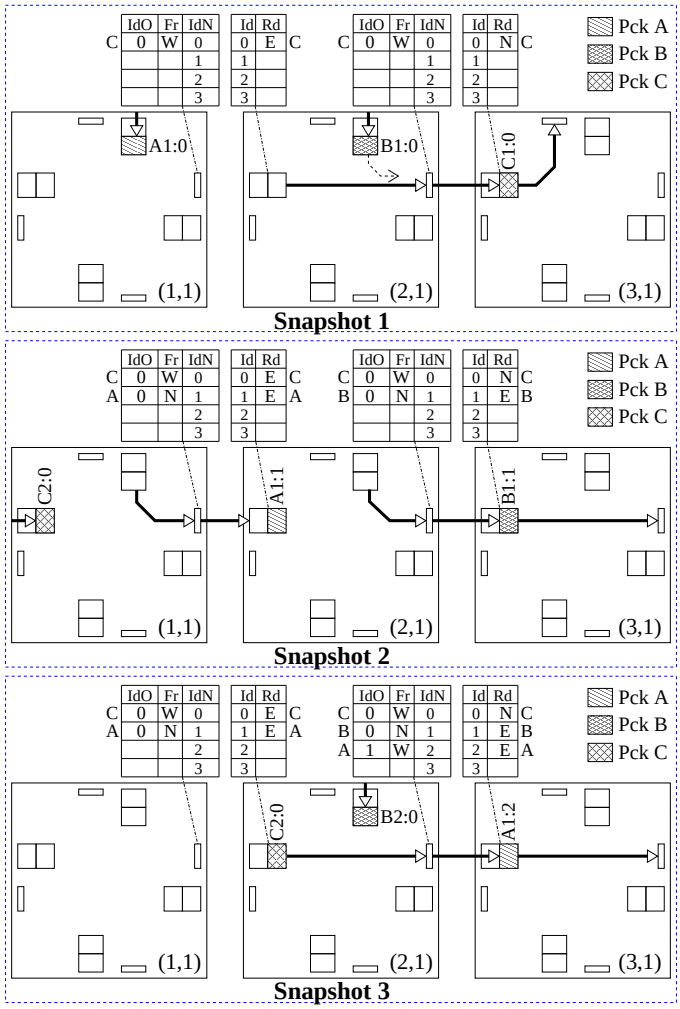


Fig. 4.6: Switching behavior in non-saturation.

In this subsection, the XHiNoC capability to tackle the blocking and non-blocking situations by using the novel wormhole switching method will be explored in detail. Fig. 4.5 shows five snapshots of the communication link sharing between three wormhole packets. Each packet is injected from a source node with maximum injection rate, i.e. consuming 100% of the maximum bandwidth capacity of the XHiNoC link. In this situation, the NoC will become saturate when the wormhole packets compete each other to share the same outgoing link as presented in the figure. The flits of packet C are contenting with the flits of packet A to acquire the East outgoing link in the router node (1,1), while the flits of packet B are competing with the flits of packet C to use the East outgoing link in the router node (2,1). The first, the second and the third flits ($C1:0$, $C2:0$ and $C3:0$) of packet C have been routed firstly, and each of them is allocated with local ID-tag 0. The flits have acquired the West input buffer of the router node (3,1), (2,1) and (1,1), respectively.

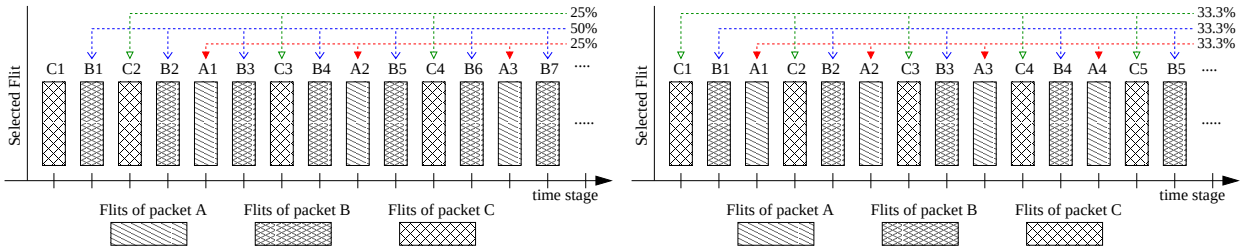
As presented in **Snapshot 1** of Fig. 4.5, the first flits of packet A ($A1:0$) and packet B ($B1:0$) come later in North input buffers of the router node (1,1) and (2,1), respectively. Due to the use of a the flit-by-flit rotating arbitration as described early in Section 3.2.3

of Chap 3, then in **Snapshot 2**, the flit $B1:0$ from North input port of router node (2,1) is selected and allocated to local ID tag 1 to use the East outgoing link, and the flit $A1:0$ from North input port of router node (1,1) is selected and allocated to local ID tag 1 to acquire the East outgoing link. Now, we can see that the flits of different wormhole packets are being interleaved with different local ID-tags.

In the next snapshots, the situations in every input buffer can be easily estimated. The arbiter units at the East output ports of the router node (1,1) and (2,1) alternate their selection between West and North input ports. When the first flit of a packet (a header flit of a new packet) acquires the shared links, then the packet will be allocated to a new free local ID slot. For example, the wormhole packet A in **Snapshot 3** is allocated to local ID slot 1 in the West input port of the router node (2,1), because ID slot 0 has been used by packet C. While, in the West input port of the router node (3,1) as presented in **Snapshot 5**, packet A is allocated to local ID slot 2 because local ID slots 0 and 1 have been used by packet C and packet B.

If the output selection results at the East outgoing link in the router node (2,1) are analyzed on each snapshot, then we can see that packet B consumes the outgoing link more frequent than the other two packets. Packet A and packet B fairly share the outgoing link between the router nodes (1,1) and (2,1), because the arbiter unit at East outgoing link in the router node (1,1) circulates fairly the arbitration between flits coming from West and North output ports. Hence, packet A and C consumes respectively 50% of the maximum bandwidth capacity of the link. In the next router node (2,1), both packets A and C compete again with packet B to share the link between the router node (2,1) and (3,1). Therefore, in the same situation, packet B coming from North input port consumes 50% of the maximum link bandwidth capacity, while the rest bandwidth space is shared by packet A and C, which come from West input port. It means that packet A and C share 25% bandwidth space of the link. This condition has been also well explained in Section 3.3.4 of Chap. 3.

Fig. 4.6 presents three snapshots of another communication media sharing between three wormhole packets, where each packet is injected from a source node to consume $\frac{1}{3}$ or 33.33% of the maximum link bandwidth capacity. At West input port in the router node (3,1) of each snapshot, we can see that the link can be shared fairly by packets A, B and C. The total bandwidth consumption of the 3 wormhole packets is 100% (i.e. about $3 \times 33.33\%$). Compared to the other situation presented in Fig. 4.5 in advance, the fairness of the communication media share is better and well-depicted. Fig. 4.7 exhibits the output selection results made by the arbiter unit at East output port of the router node (2,1) between flits of packet A, B and C.



(a) According to Fig. 4.5 (b) According to Fig. 4.6

Fig. 4.7: Flits output/outgoing selection results at East output port in the router node (2,1).

4.3 Experimental Results

In this section, the XHiNoC is clocked with $1\ GHz$. Hence, the maximum bandwidth capacity of each communication link is $B_{max-\ell}^{(1\ GHz)} = 4\ Byte \times 1\ GHz \times \frac{1}{2} = 2\ GB/s$ or $2000\ MB/s$. By using the number of *flits per cycle (fpc)* unit, the maximum data rate of every link is $0.5\ fpc$. Thus, $0.5\ fpc$ is equivalent to $2000\ MB/s$. If we have data rate R in fpc, where $0 < R \leq 0.5\ fpc$, then we have the relevant bandwidth rate $B = R \times 4000\ MB/s$, such that $0 < B \leq 2000\ MB/s$.

4.3.1 Bit Complement Traffic Scenario

This subsection presents the performance of our XHiNoC over the bit complement traffic pattern under 4x4 mesh planar topology, in which a message is injected from source node with a binary address and will be accepted in the target node bit complementary of the binary source address. For example, if a packet is injected from node $(1, 3)$, where its binary address is $(01, 11)$, then the packet will be accepted at node $(10, 00)$ in binary code address or node $(2, 0)$ in decimal code address. In the 2D 4x4 mesh with the bit complement traffic, we will have 16 node communication pairs (16 node as injector and as acceptor node at the same time).

Fig. 4.8, Fig. 4.9 and Fig. 4.10 will present the XHiNoC unique behaviors in response to the bit complement traffic pattern under saturated and non-saturated conditions. Fig. 4.8(a) shows the measurement of the average latency to transfer the tail flit (end of payload flit) from source to target node for different numbers of the total injected flits per data producer node and different injection rates (IR) in flit per cycle (fpc). The average tail flit latency is $\delta_{avg} = \frac{1}{16}\sum_{k=1}^{16}\delta_k$, where δ_k is the latency of the communication pair k in the bit complement traffic scenario.

Fig. 4.8(c) shows also the measurement of the average bandwidth over different numbers of the total injected flits per message. The average bandwidth is $B_{avg} = \frac{1}{16}\sum_{k=1}^{16}B_k$, where B_k is the actual/measured bandwidth of the communication pair k in the bit complement traffic scenario. It looks that for similar injection rate, the average actual/measured bandwidth rate is constant although the communication volumes are changed from 500

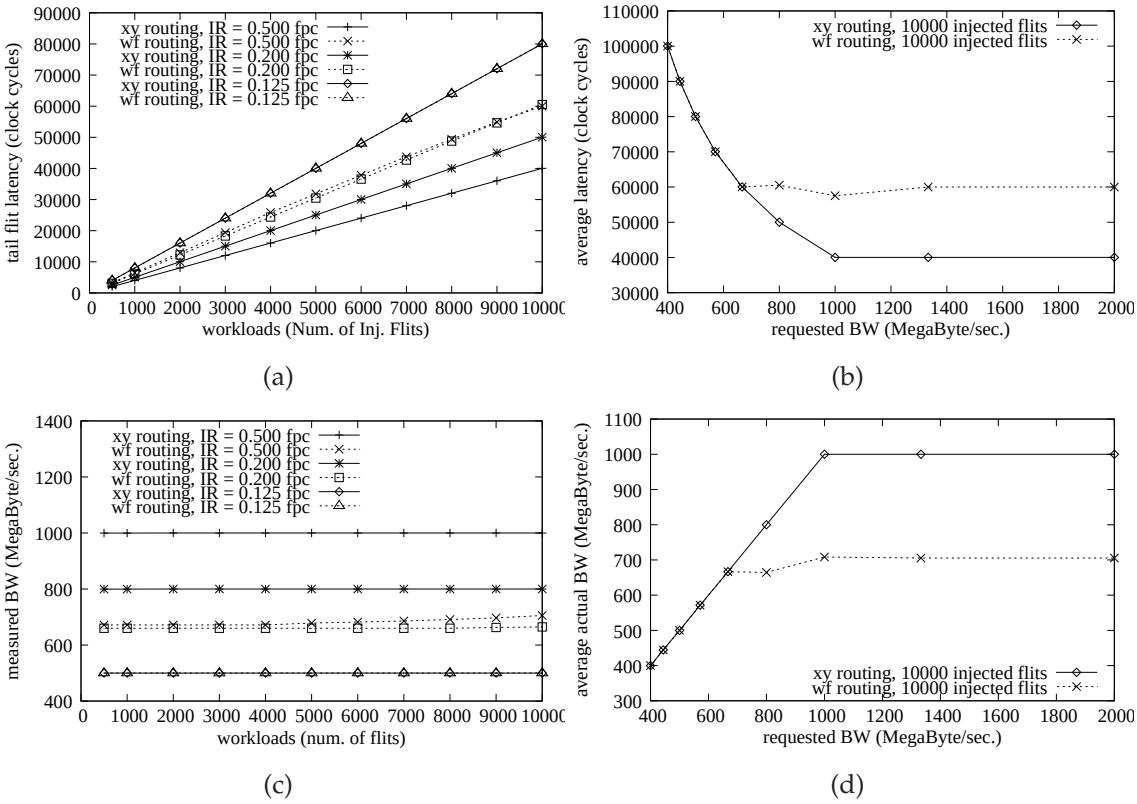
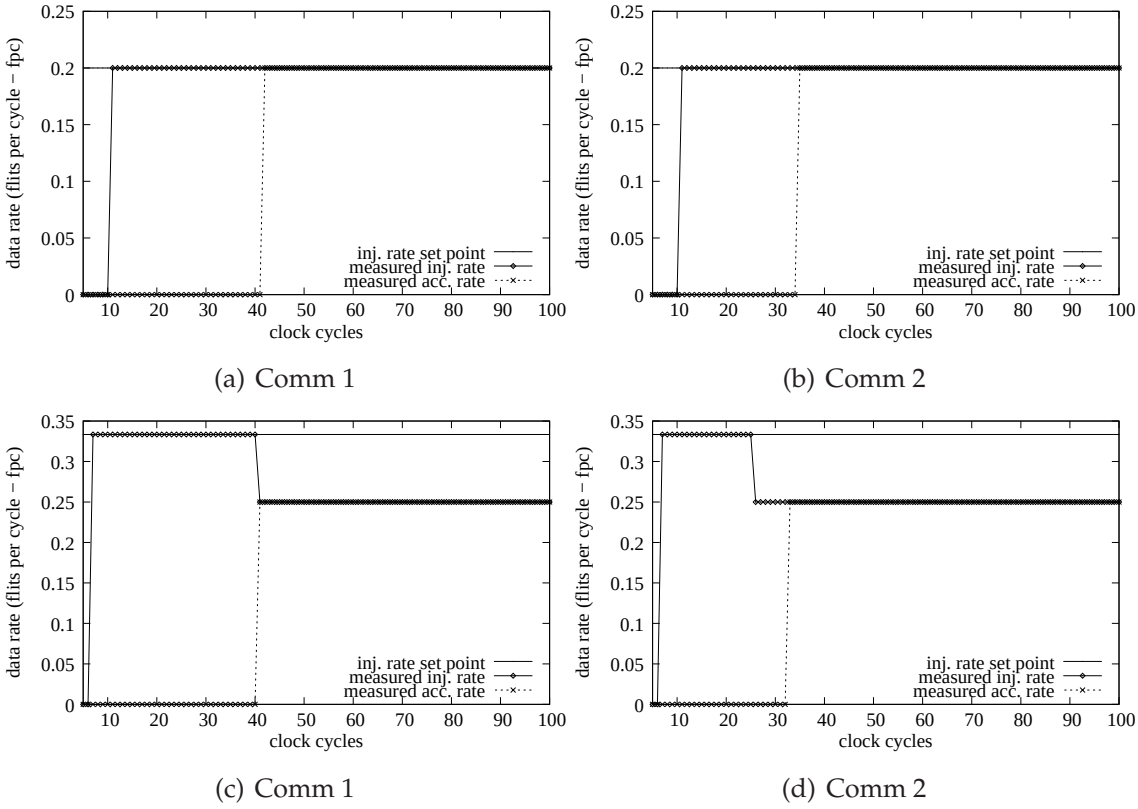


Fig. 4.8: Latency and bandwidth measurements in bit complement traffic scenario.

flits to 10000 flit per data producer node. The average transfer latency also increases linearly when the total number of injected flits is increased in this scenario even when the NoC is saturated. This behavior is unique compared with the traditional wormhole switching method because of the link sharing and flit interleaving capability as well as the mechanism to control dynamically the injection rate when the NoC is saturated.

Fig. 4.8(b) and Fig. 4.8(d) show the average latency and average actual bandwidth respectively over different requested bandwidth rates when 10000 number of flits per message are injected from every data producer node. By using static XY routing, there is a saturated latency start, when the requested bandwidth rates increases from a starting value of $1000\ MB/s$ ($0.25\ fpc$). While, by using adaptive West-First routing, the latency start being saturated, when the requested bandwidth rates increases starting from $666.67\ MB/s$ ($0.1667\ fpc$). Due to the existing mechanism which dynamically control the injection rates, the term *expected/requested bandwidth rate* or *injection rate setpoint* is different from the *actual/measured injection rate*. The former is assumed constant while the latter changes in accordance with the NoC saturation condition. Therefore, in the saturation condition, the injection rate at a source node as well as the acceptance rate at its target node changes dynamically to a certain stable rate or swings around a fixed acceptable rate.

Fig. 4.9(a) and Fig. 4.9(b) present the transient response observation/measurement of the injection and acceptance rate of two selected communication pairs, i.e. $Com\ 1$ and



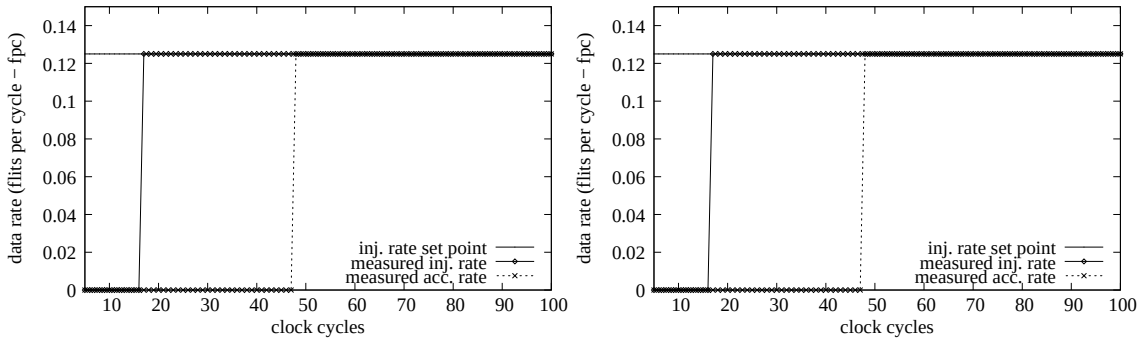
(a) Comm 1 (b) Comm 2

(c) Comm 1 (d) Comm 2

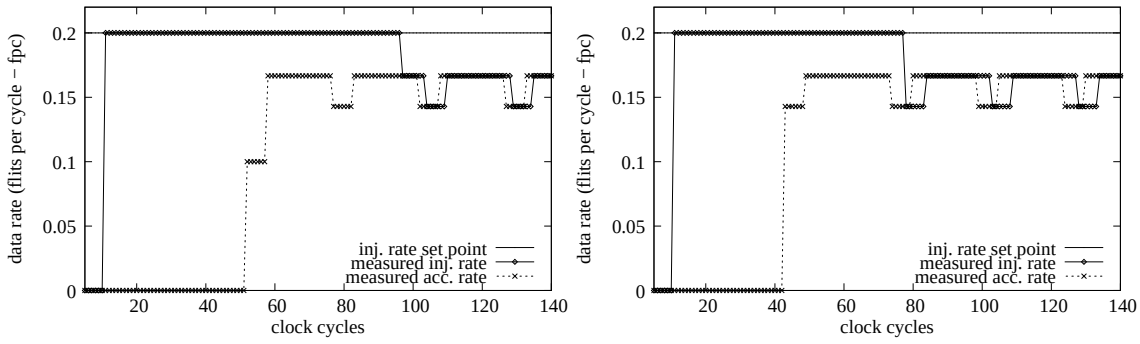
Fig. 4.9: Measurement of the actual injection and acceptance rate at two selected communication pairs using static XY routing.

$Com\ 1$ respectively by using the static XY routing algorithm. $Com\ 1$ is the communication edge from node (0,0) to node (3,3), while $Com\ 2$ is the communication edge from node (2,3) to node (1,0). As presented in the figure, the injection setpoint is $0.2\ fpc$ or similar to $800\ MB/s$. If we check again the NoC latency and bandwidth behaviors over different required bandwidth rate depicted in Fig. 4.8(b) and Fig. 4.8(d), then we can see that the NoC is not yet saturated when messages are injected with bandwidth rate of $800\ MB/s$ while using static XY routing. Hence, the injection and acceptance rates will simply follow the injection rate set point. Meanwhile, if the messages are injected with $0.333\ fpc$ or similar to $1333.33\ MB/s$, then according to Fig. 4.8(b) by using static routing, this data rate will make the NoC being in saturated condition. Therefore, as presented in Fig. 4.9(c) and Fig. 4.9(d), the injection and acceptance rates of the $Com\ 1$ and $Com\ 2$ are stable at $0.25\ fpc$ point or lower than the requested injection rate setpoint.

Fig. 4.10(a) and Fig. 4.10(b) also present the same non-saturated condition when using the adaptive West-First routing algorithm. As presented in figure, the requested injection rate setpoint is $0.125\ fpc$ or $500\ MB/s$. In accordance with Fig. 4.8(b) and Fig. 4.8(d), at $500\ MB/s$ requested bandwidth rate, the NoC is not yet saturated when using adaptive West-First routing. Hence, both the injection and acceptance rates of the $Com\ 1$ and $Com\ 2$ will be stable at $0.125\ fpc$. However, if the requested communication rate setpoint is 0.2 or $800\ MB/s$ as presented Fig. 4.10(c) and Fig. 4.10(d), then the NoC is saturated.



(a) Comm 1 (b) Comm 2



(c) Comm 1 (d) Comm 2

Fig. 4.10: Measurement of the actual injection and acceptance rate at two selected communication pairs using minimal adaptive West-First routing.

Therefore, the average injection and acceptance rates will be lower than the requested communication bandwidth. At initial clock cycles, the injection rate follows the requested injection rate. However, the injection rate at the source node follows the actual measured acceptance rate at the target node and fluctuates within a fixed average rate towards the end.

4.3.2 Hotspot Traffic Scenario

In this subsection, the NoC performance is simulated under hotspot traffic pattern, in which all nodes send a message to a single hotspot node, i.e. node (3,3). So, this node will receive all messages from all other 15 source nodes. Hence, there are 15 communication pairs in this scenario.

Fig. 4.11(a) and Fig. 4.11(c) show the NoC average latency and bandwidth behaviors over variable numbers of injected flits per data producer node and with different injection rate for the hotspot traffic scenario. If the messages on each source node are injected with lower injection rate, then the NoC will be not saturated. In the non-saturated conditions, the performance of the NoC prototypes with the static XY and adaptive West-First routing algorithms will be similar.

Fig. 4.11(c) and Fig. 4.11(d) present the NoC average latency and bandwidth responses

over different requested injection or bandwidth rates. There is a different NoC characteristic presented in Fig. 4.11(d), when the total number of injected flits per source node is different. For instance, when we select the total number of 500 and 10000 flits per data producer node. If the number of injected flits is 500 flits, then the average bandwidth starts at a saturated bandwidth of $133.33\ MHz$. We can observe that the number of messages sharing the local output port of the target node (3,3) is 15 messages. Since the maximum capacity of the outgoing port is $2000\ MHz$, then the average actual/measured bandwidth is $\frac{2000}{15} = 133.33\ MHz$. However, if the number of injected flits is 10000 flits per producer node, then the saturation point moves to a higher rate. Based on our observation in the our cycle-accurate RTL simulation, the last flits of some producer nodes located nearby the target node (3,3) are accepted early, while the other producer nodes far from the target node (3,3) are still injecting their payload flit. Therefore, the curves presented in Fig. 4.11(c) and Fig. 4.11(d) will be exponentially reduced from the $133.33\ MHz$ point until they reach the bandwidth saturation point. We call the the area in the exponentially reduced curves as the exponential region.

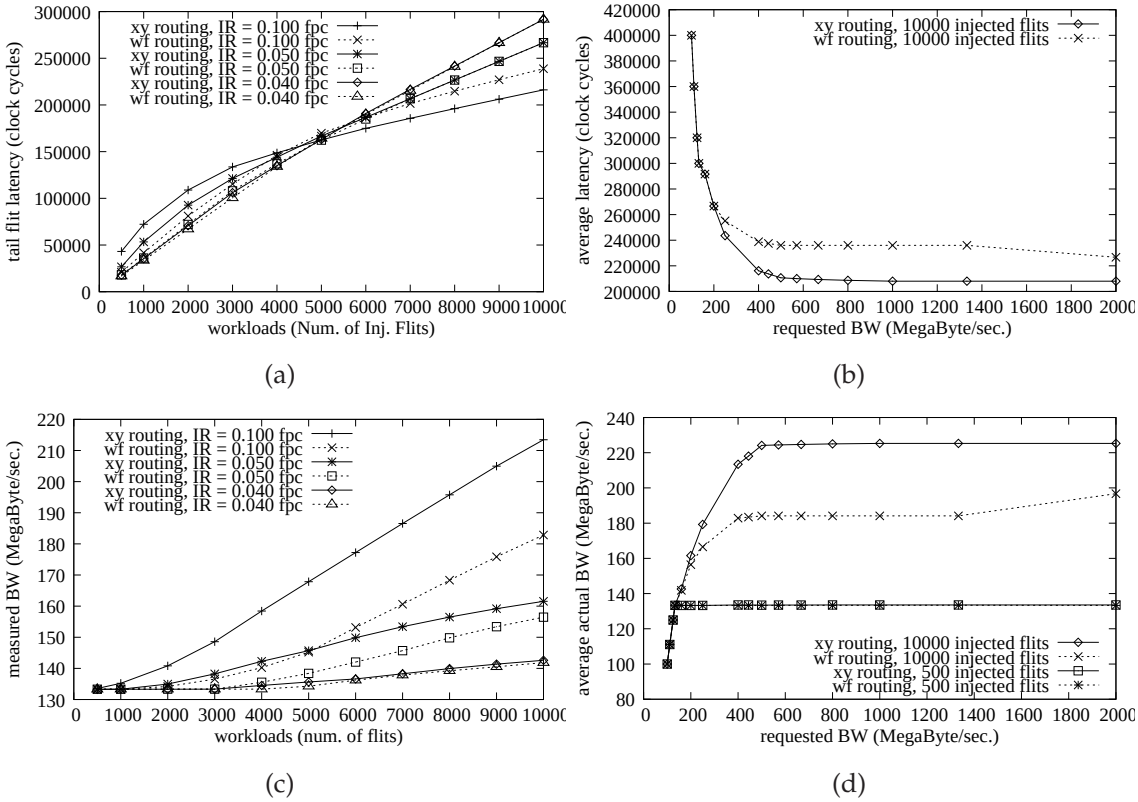
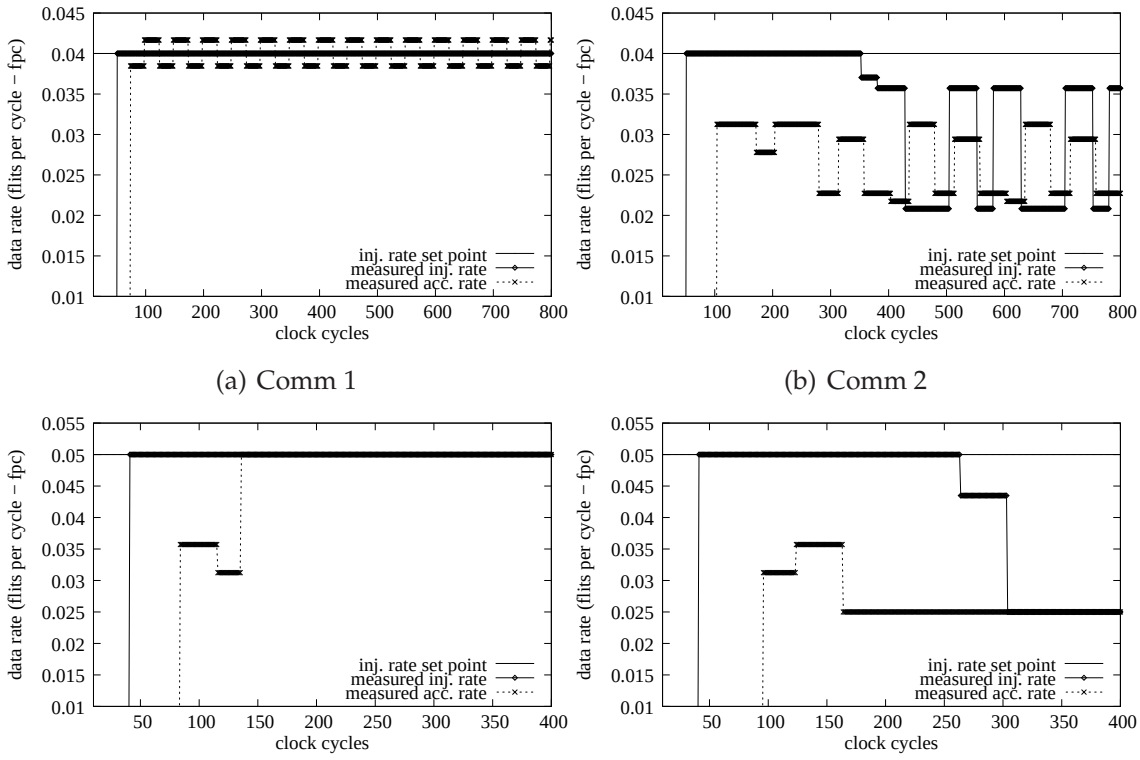


Fig. 4.11: Latency and bandwidth measurements in hotspot traffic scenario.

Fig. 4.12(a) and Fig. 4.12(b) present the injection rate and acceptance rates of two selected communication pairs in the hotspot traffic scenario by using the static XY routing algorithm. $Com\ 1$ is a communication edge that is transferring data from node (2,2) to node (3,3), while $Com\ 2$ is a communication edge that is transferring data from node (1,1) to node (3,3). The requested injected setpoints from the source nodes are $0.04\ fpc$ or simi-

lar to $160\ MB/s$. According to Fig. 4.11(b) and Fig. 4.11(d), this requested bandwidth will make the NoC be in the exponential region. Therefore, as presented in the Fig. 4.12(a) the injection rate of $Com\ 1$ can follow the expected injection rate setpoint, while its acceptance rate in the target node swings around the expected injection rate setpoint. Fig. 4.12(b) shows the transient responses of the injection and acceptance rate of the $Com\ 2$. It looks that the rates are reduced and fluctuates within certain lower rates than the expected rate.

Fig. 4.12(c) and Fig. 4.12(d) shows the other transient responses by using adaptive West-First routing in which the expected injection rate is $0.05\ fpc$ or similar to $200\ MB/s$. We can see that the injection and acceptance rates of $Com\ 1$ as shown in Fig. 4.12(c) will be stable at the expected rate. While the injection and acceptance rates of $Com\ 2$ as shown in Fig. 4.12(d) will be reduced to about $0.025\ fpc$. The expected $200\ MB/s$ data rate makes the NoC also be in the exponential region according to Fig. 4.11(b) and Fig. 4.11(d).



(a) Comm 1 (b) Comm 2

(c) Comm 1 (d) Comm 2

Fig. 4.12: Measurement of the actual injection and acceptance rate at two selected communication pairs using static XY and minimal adaptive West-First routing.

4.3.3 Matrix Transpose Traffic Scenario

Fig. 4.13(a) and Fig. 4.13(b) show the NoC average latency and bandwidth behaviors over variable numbers of injected flits per data producer node and with different injection rate for the matrix transpose traffic scenario. If the messages on each source node are injected with lower injection rate, then the NoC will not be saturated. In the non-saturated

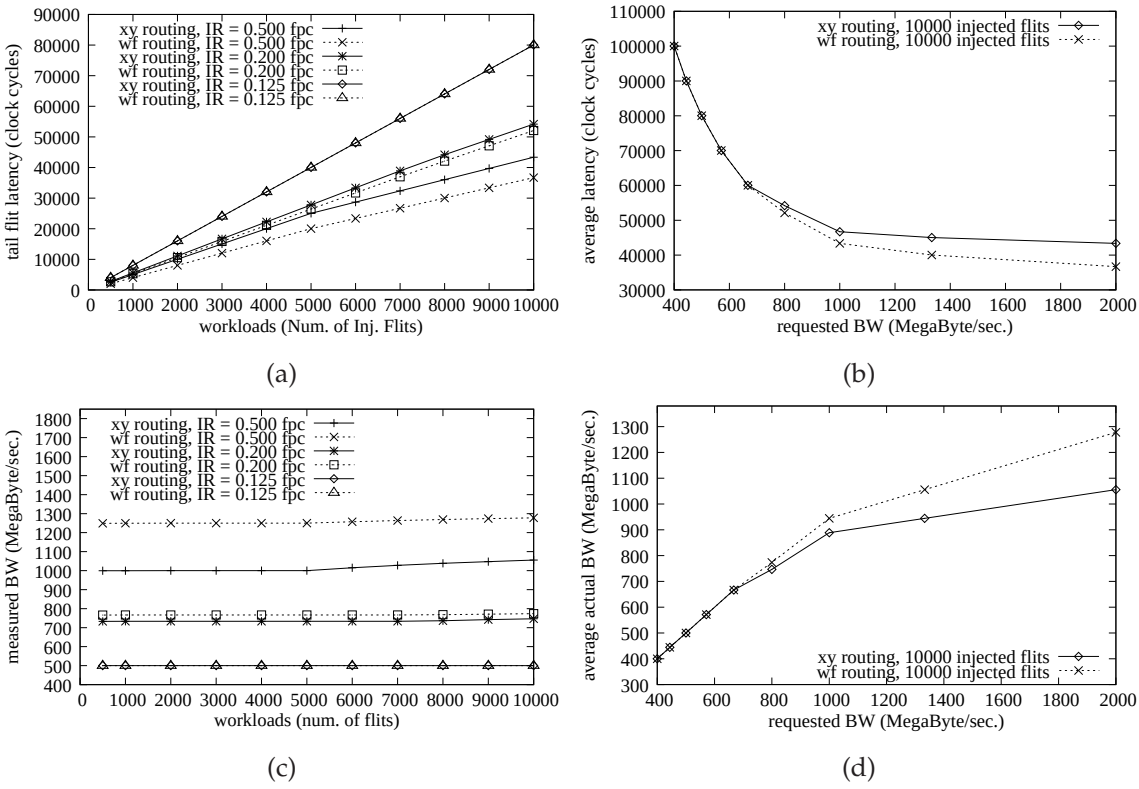


Fig. 4.13: Latency and actual bandwidth measurements in transpose traffic scenario.

conditions, the performance of the NoC prototypes with the static XY and adaptive West-First routing algorithms will be similar.

Fig. 4.13(b) and Fig. 4.13(d) show the average latency and average actual measured bandwidth respectively over different requested bandwidth rates when 10000 number of flits per message are injected from every data producer node. As presented in the figure, the performance of the NoC prototypes are similar for the static XY and adaptive West-First routing algorithms, when the data are injected less than about $666\ MB/s$ ($0.333\ fpc$). If the injected rates are increased faster than the abovementioned value, the performance of the NoC prototype with the adaptive West-First routing algorithm is better than the static XY routing algorithm.

4.3.4 Perfect Shuffle Traffic Scenario

Fig. 4.14(a) and Fig. 4.14(b) exhibit the NoC average latency and bandwidth behaviors over variable numbers of injected flits per data producer node and with different injection rate for the perfect-shuffle traffic scenario. In the perfect shuffle data distribution scenario, the performance of the NoC prototypes with the static XY and adaptive West-First routing algorithms are similar both in the saturated and non-saturated conditions.

Fig. 4.14(b) and Fig. 4.14(d) show the average latency and average actual measured bandwidth, respectively over different requested bandwidth rates when 10000 numbers

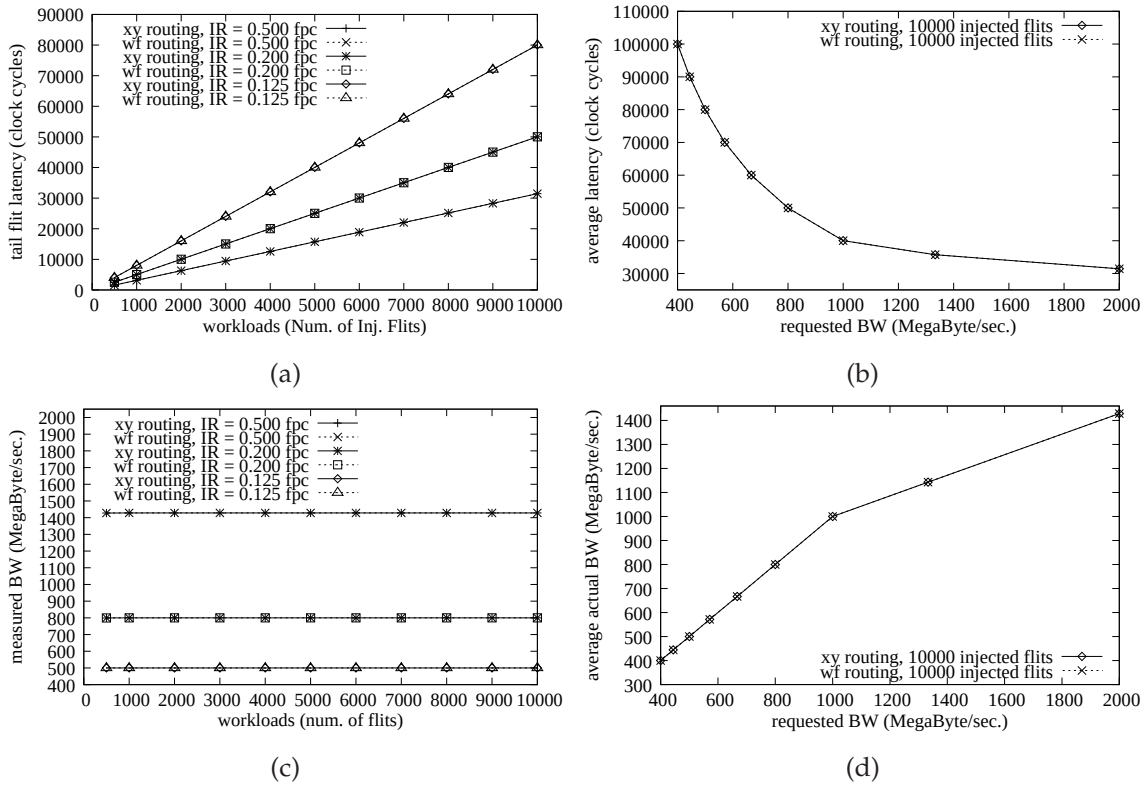


Fig. 4.14: Latency and actual bandwidth measurements in perfect shuffle (1-bit left-rotate) traffic scenario.

of flits per message are injected from every data producer node. As presented in the figure, the performance of the NoC prototypes are similar for the static XY and adaptive West-First routing algorithms. This performance characteristic is due to a situation that the messages are routed to similar paths when the static XY or adaptive West-First routing algorithm is used.

4.3.5 Bit Reversal Traffic Scenario

Fig. 4.15(a) and Fig. 4.15(b) show the NoC average latency and bandwidth behaviors over variable numbers of injected flits per data producer node and with different injection rate for the bit reversal data distribution scenario. Similar to the previous results, if the messages on each source node are injected with lower injection rate, then the NoC will not be saturated. In the non-saturated conditions, the performance of the NoC prototypes with the static XY and adaptive West-First routing algorithms will be similar.

Fig. 4.15(b) and Fig. 4.15(d) present the average latency and average actual measured bandwidth respectively over different requested bandwidth rates when 10000 number of flits per message are injected from every data producer node. As presented in the figure, the performance of the NoC prototypes are similar for the static XY and adaptive West-First routing algorithms, when the data are injected less than $666\ MB/s$ ($0.333\ fpc$). If the injected rates are increased faster than the abovementioned value, the performance of the

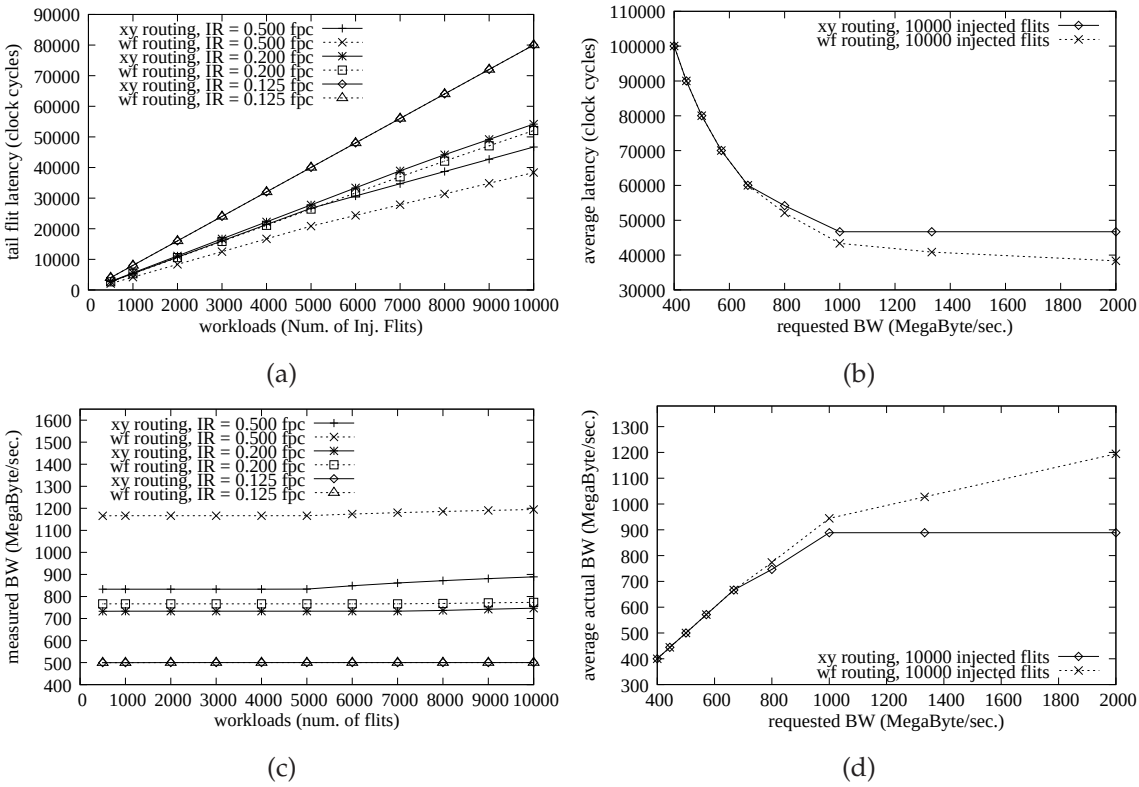


Fig. 4.15: Latency and actual bandwidth measurements in bit reversal traffic scenario.

NoC prototype with the static XY routing algorithm is better than the other one with the adaptive West-First routing algorithm. It looks from Fig. 4.15(b) and Fig. 4.15(d), the NoC with the static XY routing algorithm is saturated when the data are injected faster than $1000\ MB/s$ ($0.25\ fpc$).

4.3.6 Qualitative Comparisons with Traditional Wormhole Switching

Sections 4.3.1, Sections 4.3.2, Sections 4.3.3, Sections 4.3.4 and Sections 4.3.5 have presented the XHiNoC performance over different commonly-used data distribution scenarios. The performance evaluation results presented in this chapter is lacks of a direct performance comparisons with the traditional wormhole switching method. This section will explained qualitatively, the differences of the performance characteristics between the flit-level interleaving wormhole switching method and the traditional wormhole switching method.

A performance measurement result can be depicted as a 2D graph diagram showing the average latency changes over incremental changes of data injection rates at the source node. The latency metric can be represented as the number of clock cycle completely accepting a packet or to accept the last flit of a wormhole packet. The injection rate represents the speed of data injection that can be measured as the number of injected flits per cycle, the number of injected flits per second, or byte per second, or bit per second

Tab. 4.1: The last flit acceptance (in $clock\ cycle\ period$) and average bandwidth (in $fpc/flit\ per\ cycle$) measurements for $Comm\ 1$, $Comm\ 2$ and $Comm\ 3$ with different FIFO queue depths under transpose scenario.

Communication pair	Comm 1			Comm 2			Comm 3		
FIFO Queue Depth	2	4	8	2	4	8	2	4	8
Acceptance of the 500th flit	2011	2011	2011	2021	2025	2033	3025	3025	3025
Average bandwidth (fpc)	0.2486	0.2486	0.2486	0.2474	0.2469	0.2459	0.1653	0.1653	0.1653
Acceptance of the 1000th flit	4011	4011	4011	4021	4025	4033	6025	6025	6025
Average bandwidth (fpc)	0.2493	0.2493	0.2493	0.2487	0.2484	0.2480	0.1660	0.1660	0.1660
Acceptance of the 1500th flit	6011	6011	6011	6021	6025	6033	9025	9025	9025
Average bandwidth (fpc)	0.2495	0.2495	0.2495	0.2491	0.2490	0.2486	0.1662	0.1662	0.1662
Acceptance of the 2000th flit	8011	8011	8011	8021	8025	8033	12025	12025	12025
Average bandwidth (fpc)	0.2497	0.2497	0.2497	0.2493	0.2492	0.2490	0.1663	0.1663	0.1663

(bps). In the traditional wormhole switching method, when the injection rates of packets at every source node are increased, then the average latency will increase linearly by using some data distribution scenarios. When the injection rate is further increased, the average latency will increase exponentially. But in the flit-level message interleaving wormhole switching method, the average latency will reduce when the packet injection rates are increased during non-saturating conditions.

In general, the NoC will be saturated at different saturating points when any or some data producer nodes inject data to the NoC in such a way that the injected data traffic compete each other to share the same communication channel, where the total number of the expected rates of the competing traffics nodes exceeds the maximum bandwidth capacity of the shared channel. Due to implementation of the link-level flit flow control between the NoC routers as well as between the NoC local port and the network interface (NI) port, the congestion situation will trace back to the source node. The data injection rates at the source node are then automatically controlled to avoid input data overflows during saturating conditions, i.e. the NI will not inject a new wormhole data flit until a free space is free in the FIFO queue at the local input port. Hence, lossless data transmissions in the network are guaranteed.

In the non-saturating conditions, where the expected data injection rates are slow in such a way that the data rates in the NoC do not exceed the maximum NoC link bandwidth capacity, the injection and acceptance rates move to fixed steady state points. In this non-saturating condition, the latency and bandwidth of each considered point-to-point communication pair will be always fixed even if the workload (data burst) sizes at each data producer node are increased.

The simulation results presented in this chapter are only made without considering the size of the mesh interconnection network. The NoC performance under different network sizes and different routing algorithms could be different, but the main objective

Tab. 4.2: The last flit acceptance (in $clock\ cycle\ period$) and average bandwidth (in $fpc/flit\ per\ cycle$) measurements for $Comm\ 4$, $Comm\ 5$ and $Comm\ 6$ with different FIFO queue depths under transpose scenario.

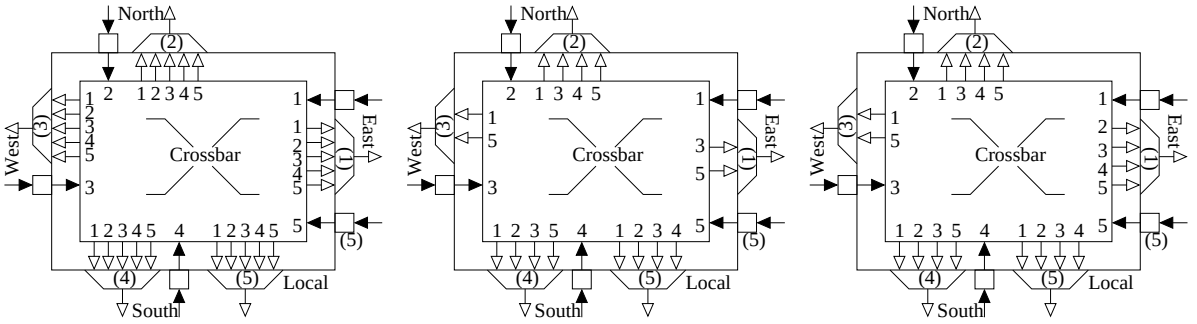
Communication pair	Comm 4			Comm 5			Comm 6		
FIFO Queue Depth	2	4	8	2	4	8	2	4	8
Acceptance of the 500th flit	2011	2011	2011	2019	2019	2019	1013	1013	1013
Average bandwidth (fpc)	0.2486	0.2486	0.2486	0.2476	0.2476	0.2476	0.4936	0.4936	0.4936
Acceptance of the 1000th flit	4011	4011	4011	4019	4019	4019	2013	2013	2013
Average bandwidth (fpc)	0.2493	0.2493	0.2493	0.2488	0.2488	0.2488	0.4968	0.4968	0.4968
Acceptance of the 1500th flit	6011	6011	6011	6019	6019	6019	3013	3013	3013
Average bandwidth (fpc)	0.2495	0.2495	0.2495	0.2492	0.2492	0.2492	0.4978	0.4978	0.4978
Acceptance of the 2000th flit	8011	8011	8011	8019	8019	8019	4013	4013	4013
Average bandwidth (fpc)	0.2496	0.2496	0.2496	0.2494	0.2494	0.2494	0.4983	0.4983	0.4983

of the simulation is not to present such difference. The main objective is to present the characteristics of the proposed novel wormhole switching method during saturating and non-saturating conditions.

4.3.7 Queue-Depth-Insensitive Performance Behavior

In this section, the throughput and latency of the XHiNoC with the novel wormhole switching method for different sizes of FIFO buffers is evaluated. The size of the FIFO buffer is also called the depth of the FIFO buffer, i.e. the maximum number of data that can be buffered in the FIFO queue. A FIFO queue having the depth of M will have M number of registers to buffer data. A matrix transpose benchmark is used to evaluate the NoC performance with different sizes of the FIFO buffers. A networked processing unit (NPU) at node (i, j) will send N number of flits (a message) to another NPU at node (j, i) (like matrix transpose operation). The message is then routed using the static XY routing algorithm.

Six internode data communication pairs are established in this scenario. An inter processor core data communication is a pair of NPU sending a message to a NPU receiving the sent message. Communication 1 (*Comm 1*) is a communication pair between node (1,0) as the data sender and (0,1) as the data acceptor. The communication pairs are represented as $Comm\ 1|(1,0) \Leftrightarrow (0,1)$. The rest of the communication pairs are represented as $Comm\ 2|(2,0) \Leftrightarrow (0,2)$, $Comm\ 3|(3,0) \Leftrightarrow (0,3)$, $Comm\ 4|(2,1) \Leftrightarrow (1,2)$, $Comm\ 5|(3,1) \Leftrightarrow (1,3)$ and $Comm\ 6|(3,2) \Leftrightarrow (2,3)$. All communication node pairs are required to communicate data with their maximum throughputs. Hence, in this scenario, $Comm\ 1$, $Comm\ 2$ and $Comm\ 3$ will compete to share the same communication path, while $Comm\ 4$ and $Comm\ 5$ will compete to share the same communication path, and the $Comm\ 6$ will consume the maximum link bandwidth capacity of its communication path.



(a) Full IO interc. (b) Custom mesh XY (c) Custom mesh WF

Fig. 4.16: Crossbar switch structure for fully and customized IO interconnects.

The main objective of the simulation is to evaluate the effects of the FIFO buffer sizes (the depth of the FIFO buffer) on the NoC performance. Table 4.1 and Table 4.2 show the tail flit acceptance latency measured in *number of clock cycles* when the data producer nodes inject 500, 1000, 1500 and 2000 flits into the NoC. The average bandwidth is then analyzed as the total number of accepted flits over the number of clock cycle to accept the last flit, and measured in *number of flit per cycle (fpc) unit*. The measurements are made for every communication pair with the FIFO queue depth of 2, 4 and 8 registers. As presented in both tables, the NoC performance is generally less sensitive to the FIFO queue sizes. Only $Comm\ 2$ shows a very small variation in the tail flit acceptance latency, where the acceptance will be delayed for 4 clock cycle periods when the FIFO queue depth is changed from 2 to 4 registers, and will be delayed for 12 clock cycles periods when the FIFO queue depth is changed from 2 to 8 registers.

4.4 Design Customization for Area Optimization

In this section, the data paths and the control paths of the fully interconnected crossbar switch of the XHiNoC based are customized for router area optimization purpose. The customization technique has also been used by some existing NoCs such as ANoC [28], Xpipes NoC [30], SPIN [90] and DSPIN [181]. See again Fig. 3.6(a) for the architecture of the fully IO-interconnected crossbar of the XHiNoC. The IO data paths customization is made based on two aspects i.e., neglecting paths for backtrace routing and neglecting paths related to prohibited turn applied in the used routing algorithm.

This design customization will give a trade-off between area minimization and router architecture flexibility. The IO switch interconnect customization will reduce the logic area of routers but it reduces the design flexibility because if the routing algorithm is exchanged, then the crossbar interconnect of the router switch must be customized again to suite the used routing algorithm. This optimized architecture is suitable for embedded NoC-based MPSoC for consumer appliances that requires a compact router area, or in post-fabricated chip-level multiprocessor systems that are specially dedicated to utilize

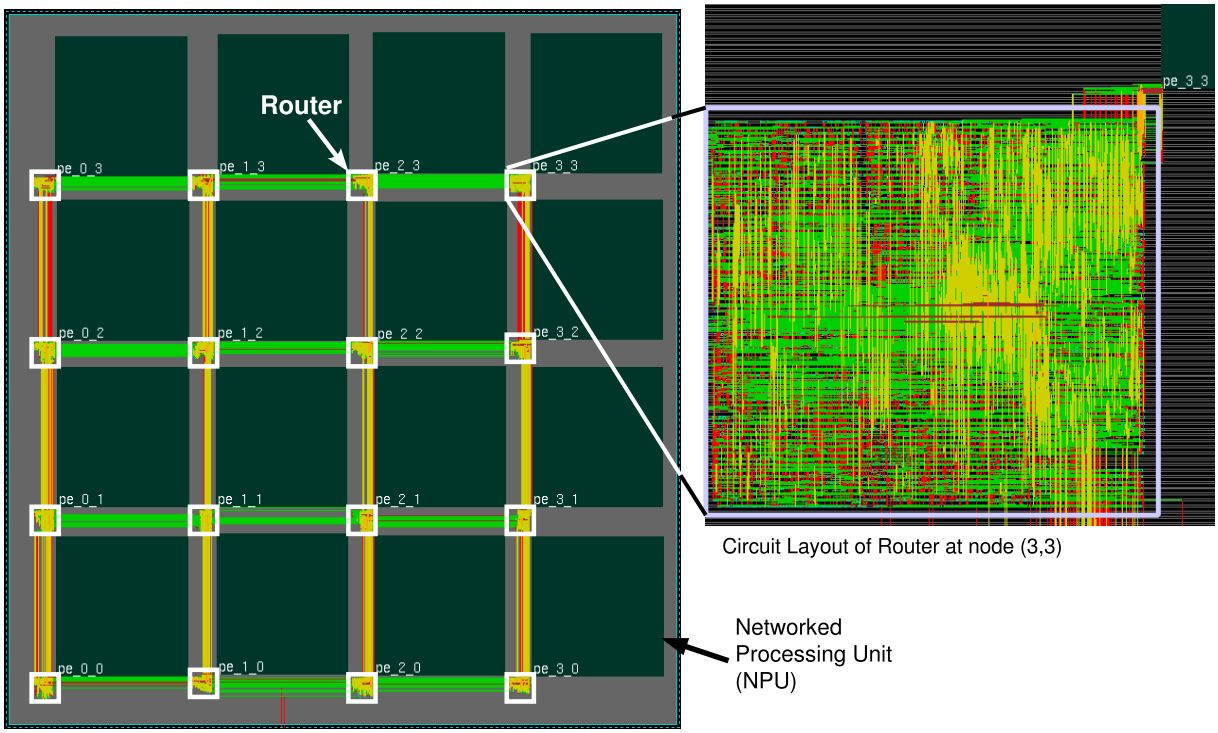


Fig. 4.17: Circuit layout of a multiprocessor system interconnected with XHiNoC routers using CMOS standard-cell technology library.

a fixed routing algorithm. Meanwhile, in the fully IO crossbar interconnect, the routing algorithm can be easily and simply exchanged without changing the data path and control path structures of the router. This architecture is suitable when the NoC will be implemented on a reconfigurable device such as a FPGA, where routing algorithm of the NoC is reconfigurable, or probably also in the post-fabricated NoC-based multiprocessor circuit, where the routing function is reconfigurable.

4.4.1 Neglecting Paths for Backtrace Routing

If we assume that our NoC will not make a backtrace routing, i.e. packets are routed in a opposite direction in the current node because of any reason, then some backtrace crossbar switch interconnects can be neglected. If we have N pairs of IO port, then we will have input ports $P_j, j \in \{1, 2, \cdots, N\}$ and output ports $P_k, k \in \{1, 2, \cdots, N\}$. Thus, the backtrace IO paths connecting port P_j and P_k such that $j = k$ can be removed from the crossbar switch interconnects. For N number of IO pairs, we can remove N number of the backtrace data paths and their related control paths from the router architecture.

If we see again the architecture of the XHiNoC in Fig. 3.6(a), then the data path d_i is removed from the input port of an MIM module in output $Port\ i$. As a consequence, the control paths $r_{i,i}$ and $a_{i,i}$ are removed from the port entities of all REB and $Arbiter$ modules.

4.4.2 Neglecting Paths related to Prohibited Turns

This customization is made based on the selected routing function in the router prototype by taking into account the prohibited turns in the turn models of the used routing function. For the sake of simplicity, we select the case for customizing router crossbar interconnect using the static routing algorithm.

In the static routing algorithm, the turns North–East, North–West, South–East and South–West are prohibited. We can set the Ports East, North, West, South and Local as the $Port\ 1$, $Port\ 2$, $Port\ 3$, $Port\ 4$ and $Port\ 5$, respectively. Based on the five-port generic architecture of the XHiNoC as presented in Fig. 3.6(a), then in both $Port\ 1$ and $Port\ 3$, the data path input d_2 and d_4 connected to the MIM modules in the both ports are removed. Consequently, the control paths $r_{2,1}$, $r_{2,3}$, $r_{4,1}$ and $r_{4,3}$ as well as the control paths $a_{2,1}$, $a_{2,3}$, $a_{4,1}$ and $a_{4,3}$ are removed from the port entities of the REB at $Port\ 2$ and $Port\ 4$ and from the port entities of the $Arbiter$ modules at $Port\ 1$, $Port\ 3$.

We can also generalize that, if turn from input $Port\ n$ to output $Port\ m$, where $n, m \in \{1, 2, \cdots, N\}$, is prohibited in the routing function, then data path input d_n from a routing engine (RE module) connected to the MIM module at $Port\ m$, as well as the control paths $r_{n,m}$ and $a_{n,m}$ can be removed from the crossbar interconnects of the switch. Section 4.5 will present the logic cell area efficiency to apply the customization of the router IO port interconnects. Fig. 4.16 presents the crossbar structure of the mesh switches with fully IO interconnects (Fig. 4.16(a)) and customized IO interconnects for mesh with XY routing algorithm (Fig. 4.16(b)) and for mesh with minimal adaptive WF routing algorithm (Fig. 4.16(c)). As presented in Fig. 4.16(b) data paths from North (5) input port to East (1) and West (3) output ports, as well as data paths from South (4) input port to East (1) and West (3) output ports are removed from the crossbar interconnects because by using the static XY routing algorithm, turns North–East, North–West, South–East and South–West are prohibited.

4.5 Synthesis Results

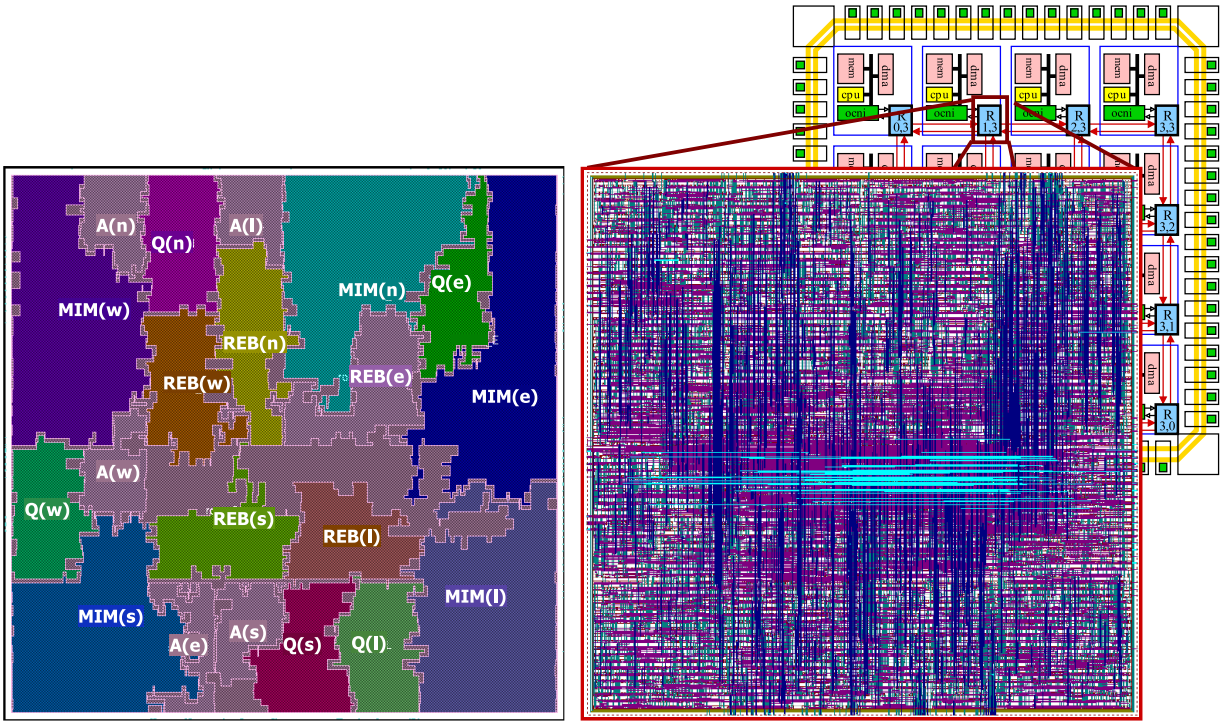
4.5.1 Synthesis with Fully and Custom Crossbar IO Interconnects

In this subsection, the XHiNoC router prototypes are synthesized using 130-nm CMOS standard-cell library from *Faraday technology Corporation*. The router is targeted to work with about $1.1\ GHz$ data frequency (or $0.9\ ns$ clock period).

Table 4.3 presents the synthesis result for the NoC prototypes with the static XY routing algorithm which are designed with fully and customized IO-port crossbar switch interconnects, as well as the NoC with the adaptive West-First (WF) routing algorithm with fully IO crossbar switch interconnects. The table also shows in detail the efficiency for each component in the router when the crossbar interconnect in the switch router is cus-

Tab. 4.3: Synthesis Results of the router with flit-level interleaved wormhole switching method using 130-nm CMOS technology with targeted working frequency of about 1.1 GHz (0.9 ns clock period).

NoC Router Component	WF with Fully IO Intc. (mm^2)	XY with Fully IO Intc. (mm^2)	XY with Custom IO Intc. (mm^2)	Efficiency: Custom/Full IO with XY routing
FIFO buffer (E)	0.003419	0.003390	0.003274	3.4 %
FIFO buffer (L)	0.003343	0.003376	0.003276	2.9 %
FIFO buffer (N)	0.003375	0.003382	0.003254	3.9 %
FIFO buffer (S)	0.003357	0.003387	0.003250	4.0 %
FIFO buffer (W)	0.003343	0.003374	0.003263	3.3 %
Total 5 FIFO buffers	0.016837	0.016909	0.016317	3.5 %
% of Total Cell Area	14.87 %	16.0 %	19.3 %	
Arbiter (E)	0.001571	0.001517	0.000195	87.2 %
Arbiter (L)	0.001588	0.001518	0.000549	63.8 %
Arbiter (N)	0.001611	0.001518	0.000580	59.9 %
Arbiter (S)	0.001659	0.001445	0.000587	61.9 %
Arbiter (W)	0.001642	0.001469	0.000201	86.3 %
Total 5 Arbiters	0.008071	0.007489	0.002112	71.8 %
% of Total Cell Area	7.13 %	7.0 %	2.4 %	
MIM (E)	0.011644	0.010969	0.008345	23.9 %
MIM (L)	0.010895	0.011592	0.009996	13.8 %
MIM (N)	0.010951	0.011153	0.010139	9.1 %
MIM (S)	0.011518	0.011300	0.009873	11.1 %
MIM (W)	0.011464	0.011230	0.008433	24.9 %
Total 5 MIMs	0.056472	0.056224	0.046786	16.8 %
% of Total Cell Area	49.88 %	53.0 %	55.7 %	
REB (E)	0.006404	0.005162	0.004140	19.8 %
REB (L)	0.006325	0.005175	0.004391	15.1 %
REB (N)	0.006378	0.005157	0.002953	42.7 %
REB (S)	0.006316	0.005155	0.002953	42.7 %
REB (W)	0.006411	0.005161	0.004626	10.4 %
Total 5 REBs	0.031834	0.025810	0.019063	26.1 %
% of Total Cell Area	28.12 %	24.0 %	22.6 %	
Total Cell Area	0.113214	0.105877	0.083703	20.9 %



(a) Cell allocation view (b) Place & Route view

Fig. 4.18: Circuit layout of the router with XY routing algorithm (e=east, n=north, w=west, s=south, l=local, Q=FIFO queue, A=Arbiter).

tomized based on the static XY routing algorithm. The turn models applied to the static XY routing function is used to implement the necessary and unnecessary data wire and control wire lines.

As shown in Table 4.3, we can see the area of the ID-management unit in multiplexor unit (MIM) implementations. The area of the MIM component is about 3 times the area of the 2-depth FIFO. Theoretically, if we want to interleave 15 messages by using VC approach where the FIFO buffer is not shared by different messages, then 15 virtual channels must be implemented on each input or output port as well as 15 VC controller with 15 virtual channel ID on both input and output ports. Hence, our proposed architecture with the local ID-Management unit is much more efficient than the VC approach.

From Table 4.3, we can see that, in term of the percentage, higher efficiency is obtained from the $Arbiter$ units, i.e. about 71.8%. But related to the logic cell area, higher efficiency is achieved from the MIM module, i.e. about $0.01\ mm^2$ (obtained from $0.056224\ mm^2 - 0.046786\ mm^2$). In general, the NoC prototype with the custom crossbar interconnects gives about 21% logic cell area efficiency over the NoC prototype with fully crossbar interconnects in cases when the customization is made for NoC using the static XY routing algorithm. FIFO buffer units do not give a significant efficiency because the customization method does not involve directly data paths and control paths of the FIFO queues.

We have also prototyped the circuit layout of a 2D 4×4 mesh NoC using XHiNoC

routers as presented in Fig. 4.17. Each XHiNoC router is connected with a networked processing unit (NPU), which is assigned as a block with empty logic cells (black box). The circuit layout is made using a *Silicon Encounter* tool from *Cadence* with 180-nm CMOS standard-cell technology from *UMC*. The right-side of the figure also shows the enlarged view of the logic cell placement and wire routing of the router at node (3,3). Fig. 4.18 shows the circuit layout result of a single switch component. The allocations of all modular cells in the switch is presented in Fig. 4.18(a). Fig. 4.18(b) shows the cell placement and wires route results of the switch.

By using 130-nm CMOS standard-cell library, the total cell area of our customized NoC router is about 0.084 mm^2 (32-bit data + 7 control bits). We compared the area of our NoC with other currently developed NoC. Although we realize that the logic area reports may not be fairly comparable because of the differences of the design parameters such as buffer sizes, the use of virtual channels (VCs) and the flit-width, the least we can see is the impact of the use the wormhole cut-through switching implementation, which does not require virtual channels. In addtion, the depth of the FIFO queue in each input port can be minimize to 2 registers. Hence, the area cost of our on-chip router is reduced. Now we can see the comparisons as follows.

The TRIPS NoC [87] that uses VCs, contains two data networks, the OPN and the OCN, in which the logic areas of the OCN and OPN routers are 1.10 mm^2 and 0.43 mm^2, respectively by using 130-nm technology. The TRIP NoC clock frequency is about 366 MHz. The Xpipes NoC [30] (without specifically describing whether VCs are used or not) has a logic area of 0.19 mm^2 at 800 MHz-implementation with 4-IO-port and 64-bit-flit router implementation using 130-nm technology. The larger area of the TRIPS NoC is due to the use of virtual channels, where four virtual channels per input port are implemented in the TRIPS router. The depth of the FIFO buffer in each channel is two flits.

Teraflops [98] NoC router that uses a double-pumped crossbar switch to reduce the routing area has a compact 0.34 mm^2 router area using 65-nm technology (32-bit data + 6 control bits). For various voltage levels ranging from 0.75 V until 1.2 V, Teraflops NoC router can be clocked from 1.7 GHz to 5.1 GHz respectively. The SCC [103] NoC router synthesis result by using 65-nm standard-cell library is 0.097 mm^2 (32-bit flow control digits/flits) with 250 MHz working frequency. The Teraflops NoC and SCC NoC do not use VCs. Specifically, SCC NoC does not use VCs because they increase the total buffer counts and results in power consumption that would exceed the SCC NoC's target constraints [103]. With similar motivation, we also do not implement virtual channels in our NoC to save area and power dissipation as well as to characterize specifically how our NoC can solve the head-of-line blocking problem without the use of VCs.

In the data output stage, we combine the ID management unit stage into crossbar data multiplexing stage. As a result, the maximum data frequency of the current VLSI architecture can be increased from $472\ MHz$ to $1.1\ GHz$ (increase about $2.3\times$ or more than 100% speed overhead). By using our current VLSI architecture, the critical path of our previous on-chip router that is located in the routing stage has been cut to increase the maximum

Tab. 4.4: Gate-level synthesis of the wormhole-switched router using 130-nm CMOS technology with 1.0 GHz target frequency (16 ID slots per link) for different FIFO buffer sizes (Queue-Depth).

FIFO Depth	2	4	8
Total logic cell area (mm^2)	0.102	0.119	0.152
Critical path (ns)	0.94	0.95	0.95
Est. power dissipation (mW)	54.575	65.674	87.572
FIFO cell area (mm^2)	0.0164	0.0326	0.0656
% of Total logic cell area	16 %	28 %	43 %

data frequency of the on-chip router. The critical path in the current architecture is now on the Multiplexor with ID Management Unit (MIM Component).

By using the NoC router static XY routing and with the fully IO port interconnects, the estimation of the dynamic power, i.e the net switching power and cell internal power of the NoC router is about $15.99\ mW$ and $44.25\ mW$, respectively. Its leakage power is estimated about $21.5\ \mu W$. While the net switching power, cell internal power of the NoC router with minimal adaptive WF routing algorithm with fully IO port interconnects is estimated about $17.99\ mW$ and $47.15\ mW$ respectively. Its leakage power is estimated to be about $22.2\ \mu W$. By using the NoC router with static XY routing algorithm and with optimized/customized IO port crossbar interconnects, the net switching power, cell internal power and leakage power is estimated $13.45\ mW$, $38.70\ mW$ and $15.4\ \mu W$, respectively.

4.5.2 Synthesis with Different FIFO Queue Depths

In this subsection, the XHiNoC router prototypes are synthesized using 130-nm CMOS standard-cell library from *Faraday technology Corporation*. The router is targeted to work with about $1.0\ GHz$ data frequency (or $1.0\ ns$ clock cycle period). Table 4.4 shows the impact of the depth of the FIFO buffer on estimated power dissipation and total logic cell area. The cell area increases about 16% if the depth is increased two times (from 2 to 4). If the FIFO depth is increased 4 times (from 2 to 8), then a 49% cell area overhead is obtained. The area contribution of the 8-depth FIFO buffer is almost half of the total logic cell of the router. As presented in the Table 4.4, with the depth of 2, the area contribution of the FIFO buffer over total cell area of the router is only about 16%. It looks like the large depth of the FIFO buffer will give a significant contribution of the total area of a NoC router.

The critical paths of the NoC router prototypes with different sizes of the FIFO buffer are shown also in Table 4.4. The NoC router prototypes are synthesized with target data (working) frequency of 1 GHz (clock period of 1 ns). It seems that the critical path is almost independent from the depth of the FIFO buffer. The critical paths with 1 GHz target frequency of the router prototypes with 2, 4 and 8 FIFO registers vary only between 0.94–0.95 ns. The critical path of the NoC router itself is not located in the FIFO buffer component.

Tab. 4.5: Gate-level synthesis of the wormhole-switched router using 130-nm CMOS technology (2-depth FIFO buffer) for different number of available ID slots per link.

Num. of ID slots per link	8	16	32
Total logic cell area (mm^2)	0.073585	0.105877	0.174709
Working clock cycle period (Data frequency)	0.9 ns (1.1 GHz)	0.9 ns (1.1 GHz)	1.0 ns (1.0 GHz)
Critical path (ns)	0.82	0.82	0.95
Est. power dissipation (mW)	45.872	60.255	83.507
REB cell area (mm^2)	0.018356	0.025810	0.041581
% of Total logic cell area	24.9 %	24.0 %	23.9 %
MIM cell area (mm^2)	0.031389	0.056224	0.108402
% of Total logic cell area	42.7 %	53.0 %	62.0 %

4.5.3 Synthesis with Different Number of Available ID Slots

The wormhole-switched XHiNoC router with fully crossbar IO interconnects and with static XY routing algorithm has been synthesized for different number of available ID slots per data communication link. Table 4.5 shows the synthesis results when the number of available ID slots per link are 8, 16 and 32 ID slots. The number of available ID slots can be controlled through a parameter in the VHDL package file at design design. Beside the total logic cell area, Table 4.5 presents also the impact of the amount of ID slot on the logic cell area of the REB and MIM components, as well as their logic are contributions to the total logic cell area. The changes of the ID slot availability parameter will directly affect two components in the XHiNoC router, i.e. the REB and the MIM components since both components contain an ID-based routing reservation table and ID slot table.

When the number of ID slots per link is increased two times (from 8 to 16 slots), the area overhead is about 43.88% or the logic cell area increases When the number of ID slots per link is about 1.44 times. increased four times (from 8 to 32 slots), the area overhead is about 137.42% or the area increases about 2.37 times. When the number of ID slots per link is increased two times (from 16 to 32 slots), then the area overhead is about 65.01% or 1.65 times. It also seems that the logic cell areas of the REB and MIM components increase as the number of ID slots per link is set larger.

When the number of ID slots per link is set to 8 or 16 slots, the XHiNoC router can be synthesized with a data frequency of 1.11 GHz. When the number of available ID slots is set to 32 ID slots, then the router cannot be synthesized to work at 1.11 GHz, but it can be synthesized at a 1.0 GHz working frequency. The table also presents the signal delay of the critical path of the router with different numbers of ID slots per link. According to the synthesis report files, the critical path of the router is in the MIM component. Therefore, the signal delay in the critical path of the router can be longer, when the number of available ID slots is increased.

Tab. 4.6: Synthesis of the wormhole-switched router with customized crossbar IO interconnects on a Xilinx FPGA device (Target device: Spartan3 xc3s4000).

	Utilization	Percentage of Total
Number of slice flip-flops	1247	2.0 %
Number of 4-input LUTs	3078	5.0 %
Number of occupied slices	1772	6.0 %
Minimum Delay	12.290 ns	
(Maximum Frequency)	(81.367 MHz)	

4.5.4 Synthesis on an FPGA Device

The XHiNoC wormhole-switched router with static XY routing algorithm and with the customized crossbar IO interconnects has been synthesized for an FPGA target device. The target device is a Spartan3 (*xc3s4000*) device from *Xilinx*. Table 4.6 shows the synthesis summary, which presents the utilization of the occupied slices, slice flip-flops and 4-input look-up tables (LUTs). As shown in the table, the maximum allowable working (data) frequency is 81.367 MHz, which in general is slower than the CMOS standard-cell technology implementation. It seems that the utilization of the slices on the target FPGA device of the wormhole-switched router with the customized IO interconnects is about 6% of the total slices.

4.6 Summary

This chapter has presented a new router architecture for NoC augmented with a novel mechanism able to extend the wormhole switching technique to support the interleaving of the flit belonging to different packets in the same communication link. The work presented in this chapter deals with a very important problem which limits the performance of the traditional wormhole-switched network, i.e. the head-of-line blocking problem. By using the proposed technique, the head-of-line blocking problem can be strongly attenuated, similar to another technique that uses virtual channels. But the virtual-channelles technique solves the problem without experimenting the large overhead of the virtual-channel-based router implementation due to the additional FIFOs, arbitration units and control logic units.

The main critic of the NoC router design with the wormhole switching capable of interleaving different messages at flit level is the implementation of two tables on every one-directional link that can lead to an area overhead. However, when a number of N messages is allowed in-flight (mixed) in the same physical link, then compared to a VC-based solution, our concept theoretically requires less logic area and power, since the size of the two tables having N slot registers will be less than the size of N number of VC buffers.

Chapter 5

Multicast Routing for Collective Communication Service

Contents

Services in terms of efficient routing and scheduling are critical with respect to the performance of NoC-based multicore processor systems. Historically, the first generation multicomputers only supported unicast communication (single PE sends a message to single PE unit). Nowadays, multicomputers have been developed towards collective communication services. The collective communication services include *one-to-many* communication such as *multicast* (the same message is sent from a source node to an arbitrary number of destination nodes), *one-to-all* communication such as *broadcast* (the same message is sent from a source node to all nodes (entries) in the network) and *scatter* (different messages are sent from a source node to all entries in the network), *many-to-one* communication (a destination node receives different messages from an arbitrary number of source nodes), and *all-to-one* communication such as *reduce* (a destination node combines different messages from an arbitrary number of source nodes by performing a certain operation such addition, multiplication, maximum, minimum, or a logical operation).

Among the aforementioned collective communications, the multicast and broadcast communications are the most interesting communication modes. Since both communications are not only required in many parallel algorithms and applications in multiprocessor system but also demand special attentions in the network communication protocol layers. With software implementation, a multicast message can be injected into the network by sending separate copies of the message from the source to every destination node (unicast-based multicast delivery). However, this approach is inefficient in terms of communication latency and energy.

The need for collection communication service including multicast routing in parallel computing and multicomputer-based applications is described in Section 5.1. The state of the art in multicast routing methodology and theory is presented in Section 5.2. The multicast routing can be divided generally into *path-based multicast routing* and *tree-based multicast routing*. The difference between both methods is described in Section 5.2.1. A new theory of a deadlock-free multicast routing is presented in this chapter. The theory is supported by a *simple and smart* multicast contention/conflict management callled *Hold-Release Multicast Tagging Mechanism* which is explored in Section 5.3.

Microarchitecture, components and implementation of the best-effort version of the deadlock-free multicast routers are presented in Section 5.4. The extended adaptive multicast router version is also explored in Section 5.5 together with an inefficient spanning tree (branches of tree) problem that probably occurs when using adaptive tree-based multicast routing algorithm. Therefore, a special technique that is directly implemented on the multicast routing engine of each NoC router is used to solve the inefficient spanning tree problem. The effectiveness of the proposed multicast routers with different routing algorithms is evaluated under a random mixed unicast-multicast traffic scenario as presented in Section 5.6. Three performance metrics are used to in the evaluation, i.e. tail flit transfer latency, message throughput and the number of traffics performed in the

NoC communication links. The synthesis results of the multicast routers using CMOS standard-cell technology are also presented in this chapter (Section 5.7). The work in this chapter is then summarized in Section 5.8.

5.1 The Need for Collective Communication

The multicast delivery service has been intensively used in large-scale multiprocessor systems, and has been a fundamental service of some data parallel computer languages. The following points present the need for multicast services in parallel computing and multicomputer applications that have been cited and well summarized in [137], [138] and in [110] from many works in the literature.

- In several parallel algorithms, e.g. parallel search algorithm [61] and parallel graph [126] algorithm, have made use of multicast communication. In the parallel algorithms, a set of independent computational processes is collectively run to find a global optimum state. When a computational process finds an optimal state, this information is sent to other processes that will be efficiently made by using multicast delivery.

- In parallel numerical algorithm involving a variety of linear algebra computations, multicast communication is used to perform some matrix-based operations [76] [109], such as matrix-vector and matrix-matrix multiplication, LU-factorization and Householder transformations.

- In single-instruction multiple-data (SIMD) machines, in which the same program is executed on different processors with different data in parallel, and in multiple-instruction multiple-data (MIMD) machines, multicast communication is an efficient operation, especially when long data streams will be sent concurrently to several processor cores.

- In a data parallel programming model designed by using data parallel programming languages, a variety of process control operations and global data movement such as *reduction*, *replication*, *permutation*, *segmented scan* and *barrier synchronization* require collective communication models. Specifically, the *replication* [150] and the *barrier synchronization* [214] are performed by using multicast data delivery.

- In a distributed shared-memory paradigm, multicast services may be used to efficiently support shared-data invalidation and updating [133], i.e. when a core change the value of a variable in its local cache, then this change must be informed to other cores in the system that also have the copy of the variable in their caches to maintain data consistency (cache coherence issue).

Recently, development of programming models for the NoC-based multiprocessor systems has been recently a hot topic in multicomputer research area. Ultimately, multicast communication service has been a standard service in data parallel programming languages such as *Fortran-D* [73], *Distributed Fortran 90* [155] and *High Performance Fortran (HPF)* [95]. Message passing libraries such as *Message Passing Interface (MPI)* [156] and *Parallel Virtual Machine (PVM)* [79], [80], which are commonly used to design message passing programming models, also includes some standard procedures to perform collective communications such multicasting and broadcasting. Both libraries have been developed for computer languages such as Fortran and C/C++.

Multicast communications in the programming models can be effectively and efficiently implemented in the application layer of the NoC-based multiprocessor systems, as long as hardware infrastructures in network, data-link and physical layers supports the multicast services. Indeed, the multicast support as one of the collective communication services can simplify the programming models and alleviate programming efforts for NoC-based multiprocessor systems.

In internet community, multicast data communication has been an interesting topic. The works presented in [71], [170] and in [112] have presented some protocols utilized to support a multicast data transmission. The work in [112] especially provides a reliable multicast communication by involving the use of multiple multicast channels for reducing receiver processing costs and reducing network bandwidth consumption in a multicast session. The works mentioned above are dedicated for off-chip networks not for on-chip network platform. However, the implemention of the multicast protocol in both different platforms has the same motivation, i.e. to reduce communication time and energy.

5.2 State-of-The-art in Multicast Routing Methodology and Theory

5.2.1 Path-based and Tree-based Multicast Routing Methods

Multicast messages can be routed in the network by using *path-based* or *tree-based* multicast routing method. Fig. 5.1 presents the different traffics formations by using the tree-based and path-based multicast routing. In the tree-based multicast routing, the traffics are formed like a tree, in which the destination nodes are located at each end-branch of the tree. The multicast tree tends to increase contention probability between multicast messages in the network, because each multicast message tree can acquire more than two sinking (output) ports in an intermediate node, which probably compete with other multicast message trees to acquire the same sinking ports. In order to eliminate such situation, the path-tree multicast routing is proposed. In the path-based multicasting, the message is guided in the network such that the all destination nodes are transited within

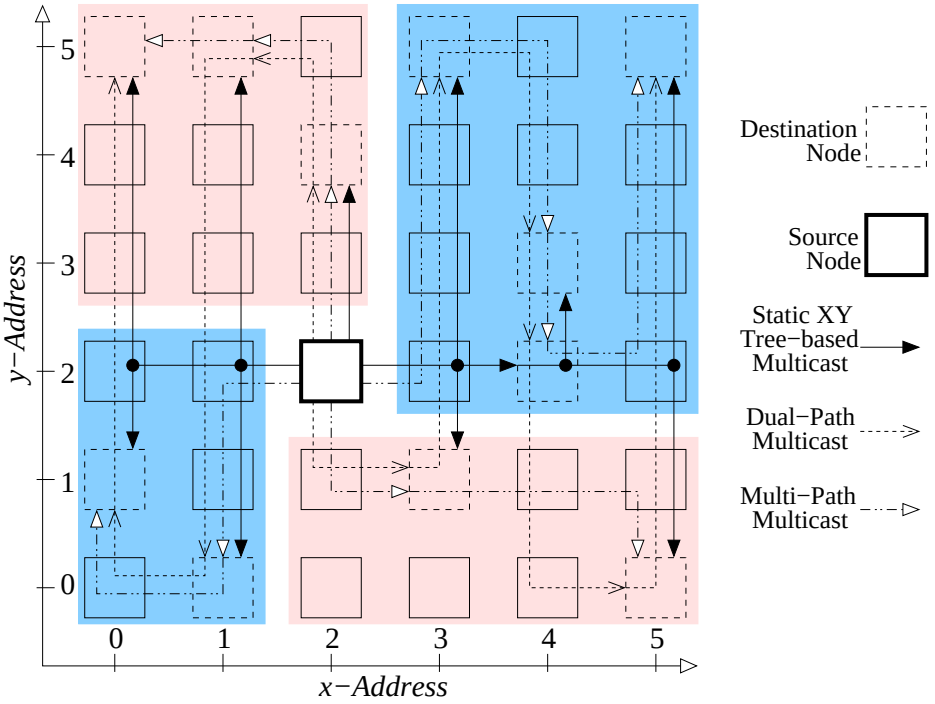


Fig. 5.1: The traffic formations by using static tree-based, dual-path and multi-path multicast routing methods.

the message with minimum number of paths.

The works in [137], [63] and [39] present the multicast methodology using path-based method. In path-based multicast routing, PEs that inject the message have to set up the order of headers containing the addresses of all multicast destination nodes in order to find optimum paths from the PEs to the destination nodes. Therefore, there will be time-overhead for the message preparation at source nodes. The path-based multicast routing is aimed at reducing or probably preventing the multicast messages conflicts in intermediate nodes. Each multicast packet will acquire at most two sinking ports in a destination node to forward the multicast message i.e., LOCAL port (connected directly to a resource tile) and the other (one) port for forwarding/duplicating the multicast message to other destination nodes. In general, path-based multicast routing can be classified into dual-path and multi-path multicast routing. In the dual-path multicast routing, the number of maximum paths performed in the network is two, while in the multi-path multicast routing, the number of maximum path is four. However, the path-based multicast routing avoid to do branching. In each intermediate destination node, a packet is firstly forwarded from an input port to the LOCAL output port, while keeping the packet in the input port. Afterwards, the packet is routed to another requested output port. Fig. 5.1 presents the different routing paths performed by the dual-path and multi-path multicast routing methods.

The works in [19], [147], [124] present the routing methodology based on multicast tree. In the tree-based multicast routing, the header ordering in source nodes is not re-

quired (the order of the destination addresses can be freely determined). The multicast routing will form communication paths like branches of trees connecting the source node with the destination nodes at the end points of the tree branches. A higher probability that multicast deadlock occurs in intermediate nodes is the disadvantage of the tree-based multicast routing. However, the novel multicast scheduling for adaptive tree-based multicast routing presented in this thesis can solve effectively and efficiently the multicast deadlock problem in the intermediate nodes, which makes the methodology more interesting.

In general, the multicast routings presented in [137], [63], [39], [19], [147] and [124] are not suitable for on-chip networks. All these works utilize virtual channels to solve multicast deadlock problems. In general, FIFO queues as the main components in virtual channels dominate significantly the logic gate consumption. In the XHiNoC, the adaptive routing algorithms used to route unicast and multicast packets are the same and multicast contentions are solved without using virtual channels, resulting in a very efficient gate-level implementation of the routing function and data buffers.

The work presenting a path-based multicast routing dedicated for NoC has been introduced in [146]. The path-based multicast routing is designed to avoid multicast deadlock in the destination nodes by reserving virtual channels and giving priority to the multicast message over the unicast message on arbitration of link bandwidth. Experiments in the work show that the proposed multicast technique improves throughput, and does not exhibits significant impact on the unicast performance in a network with mixed unicast-multicast traffic "only if" the network is not saturated.

Compared to the work presented in [146], the proposed tree-based multicast scheduling presented in this thesis does not give priority to multicast messages (fair flit-by-flit arbitration between the unicast and multicast messages). Hence, our multicast technique does not have a significant impact on the unicast performance "even if" the network is saturated. The multicast routing methodology used in the XHiNoC also presents an interesting performance characteristic during saturating and non saturating conditions. Moreover, the NoC router in [146] has not been synthesized into logic gate level.

5.2.2 Source and Distributed Multicast Routing

According to the place where the routing paths and routing decisions are made, the multicast routing can be divided into *centralized (source) multicast routing* and *distributed multicast routing*. By using the static tree-based multicast routing in a mesh-based regular network for instance, the routing decision can be made by using the distributed routing approach, because the network orientation can be easily mapped to every network router to make correct routing paths.

The work in [216] presents the problem of synthesizing custom NoC architectures that are optimized for a given application, and considers both unicast and multicast traffic

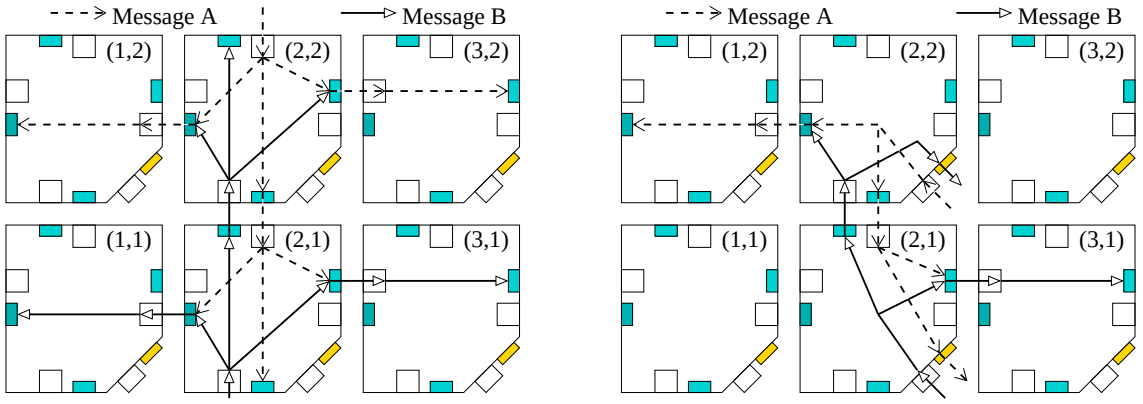
flows in the input specifications. Several algorithms that can systematically examine different flow partitioning are proposed. Algorithms based on Rectilinear-Steiner-Tree are then used to generate efficient network topology. The design flow of the work integrates floorplanning and deadlock-free routing determination. The work proposes a static solution for deadlock-free multicast routing that is fixed to specific NoC application. Hence, it look that the work in [216] can be classified into the centralized multicast routing with off-line (at design time) multicast routing paths computation.

The work in [138] presents a new heuristic multicast routing scheme that combines the distributed routing and source routing methods. The proposed path-based multicast routing scheme consists of two routing algorithms, i.e. a preprocessing algorithm for message preparation to find routing control information that will be carried by the message that are run at source node, and an algorithm for message routing that are made distributively in the intermediate nodes. The generated routing control information are in conjunction with destination address such that efficient routing decision can be made by forward nodes.

So far, there have been some other works that have introduced a NoC router with multicast routing service. The work in [1] for example presents a *Multicast Router Rotary* (MRR). The multicast routing algorithm in the MRR can be classified into a distributed routing method. The multicast contention in MRR is solved by implementing two single-direction internal rings in the switch, one in clock-wise direction and the other one in counter clock-wise direction. Without careful data flow rule, a dangerous permanent deadlock can occur especially when packets come from all different input ports, and each of them requests all output ports simultaneously. The proposed data flow rule in the MRR must even allow misrouting to avoid deadlock in a case that a packet cannot find a free output port. In any circumstance, misrouting can increase data communication energy due to the overhead misrouting traffic which can lead to a livelock situation. The work in [1] has not yet addressed this livelock issue. Moreover, an additional 10 internal buffers (5 for each ring) in the MRR will increase the area overhead of the router.

The work in [189] presents a *Broadcast-multicast-enabled Logic-based Distributed Routing* ($BLBDR$). Another routing approach called *Recursive Partitioning Multicast* (RPM) method is also presented in [208]. The BLBDR and RPM methods need for global network view and preprocessing algorithm for network partitioning. In the RPM method, a routing decision is made based on the current network partitioning that has been previously computed recursively in a source node. The whole network is divided into at most eight subnets by the source node. The objective of the network partitioning is to minimize packet replication time. In general, a pre-processing network partitioning algorithm methods will lead to an initiation time overhead.

A *Virtual Circuit Tree Multicasting* ($VCTM$) method is presented in [107]. In the VCTM method, a setup packet must be sent in the network to configure a switched tree-based multicast virtual circuit. The virtual circuit configuration is implemented by using virtual channels. Hence, like the RPM method [208], both multicast routers have large logic area



(a) Tree-based multicast routing

(b) Branching in dual/multi-path multicasting

Fig. 5.2: Multicast deadlock configurations when using tree-based and path-based multicast routing in mesh networks.

cost due to the replication of buffers and control logics for the VCs arbitration.

5.3 Theory for Deadlock-Free Multicast Routing

Multicast deadlock configuration is a situation in which multicast packets, which are switched in the network with wormhole switching technique, cannot move further due to multicast dependency occurs in some NoC routers. The multicast dependency occurs because two or more multicast packets are competing with each other to access the same output ports in any NoC router, while in other NoC routers, the same situation occurs, i.e. the same competing multicast packets compete also to acquire the same output ports. In order to understand a better insight about the multicast deadlock configuration, an example of the multicast deadlock configurations is presented in Fig. 5.2.

The deadlock configuration when using tree-based multicast routing with wormhole switching method is shown in Fig. 5.2(a). In node (2,2), message A cannot move further because the East and West output ports are acquired by message B. Meanwhile, message B cannot move further in node (2,1), because the East and West output ports are acquired by message A. The multicast dependencies of the contenting multicast packets in many network nodes will lead to a deadlock configuration.

In a path-based multicasting, a source node arranges the ordered list of headers containing destination address. When a message is injected to the network, it will be routed to a destination node according to the address attached in the leading header flit. When the message arrives the destination node, the leading header flit is removed. Hence, the next header will be the leading flit and guides the message into the next destination. The path-based multicasting is a mechanism to avoid branching in intermediate nodes. Two branches of the paths are formed only in destination nodes, i.e. one to Local port and

one to another port. In the source node, the number of branches can be maximum 2 for dual-path and more than 2 for the multipath-based multicasting. Deadlock configuration can occur when a multicast router does tree branch (even by using dual/multi-path multicast) as shown in Fig. 5.2(b).

Theories of deadlock-free adaptive multicast routing in wormhole network has been presented in [110], [137] and [63]. The work in [110] has presented an optimum broadcasting method and personalized communication in hypercube interconnection networks. In particular, the theory presented in [63] is only valid when using the path-based multicast routing model in wormhole-switched network.

The NoC presented in [142] uses a time-space-time switch designed for time-division-multiplexing (TDM-based) NoCs and introduces a basic formalism for multicast routing. Slot map tables as central components are used as time slot interchangers to directly control the read and write operation in random access frame buffers. Although this work has mentioned the feasibility of implementing the multicast scheduling technique, a concrete multicasting procedure, system-level or RTL-level simulations for measuring the NoC performance over multi message multicast traffics and the NoC's capability to handle the multicast deadlock problem has not been presented so far.

However, most of the aforementioned proposed theories are generally not dedicated for networks-on-chip. The work in [142] has found that the problem of finding optimal coloring for TDM-based multicast solution is a non-deterministic polynomial-time (NP) hard problem. The work has tried to use a *Q-Coloring Algorithm* that has been previously introduced in [86]. However, the work shows only the existence of feasible scheduling algorithm supporting multicast routing without further formal prove. The following Section 5.3.1 proposes a simple and smart mechanism to tackle multicast deadlock configuration, which is suitable and dedicated for networks-on-chip, in which virtual channels are not involved to solve the multicast dependency problem.

5.3.1 New Multicast Method based on Hold-Release Tagging Policy

The tree-based multicast routing is prone to deadlock. The deadlock occurs in a intermediate node when one or more outgoing links are simultaneously requested by the same multicast packets. Therefore, we propose a new methodology to handle the multicast deadlock. Fig. 5.3 presents 6 snapshots of the proposed multicast scheduling method and a fair flit-by-flit round arbitration of a so called *hold-release multicast fair scheduling policy* for the deadlock handling mechanism.

- In **Snapshot 1**, three multicast packets, i.e A coming from Port 1, B from Port 3 and C from Port 4, request different and the same outgoing links. Port 2 and Port 3 outgoing links are requested by Packets A and C. The other outgoing links are only requested by one Packet, i.e. Port 1 and Port 4 are requested by Packet B, and Port 5 by Packet C. The flits $A1^0$, $B1^0$ and $C1^0$ represent the flits with local ID-tag 0.

- Although the outgoing links are requested by more than one packet, each one of every single flit can be granted access to the outgoing link at every stage as shown in **Snapshot 2**. In this stage, we assume that flit $C1$ is firstly selected to access the Port 2 outgoing link, while flit $A1$ is granted access to the Port 3 outgoing link. The other outgoing links, i.e. Port 1, 4 and 5, also select their single request from flits in the incoming port.

- In the next stage as presented in **Snapshot 3**, all granted flits are accepted in the outgoing links. However, the states of all flits in incoming are different and depend on whether their multicast requests have been granted by their required outgoing ports. For instance, all multicast request of Packet B to access Port 1 and Port 4 have been granted by these ports. Hence, flit $B1$ (with *R* state) can be released from Port 3 input buffer and its request is now replaced by the request of the new incoming flit $B2$. But flits $A1$ and $C1$ (with *H* state) must still be withheld in input buffers because their other requests (presented in dashed lines) to access another port have not been granted in this stage. In this stage, all ID-tags of the packets are mapped and updated with new ID-tag 0.

- In the next stage as shown in **Snapshot 4**, by using the flit-by-flit round arbitration method, arbiters at Port 2 and Port 3 change now their selection to other flits, which also request the ports. Port 2 selects now flit $A1$, while Port 3 selects flit $C1$. Port 1 and Port 4 select again the flit coming from Port 3 input buffer (i.e. flit $B2$), because these ports are only requested by Packet B from Port 3 input buffer. But the Port 5 outgoing arbiter will not grant flit $C1$ again because flit $C1$ has been granted in the previous stage. This decision is made to avoid flit C1 being transported twice into the Port 5 (avoiding improper multicast replication).

- In the next stage as presented in **Snapshot 5**, flits $A1$, $B2$ and $C1$ are transferred to the outgoing links, and can be released from Port 1, Port 3 and Port 4 input buffers (with *R* state) respectively because their multiple requests have been granted previously step by step in **Snapshot 2** and **Snapshot 4**. Their request are now replaced by the requests of new incoming flits i.e., flits $A2$, $B3$ and $C2$. Because ID-tag 0 has been used by packet C in Port 2, then packet A in the Port 2 outgoing link is assigned with a new local ID-tags 1 ($A1^1$). The same situation is presented in Port 3, where packet A has used ID-tag 0. Hence packet C in the Port 3 outgoing links is assigned with a new local ID-tags 1 ($C1^1$).

- **Snapshot 6** generally shows the same mechanism with the situation shown in **Snapshot 2**.

The philosophy of the *Hold-Release Tagging Mechanism* is as follows. "If a multicast flits from an input port n has an $N_{s,n}^{req}$ number of requests at any instant time t_s, then each single request to an output port m can be forwarded from the input port n to the output port m in the next time stage only if it receives a grant by an arbitration unit at

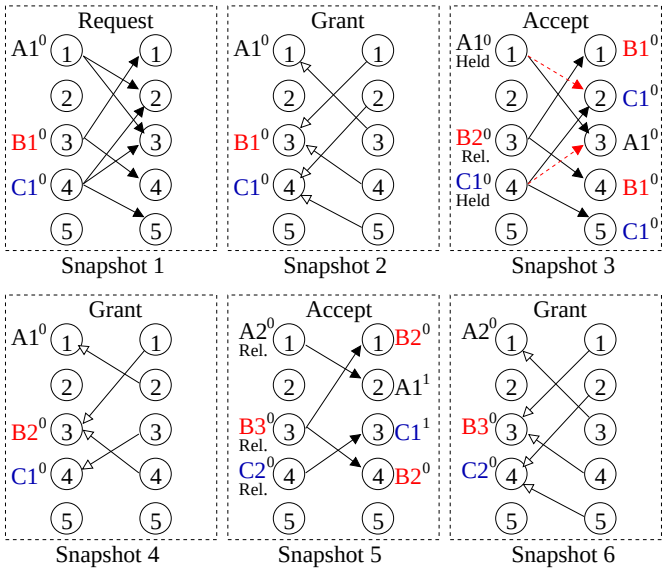


Fig. 5.3: Hold and Release Multicasting Policy.

input port n, while the other requests must be held in the input port if it is not granted by their requested output ports. In each next time stage, a single request, which has been granted before, must be reset to prevent improper flit replication. If all requests have been granted, then the multicast flit can be released from the queue in input port n".

Fig. 5.4 shows another example of a high contention of multicast routing requests in a NoC router, in which the solution is described in a different manner. As presented in the figure, five messages coming from different input ports compete with each other to share the same output ports. Message A coming from input port 1 for example has four multicast routing requests, i.e. to output port 2, 3, 4 and 5. The other messages (message B, C, D and E) have also four multicast requests. At the output port 1 for example four multiple requests must be served, i.e. requests from message B, C, D and E.

Since all messages come at the same *input time stage*, i.e. input time stage 1 ($i1$), we assume that the first input selection of each output port is different and each output port rotates their selection flit-by-flit and port-by-port. For example, the output port 1 selects firstly the flit from input port 2, i.e. flit $B1$. At the output time stage $o1$, the number of the granted multicast routing requests of each message is 1 as shown in a table at the right side of the Fig. 5.4(a). Therefore, all the flits must still be held in the input buffers, because the other three requests of each message have not been granted. At the time stage $o2$ as shown in Fig. 5.4(b), each arbiter at the output port now rotates its selection to another input port. We assume that the rotation is made in an incremental manner (up counting). Thus the output port 1, for example selects the flit from input port 3, i.e. flit $C1$.

At the time stage $o3$ as shown in Fig. 5.4(c), the flit $A1$ and $D1$ can be released from input buffers because all their multicast routing request have been granted. For the flit $A1$ for instance, its requests to output port 2 and 4 are granted at time stage $o1$ and $o2$, respectively. Afterwards, its requests to output port 3 and 5 are concurrently granted at

the time stage $o3$. Hence, at the time stage $o3$, all of four multicast routing requests of the flit $A1$ have been granted. Thus, it can be released from input port 1. Meanwhile, since the number of the granted requests of flits $B1$, $C1$ and $E1$ from time stage $o1$ until $o3$ are less than their total multicast requests, i.e. 2, 3 and 2, respectively, then they must be still held in the input buffers. Finally at the time stage $o4$ as shown in Fig. 5.4(d), flits $B1$, $C1$ and $E1$ are switched out to their requested output ports, and can be released from the input ports.

By using the proposed *Hold and Release Multicasting Policy* explained above, the contenting multicast flits can move out from the multicast dependency after four *output switching time stages*. As presented in the figure, all the first line flits ($A1$, $B1$, $C1$, $D1$ and $E1$) of the competing multicast messages can be released from input buffers at the output time stage $o4$. The same multicast contention solution is experienced by the second line flits ($A2$, $B2$, $C2$, $D2$ and $E2$) as well as the next line of flits messages.

5.3.2 Multicast Flit Replication Control based on Hold/Release Tagging Mechanism

In order to avoid an improper flit replication during multicast contention handling mechanism, a multicast flit replication control must be applied. This subsection will describe formally how the replication control works in line with hold/release tagging mechanism.

Definition 5.1 ***Routing Request Matrix*** *$R(t)$ describes the requests of all incoming flits to access the output ports at time-stage unit t. The elements of the routing request matrix $R(t)$ consist of the elements of the time-varying input n binary request or the time-varying output m binary request defined in Def. 3.13 such that*

$$R_{N_{inp}\times N_{outp}}(t) = \left(r_{n,1}(t),\ r_{n,2}(t),\ \cdots,\ r_{n,N_{outp}}(t)\right) \tag{5.1}$$
$$n = \{1, 2, \cdots, N_{inp}-1, N_{inp}\} = \{1 : N_{inp}\}$$

$$R_{N_{inp}\times N_{outp}}(t) = \begin{pmatrix} r_{1,m=1:N_{outp}}(t) \\ r_{2,m=1:N_{outp}}(t) \\ \cdots \\ r_{N_{inp},m=1:N_{outp}}(t) \end{pmatrix} \tag{5.2}$$

We can also define that $R(t) : r_{n,m}(t) \in \{0, 1\}$, where n and m represent the row and column coordinates of each matrix element. According to Def. 3.8, n and m are interpreted as the input and output port number of the router IO port, respectively. The value of the $r_{n,m}(t)$ are either 0 or 1. The element $r_{n,m}(t) = 1$ if there is a routing request from input port n to output port m, else its value is $r_{n,m}(t) = 0$. For a unicast request, $\forall n:\ \sum_{m=1}^{N_{outp}} r_{n,m}(t) = 1$, and for multicast request $\forall n: 1 < \sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq N_{outp}$.

If $0 \leq \sum_{n=1}^{N_{inp}} r_{n,m}(t) \leq 1$, then there is no contention to access the output port m. Equ. 5.3 (left-side) shows an example of the $R(t)$ for the Snapshot 1 in Fig. 5.3 where the IO ports are represented as port numbers 1, 2, 3, 4 and 5, respectively.

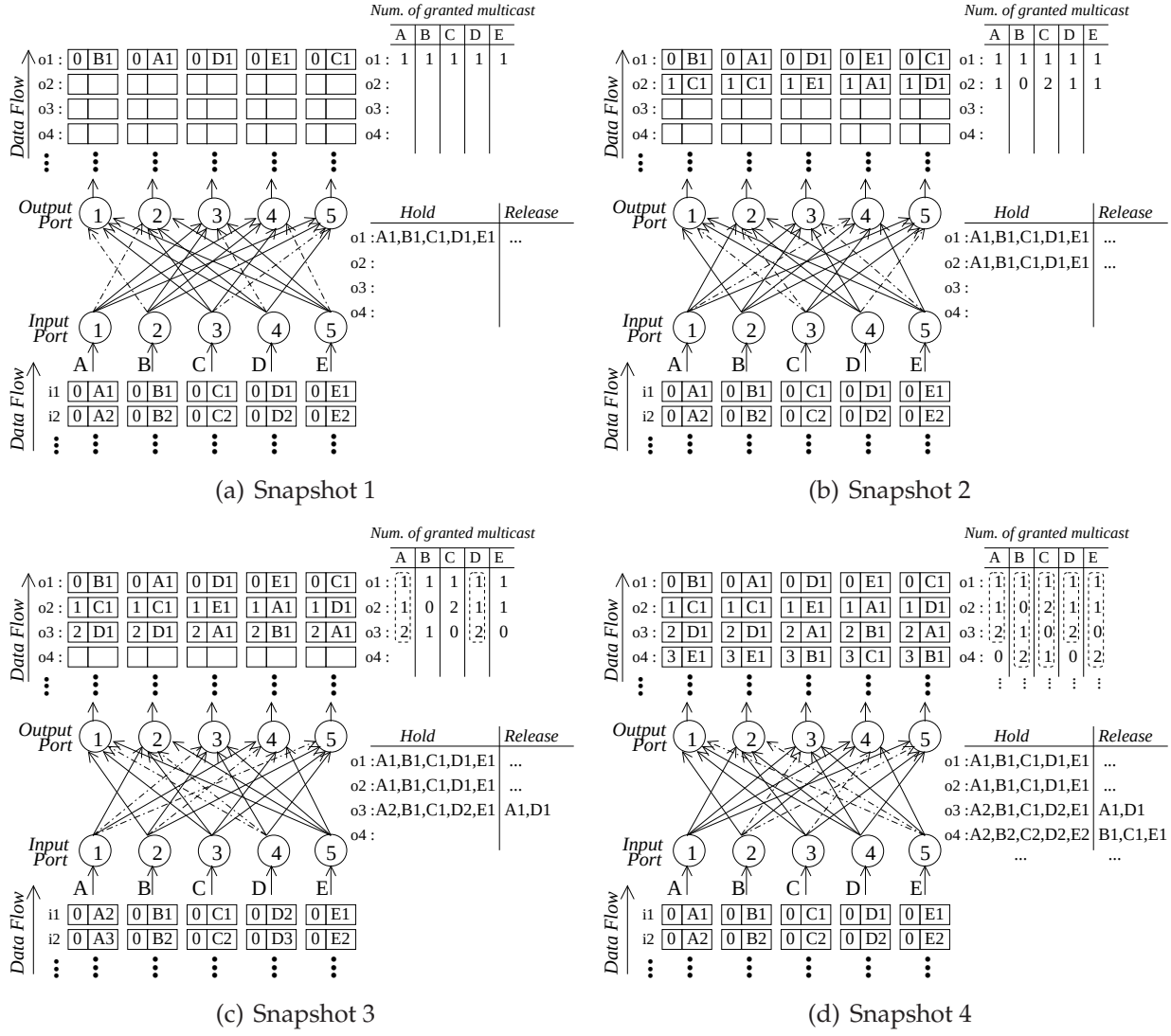


Fig. 5.4: High multicast traffic contentions in a router and solution with the Hold and Release Multicasting Policy.

$$R(1) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; A(1) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.3}$$

Definition 5.2 ***Arbitration Matrix*** *$A(t)$ describes the grant signal from an arbiter unit to select one flit from the input port to access its requested output port at time stage t. The Arbitration Matrix and its array elements are defined as $A(t) : a_{n,m}(t) \in \{0, 1\}$. The form of the Arbitration Matrix $A(t)$ is strongly dependent on $R(t)$.*

$$A_{N_{inp} \times N_{outp}}(t) = \left(a_{n,1}(t),\ a_{n,2}(t),\ \cdots,\ a_{n,N_{outp}}(t)\right) \tag{5.4}$$
$$n = \{1, 2, \cdots, N_{inp} - 1, N_{inp}\} = \{1 : N_{inp}\}$$

For example, if the output port $m = 2$ has two requests from input ports as shown in column 2 of matrix $R(t)$ in Equ. 5.3, i.e. $r_{n=1:5,m=2}(t) = [1\ 0\ 0\ 1\ 0]^T$, then based on Def. 3.16, Def. 3.17 and Def. 3.18, the set of two possible combinations of the column 2 of the arbitration matrix is $a_{n=1:5,m=2}(1) = [0\ 0\ 0\ 1\ 0]^T$ and $a_{n=1:5,m=2}(2) = [1\ 0\ 0\ 0\ 0]^T$. In other words, in each time-stage t, where the arbiter rotates the selection among existing requests, the arbiter can only select one flit from an input port. This means, the sum of the column elements in A must be either 0 or 1, or $0 \leq \sum_{n=1}^{N_{inp}} a_{n,m}(t) \leq 1$. Equ. 5.3 (right-side) shows an example of the $A(t)$ for the Snapshot 2 in Fig. 5.3.

Definition 5.3 ***Tagged Matrix*** *$R^*(t) : r^*_{n,m}(t)$ is a support matrix that is useful to determine whether a flit must be "held" or can be "released" from the input port, and to compute the next routing request matrix $R(t+1)$. For each time-stage unit t, the matrix request $R(t)$ will be updated as presented in Equ. 5.5. The function ϕ contains two subfunction, i.e. $f_{FR*} : R(t), A(t) \rightarrow R^*(t)$ contains operator to form tagged matrix $R^*(t)$ and $f_{UPR} : R^*(t) \rightarrow R(t+1)$ to update each element in $R(t+1)$.*

$$R(t+1) = \phi\left(R(t), R^*(t), A(t)\right) \tag{5.5}$$
$$f_{FR*} : R(t), A(t) \rightarrow R^*(t)$$
$$f_{UPR} : R^*(t) \rightarrow R(t+1)$$

According to Equ. 5.6 and Equ. 5.7, the form of tagged matrix $R^*(t)$ depends on the current form of the $R(t)$ and $A(t)$.

$$\textbf{\textit{if}}\ n = constant\ \textbf{\textit{and}}\ \exists m : r_{n,m}(t) \neq a_{n,m}(t)\ \textbf{\textit{then}}$$
$$\forall m : r^-_{n,m}(t) = \begin{cases} 1^- & : r_{n,m}(t) = a_{n,m}(t) \\ 1^* & : r_{n,m}(t) \neq a_{n,m}(t) \\ 0 & : r_{n,m}(t) = 0 \end{cases} \tag{5.6}$$

$$\textbf{\textit{if}}\ n = const\textbf{\textit{ and}}\ \forall m\colon r_{n,m}(t) = a_{n,m}(t)\ \textbf{\textit{then}}$$
$$\forall m\colon r^{+}_{n,m}(t) = \begin{cases} 0 & : r_{n,m}(t) = 0 \\ 1^{+} & : r_{n,m}(t) = a_{n,m}(t) \end{cases} \tag{5.7}$$

If we compare matrices in Equ. 5.3, then according to Equ. 5.6, we can see that there are two elements which do not match each other, i.e. $r_{1,2}\,(1)$ does not match with $a_{1,2}\,(1)$, and $r_{4,3}\,(1)$ does not match with $a_{4,3}\,(1)$. Therefore, these two elements are tagged with the symbol $(^{*})$ as presented in Equ. 5.9. If minimal one element of row n is tagged with $(^{*})$, then the others elements having the value 1 in same row n will be marked with the symbol $(^{-})$. As presented in Equ. 5.9, the element $r^{-}_{1,3}\,(1)$, $r^{-}_{4,2}\,(1)$ and $r^{-}_{4,5}\,(1)$ are assigned with $(^{-})$. The other elements of the row n having no 1-element, being tagged with symbol $(^{*})$ or $(^{-})$, are assigned with symbol $(^{+})$ in accordance with Equ. 5.7. As presented in Equ. 5.9, all elements in row 3, i.e. $r^{+}_{3,1}\,(1)$ and $r^{+}_{3,4}\,(1)$ are assigned with $(^{+})$, because there is no element in the row 3 having tag symbol $(^{*})$.

$$R^{*}_{N_{inp}\times N_{outp}}(t) = \begin{pmatrix} \forall m\colon r^{-}_{1,m}(t)\ or\ r^{+}_{1,m}(t) \\ \forall m\colon r^{-}_{2,m}(t)\ or\ r^{+}_{2,m}(t) \\ \cdots \quad \cdots \quad \cdots \\ \forall m\colon r^{-}_{n,m}(t)\ or\ r^{+}_{n,m}(t) \end{pmatrix} \tag{5.8}$$

$$R^{*}(1) = \begin{pmatrix} r^{-}_{1,m}(1) \\ r^{+}_{2,m}(1) \\ r^{+}_{3,m}(1) \\ r^{-}_{4,m}(1) \\ r^{+}_{5,m}(1) \end{pmatrix} = \begin{pmatrix} 0 & 1^{*} & 1^{-} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1^{+} & 0 & 0 & 1^{+} & 0 \\ 0 & 1^{-} & 1^{*} & 0 & 1^{-} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.9}$$

Definition 5.4 *The* Hold/Released Tagging Policy *can be applied by observing whether the row array element of Equ. 5.8 falls in the case according to Equ. 5.6 i.e.* $r^{-}_{n,1:N_{outp}}(t)$*, or in the case according to Equ. 5.7 i.e.* $r^{+}_{n,1:N_{outp}}(t)$*. Both Equ. 5.6 and Equ. 5.7 comprise of an* antecedence *or* condition *part and a* consequence *part.*

Definition 5.5 (Data Hold Policy) *If the array elements of the row* n *of the Tagged Matrix* $R^{*}(t)$ *are* $r^{-}_{n,1:N_{outp}}(t)$*, then the flit coming from the input port* n*, where* $n = const$*, must be held in the input port* $l = n$*, because at time stage* t *and* $\forall m,\ n = const. \Rightarrow \exists m\colon r_{n,m}(t) \neq a_{n,m}(t)$ *or there is minimal one element of the* $r_{n,1:N_{outp}}(t)$ *that has not been granted to be switched out to the requested output port. According to Equ. 5.6, the non-granted element is tagged with symbol* $(^{*})$*, while the granted element is tagged with symbol* $(^{-})$*. Therefore, according to Equ. 5.10, the next request at time stage* $t+1$ *of granted element will be dropped to avoid improper multicast flit replication.*

$$\begin{aligned} &\textbf{\textit{if}}\ n = const.\ \textbf{\textit{and}}\ r^{*}_{n,m}(t) = r^{-}_{n,m}(t)\ \textbf{\textit{then}} \\ &\forall m\colon r_{n,m}(t+1) = \begin{cases} 0 & : r^{*}_{n,m}(t) = 0\ or\ 1^{-} \\ 1 & : r^{*}_{n,m}(t) = 1^{*} \end{cases} \end{aligned} \tag{5.10}$$

Definition 5.6 (Data Release Policy) *If the array elements of the row n of the Tagged Matrix $R^{*}(t)$ are $r^{+}_{n,1:N_{outp}}(t)$ or all elements of the $r_{n,1:N_{outp}}(t)$ has been granted to be switched out to the requested output ports, then the flit coming from the input port n can be released from the input port n. In Equ. 5.11, we can define that if the flit from input port n is released from the input port and switched to the output port, then the next considered flit at time stage $t+1$ may be 1) a zero flit (not a data flit, $r_{n,1:N_{outp}}(t+1) = \emptyset$), 2) a flit of different message with different unicast or multicast output routing direction ($r_{n,1:N_{outp}}(t+1) \neq r_{n,1:N_{outp}}(t)$), or 3) a flit that belongs to the flit that has been released from the input port ($r_{n,1:N_{outp}}(t+1) = r_{n,1:N_{outp}}(t)$).*

$$\begin{aligned} &\textbf{\textit{if}}\ n = const.\ \ \textbf{\textit{and}}\ r^{*}_{n,m}(t) = r^{+}_{n,m}(t)\ \textbf{\textit{then}} \\ &\forall m\colon r_{n,m}(t+1) = r^{F_{next}}_{n,m}(t+1) \end{aligned} \tag{5.11}$$

$$R(2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}; A(2) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.12}$$

$$R^{*}(2) = \begin{pmatrix} r^{+}_{1,m}(2) \\ r^{+}_{2,m}(2) \\ r^{+}_{3,m}(2) \\ r^{+}_{4,m}(2) \\ r^{+}_{5,m}(2) \end{pmatrix} = \begin{pmatrix} 0 & 1^{+} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1^{+} & 0 & 0 & 1^{+} & 0 \\ 0 & 0 & 1^{+} & 0 & 1^{+} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{5.13}$$

Lemma 5.1 *By using the "hold/release tagging policy" defined in Def. 5.4, improper multicast flit replication on every router can be avoided.*

Proof of Lemma 5.1 *If the number of requests of a flit coming from input port n is defined as $N^{req}_{s,n}$ such that at time stage $t = t_s$, $N^{req}_{s,n} = \sum_{m=1}^{N_{outp}} r_{n,m}(t_s)$, then according to Equ. 5.10, a routing request $r_{n,m}(t)$ that has been granted will be reset at the next time stage $t = t_s + 1$. Therefore, every routing request $r_{n,m}(t)$ will be only forwarded once to the output port.*

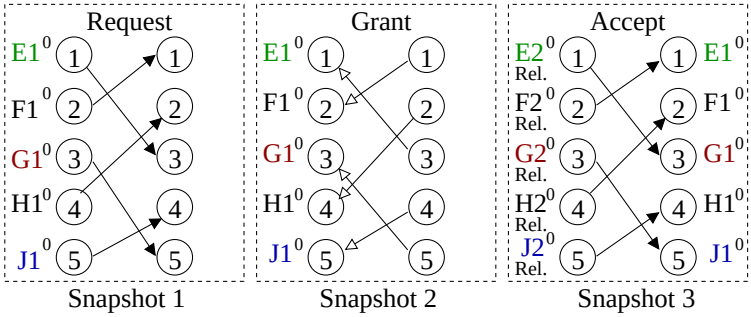


Fig. 5.5: Scheduling unicast requests without contention.

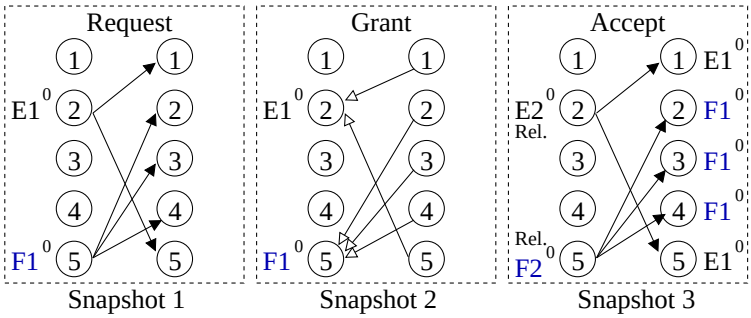


Fig. 5.6: Scheduling multicast requests without contention.

5.3.3 Proof of the New Theory for Deadlock-Free Multicast Routing

Postulate 5.1 *If unicast packets are routed in a certain router such that* $\forall n\colon \sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1$ *(See Def. 5.1), or if multicast packets are routed in a certain router such that* $\exists n\colon \sum_{m=1}^{N_{outp}} r_{n,m}(t) > 1$*, and there is no contention between them to access output link such that* $\forall m\colon \sum_{n=1}^{N_{inp}} r_{n,m}(t) \leq 1$*, then according to Def. 5.2, Def. 3.16 and Def. 3.17,* $\because \forall m\colon T_{s,m} = 1 \Rightarrow \forall t\colon A(t+1) = A(t)$*. Thus, according to Def. 5.3, then* $\nexists r^*_{n,m}(t) = 1^* \therefore \forall t\colon R(t) = A(t)$ *or there is no need to apply for the "Hold/Release Rule". All unicast packets (without contention) can be released from every input port, or will not be withheld at every input port at each time stage* t*.*

Fig. 5.5 and Fig. 5.6 show examples of the unicast and multicast requests in a router without contention, respectively. In Fig. 5.6, we can see that multicast packet E from input port 2 acquires output ports 1 and 5 without competing with multicast packet F from input port 5 that acquires the other output ports, i.e. output port 2, 3 and 4.

Postulate 5.2 *If unicast packets are routed in a certain router such that* $\forall n\colon \sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1$ *(See Def. 5.1), and there are one or more contentions between them to access output link such that* $\exists m\colon \sum_{n=1}^{N_{inp}} r_{n,m}(t) > 1$*, according to Def. 5.2, Def. 3.16 and Def. 3.17, the contention on each output port* m *can be solved at* $t = T_{s,m}$ *where* $\forall m\colon \exists T_{s,m}\colon r_{n,m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{n,m}(t)$*, where* $\forall m\colon 1 \leq T_{s,m} \leq N_{inp}$*. Furthermore according to Def. 5.3,* $\because \forall n\colon \sum_{m=1}^{N_{outp}} r_{n,m}(t) \leq 1 \Rightarrow \forall t, n, m\colon \nexists r^*_{n,m}(t) = 1^-$*, or* $\forall t, m, n\colon r^*_{n,m} = r^+_{n,m}$ *according to Equ. 5.8. Therefore, the "Hold/Release Tagging Policy" is only partially applied, i.e. a unicast request* $r_{n,m}(t)$ *that has not*

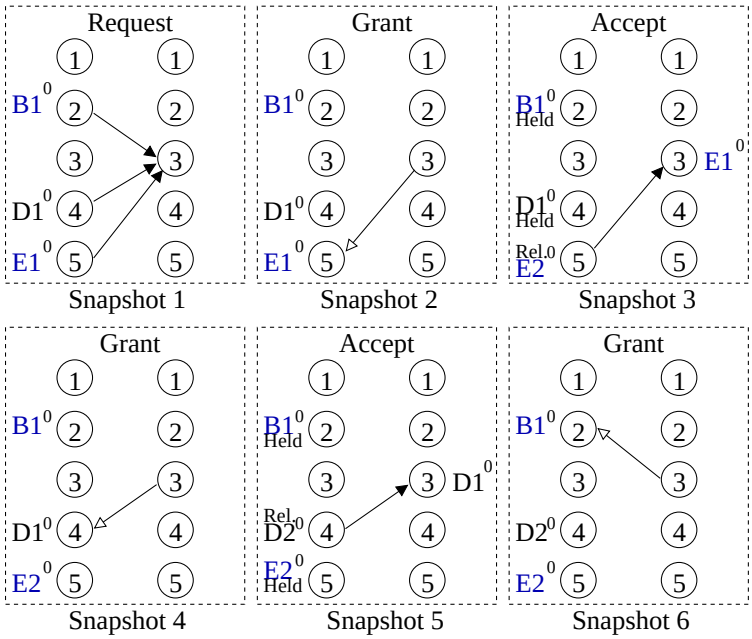


Fig. 5.7: Scheduling unicast requests with contention.

been granted at time stage t must wait in the input port $l = n$. Fig. 5.7 shows an example of this situation and how the unicast contention is solved.

Based on the postulates mentioned above, the problem can be extended in a situation called *Multicast Routing Requests with Outgoing Contention* where multicast packets are routed in a router with multicast requests such that $\exists n\colon \sum_{m=1}^{N_{outp}} r_{n,m}(t) > 1$, and there are one or more contentions between the multicast requests to acquire the same output ports or $\exists m\colon \sum_{n=1}^{N_{inp}} r_{n,m}(t) > 1$. This extended problem is actually the key problem that will be solved by our new theory to perform deadlock-free multicast routing without implementing virtual channels in a wormhole-switched NoC.

Theorem 5.1 *The ID-field being part of every flit allows the implemention of a flit-by-flit arbitration and an ID-based routing for interleaving different packets in the same queue, where flits belonging to the same packet have the same ID-tag on every local communication link. Hence, multicast deadlock problem can be solved at each router by further applying a "Hold/Release Tagging Policy" to control and manage conflicting multicast requests.*

Proof of Theorem 5.1 *The circulating arbitration mechanism can guarantee that one flit of unicast or multicast packets can be forwarded to each outgoing link at each router node, where multicast conflict may occur. After arbitration process at each time t, a hold-release tagging mechanism can also guarantee that improper replication of the multicast packets can be avoided, because: 1) the granted multicast bit-requests will be assigned and will not be included again in the next arbitration process, and 2) the flits having multicast bit-requests will be kept in the FIFO queue until all its multiple bit-requests are granted.*

The circulating selection result of the arbitration process at each output port may be random and not uniform. Therefore, there are two possible configurations after the arbitration process, i.e. 1) all requests of a multicast flit from an input port n are granted at the same time-stage t ($r_{n,1:N_{outp}}(t) = a_{n,1:N_{outp}}(t)$), or 2) not all the multicast requests from an input port n are granted ($\forall h \in \varphi_n^{req}\colon \exists r_{n,h}(t) \neq a_{n,h}(t)$). In the situation 1), the multicast flit can be released from FIFO queue, and in the situation 2), the multicast flit must be held in FIFO queue, and the hold-release tagging policy and the circulating/rotating flit-by-flit arbitration will then cover the situation.

By circulating the bit-set selection in every column m of $A(t)$ at each time-stage t, where the circulating combinations of $a_{1:N_{inp},m}(t)$ for $\forall m$ (all output ports) are independent each other, then it is possible, in finite time T_f to find $A(T_f)$ in such a way, that all conflicting multicast flits can be rescued from multicast dependency.

1. *If the amount of requests in every output port m at $t = t_s$ is $N_{s,m}^{req}$ such that $1 \leq N_{s,m}^{req} \leq N_{inp}$ (See Equ. 3.6), then the required number of circulating arbitration time to grant the request from the input port $l \in \Phi_m^{req}$ to output m appear at t_s is $T_{s,m} = N_{s,m}^{req}$. The probability that the request $r_{l,m}(t_s)$ is selected by grant acknowledge $a_{l,m}(t_s)$ is $Prob(r_{l,m}(t_s) = a_{l,m}(t_s)) = \frac{1}{N_{s,m}^{req}}$. According to Def. 3.15, Def. 3.16, Def. 3.17 and Def. 3.18, then at $t = T_{s,m}$ we will achieve that $r_{1:N_{inp},m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$. The maximum number of requests to an output port m is N_{inp}. Hence, if $\Phi_m^{req} \subseteq \Phi$, then $r_{1:N_{inp},m}(t_s) = \bigcup_{t=1}^{N_{inp}} a_{1:N_{inp},m}(t)|t_s = 1$.*

2. *If at $t = t_s$ there is multicast requests from an input port n such that $N_{s,n}^{req} > 1$ (See Def. 3.14), then we obtain a set of input ports φ_n^{req} in such a way that $r_{n,h}(t_s) = 1$ iff an output port $h \in \varphi_n^{req}$ (See Equ. 3.5). The probability that every single request of the multicast request $r_{n,h}(t_s)$ from the input port n is selected by the arbitration unit at the requested output port $h \in \varphi_n^{req}$, also depends on the number of requests from other input ports in the set Φ_h^{req} to the same output port h.*

In accordance with items 1) and 2) mentioned above, then we can derive a conditional equation such that the multicast deadlock problem is solved as described in the following equation.

$$R(t_s) = \left(\bigcup_{t=t_s}^{T_{s,1}} a_{1:N_{inp},1}(t) \ \cdots \ \bigcup_{t=t_s}^{T_{s,N_{outp}}} a_{1:N_{inp},N_{outp}}(t) \right) \tag{5.14}$$

The typical contentionless switching situations presented in Fig. 5.5 and Fig. 5.6 can be solved at every single time stage such that $\forall t, m, n\colon r_{n,m}(t) = a_{n,m}(t)$. The typical problem presented in Fig. 5.7 can be solved per output port basis, where at every output port the problem is solved at $t = T_{s,m}$ such that $\forall m\colon r_{1:N_{inp},m}(t_s) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$. The multicast contention problem presented e.g. in Fig. 5.3 must be solved per input-output basis because of the existing multicast requests.

Because the circulating arbitration order at every input port m is not uniform or independent from each other, then there will be many possible combinations of $A(t)$ at every time stage t. The arbitration time solution $T_{s,m}$ at every output $m \in \varphi$ may vary and depends on the number of requests N_m^{req} to the output port m ($T_{s,m} = N_m^{req}$). However, we can guarantee that, at the maximum

time of $t = T_f$ such that $\forall m\colon T_f = T_{s,m}^{max} = max(T_{s,m})$ or $T_f = max(T_{s,1}, T_{s,2}, \cdots, T_{s,N_{outp}})$, then the multicast deadlock dependency problem on each router is solved at T_f if there is no congestion in the outgoing links, $\because$ at $t = T_f \Rightarrow \forall m\colon \bigcup_{t=t_s}^{T_f} a_{1:N_{inp},m}(t) = \bigcup_{t=t_s}^{T_{s,m}} a_{1:N_{inp},m}(t)$.

Therefore, the conditional equation (Equ. 5.14) is fulfilled, and by following the Proofs of Lemma 4.1, Lemma 4.2 and Lemma 5.1 then the multicast deadlock and dependency problems on each router is solved without improper multicast flit replication. In other words, all requests depicted in $R(t_s)$ (at initial time t_s) will finally receive a grant acknowledge $A(T_f)$ in such a way that all flits appear at $t = t_s$ from all input ports $n \in \Phi$ would have been switched out to the output ports $m \in \varphi$ or would have been rescued from the multicast contention in the router in finite time stage T_f where $1 \leq T_f \leq N_{inp}$. If congestion occurs at any outgoing link then the solution is postponed for T_{wf} time stage. Hence, the problem is solved at $t = T_f + T_{wf}$, where T_{wf} is the number of time stages to wait for a free data slot available in the queue of the congested link connected directly to the most requested output port.

The descriptions given above have proved the Theorem 5.1 because the multicast problem can be solved in such a way that the contenting or conflicting multicast packet can be rescued from the multicast dependency in a router. If the multicast dependency (deadlock) problem can be solved on every router $R_c \in \Re$, then the network is free from multicast deadlock problem as long as the routing algorithm used to route unicast and multicast packets does not form cyclic dependencies. The detailed proof of the last statement can be found in [83], [58], [62].

T_f depends on the concrete multicast conflict situation in each router. For instance, in the multicast conflict case presented in Fig. 5.3, the flit coming from the $PORT\ 3$ port can be rescued from the multicast-dependency after generating one in-column bit-set combination of the arbitration matrix $A(1)$ as shown in Snapshot 2. While the flits coming from $PORT\ 1$ and $PORT\ 4$ ports can be rescued after generating two in-column bit-set combinations of the arbitration matrices $A(1)$, and $A(2)$ as shown in Snapshot 2 and Snapshot 4, respectively. From Fig. 5.3, we can see that for $N_{inp} = N_{outp} = 5$, then the numbers of request at every output port $m \in \{1, 2, 3, 4, 5\}$ are $N_{s,1}^{req} = N_{s,4}^{req} = N_{s,5}^{req} = 1$ and $N_{s,2}^{req} = N_{s,3}^{req} = 2$. Thus, $\forall m = \{1, 2, 3, 4, 5\}$ then $T_f = max(T_{s,1}, T_{s,2}, T_{s,3}, T_{s,4}, T_{s,5}) = max(1, 2, 2, 1, 1) = 2$. Therefore, the multicast dependency deadlock depicted in Fig. 5.3 can be solved in the next finite time stage $T_f = 2$. The rotating output selection per output port in four successive time stage of the problem shown in Fig. 5.3 is presented in the following tabular.

t	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
$1:$	$a_{3,1}(1)$	$a_{4,2}(1)$	$a_{1,3}(1)$	$a_{3,4}(1)$	$a_{4,5}(1)$
$2:$	$a_{3,1}(2)$	$a_{1,2}(2)$	$a_{4,3}(2)$	$a_{3,4}(2)$	$a_{4,5}(2)$
$3:$	$a_{3,1}(3)$	$a_{4,2}(3)$	$a_{1,3}(3)$	$a_{3,4}(3)$	$a_{4,5}(3)$
$4:$	$a_{3,1}(4)$	$a_{1,2}(4)$	$a_{4,3}(4)$	$a_{3,4}(4)$	$a_{4,5}(4)$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
	$T_{s,1} = 1$	$T_{s,2} = 2$	$T_{s,3} = 2$	$T_{s,4} = 1$	$T_{s,5} = 1$

Theorem 5.2 *If the multicast dependency and deadlock problems can be solved at each router as mentioned in Theorem 5.1, then multicast deadlock configurations in the network can be solved, if: 1) the routing algorithm used to route the unicast and multicast packets does not perform cyclic dependency, and 2) a data dropping mechanism at each outgoing communication link is applied to packets that cannot be assigned to an ID slot on the communication link.*

Proof of Theorem 5.2 *The Theorem 5.2 can be proved if Theorem 5.1 can be proved and the conditions mentioned in Theorem 5.2 are fulfilled. Therefore, we will explain and prove the need for the necessary conditions mentioned in the Theorem 5.2 as follow.*

1. *The necessary condition mentioned by item 1) in the Theorem 5.2 is needed because routing algorithm used to route unicast and multicast packets are the same according to the proposed hardware solution. Therefore, if the used routing algorithm does not perform cyclic dependency, then the proposed tree-based multicast routing is also free from deadlock configuration. The proof of the deadlock-freeness in term of the cyclic dependency problem is presented in detail in [83], [58] and [62].*

2. *The necessary condition mentioned by the item 2) in the Theorem 5.2 is required because if the data flits are not dropped, then they will stall in the router especially if they must wait for other messages to free one ID slot for a very long time. In this case, the data flits will be stagnant and occupy many buffers in the upstream channels, and do not give spaces for other messages to flow (chained blocking).*

5.4 Tree-based Multicast Router Implementation with Best-Effort Communication Protocol

Two specific routing behaviors of the tree-based multicast routing will be presented. The specific multicast routing behaviors which differentiate it from the unicast routing version are exhibited when the multicast header flits will program the routing reservation table autonomously and reserve a local ID slot on every communication link. The multicast routing presented in this section is the *best-effort* version of the multicast routing. The *guaranteed-service* version will be further explored in Chap. 7.

Fig. 5.8 presents the specific packet format that should be used to perform a deadlock-free multicast routing and to enable the hold-release tagging mechanism described in Section 5.3.1. A message or a streaming data is divided into several flow control digits called *flits*. The total bit-width of each flit of the message or streaming data is $b_{total} = b_{type} + b_{tag} + b_{word}$, where b_{type} is the bit-width of the flit type field, b_{tag} is the bit-width of the ID-tag field and b_{word} is the bit width of the data word. Together with a data word, each flit brings the two additional control bit-fields.

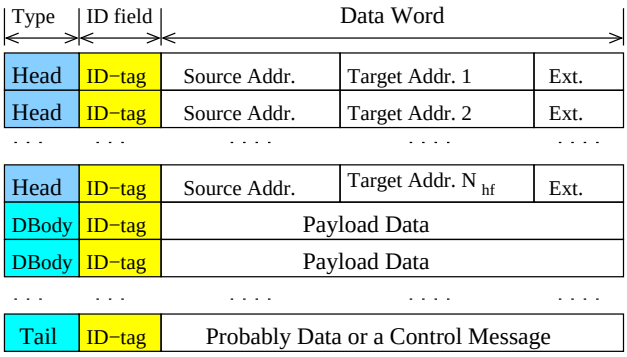


Fig. 5.8: Specific multicast packet format.

The number of header flit N_{hf} represents the number of the multicast destination N_{dest} ($N_{hf} = N_{dest}$). The header flits contains information of the source address from which node the message is injected and the target address to which nodes the message will be sent. If N_{df} number of data flits will be sent to N_{dest} number of multicast target, then the total number of flits injected to the NoC is $N_{Flit} = N_{df} + N_{hf} + [Tail]$.

5.4.1 Runtime Programming of Multicast Routing Reservation Table

Fig. 5.9 exhibits snapshots of the multicasting procedure, when three types of flits, i.e. header, databody and tail flits, are routed in the NoC router. The figure also presents how the routing reservation table (RRT) is programmed autonomously by the multicast header flits. Afterwards, the databody and tail flits will just follow the routing paths made by the header flits in every NoC router. For the sake of simplicity, only the *RRT* unit of the Western incoming port is presented in Fig. 5.9.

- 1^{st} *Header flit*. In Fig. 5.9(a), the first header flit is coming from the West (W) port with ID-tag 2. The first header flit is now being forwarded to the Local (L) port with the new ID-tag 0 (It is assumed that the packet is the first packet which uses the outgoing port. Hence, the packet header is allocated to the first free ID-slot, i.e. ID-tag 0). The Routing State Machine (*RSM*) unit has found an appropriate routing direction (the LOCAL direction in this case) and set the LOCAL slot of routing request slots in the register number 2 (in accordance with its ID-tag number) of the *RRT* unit.

- 2^{nd} *Header flit*. In Fig. 5.9(b), the second header flit is now being forwarded to the North (N) outgoing link with the new ID-tag 1. The *RSM* unit has found again an appropriate routing direction (the North output direction in this case) and set the East slot of routing request slots in the register number 2 of the *RRT* unit.

- 3^{rd} *Header flit*. The situation in Fig. 5.9(c) depicts the same mechanism as shown in the two previous snapshots, where in this case the third header flit is routed to the East (E) outgoing link with the new ID-tag 1.

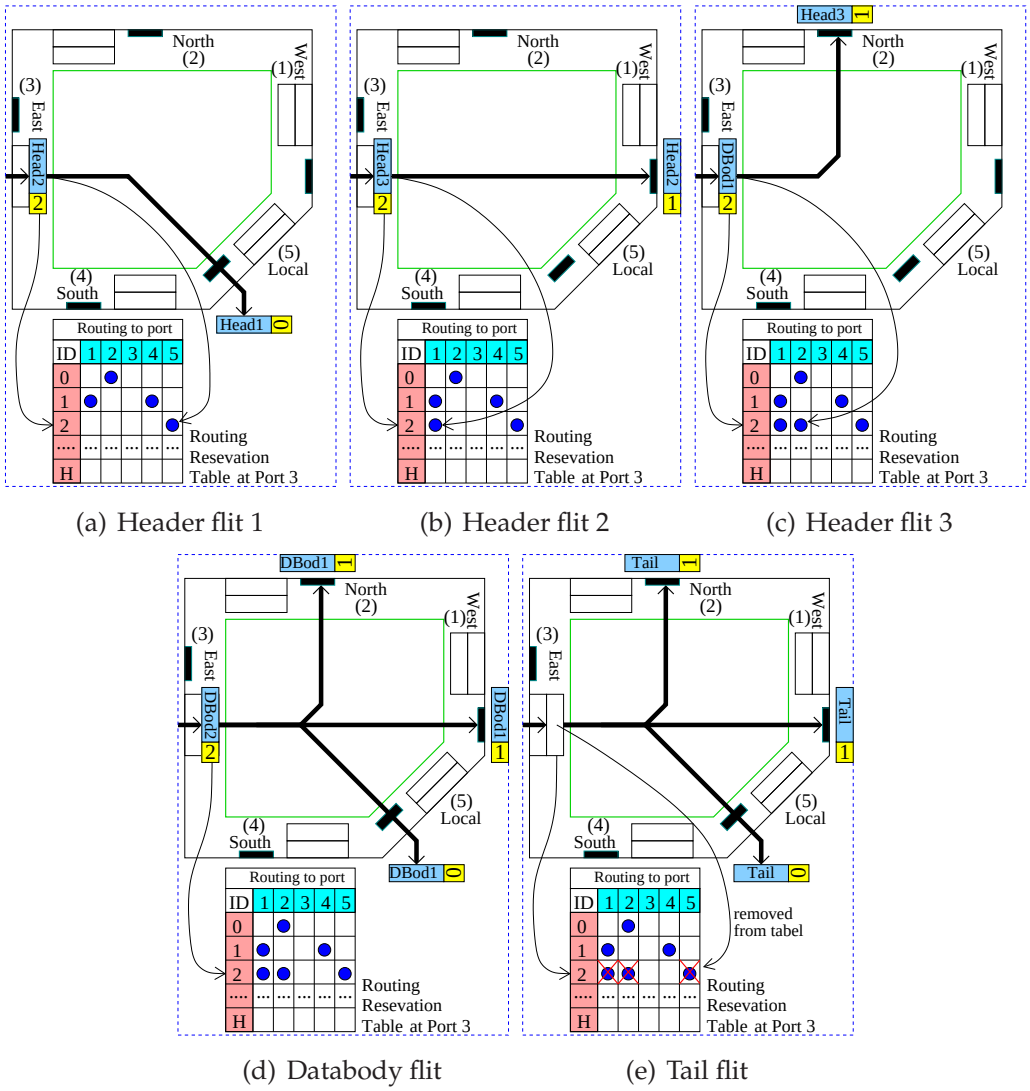


Fig. 5.9: Multicast Routing Phases.

- *Databody/Payload flit*. The register with index number 2 in the RRT shown in Fig. 5.9(d) has now three routing direction assignments in the routing request slots i.e., East (E), North (N) and Local (L). Therefore, all payload (databody) flits coming from the East port with ID-tag 2 will be forwarded simultaneously into the three outgoing links to track the paths that have been set up by the multicast header flits as shown in Fig. 5.9(d).

- As long as the routing paths set up by the header flits from a source to multiple (multicast) destinations have not closed or terminated by a special control flit called *tail* flit, then the reserved local ID slots along the routing paths can be still used by a processing element unit in the source node to send multicast data continuously. As presented in Fig. 5.9(e), a tail flit with ID-tag 2 has been routed simultaneously to the multiple output directions. Hence, the assignments of the routing request slots in the register number 2 of the RRT unit are removed from the table.

The routing engine algorithm for XHiNoC unicast version has been presented in Alg. 7 in Chap. 3. The routing engine algorithm for the XHiNoC multicast version is presented in Alg. 10. In Fig. 5.9, the mechanism to program the routing request slots of the RRT unit by the multicast header flits has been presented clearly. Alg. 10 presents the logical view to realize the multicast routing operation explained in the above items. In order to comprehend easily the operation in the Alg. 10, Def. 5.7 is given.

Definition 5.7 (Multicast Routing Reservation Slot) *A routing slot of a Multicast Routing Reservation Table of the RE unit at an input port of a multicast router is defined as*

$$T_{mcs}(k, r_{dir}) \in \{0, 1\} \tag{5.15}$$

where $k \in \Omega$ and $r_{dir} \in D = \{1, 2, 3, \cdots, N_{outp}\}$. The definition of the Multicast Routing Reservation Table can be still defined as $T(k)$, i.e. similar to the definition of the Routing Reservation Table as previously defined by Def. 3.11 in Chap.3. Therefore, we can further define $T(k)$ as a binary-element vector such that

$$T(k) = [T_{mcs}(k,1)\ \ T_{mcs}(k,2)\ \ T_{mcs}(k,3)\ \ \cdots\ \ T_{mcs}(k,N_{outp})]\,. \tag{5.16}$$

Hence, the routing reservation table T has 2D size of $row \times column = N_{slot} \times N_{outp}$.

Based on Def. 5.7, a binary-encoded multicast routing direction r_{dir}^{bin} is introduced and has a size of N_{outp} number of bits (binary elements). If we see the operation to program the RRT as presented in Fig. 5.9(a), Fig. 5.9(b) and Fig. 5.9(c), then it looks that the slot number 2 in the RRT is programmed based on Def. 5.7, when the header flits with ID-tag 2 are coming to the input port. The routing directions (and their binary-encoded value) made based on destination address information (A_{dest}) on the header flits are $r_{dir} = 5$ [0 0 0 0 1], $r_{dir} = 1$ [1 0 0 0 0] and $r_{dir} = 2$ [0 1 0 0 0], respectively. Therefore, the routing directions are written in the slot column numbers 5, 1 and 2 of the RRT slot row number 2 in accordance with the routing direction made for the headers and the ID-tag of the header flits, i.e. $T_{mcs}(2,5) = 1$, $T_{mcs}(2,1) = 1$ and $T_{mcs}(2,1) = 1$.

Thus, in conjunction with Def. 5.7, we obtain that $T(2) = [1\ 1\ 0\ 0\ 1]$. According to Def. 5.7, the number of the routing reservation row-column slots in the RRT is $(H+1) \times 5$.

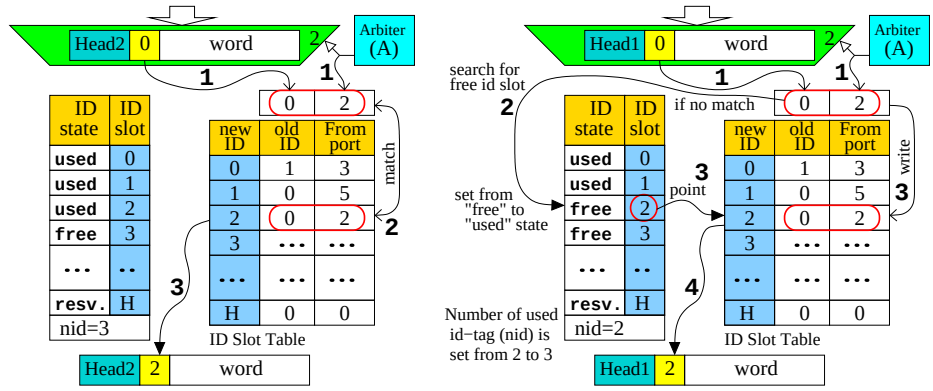
5.4.2 Runtime Multicast Local ID Slot Reservation

In this chapter, a specific behavior of the XHiNoC multicast version to update and manage the ID slot table is presented in Fig. 5.10. In Alg. 9, the operation of the the ID Update and Management has been presented in Chap. 3 for general XHiNoC version with unicast data communication protocol. In this chapter, the logical/algorithmical view to update and manage the ID slot table for XHiNoC version with multicast data communication service is presented again in Alg. 11. The main difference between both algorithms is shown in the ID update operation for the header flit type.

Alg. 10 Runtime ID-based Multicast Routing Mechanism

Read Data Flit from Queue : $F_n\ (type, ID)$

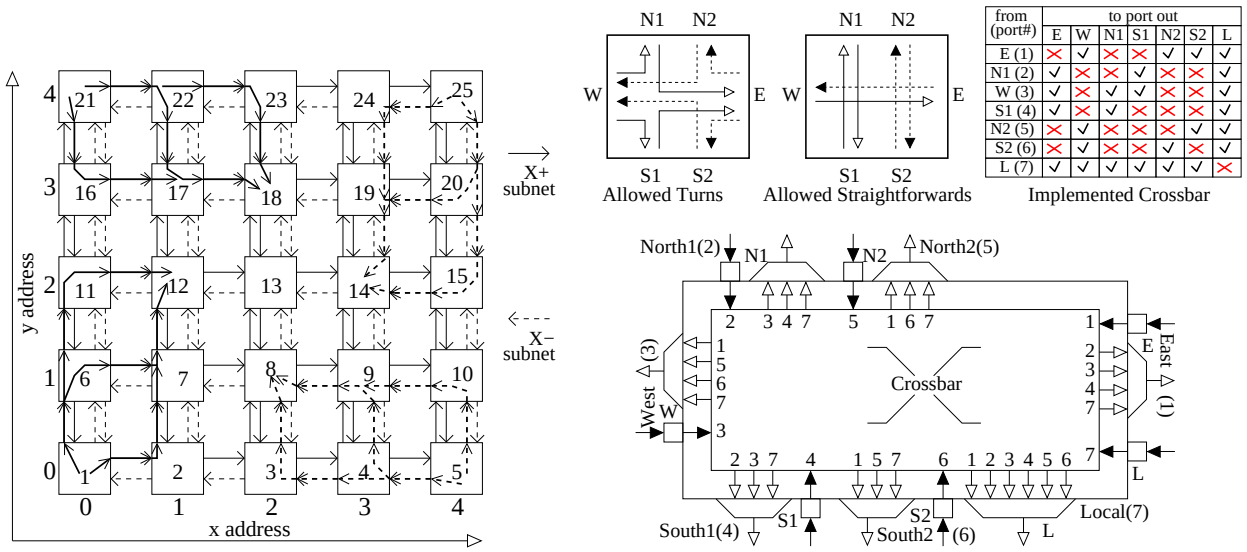
1: $F_{type} \Leftarrow type$
2: A_{dest} is obtained from Header flits
3: BEGIN $Multicast\ routing\ (r_{dir}^{bin})$
4: **if** F_{type} is $Header$ **then**
5: $\quad r_{dir} \Leftarrow f_{RSM}\ (A_{dest})$
6: $\quad T(ID, r_{dir}) \Leftarrow 1$
7: $\quad r_{dir}^{bin} = enc(r_{dir})$
8: **else if** F_{type} is $Response$ **then**
9: $\quad r_{dir} \Leftarrow f_{RSM}\ (A_{dest})$
10: $\quad r_{dir}^{bin} = enc(r_{dir})$
11: **else if** F_{type} is $Databody$ **then**
12: $\quad r_{dir}^{bin} \Leftarrow T(ID)$
13: **else if** F_{type} is $Tail$ **then**
14: $\quad r_{dir}^{bin} \Leftarrow T(ID)$
15: $\quad T(ID) \Leftarrow \emptyset$
16: **end if**
17: END $Multicast\ routing$



(a) A match information is found (b) A match information is not found

Fig. 5.10: Local ID-tag update for multicast header flits.

The conceptional of view of the ID-tag update for header flits of the multicast messages is shown in Fig. 5.10. When a leading multicast header is switched out to an output port, the information of the current ID-tag and the input port number from where the header comes will be checked in the ID Slot Table. If a match information is found then the new ID-tag can be fetched directly from the ID Slot Table. As shown in Fig. 5.10(a), a header flit with current ID-tag 2 from input port 2 finds the match information in slot number 2. Hence, it uses the ID slot number 2 as its new ID-tag. If a match is not found, then a free ID slot must be found from the Table. as presented in Fig. 5.10(b). ID slot number 2 is then found free and will be used as the new ID-tag for the header. Concurrently, the previous ID-tag and the input port number is written in the newly found slot number 2.



(a) Possible minimal adaptive routing

(b) Switch structure and turn model

Fig. 5.11: Mesh-Planar-based network and possible minimal planar adaptive routing paths.

This mechanism is run to guarantee that multicast headers belonging to the same multicast group will have the same ID-tag in order to keep the principle data flow regulation of the XHiNoC concept. The example ID update mechanisms show us indirectly that the first leading multicast header will always find firstly a free ID tag. The consecutive headers and payload data, which flow on the same link and belong to the same multicast group with the first leading header, will then just fetch the new ID-tag by indexing the ID Slot Table with the matching information.

5.5 Adaptive Tree-based Multicast Routing

Alternative approach to provide a higher degree of routing adaptiveness is by using a 2D planar adaptive routing. The planar adaptive routing algorithm is firstly introduced in [46]. The main difference between the method shown in [46] and our current NoC router implementation is the replacement of virtual channels with double-physical channels connection connecting North and South input-output ports of the router. The planar network architecture will be explored in the following subsection.

5.5.1 2D Planar Adaptive Routing Algorithm

Fig. 5.11(a) shows a 2D mesh-planar topology, where the NoC is divided into two subnets, i.e. X^+ (increment) subnetwork in solid lines and X^- (decrement) subnetwork. The links of the X^+ and X^- subnets are described in solid and dashed lines, respectively. If the $x_{offset} = x_{target} - x_{source} \geq 0$, packets will be routed through the X^+ subnetwork, while

Alg. 11 Runtime Local ID-tag Update for Multicast Routing

Outgoing Data Flit : $F_n\,(type, ID)$
Input Arbitration : $n = \{1, 2, \cdots, N_{inp}\}$
N_{usedID} : number of used/reserved ID slots
$N_{slot} - 1$: Slot reserved for control purpose ($N_{slot} - 1 = H$)

$F_{type} \Leftarrow type$; $ID_{old} \Leftarrow ID$; $F_{from} \Leftarrow n$
BEGIN $ID\ Update$
if F_{type} is $Header$ **then**
  if $ID_{old} = N_{slot} - 1$ **then**
    $ID_{new} \Leftarrow N_{slot} - 1$
  else if $ID_{old} \neq N_{slot} - 1$ **then**
    for $k = 0$ to $k = N_{slot} - 2$ **do**
      if $\exists k\colon S(k) = (ID_{old}, F_{from})$ **then**
        $ID_{new} \Leftarrow k$
      else if $\nexists k\colon S(k) = (ID_{old}, F_{from})$ **then**
        for $k = 0$ to $k = N_{slot} - 1$ **do**
          if $\exists k\colon k \neq N_{slot} - 1\colon S^k$ is true **then**
            $S(k) \Leftarrow (ID_{old}, F_{from})$; $S^k \Leftarrow false$ /* the ID Slot is used now */
            $ID_{new} \Leftarrow k$; $N_{usedID} \Leftarrow N_{usedID} + 1$
          else
            $ID_{new} \Leftarrow N_{slot} - 1$
          end if
        end for
      end if
    end for
  end if
else if F_{type} is $Databody$ **then**
  for $k = 0$ to $k = N_{slot} - 2$ **do**
    if $\exists k\colon S(k) = (ID_{old}, F_{from})$ **then**
      $ID_{new} \Leftarrow k$
    else if $\nexists k\colon S(k) = (ID_{old}, F_{from})$ **then**
      $ID_{new} \Leftarrow \emptyset$; The $Databody$ flit is dropped
    end if
  end for
else if F_{type} is $Tail$ **then**
  $N_{usedID} \Leftarrow N_{usedID} - 1$
  for $k = 0$ to $k = N_{slot} - 2$ **do**
    if $\exists k\colon S(k) = (ID_{old}, F_{from})$ **then**
      $ID_{new} \Leftarrow k$; $S(k) \Leftarrow (\emptyset, \emptyset)$; $S^k \Leftarrow true$ /* the ID Slot is now free */
    else if $\nexists k\colon S(k) = (ID_{old}, F_{from})$ **then**
      $ID_{new} \Leftarrow \emptyset$; $S(k) \Leftarrow (\emptyset, \emptyset)$; $S^k \Leftarrow true$ /* the ID Slot is now free */
      The $Tail$ flit is dropped
    end if
  end for
else if F_{type} is $Response$ **then**
  $ID_{new} \Leftarrow N_{slot} - 1$
end if
$ID_{new} \Rightarrow ID$
END $ID\ Update$

if the $x_{offset} \leq 0$, packets will be routed through the X^- subnetwork. Once a packet is routed to a subnetwork, it will not move to another subnet. By using such routing rule, the minimal planar adaptive routing algorithm will be free from cyclic dependency (free from deadlock configuration).

The main advantage of the this NoC topology architecture compared with the turn models approach commonly used in the standard-mesh structure is that minimal routing adaptivity can be made in all non-zero offset directions with maximal two alternative routing directions. For examples as shown in Fig. 5.11(a), when the target is located in North-East area (node 1 to 12), South-East area (node 21 to 18), North-West area (node 5 to 8), South-West area (node 25 to 14), then packets can form three alternative paths adaptively.

Alg. 12 presents the 2D planar adaptive routing algorithm used for the 2D mesh planar multicast router. The routing algorithm is divided into two subrouting codes for X^+ and X^- subnetwork in the 2D mesh planar topology. In the X^+ Subnet, the set of output ports that can be selected are $\{EAST, SOUTH\ 1, NORTH\ 1, LOCAL\}$. In the X^- Subnet, the set of output ports that can be selected are $\{WEST, SOUTH\ 2, NORTH\ 2, LOCAL\}$.

5.5.2 Inefficient Spanning Tree Problem

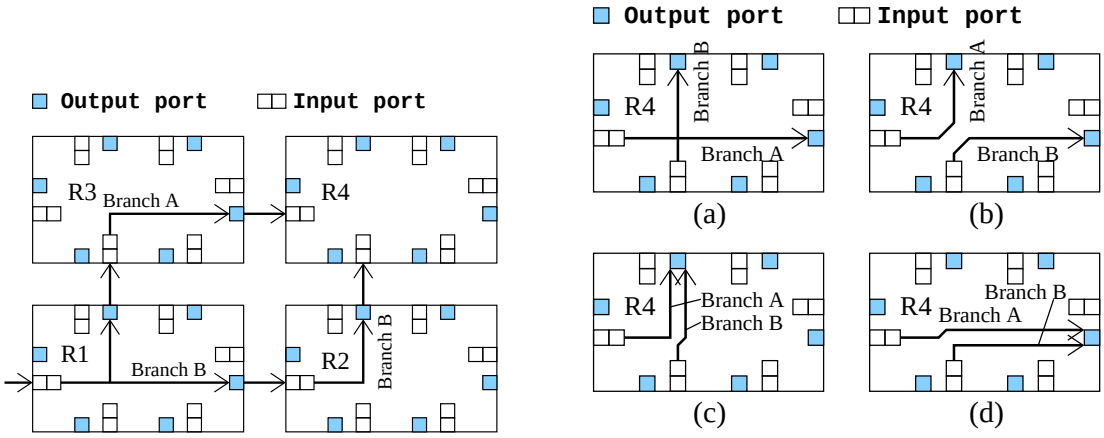
An efficient adaptive multicast routing is required to optimize communication energy. Fig. 5.12(a) shows an example of an inefficient tree-based adaptive multicast routing. A tree-based multicast message coming from the WEST input port of the router $R1$ forms two crossing branches in different routing direction i.e., a branch to NORTH (branch A) and a branch to EAST (branch B) direction. We can assume that the branches A and B are made by header flit 1 and header flit 2, respectively, which belongs to the same multicast message that will be routed to multicast destinations (x_{t1}, y_{t1}) and (x_{t2}, y_{t2}), respectively. In the router $R3$, the multicast message is routed from SOUTH to EAST (branch A). While in the router $R2$, the multicast message is routed from WEST to NORTH (branch B). Hence, these two branches are then routed to the same router (router $R4$). In this case, the multicast tree branches are inefficient in term of communication energy. The communication energy can be reduced if the router $R1$ performs only the multicast tree branch A or branch B.

Fig. 5.12(b) shows 4 possible situations which occur in the router $R4$ as the further disadvantageous consequences of the inefficient multicast tree branches formed in Fig. 5.12(a). The situations could happen because the number of free ID slots on each communication link as the parameter of the adaptive routing algorithm may change dynamically. Fig. 5.12(b)(a) and (b) show tree-branch crossing problems, in which the inefficient paths of the branches are further routed in different outgoing ports. If we assume that the current address of router $R4$ is (x_{curr}, y_{curr}) and the target nodes of the tree branches A and B are (x_{t1}, y_{t1}) and (x_{t2}, y_{t2}) such that $x_{offset1} = x_{t1} - x_{curr} > 0$ and $y_{offset1} = y_{t1} - y_{curr} > 0$ as well as $x_{offset2} = x_{t2} - x_{curr} > 0$ and $y_{offset2} = y_{t2} - y_{curr} > 0$, then in any circumstance,

Alg. 12 2D Planar Adaptive Routing Algorithm

Network is partitioned into two subnets: X^+ and X^- Subnet.
Set of output ports in X^+ Subnet: $\{EAST, SOUTH\ 1, NORTH\ 1, LOCAL\}$.
Set of output ports in X^- Subnet: $\{WEST, SOUTH\ 2, NORTH\ 2, LOCAL\}$.
Select(m_1, m_2) is selection function between output port m_1 or m_2.

1: $X_{offs} = X_{target} - X_{source}$
2: $Y_{offs} = Y_{target} - Y_{source}$
3: **while** $Packet$ is in $Subnet\ X^+$ i.e.$(X_{offs} \geq 0)$ **do**
4: **if** $X_{offs} = 0$ and $Y_{offs} = 0$ **then**
5: $Routing = LOCAL$
6: **else if** $X_{offs} = 0$ and $Y_{offs} > 0$ **then**
7: $Routing = NORTH\ 1$
8: **else if** $X_{offs} = 0$ and $Y_{offs} < 0$ **then**
9: $Routing = SOUTH\ 1$
10: **else if** $X_{offs} > 0$ and $Y_{offs} = 0$ **then**
11: $Routing = EAST$
12: **else if** $X_{offs} > 0$ and $Y_{offs} > 0$ **then**
13: $Routing$=**Select**$(NORTH\ 1, EAST)$
14: **else if** $X_{offs} > 0$ and $Y_{offs} < 0$ **then**
15: $Routing$=**Select**$(SOUTH\ 1, EAST)$
16: **end if**
17: **end while**
18: **while** $Packet$ is in $Subnet\ X^-$ i.e.$(X_{offs} \leq 0)$ **do**
19: **if** $X_{offs} = 0$ and $Y_{offs} = 0$ **then**
20: $Routing = LOCAL$
21: **else if** $X_{offs} = 0$ and $Y_{offs} > 0$ **then**
22: $Routing = NORTH\ 2$
23: **else if** $X_{offs} = 0$ and $Y_{offs} < 0$ **then**
24: $Routing = SOUTH\ 2$
25: **else if** $X_{offs} < 0$ and $Y_{offs} = 0$ **then**
26: $Routing = WEST$
27: **else if** $X_{offs} < 0$ and $Y_{offs} > 0$ **then**
28: $Routing$=**Select**$(NORTH\ 2, WEST)$
29: **else if** $X_{offs} < 0$ and $Y_{offs} < 0$ **then**
30: $Routing$=**Select**$(SOUTH\ 2, WEST)$
31: **end if**
32: **end while**



(a) Inefficient multicast branches (b) Crossing and interference branches

Fig. 5.12: Inefficient branches of multicast tree problem.

the inefficient tree branch might happen again in the next intermediate nodes.

Fig. 5.12(b)(c) and (d) exhibit tree-branch interference problem, in which the inefficient branches are further interferred into the same outgoing port. This situation will lead to inefficient multicast communication time (increase of communication latency) because of the self-contention problem. Two multicast messages will be forwarded from different input ports to the same output port but with regard to contents, the message is similar.

5.5.3 Solution for the Inefficient Spanning Tree Problem

The problems presented in Fig. 5.12(a) and Fig. 5.12(b) are not only inefficient in terms of communication energy because the inefficient path overburden the NoC, but also it can degrade the data rate of the multicast traffics. Thus, it reduces the NoC performance while increases power consumption.

We solve the aforementioned problem not by designing a specific multicast path optimization algorithm that should be run at compile time. The path optimization algorithms such as optimal spanning tree algorithm are only suitable for *source routing approach*, where routing paths for the overall paths of a multicast message from source to destination node are made at source node before the message is injected to the network. In our NoC, the routing algorithm used to route unicast and multicast messages is the same, and the routing functions are distributed locally on every port of each router. Hence, we do not implement the path optimization algorithm for initiation-time-efficiency purpose.

In order to avoid such problem, each time a routing engine has two alternative output ports to make a routing decision, then a selection strategy between two alternative output ports is made. The simple abstract view of the adaptive selection strategy is presented in the Alg. 13. The basic concept of the proposed algorithm is the identification of track records of other previously-routed header flits that belong to the same multicast message in order to find the energy-efficient routing branches of the multicast tree. The logical

Alg. 13 Multicast Adaptive Routing Selection Strategy (Abstract view)

1: Begin Function **Select**($port\ m_1$, $port\ m_2$)
2: **if** $Routing$ can be made to $port\ m_1$ and $port\ m_2$ **then**
3: **if** $Routing$ for the same $Multicast\ Packet$ has been made to $port\ m_1$ and not yet to $port\ m_2$ **then**
4: Return $Routing = port\ m_1$
5: **else if** $Routing$ for the same $Multicast\ Packet$ has been made to $port\ m_2$ and not yet to $port\ m_1$ **then**
6: Return $Routing = port\ m_2$
7: **else if** $Routing$ for the same $Multicast\ Packet$ has not been made to $port\ m_1$ and $port\ m_2$, or has been made both to $port\ m_1$ and $port\ m_2$ **then**
8: **if** $UsedID(port\ m_1) < UsedID(port\ m_2)$ **then**
9: Return $Routing = port\ m_1$
10: **else**
11: Return $Routing = port\ m_2$
12: **end if**
13: **end if**
14: **end if**
15: End Function

implementation of the abstract view of the proposed adaptive selection strategy to avoid inefficient spanning tree (branches of tree) is presented in Alg. 14. In the algorithm, it looks like routing decision is made based on the contents of the routing reservation slots and the number of used (reserved) ID slots in the two alternative output ports.

Alg. 14 Multicast Adaptive Routing Selection Strategy (Logical view)

Incoming Data Flit : $F_n\ (type, ID)$
$T(k, m)$: Routing reservation slot of a flit with ID=k to direction m
$UsedID(m)$: Number of used/reserved ID slots in direction m
1: Begin Function **Select**(m_1, m_2)
2: $k \Leftarrow ID$
3: **if** $\neg T(k, m_1)\ \&\ UsedID(m_1) \leq \neg T(k, m_2)\ \&\ UsedID(m_2)$ **then**
4: Return m_1
5: **else if** $\neg T(k, m_1)\ \&\ UsedID(m_1) > \neg T(k, m_2)\ \&\ UsedID(m_2)$ **then**
6: Return m_2
7: **end if**
8: End Function

Before running a real experiment, Fig. 5.13 will show us the different routing paths that will be performed by the aforementioned tree-based multicast routing methods (except for the wf-v1 multicast method). As presented in the figure, a multicast message is injected from node (2,2) to 10 multicast destinations. The 'xy' multicast router performes 24 traffics in the NoC. While the 'plnr' and 'wf-v2' multicast routers perform only 19 and 21 traffics in the NoC, respectively. It means that the planar adaptive multicast router can potentially reduce the communication energy of the multicast data transmission. The following experiment will show us the result of a more complex data distribution scenario.

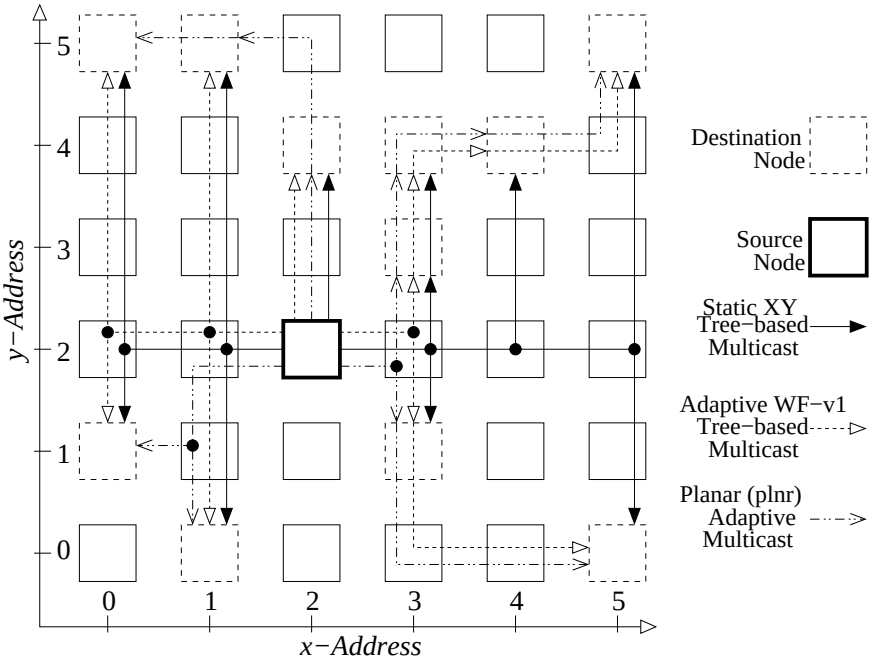


Fig. 5.13: The traffic formations by using static tree-based, minimal adaptive west-first and minimal planar adaptive multicast routing methods.

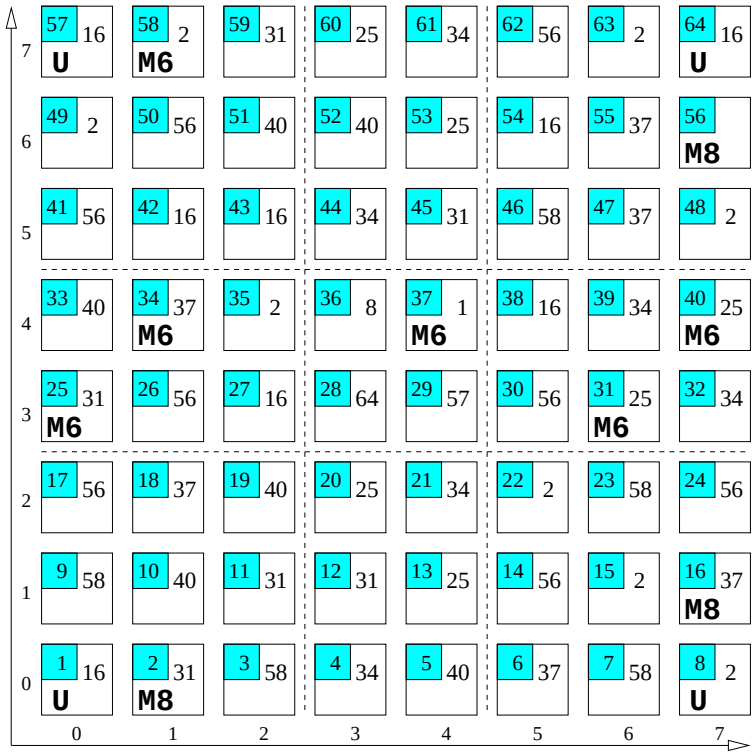
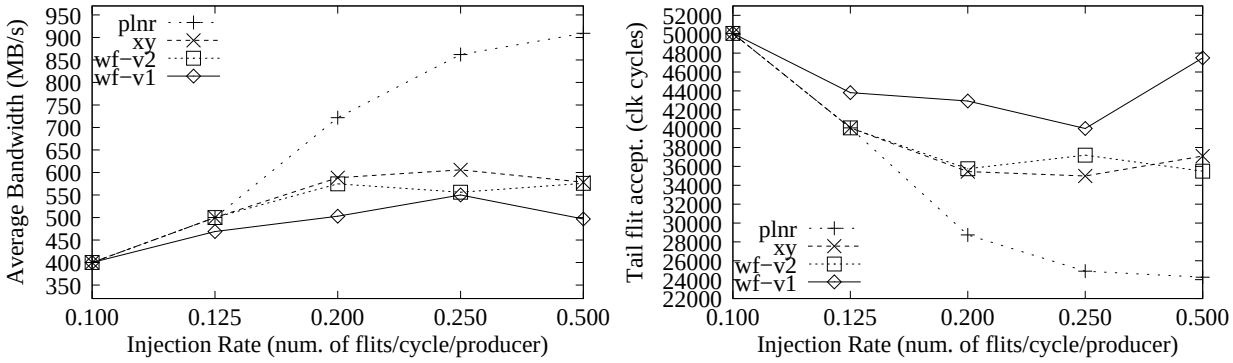


Fig. 5.14: Distribution of the source-destination communication partners.

Tab. 5.1: Unicast and Multicast communication groups for the random multicast test traffic scenario.

Comm. Group	Type	source	targ. 1	targ. 2	targ. 3	targ. 4	targ. 5	targ. 6	targ. 7	targ. 8
Comm. 1	M6	(1,4)	(6,4)	(7,3)	(4,2)	(3,0)	(3,5)	(4,7)	–	–
Comm. 2	M6	(7,4)	(0,4)	(3,6)	(2,6)	(2,2)	(1,1)	(4,0)	–	–
Comm. 3	M6	(0,3)	(6,3)	(3,2)	(4,1)	(7,4)	(4,6)	(3,7)	–	–
Comm. 4	M6	(6,3)	(0,3)	(4,5)	(3,1)	(2,1)	(1,0)	(2,7)	–	–
Comm. 5	M6	(1,7)	(7,6)	(5,5)	(6,2)	(6,0)	(2,0)	(0,1)	–	–
Comm. 6	M6	(4,4)	(1,4)	(1,2)	(7,1)	(5,0)	(6,5)	(6,6)	–	–
Comm. 7	M8	(1,0)	(7,0)	(6,1)	(5,2)	(2,4)	(7,5)	(6,7)	(0,6)	(1,7)
Comm. 8	M8	(7,1)	(2,3	(5,4	(2,5	(1,5)	(5,6)	(0,7)	(7,7)	(0,0)
Comm. 9	M8	(7,6)	(1,6)	(0,5)	(5,3)	(1,3)	(0,2)	(5,1)	(5,7)	(7,2)
Comm. 10	U	(0,0)	(4,4)	–	–	–	–	–	–	–
Comm. 11	U	(0,7)	(4,3)	–	–	–	–	–	–	–
Comm. 12	U	(7,0)	(3,4)	–	–	–	–	–	–	–
Comm. 13	U	(7,7)	(3,3)	–	–	–	–	–	–	–



(a) Average Bandwidth

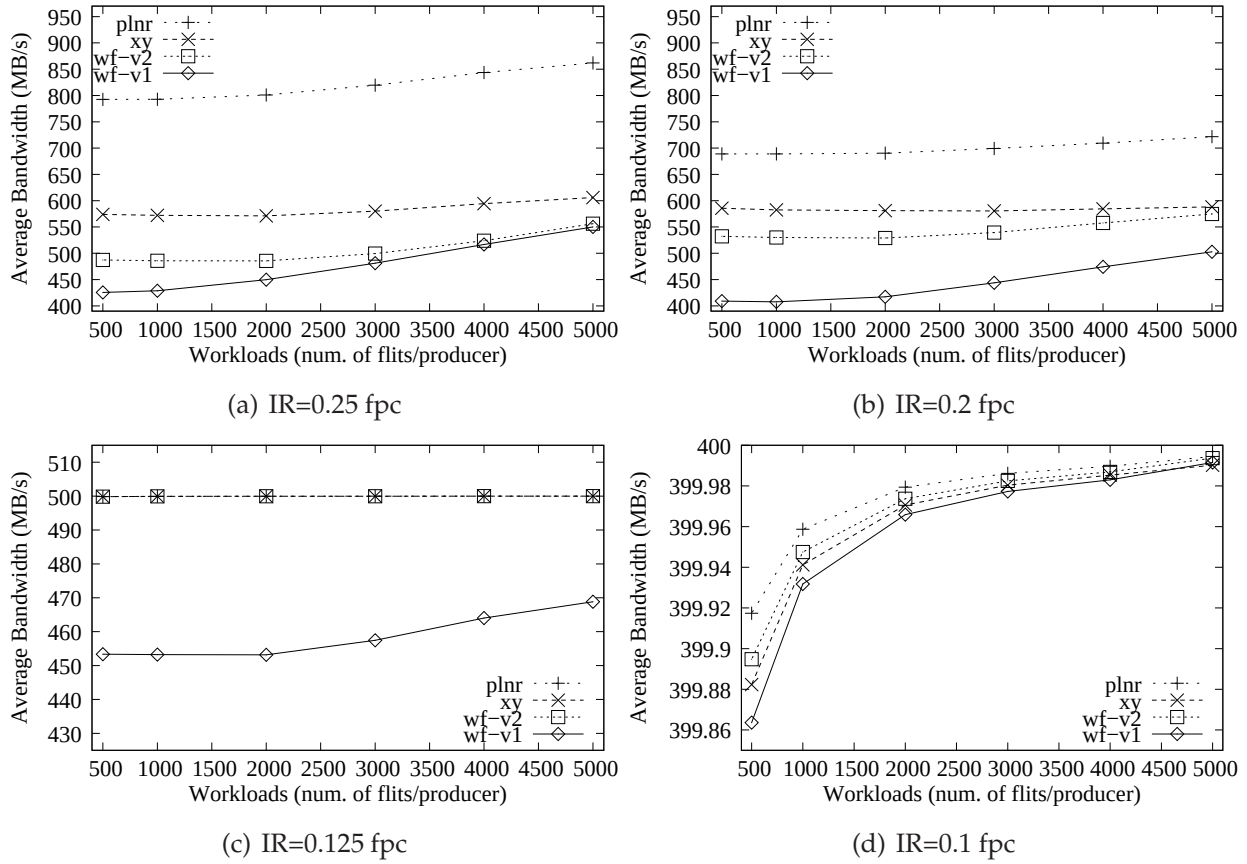
(b) Average Tail flit latency

Fig. 5.15: Average bandwidth and tail flit acceptance latency measurement versus expected data injection rates for multicast random test scenario.

5.6 Experimental Result

In the experimental results presented in this section, four XHiNoC multicast router prototypes with different multicast routing algorithms were compared. The first prototypes is the multicast router with planar adaptive routing algorithm in the mesh planar NoC architecture, which is presented with **'plnr'** acronym in the figures. The second prototypes uses XY static multicast routing algorithm in the mesh standard architecture (**'xy'**). The third and fourth prototypes are the multicast routers in the standard mesh architecture with adaptive West-First (WF) routing algorithm (**'wf-v1'** and **'wf-v2'**). The adaptive WF multicast router version 1 (**'wf-v1'**) is the multicast router without the implementation of the adaptive selection strategy to avoid inefficient spanning tree (branches of the multicast tree). Thus, the multicast trees are formed freely without considering the track records of the other previously-routed header flits belonging to the same multicast group. The adaptive WF multicast router version 2 (**'wf-v1'**) implements the adaptive selection strategy presented in the Alg. 14 to avoid such inefficient spanning tree problem.

The experiment is set up by using a multicast random data distribution (traffic) scenario to verify the theorem and methodology of the proposed deadlock-free multicast



(a) IR=0.25 fpc (b) IR=0.2 fpc

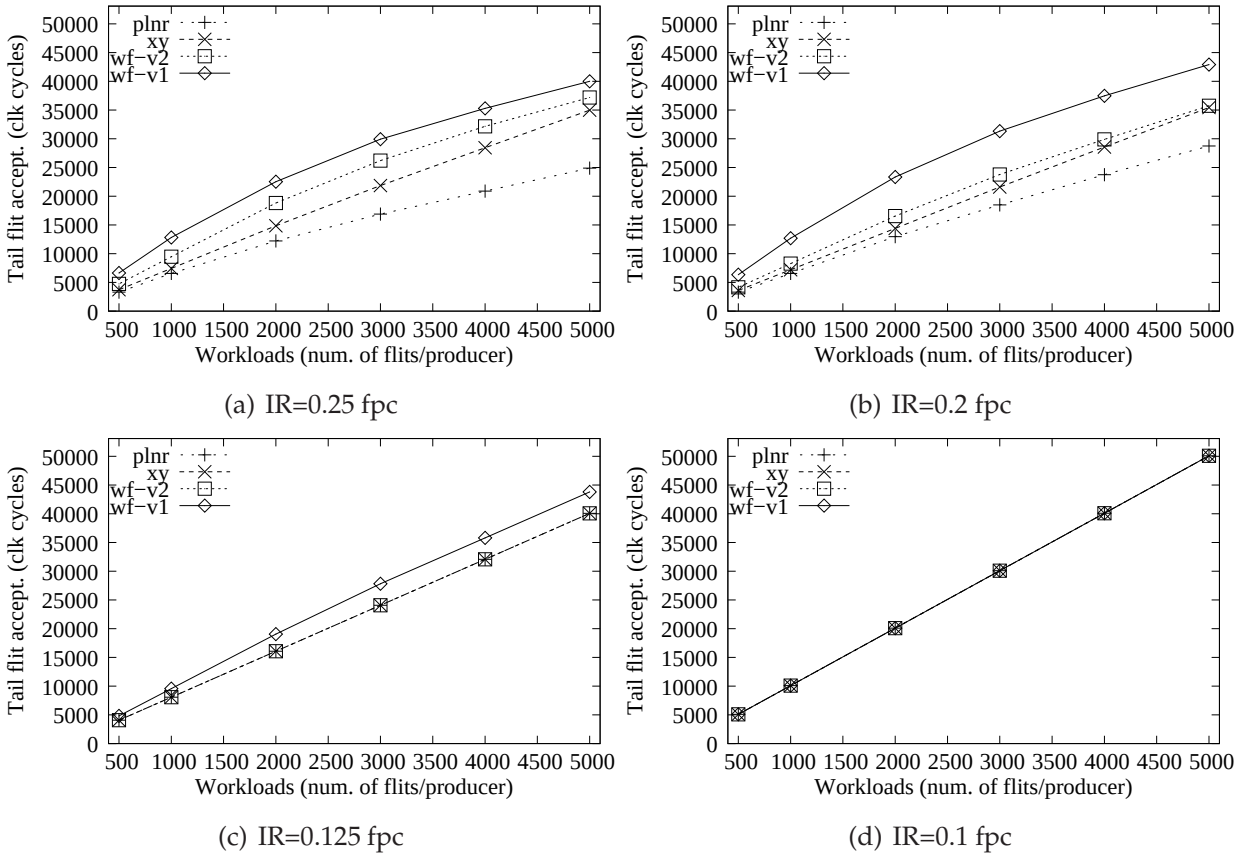
(c) IR=0.125 fpc (d) IR=0.1 fpc

Fig. 5.16: Average actual bandwidth versus workloads for multicast random test scenario.

routing. Fig. 5.14 shows the distribution of the source-destination unicast-multicast communication partners in the 2D 8×8 mesh architecture (64 network nodes). The numerical symbol in the bottom and the left sides of the mesh network represent the 2D node x-address and y-address. In the figure, it looks like 9 multicast communication partners (**M6**, **M8**) and 4 unicast communication pairs (**U**) are presented. Three of 9 multicast communication sources have 8 multicast destinations (**M8**), while the remaining six multicast sources have 6 multicast targets (**M6**).

Every NoC node in Fig. 5.14 is depicted with a square block together with the numerical symbols. A numerical symbol in the small square block at the top-left side of a NoC router node represents the node number of the node. The numerical symbol at the top-right side in the NoC router node represents the communication partner of the node, from which the NoC router node will receive a message. For example, the network node at node address $(2, 1)$ (2D node address, 2 is the x-horizontal address and 1 is the y-vertical address) has node number 11. At the right side in the mesh node 11, we see the numerical value 31. It means that the mesh node number 11 located in the node address $(2, 1)$ will receive packet from mesh node 31 located in the node address $(6, 3)$.

The boldface symbols (**U**, **6** and **M8**) at the bottom-left of the white-colored box represent that the network node will send a unicast message (**U**) or a multicast message with a number of 6 target nodes (**M6**) or 8 target nodes (**M8**). For example, the mesh node



(a) IR=0.25 fpc

(b) IR=0.2 fpc

(c) IR=0.125 fpc

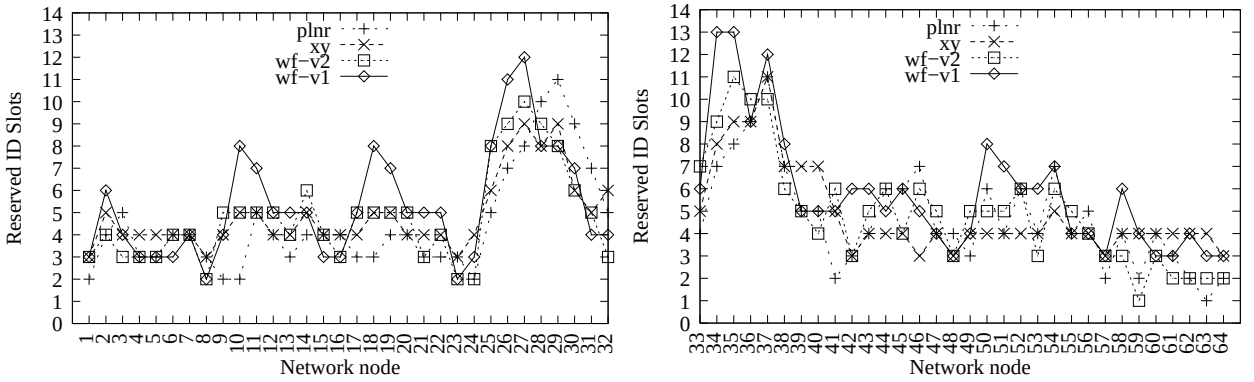
(d) IR=0.1 fpc

Fig. 5.17: Tail flit acceptance latency versus workloads for multicast random test scenario.

address $(7, 1)$ (mesh node number 16) is symbolized with (**M8**). It means that this mesh node will send a multicast message into 8 destination nodes. We can find the target nodes of the multicast message sent from the mesh node number 16 by looking for mesh nodes having numerical symbol 16 at the right-side in each mesh node. In order to find easily the partners of each unicast and multicast communications, Table 5.1 presents the unicast and multicast communication partners/groups of the source-destination distribution presented in Fig. 5.14.

The measurements of the average bandwidth and tail flit acceptance latency with various expected data injection rates are presented in Fig. 5.15. The measurements are made for five different expected data injection rates, i.e. 0.1, 0.125, 0.2, 0.25 and 0.5 flits/cycle (fpc). It looks that the tree-based multicast router with planar adaptive routing algorithm shows the best performance both in terms of the average bandwidth (Fig. 5.15(a)) and the average latency of the tail flits acceptance (Fig. 5.15(b)) for all expected data rates. If the expected data injection rates are very low (e.g. $0.1\ fpc$), then the performance of the multicast routers will be the same. The performance of the planar adaptive multicast router will be significantly better compared to the other multicast routers when the data are expected to be transmitted with higher data rates.

Fig. 5.16 presents the average actual bandwidth measurements for different sizes of workload per data producer nodes with different expected data injection rates. When the



(a) Node 1–32 (b) Node 32–64

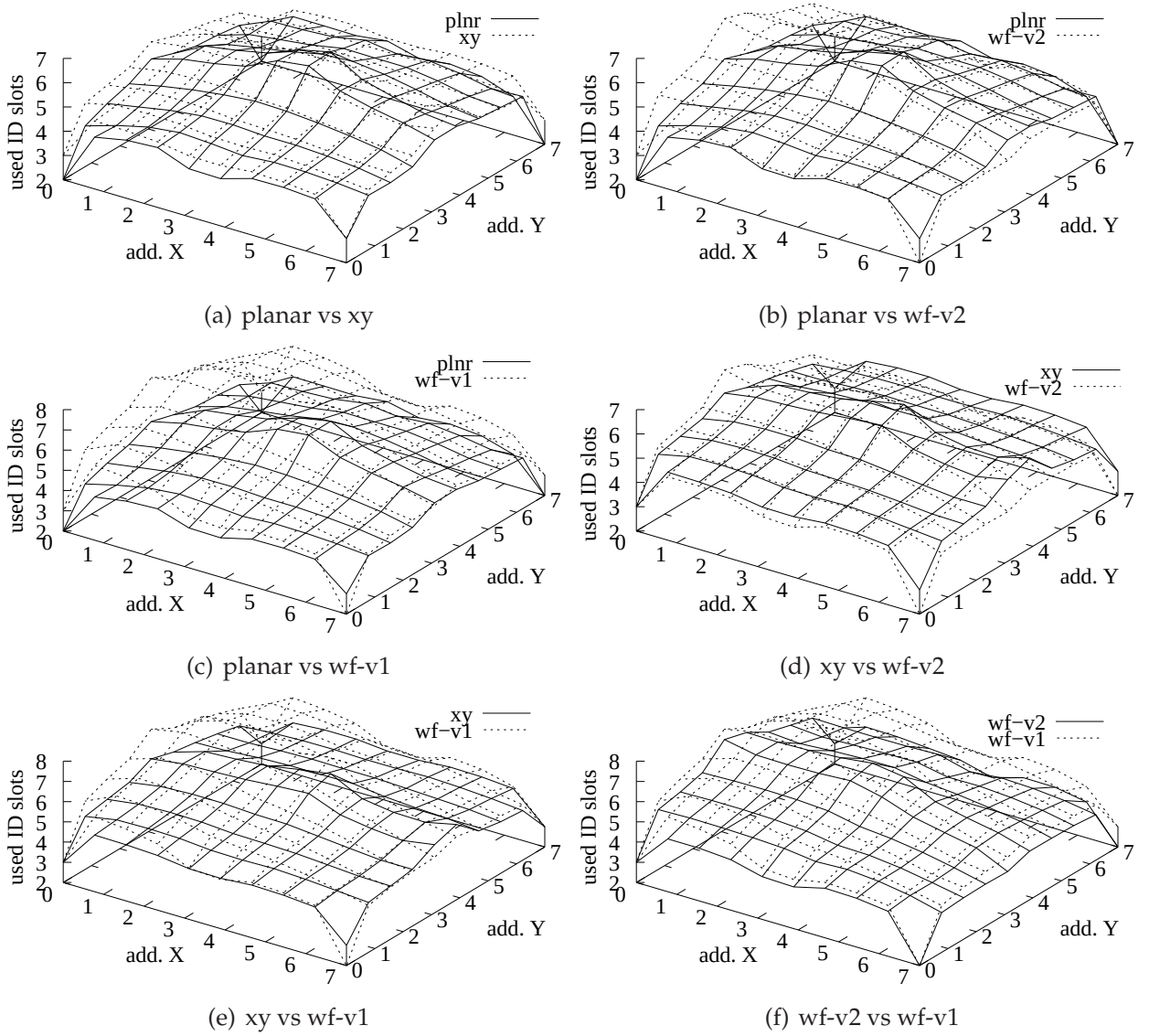
Fig. 5.18: Reserved (used) total ID slots for multicast random test scenario.

expected data injection rates are relatively fast (approximate the maximum allowable data injection rate, $0.5\ fpc$), then differences of the actual bandwidth measurements between the four different multicast routing algorithms are significant. When the expected data injection rates are slower, then the performances of the four tree-based multicast routing algorithms are nearly similar.

As presented in Fig. 5.16(d), when the expected injection rate is about $0.1\ fpc$ (flits per cycle) for every data producer node, then the average actual measured communication bandwidth is almost similar for all tree-based multicast routing algorithms, i.e. about $399.86 - 399.99\ fpc$. In general, the average bandwidth for different workload sizes (with fixed expected data rates per producer node) tends to move in a steady value. The tendency will be more significant when the expected data rates are slower such that the NoC will not be saturated.

Fig. 5.17 presents the average tail flit acceptance latency for different sizes of workload per data producer nodes with different expected data injection rates. In line with the results presented in Fig. 5.16, when the expected data injection rates are relatively fast, then differences of the tail flits acceptance latency measurements between the four different multicast routing algorithms are also significant. The average latency of the tail flits acceptances of the different tree-based multicast routings also tends to reach a steady value when the expected data injection rate per producer node is slower such that the NoC is not saturated.

Fig. 5.17(d) shows that the average latency of the tail flit acceptance of the different multicast routings are similar for various workload sizes when the expected data rate is about $0.1\ fpc$. We can see that the similar average tail flit acceptance latency of all proposed tree-based multicast routing algorithms exhibited in Fig. 5.17(d) is due to the closely similar average communication bandwidth shown in Fig. 5.16(d) The average latencies tend to increase linearly if the workload sizes are increased for each multicast routing algorithms in the non-saturating condition, i.e. in the case of slower expected data rates.



(a) planar vs xy

(b) planar vs wf-v2

(c) planar vs wf-v1

(d) xy vs wf-v2

(e) xy vs wf-v1

(f) wf-v2 vs wf-v1

Fig. 5.19: 3D views of the total ID slot reservation on every NoC router for multicast random test scenario.

Fig. 5.18 shows the 2D view of the total ID slot reservations on every NoC router node. Fig. 5.18(a) presents the total reserved ID slots for the NoC router node 1 until node 32, while Fig. 5.18(a) presents the total ID slots reservation for the NoC router node 33 until node 64. The 3D views of the reserved ID slots distribution are shown in Fig. 5.19. Each figure presents the comparison of the ID slots distribution between two tree-based multicast routing algorithms. For example, Fig. 5.19(a) exhibits the comparison between the tree-based planar adaptive multicast routing ('plnr') and the tree-based static multicast routing algorithm ('xy'), and Fig. 5.19(f) exhibits the comparison between the tree-based west-first adaptive multicast routing algorithm with ('wf-v2') and without ('wf-v1') selection strategy to avoid inefficient branches of tree-based multicasting.

As shown in Fig. 5.19(a), the ID slot reservation of the tree-based multicast router by using static XY routing algorithm is almost uniform. This traffic distribution can be even

Tab. 5.2: Total performed traffics on each link direction for different tree-based static and adaptive multicast routing methods.

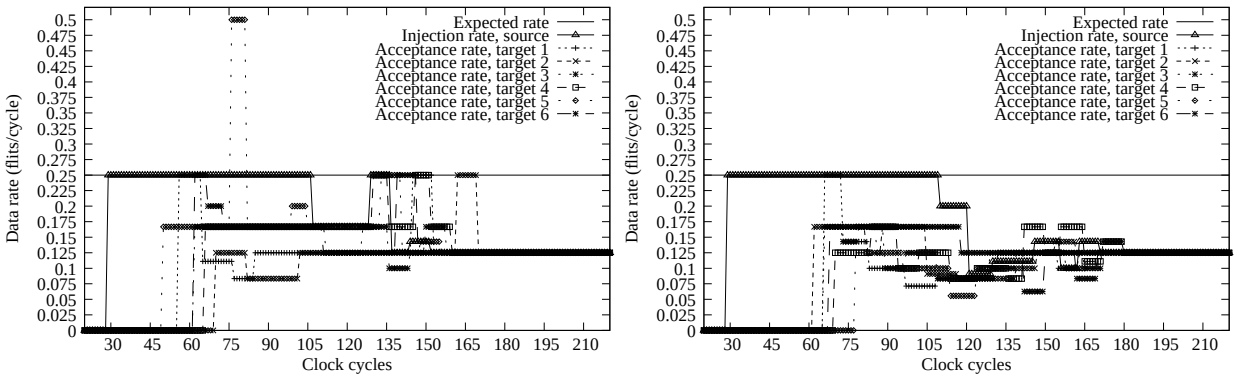
Routers	South2	North2	South	West	North	East	TOTAL
plnr	30	28	34	65	27	46	230
xy	–	–	83	40	89	36	248
wf-v2	–	–	79	40	80	46	245
wf-v1	–	–	85	40	81	86	292

predicted and can be estimated easily. In general, the tree-based multicast routers with adaptive routing algorithms tends to move the traffic into the center area in the NoC. This characteristic can be overviewed through the distributions of the ID slot reservation on each router in the NoC. However, the ID slot reservation patterns are strongly dependent on the patterns of the given traffic scenarios.

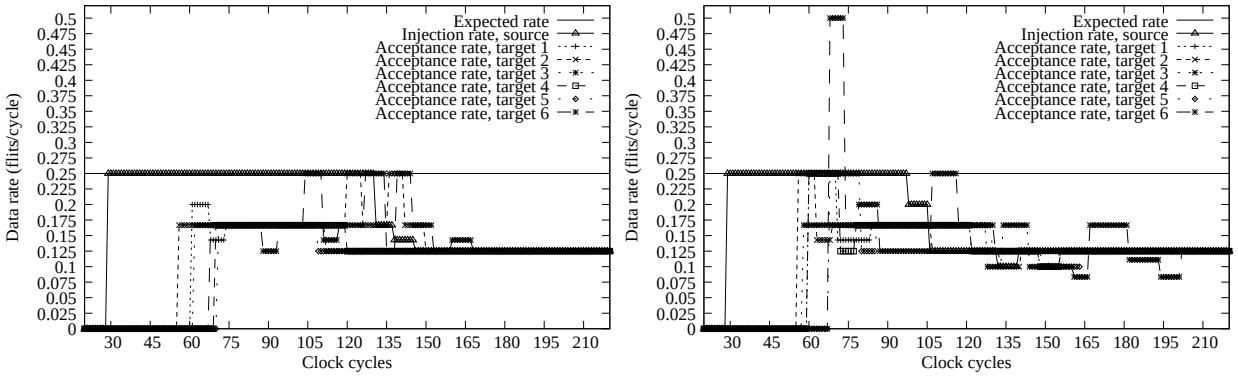
Table 5.2 shows the comparisons of the total performed communication traffics between the four tree-based multicast router prototypes. The number of the traffics represents the number of communication resources (communication links) used to route the unicast/multicast message from source to destination nodes. Therefore, this performance metric can be a representation of the communication energy of the evaluated multicast routers. The total traffic on each link direction for the four different multicast routing implementation are also presented in the table. It seems that the planar adaptive multicast router ('plnr') consumes less communication resources than the other multicast routers, i.e. about 230 communication links followed by the adaptive multicast routing with efficient spanning tree method ('wf-v2'), and then the static tree-based multicast router that uses XY routing algorithm ('xy').

As shown in Table 5.2, the multicast router with minimal adaptive routing algorithm, without using the technique to avoid inefficient spanning tree ('wf-v1'), performs the largest number of traffic, i.e. 292 traffics. This value is even greater than the static tree-based multicast router, i.e. 248 performed traffics. By using the same adaptive routing algorithm with the technique to avoid the inefficient spanning tree ('wf-v2'), the total number of traffics is reduced to 245 traffics, which is also more efficient than the multicast router with static XY routing. The reduction of the total traffic denotes the effectiveness of the solution to avoid the inefficient spanning tree as explained in Section 5.5.3.

In Chap. 3, the implementation of the link-level flit flow control together with the automatic injection rate control mechanism have been explained. By using such control mechanisms, the average actual injection and acceptance rates at each source-destination node (end-to-end communication flow) can be kept equivalent in order to guarantee loseless data delivery. Fig. 5.20 present the transient response measurements of the actual injection rate at source node and actual acceptance rates at multicast target nodes for multicast communication 1, 2, 3 and 4 by using tree-based multicast router with static XY routing algorithm. The four multicast communications have six multicast destinations



(a) Multicast Comm. 1 (b) Multicast Comm. 2



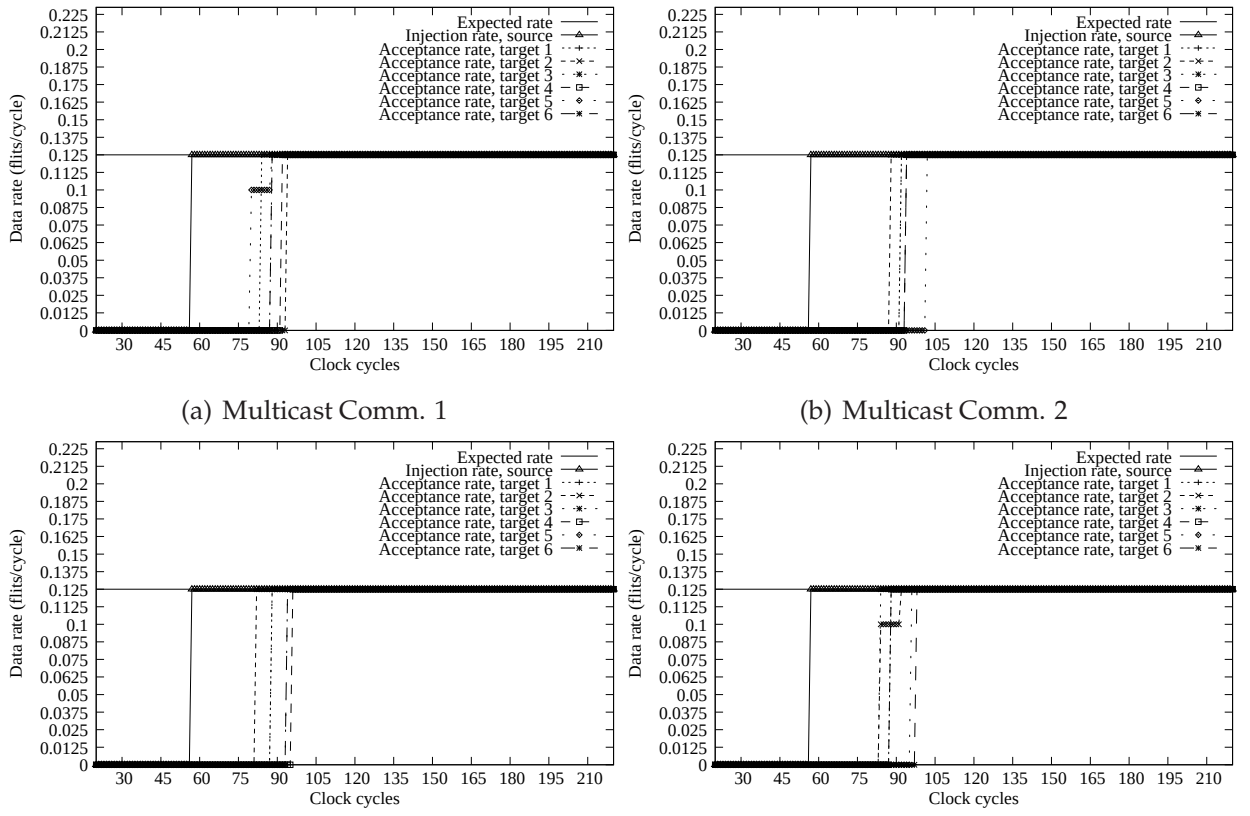
(c) Multicast Comm. 3 (d) Multicast Comm. 4

Fig. 5.20: Expected, actual injection rate at source node, and actual acceptance rates at multicast target nodes during NoC saturating condition by using the static tree-based multicast router (Expected injection rate is 0.25 flits/cycle).

respectively. The transient responses are measured until about the 213^{rd} clock cycles. The expected injection rate is $0.25\ fpc$, where with such expected data rate, the four selected multicast communication partners in the NoC will be in saturating condition.

The saturating condition is generally due to the contentions between several multicast messages in the intermediate nodes. In this situation, the actual measured acceptance rates at the six multicast destination nodes will be slower than the expected one. Therefore, the actual measured injection rates at the source nodes will be automatically reduced to a steady point following the actual measured data acceptance rates at the multicast destination nodes. As presented in the figures, the actual measured injection rate at the source node and the actual acceptance rates at the destination nodes of each multicast communication are reduced to a steady state point of about $0.125\ fpc$, which is lower than the expected rate point ($0.25\ fpc$).

As presented in Fig. 5.20, before the actual measured injection and acceptance rates are steady to the stable point, the injection and acceptance rates change dynamically in transient time. The dynamical responses in the transient time can be seen as the contention behaviors of the traffic. As presented in Fig. 5.20(a), for example, a high overshot is shown by the acceptance rate at the multicast target number 5 of the multicast Com-



(a) Multicast Comm. 1

(b) Multicast Comm. 2

(c) Multicast Comm. 3

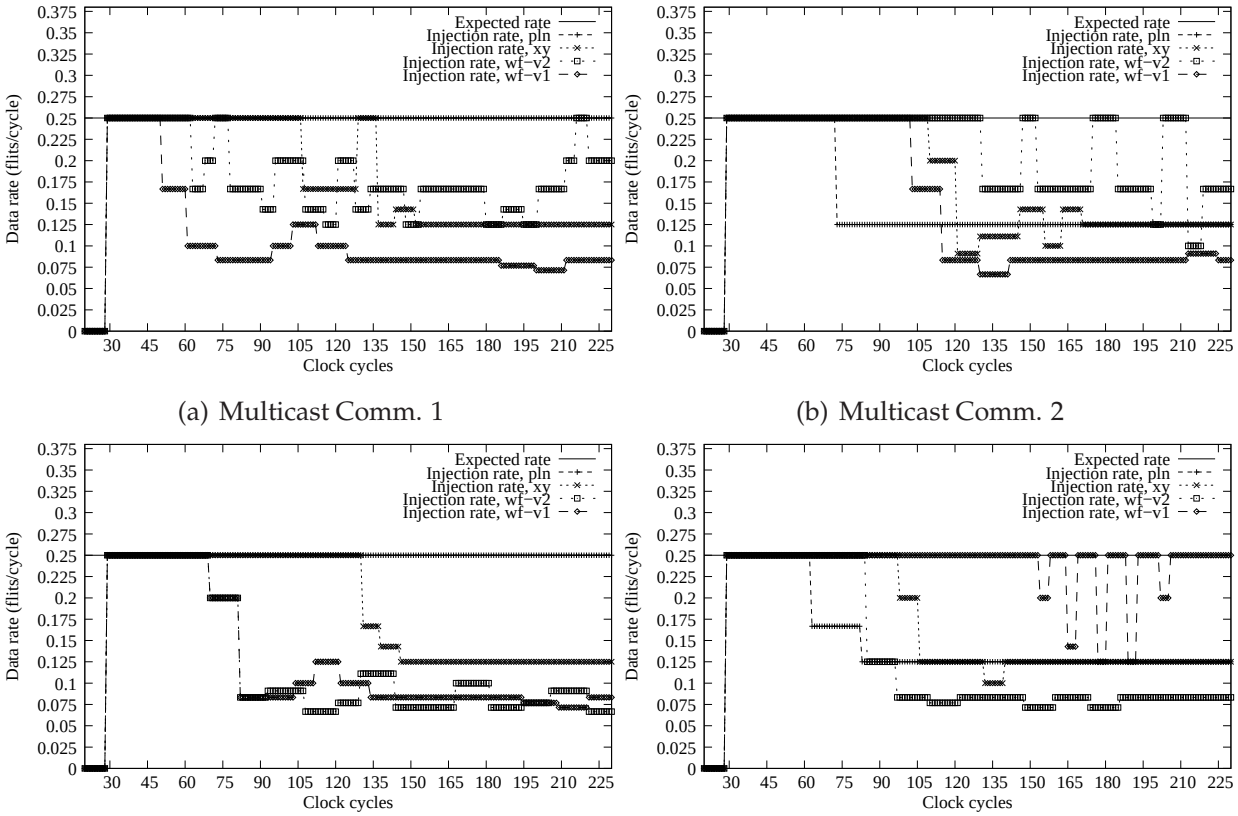
(d) Multicast Comm. 4

Fig. 5.21: Expected, actual injection rate at source node, and actual acceptance rates at multicast target nodes during NoC non-saturating condition by using the static tree-based multicast router (Expected injection rate is 0.125 flits/cycle).

munication 1 during about 6 cycle periods. The same overshot condition is shown by the acceptance rate at the multicast target number 6 of the multicast Communication 4 as shown in Fig. 5.20(d).

Fig. 5.21 present the other transient response measurements of the actual injection rate at source node and actual acceptance rates at multicast target nodes for multicast communication 1, 2, 3 and 4 by using tree-based static XY multicast routing. The expected injection rate in this case is $0.125\ fpc$, where with such expected injection rate, the NoC will be not saturated. In this situation accordingly, no contention occurs between the four selected multicast communication partner groups, or the four selected multicast communication groups do not have contention with traffics of other multicast communication groups, which are in saturating points. Therefore, the actual measured injection rates at the source nodes can follow the actual measured data acceptance rates at the multicast destination nodes. As presented in the figures, all actual injection and acceptances rates at the six multicast destination nodes move to a steady state value of about $0.125\ fpc$, which is similar to the expected data injection rate.

The comparison of the actual measured data injection rates between the multicast router prototypes for four selected multicast communication groups is shown in Fig. 5.22.



(a) Multicast Comm. 1

(b) Multicast Comm. 2

(c) Multicast Comm. 3

(d) Multicast Comm. 4

Fig. 5.22: Comparisons of the actual injection rates at source nodes for different routing algorithms during NoC saturating condition (Expected injection rate is 0.25 flits/cycle).

The comparison is made for actual injection rate transient responses with an expected data rate of $0.25\ fpc$. For the multicast communication 1 and 3 (Fig. 5.22(a) and Fig. 5.22(c)), the actual injection rate of the planar adaptive multicast router can follow the expected data rate. For the multicast communication 1 and 2 (Fig. 5.22(a) and Fig. 5.22(b)), the WF multicast router version 1 ('wf-v1') has the worst steady-state response. However, as presented in the Fig. 5.22(d), the WF multicast router version 1 shows the best performance, because its steady state injection rate moves around the expected data rate.

5.7 Synthesis Results

The logic synthesis results of the four XHiNoC tree-based multicast router prototypes, i.e. the static XY tree-based multicast router (*xy*), the minimal adaptive West-First tree-based multicast router (*wf-v1*), the minimal adaptive West-First tree-based multicast router with efficient spanning tree solution (*wf-v2*) and the minimal 2D planar adaptive tree-based multicast router (*plnr*) are presented in Table 5.3. The logic synthesis of the multicast routers is made by setting the target working frequency to 1 GHz. It appears that the area overhead of the tree-based multicast router prototype with the planar adaptive routing

Tab. 5.3: Synthesis results of the multicast routers using 130-nm CMOS technology library.

	xy	wf-v1	wf-v2	plnr
Target frequency	1 GHz	1 GHz	1 GHz	1 GHz
Total logic cell area	0.1089 mm^2	0.1133 mm^2	0.1168 mm^2	0.1378 mm^2
Est. net switch. power	14.652 mW	16.555 mW	17.149 mW	21.335 mW
Est. cell intern. power	40.330 mW	42.180 mW	43.999 mW	53.960 mW
Est. cell leakage power	21.60 μW	22.20 μW	23.50 μW	25.90 μW

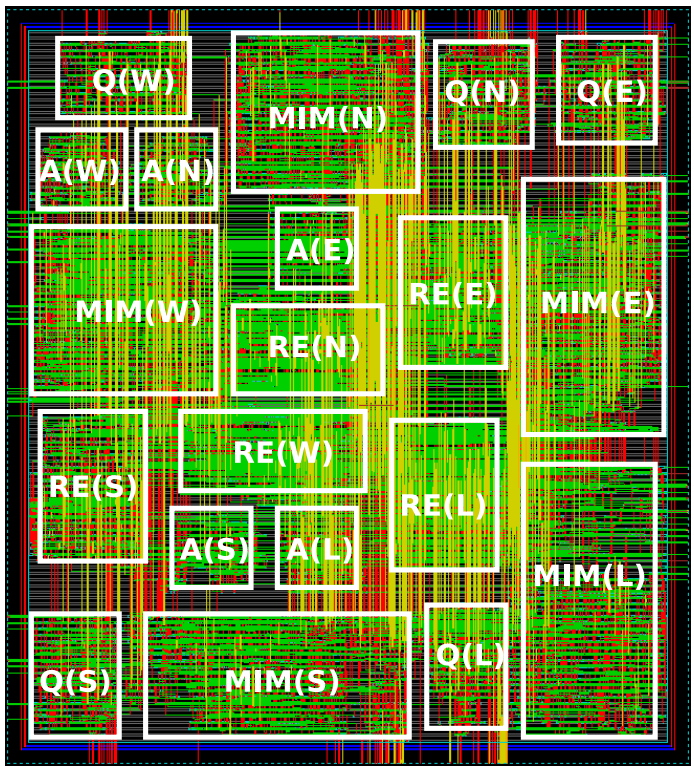


Fig. 5.23: Circuit layout of the XHiNoC router with tree-based XY multicast routing using CMOS standard-cell technology library.

algorithm is higher than the other multicast routers.

Fig. 5.23 shows the circuit layout (logic cell placement and wire routing) of the tree-based multicast router with static XY routing algorithm. The circuit layout is made by using 180-nm CMOS standard-cell technology with block partitioning method. The components for each port presented in the circuit layout floorplan are the FIFO queues/buffers (*Q*), the multiplexor with ID management (*MIM*), the routing engine with single buffer (*RE*) and the arbiter unit (*A*). Each component is placed in the circuit floorplan according to the port location of the mesh-based router. We can see in the figure that the multiplexor with ID management (*MIM*) components dominate the total area of the router. The arbiter (*A*) units occupy the smallest area of the overall circuit floorplan.

5.8 Summary

This chapter has presented the XHiNoC router prototypes capable of routing unicast and multicast messages using static or adaptive routing algorithms. The messages are routed with the deadlock-free tree-based multicast method, where the routing engines used to route the unicast and the multicast messages are the same, resulting in an efficient routing algorithm gate-level implementation. A new theory for deadlock-free multicast routing has also been introduced in this chapter. Formal definition and description for the VLSI architecture and implementation of the deadlock-free tree-based multicast routers have been given in this chapter.

The performance evaluation result under the random selected mixed unicast-multicast traffic has presented that the tree-based multicast router with planar adaptive routing algorithm shows the best performance over the static tree-based multicast router and the adaptive tree-based multicast routers with the minimal West-First routing algorithm. Certainly there are still many traffic patterns that can be used to verify the latency, throughput and communication energy (the number of the performed traffics in the network).

The tree-based multicast routing presented in this thesis is a class of distributed routing, in which routing decisions are made locally on every switch node based on the node destination address in the multiple header probes of the message packets. Hence, the technique to prevent inefficient spanning tree problem presented in this thesis will probably result in a *suboptimal* or *near-optimal* multicast spanning tree, or in certain cases, an *optimal* spanning tree may be obtained. When the adaptive routing algorithms are used, then the spanning tree formed independently at runtime by the header probes would be probably different if the order of the header probes is different. When the static tree-based multicast routing is used, then the formed multicast spanning tree will be the same although the order of the header probes is changed.

A preprocessing algorithm before sending the multicast traffics to prepare a certain order of the header flits that can perform optimal spanning tree in the network can be further developed. However, this algorithm will lead to computational time and energy overheads. Further investigation and analysis on the computational time overhead of the preprocessing algorithm and its impacts on communication energy of the sub-tree in the optimal performed spanning tree should be investigated. However, in some cases, the gain of the communication energy efficiency due to the preprocessing algorithm may be lower than its computational time and energy overhead.

Chapter 6

Contention- and Bandwidth-Aware Adaptive Routing Algorithm

Contents

Adaptive routing selection strategies for a wormhole switched network-on-chip (NoC) based on bandwidth (BW) space occupancy, congestion information (FIFO buffer occupancy), and contention information through the reservation numbers of communication links are presented in this chapter. Motivations behind the implementations of adaptive routing algorithms in NoCs are described in Section 6.1. State-of-the-art of several adaptive routing selection strategies is shown in Section 6.2. Basically, the BW-oriented adaptive routing selection strategy can be combined with the other strategies such as contention-based or queue-length-based strategy.

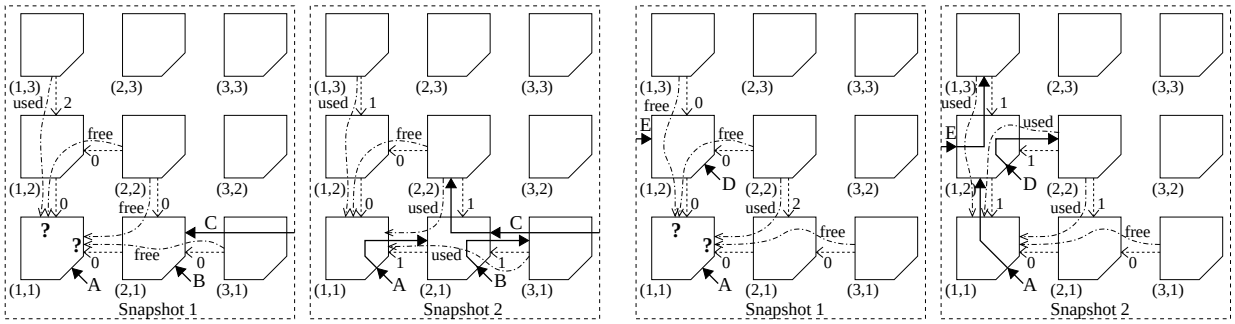
Section 6.3 in particular will show a contention- and bandwidth-aware (CBWA) adaptive routing selection strategy that makes routing decisions by considering two information between alternative output ports. The adaptive routing decisions are made at runtime by considering the bandwidth spaces that have been reserved so far by other messages/streams as well as the number of messages/streams that have currently acquired the the alternative output ports. The number of messages acquiring the output port is reflected by the number of local identity (ID) slots that has been reserved by messsages routed through the output port. The routing engines in the NoC router read both signals and route messages/streams into an output port having occupied less BW spaces and ID slots. Micoarchitectures and routing function algorithms of the NoC routers with different selection strategies are also presented in Section 6.3.

Experimental results under two selected traffic scenarios with different network sizes are presented in Section 6.4. The experimental results will evaluate five NoC router prototypes having different adaptive routing selection strategies, which have been explored previously in Section 6.3. The synthesis results of the five adaptive NoC routers are presented in Section 6.5.

6.1 Motivation Behind Adaptive Routing Implementation

Network designers are motivated to design adaptive routing algorithms due to two main objectives, i.e. to avoid entering hotspot links, and to avoid entering faulty network components (faulty switch or link). The works in [172], [140] and [163] for instance, propose fault-tolerance adaptive routing algorithms. Network faults can lead a regular network to a non-regular network. The work in [135] presents a fault tolerance routing algorithm by balancing traffics over network faults and non-regularity due to the network component faults. The work in [194] presents a deadlock-free adaptive routing method to cover the problem of oversized IP components placement in irregular mesh-based network.

The adaptive routing methods, which are aimed at increasing the degree of the message routing adaptiveness in such a way that hotspot in the network can be reduced, and network performance can be increased accordingly, have been presented in the literature. Based on routing locality, routing paths (routing decisions) can be made by in source nodes by running a pre-processing algorithm to compute the set of routing paths (source/centralized routing) or made locally in every switch node (local/distributed routing). Based on routing setup time, routing algorithms can be set up at design-time or at runtime. Most of routing implementations made at design time uses routing tables to route messages (packets). The contents of the routing tables are programmed at design time, and then adaptive routing paths are assigned in every routing table in the network nodes by using some technique. The work in [178] for example, presents a design-time routing method called "Application-Specific Routing Algorithm" (APSRA) used to increase the degree of routing adaptivity for hotspot avoidance. The "Segment-based Rout-



(a) Unpredictable traffic problem 1 (b) Unpredictable traffic problem 2

Fig. 6.1: Problem in the unpredictable two-hop neighbor-on-path congestion measurement.

ing" (SR) presented in [153] can be classified also into an adaptive routing implementation at design-time, in which the network is segmented into some subnets and restrictions are applied to avoid deadlock configurations. The dynamic routing protocol in [144] for balancing distribution of traffics in NoCs can also be classified into the design-time routing approach.

In look-up-table-based routing algorithms, the size of the tables will increase as the network size increases, since all entries must be added in the tables. Some works then propose different techniques to reduce the size of the routing tables. The work in [154] presents a region-based routing algorithm aimed at reducing the size of routing tables for NoCs by grouping destination network into network regions. The work in [118] shows a simple data transfer technique by applying local addresses (labels), which are computed off-line for each traffic in an application (source (design-time) routing approach). With the same background mentioned in [118], where traffics in embedded MPSoC are predictable, our proposed methodology can also be implicitly viewed as a technique to reduce the number of entries in the routing tables based on runtime variable (dynamic) local message identity (ID) technique that will be explained later. In our experiments, (see Section 6.4.2 for example), all considered traffics can still be routed under several scenarios, although the number of available ID slots per link is set less than the number of node entries in the NoC. Our methodology can be classified into runtime distributed routing approach, where the routing is made locally in every NoC router at runtime during application execution time.

6.2 State-of-the-Art in Adaptive Routing Strategy

6.2.1 Selection based on FIFO Queue Occupancy

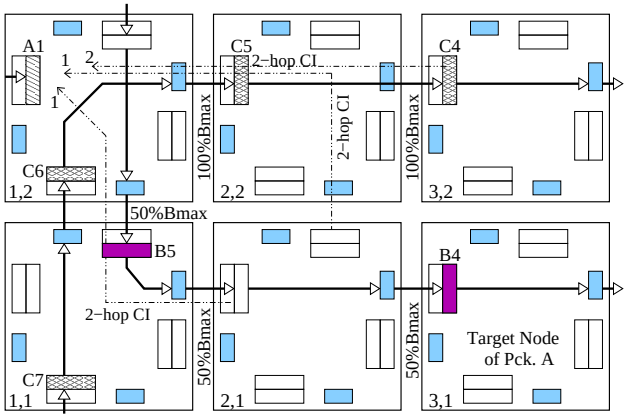
A commonly used adaptive routing policy is based on buffer occupancy, where the "congestion information" (CI) of a set of possible admissible output ports connected to the downstream (next-hop) routers are traced back to upstream routers. The CI information

can be the length of data queues in the FIFO buffer that can be indicated by multiple-bit signal, or the buffer status (free or busy) that can be indicated by single-bit signal. These CI signal will be used by a packet on a current router to select one best routing direction between alternative downstream outgoing link at any instant time. A "stress value", which indicates how many packets coming into the downstream outgoing links at a unit time [217], can also be used as an alternative consideration for packet-switched routers to make routing decisions. Many works have used this queue-length-oriented adaptive routing selection such as in [134], [168] [100] and [217].

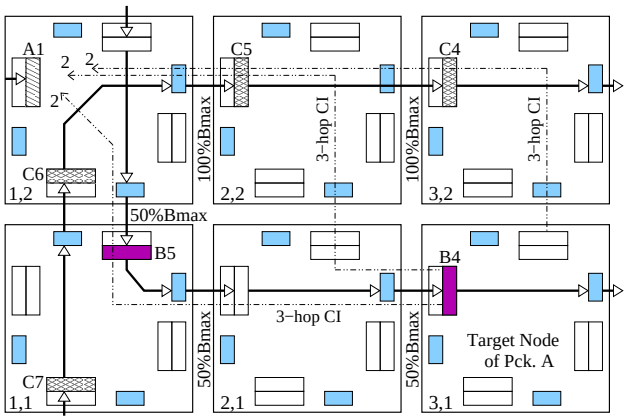
A specific technique to drain messages from hotspot areas called "Contention-Aware Input Selection" (CAIS) is presented in [213]. Rather than adaptively selecting less congested outgoing downstream direction, the CAIS method focuses on selection of input ports in the upstream (backtrace) direction. When two or more input ports request the same output port, an arbiter unit at the output port will select an input port having more waiting packets in its upstream direction. It looks that adaptive routing path selection is made by the arbitration unit rather than by the routing engine unit.

Because the FIFO buffers are implemented at input or output or both at input and output ports of every router node, then this strategy can be divided into a single hop or multi-hop congestion-aware adaptive routing strategies. The work in [14] has presented an interesting method to make adaptive routing selection based on the number of free buffer slots and availability of buffer in the two-hop adjacent neighbors "Neighbor-on-Path Routing Selection Strategy". However, the main critic to apply such methodology is the problem of unpredictable traffic situation as shown in Fig. 6.1(a).

Packet A will be routed from node (1,1) to (3,3). By measuring two-hop neighbor congestion information, the packet A at node (1,1) can overview four alternative paths, i.e. node-to-node paths $(1,1)-(1,2)-(1,3)$, $(1,1)-(1,2)-(2,2)$, $(1,1)-(2,1)-(3,1)$ and $(1,1)-(2,1)-(2,2)$. However, packet header A has only two alternative output selection, i.e. to North or East output port as depicted in Snapshot 1 of the figure. All four adjacent neighbors send back congestion information, i.e. the length of data queues in the FIFO buffers and the buffer status ("free" or "busy"). At the same time packet headers B and C come to node (2,1) via Local and East input port, respectively, in which packet A certainly does not know such situation. In this case, packet A will not use $(1,1)-(1,2)-(1,3)$ path. As shown in Snapshot 2, we assume that the routing engine finally decides to route packet header A to East output port, and at the same time at node (2,1), packet headers B and C are routed to West and North, respectively. Now, an unexpected situation occurs, where packet A has selected a non optimal path because of the unpredictable traffic situation. The same successive situations is shown in Fig. 6.1(b), when the packet A would be routed to the North output port.



(a) 2-hop congestion information traceback



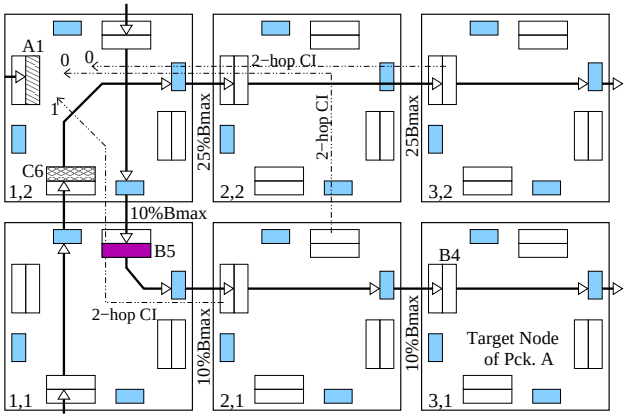
(b) 3-hop congestion information traceback

Fig. 6.2: A Situation of 2-hop and 3-hop congestion information (CI) traceback and actual link bandwidth consumption.

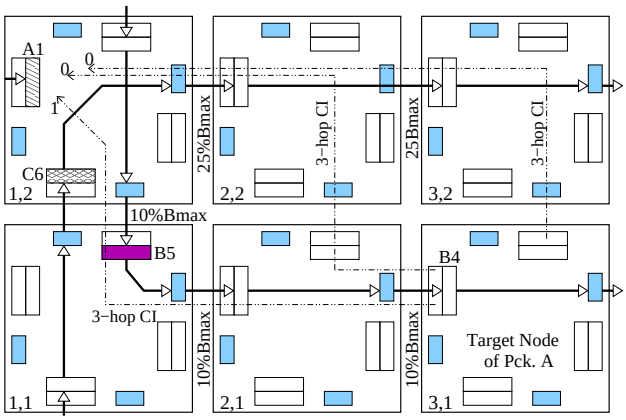
6.2.2 Selection based on Bandwidth-Space Occupancy

Another approach to make an adaptive routing decision is a strategy based on bandwidth-space occupancy. Fig. 6.2 shows us a snapshot of a network situation of the difference in orientations between the adaptive routing strategies based on FIFO queue occupancy (FQO) or also known as *Congestion-Aware Adaptive Routing Selection Strategy* and bandwidth space occupancy which is also known as *Bandwidth-Aware (BWA) Adaptive Routing Selection Strategy*. Fig. 6.2(a) presents a snapshot of the network situation where the FQO strategy reads congestion information (CI) traceback from 2-hop possible downstream neighbor routers. While Fig. 6.2(b) presents a snapshot of the network situation where the FQO strategy reads congestion information (CI) traceback from 3-hop possible downstream neighbor routers.

In Fig. 6.2, packet A coming to West input port at node (1,2) will be routed to node (3,1). We can see that packet A has 3 alternative paths to attain node (3,1), i.e. node-to-node paths $(1,2)-(2,2)-(3,2)-(3,1)$, $(1,2)-(2,2)-(2,1)-(3,1)$ and $(1,2)-(1,1)-(2,1)-(3,1)$. The header of the packet A (flit $A1$) has alternative output ports at the instant time, i.e.



(a) 2-hop congestion information traceback



(b) 3-hop congestion information traceback

Fig. 6.3: Another Situation of 2-hop and 3-hop congestion information (CI) traceback and actual link bandwidth consumption.

East and South. While the header flit of packet A is coming to node (1,2), at the same router node, the payload flit of packet B is coming from North input port and the payload flit of packet C is coming from South input port. They have acquired in advance the South and East output ports and have reserved 50% and 100% of the maximum BW space (B_{max}) of the output ports, respectively.

As presented in Fig. 6.2(a), the 2-hop congestion-information (CI) signals are sent back to node (1,2). If the packet A reads the 2-hop CI signals and buffer availability (like the strategy used in [14]), then packet A will select South output port as the best output path, because the South output port presents the CI signal value of 1 and the East output port presents the CI signal value of 1 and 2 for two consecutive paths, i.e. the formed paths when the packet A would be routed to the East port. But if only 1-hop CI signals are considered (not presented in the figure), then there is no different hotspot situation based on the view of the packet A, because both East and South neighbor send back the same queue-length (data queue occupancy), i.e. 1 data queue.

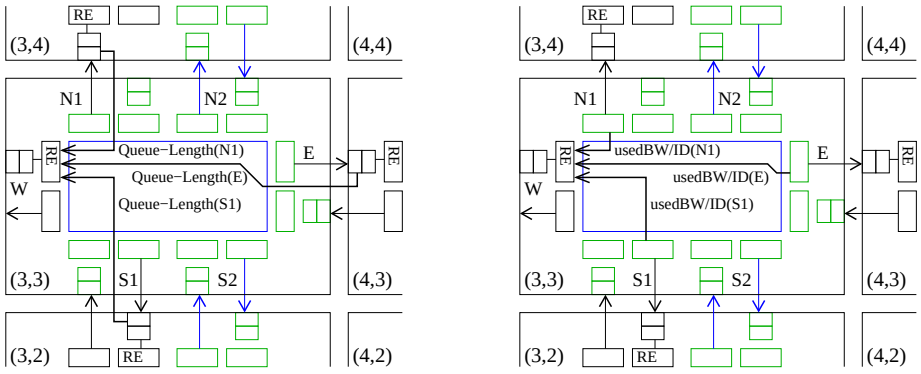
The situation presented in Fig. 6.2 is actually the main functionality of using the N-

hop neighbor congestion information without considering its drawback mentioned in Section 6.2.1. However, if the packet A just reads the actually bandwidth (BW) space occupancy of the two alternative output port at the current node (1,2), then packet A can view the different hotspot situation. Hence, packet A will also be routed to South output port when using the BWA strategy because it has more free BW spaces.

The 3-hop CI traceback is presented in Fig. 6.2(b). When the header flit of packet A reads the 3-hop CI signals from the South and East output port at the instant time of the network situation, then both alternative output ports give an equal CI signal value of 2 as presented in the figure. In this case, the header surely cannot make the best selection. However, by using the BWA adaptive routing selection strategy, the header flit of the packet A will select South output port because the South output port provides still 50% BW space compared to the BW space of the East output port that has been consumed 100% by the packet B.

Another situation is presented in Fig. 6.3. The paths and participating packets in the figure is similar to Fig. 6.2 but with different BW space occupancy. In the Fig. 6.3, we can see that packet B consumes 10% B_{max} of its acquired links, while packet C consumes 25% B_{max} of its acquired links. By reading the CI signals of the 2-hop and 3-hop adjacent neighbors, packet A indicates that East output port is the best output port, because both 2-hop and 3-hop CI signals from the downstream routers that are sent back to the East output port have a zero value (no data occupies the buffer). The 2-hop and the 3-hop neighbor CI signals that are sent back to South output port are 1 data queue-length as presented in Fig. 6.3(a) and Fig. 6.3(b) respectively. However, if we use BWA adaptive routing strategy, then packet A will be routed to South output port because it has less BW space occupancy (10%) compared to East output port in which its maximum BW capacity has been consumed by 25% of packet B.

The work in [179] presents bandwidth-aware adaptive routing method. However, this method computes adaptive routing paths off-line (at design time). A runtime bandwidth-aware adaptive routing function called AdNoC is presented in [6]. The proposed method selects an output port having more free bandwidth spaces. The bandwidth-aware adaptive routing selection of the XHiNoC has the same strategy as AdNoC's strategy. However, AdNoC uses virtual channel buffers leading to extra large area overhead, while XHiNoC do not use them. The "bandwidth/contention/congestion look-ahead" method used in XHiNoC can compute immediately a routing decision in one cycle period. Meanwhile, the AdNoC implementation result requires 4 cycle periods to make routing decision leading to routing computation time overhead. Moreover, the XHiNoC can also implement many strategies or the combination of the strategies.



(a) Congestion-aware (b) BW/Contention-aware

Fig. 6.4: Alternative information that can be used to make adaptive output routing selection.

6.3 Architectures and Algorithms for Adaptive Routing Selection Functions

6.3.1 Local ID-based Data Multiplexing

We use a wormhole switching technique, where flits of different messages can be interleaved, and share the same communication media based on the locally organized message identity. Flits belonging to the same message will always have the same local ID-tag when acquiring a communication medium (network link). The wormhole messages can be interleaved at flit-level because every flit has a unique local ID-tag to differentiate it from other flits belonging to different flits in the same link. The local ID tag of a message is updated by an ID management unit implemented at output port, when the message enters a new communication channel. By using this kind of wormhole switching, the head-of-line blocking problem commonly happen in the traditional wormhole switching can be solved without implementing virtual channels. The concept of the wormhole switching has been depicted in Fig. 4.3 in Chap. 4, where different messages can be interleaved in the same buffer pool or can virtually cut-through at flit level. Each message reserves one ID slot in order to be able to use the link. Based on such situation, contention information in the output port can be achieved by counting the number of the reserved ID slots in the link. Moreover, if the ID slot reservation is followed by bandwidth reservation, then a bandwidth-aware adaptive routing selection strategy can be also implemented in our NoC.

6.3.2 Adaptive Routing Selection Functions

Five router implementations based on information that are considered to make routing decision and based on the viewpoint of our NoC microarchitecture will be presented. The three considered information are described in the following.

Alg. 15 CBWA Adaptive Routing Function–BW-ID version (Abstract view)

```
1: Begin Function Select(port m1, port m2)
2: if Routing can be made to port m1 and port m2 then
3:    if the used BW space in port m1 is less than the used BW space in port m2 then
4:       Return Routing = port m1
5:    else if the used BW space in port m1 is greather than the used BW space in port m2 then
6:       Return Routing = port m2
7:    else if the used BW space in port m1 is the same as the used BW space in port m2 then
8:       if the used ID slots in port m1 is less than or equal to the used ID slots in port m2 then
9:          Return Routing = port m1
10:      else
11:         Return Routing = port m2
12:      end if
13:   end if
14: end if
15: End Function
```

- *Identity (ID) slot occupancy* (the number of free ID slots). This information can be called as *Contention Information* of an output port, i.e. the number of messages that have contented (competed) so far to access the output port. Since our router can interleave different wormhole messages at flit-level in the same link without using virtual channels, then the number of reserved ID slots will represent the number of the wormhole messages that have been mixed in the outgoing link.

- *Bandwidth (BW) space occupancy* (the number of free BW space). This information can be called as *BW-Reservation Information* of an output port, i.e. the number of BW spaces that have been reserved by messages to access the output port.

- *Buffer space occupancy* (the number of data queue in a FIFO buffer). This information can be called as *Congestion Information* of an output port, i.e. the queue-length in the FIFO buffer at the input port of the next neighbor switch connected directly to the output port.

Fig. 6.4(a) shows how the routing engine (RE) unit at the West input port of router node $(3, 3)$ receives congestion information in term of *Queue-Length* in the FIFO buffers at the input ports of the neighbor nodes $(3, 4)$ (North 1), $(4, 3)$ (East) and $(3, 2)$ (South 1). The North 1 (N1), East (E) and South 1 (S1) are alternative output ports to route messages adaptively from the West (W) input port. Fig. 6.4(b) shows the same situation, but in this case, the number of reserved bandwidth (*usedBW*) or reserved ID slots (*usedID*) at the alternative output ports are sent to the RE unit. It appears that information from neighbors are required when the queue-length-oriented selection function is implemented. Meanwhile, the contention-oriented and bandwidth-oriented selection strategies need only local information from each considered router like general off-chip routers. By using the three considered information explained above or combination of them, then five NoC router prototypes (version) are proposed as mentioned in the following.

- *BW-ID version.* This prototype uses two information signals to make routing decisions. The first prioritized signal is the number of the reserved bandwidth spaces. Messages are routed to an output direction having less reserved bandwidth spaces. If the numbers of the reserved bandwidth spaces between two output ports are equal, then the second prioritized signal is used, i.e. the number of reserved ID slots. The messages are then routed to an output direction having less reserved ID slots when the numbers of the reserved bandwidth spaces between the alternative output ports are equal. This adaptive routing strategy can be called as *Contention- and Bandwidth-Aware (CBWA) Adaptive Routing Selection Strategy*.

- *FQ-ID version.* This prototype also uses two information signals to make routing decisions. The first prioritized signal is the number of used buffer spaces. Messages are routed to an output direction having less utilized buffer spaces. If the number of the used buffer spaces between two output ports are equal, then the second prioritized signal (the number of reserved ID slots) is used. The messages are then routed to an output direction having less reserved ID slots when the numbers of FIFO queue occupancies between the alternative output ports are the same. This adaptive routing strategy can be called as *Contention- and Congestion-Aware (CCA) Adaptive Routing Selection Strategy*.

- *BW version.* This prototype uses single information signals to make routing decisions. Messages are routed to an output direction having less reserved bandwidth spaces. This adaptive routing strategy can be called as *Bandwidth-Aware (BWA) Adaptive Routing Selection Strategy*.

- *FQ version.* This prototype uses single information signals to make routing decisions. Messages are routed to an output direction having used less FIFO buffer spaces. This adaptive routing strategy can be called as *Congestion-Aware Adaptive Routing Selection Strategy*.

- *ID version.* This prototype uses single information signals to make routing decisions. Messages are routed to an output direction having used less ID slots. This adaptive routing strategy can be called as *Contention-Aware Adaptive Routing Selection Strategy*.

The abstract view of the BW-ID version or the CBWA adaptive routing selection function is presented in Alg. 15. In the algorithm, the function will firstly consider and select an output port having more free-available BW spaces. When the free BW spaces between two alternative output ports are equal, then the function will consider and select an output port having more free-available ID slots. When both considered information are equal, then as shown in Alg. 15, the router will select output port m_1.

The logical view of the BW-ID version or the CBWA adaptive routing selection strategy is shown in Alg. 16. The logical equations are derived from the abstract view of the routing selection function described in Alg. 15, where the BW space occupancy is the first

Alg. 16 CBWA Adaptive Routing Function–BW-ID version (Logical view)

$usedBW(m)$: Number of used/reserved Bandwidth space in direction m
$usedID(m)$: Number of used/reserved ID slots in direction m
1: Begin Function **Select**(m_1, m_2)
2: **if** $usedBW(m_1)$ & $usedID(m_1) \leq usedBW(m_2)$ & $usedID(m_2)$ **then**
3: Return m_1
4: **else if** $usedBW(m_1)$ & $usedID(m_1) > usedBW(m_2)$ & $usedID(m_2)$ **then**
5: Return m_2
6: **end if**
7: End Function

priority signal and the ID slot occupancy is the second priority signal. The concatenated $usedBW$ & $usedID$ signal of the BW space occupancy ($usedBW$) and the ID slot occupancy ($usedID$) as presented in the Alg. 16 will represent the level of signal priority. In digital signals, the $usedBW$ and the $usedID$ are represented in binary codes. Hence, by locating the $usedBW$ signal in the left side, then it will be directly allocated in the most significant bit (MSB) of the concatenated signal. While the $usedID$ put in the right side will be the least significant bit (LSB) of the concatenated signal. The $usedBW$ signal is allocated in the MSBs of the concatenated signal, because during runtime selection, the BW space occupancy has higher priority than the ID slot occupancy, which is represented by the $usedID$ signal.

The logical view of the CCA (Contention- and Congestion-Aware) adaptive routing selection function is shown in Alg. 17. Like the CBWA selection function, the CCA routing function considers two information, i.e. the queue-length in FIFO buffer in the next neighbor of two alternative outgoing links and the number of reserved ID slots in the two alternative output ports.

Alg. 17 CCA Adaptive Routing Function–FQ-ID version (Logical view)

$QueueLength(m)$: Number of data in FIFO buffer of neighbor in direction m
$usedID(m)$: Number of used/reserved ID slots in direction m
1: Begin Function **Select**(m_1, m_2)
2: **if** $QueueLength(m_1)$ & $usedID(m_1) \leq QueueLength(m_2)$ & $usedID(m_2)$ **then**
3: Return m_1
4: **else if** $QueueLength(m_1)$ & $usedID(m_1) > QueueLength(m_2)$ & $usedID(m_2)$ **then**
5: Return m_2
6: **end if**
7: End Function

The logical view of the BW version or the BWA adaptive routing selection strategy is shown in Alg. 18. The BW version adaptive routing selection function is simpler than the BW-ID version because the *usedID* signals from both alternative output ports are removed from the selection mechanism. Alg. 19 and Alg. 20 present the logical view of the congestion-aware and the contention-aware adaptive routing selection function, respectively.

Alg. 18 Bandwidth-Aware Adaptive Routing Function (Logical view)

$usedBW(m)$: Number of used/reserved Bandwidth space in direction m

```
1: Begin Function Select(m1, m2)
2: if usedBW(m1) ≤ usedBW(m2) then
3:     Return m1
4: else if usedBW(m1) > usedBW(m2) then
5:     Return m2
6: end if
7: End Function
```

Alg. 19 Congestion-Aware Adaptive Routing Function (Logical view)

$QueueLength(m)$: Number of data in FIFO buffer of neighbor in direction m

```
1: Begin Function Select(m1, m2)
2: if QueueLength(m1) ≤ QueueLength(m2) then
3:     Return m1
4: else if QueueLength(m1) > QueueLength(m2) then
5:     Return m2
6: end if
7: End Function
```

6.3.3 Router Microarchitecture and Packet Format

The microarchitecture of the NoC router using the contention- and BW-aware (CBWA) adaptive routing selection strategy is presented in Fig. 6.5(a). The router is designed based on 2D mesh-planar topology, where each router has seven IO port, i.e. East, North1, North2, West, South1 and South2 ports. Crossbar interconnect is customized to optimize the logic area of the router based on the allowed turns in the 2D planar adaptive routing algorithm. The rest router internal IO connections representing the prohibited turns are removed from the architecture. Fig. 6.5(b) shows the NoC router that uses the contention- and congestion-aware (CCA) adaptive routing function. We can see that we need a little bit of effort to reconfigure the microarchitecture of the XHiNoC at design time from BW-ID to FQ-ID version. The required modifications are (1) add new output and input ports for the $QueueLength$ signal from the FIFO buffer, (2) replace the RSM with a new RSM, and (3) remove the BW accumulator unit.

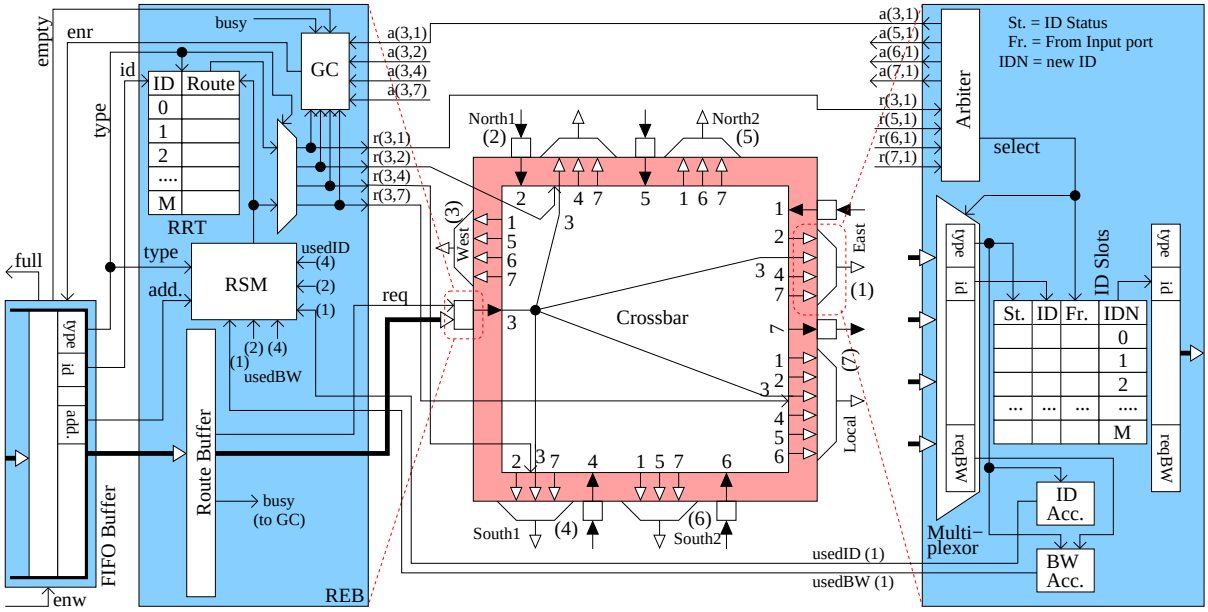
For the sake of simplicity, only the router components in East input port and in West output port are depicted. Two main components in the input port are FIFO queue and Routing Engine with Data Buffering (REB). In the REB unit, the combination of the routing state machine (RSM) and routing reservation table (RRT) is implemented to support runtime adaptive routing mechanism.

The detail packet format and the control bits used in the XHiNoC architecture for the CBWA and BWA adaptive router is presented in Fig. 6.6. The message is split into several flits and has 39-bit width, 32 bits for dataword plus 9 extra bits i.e., 3-bit field to define the type of flits and 4-bit field to determine the local identity label or ID-tag of a message. An extra 12-bit field in the header and tail flits is used to present the expected communication

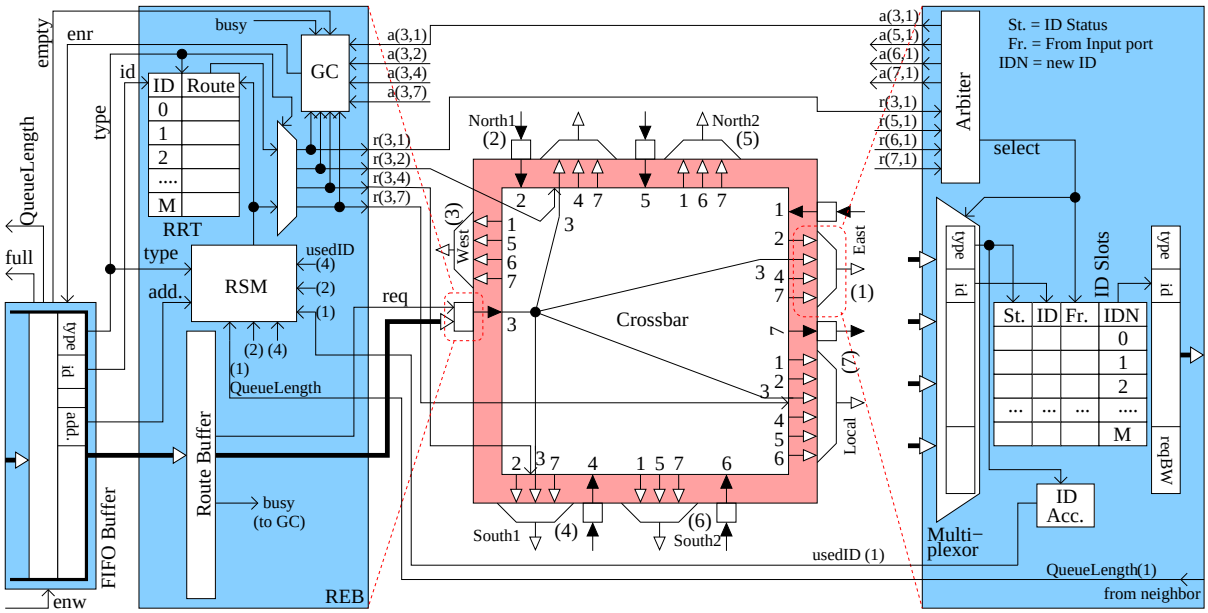
Alg. 20 Contention-Aware Adaptive Routing Function (Logical view)

$usedID(m)$: Number of used/reserved ID slots in direction m

1: Begin Function **Select**(m_1, m_2)
2: **if** $usedID(m_1) \leq usedID(m_2)$ **then**
3: Return m_1
4: **else if** $usedID(m_1) > usedID(m_2)$ **then**
5: Return m_2
6: **end if**
7: End Function



(a) Router with Contention- and Bandwidth-Aware Adaptive Routing Function



(b) Router with Contention- and Congestion-Aware Adaptive Routing Function

Fig. 6.5: Switch microarchitectures for routers with contention- and bandwidth-aware and with congestion- and contention-aware adaptive routing selection strategy.

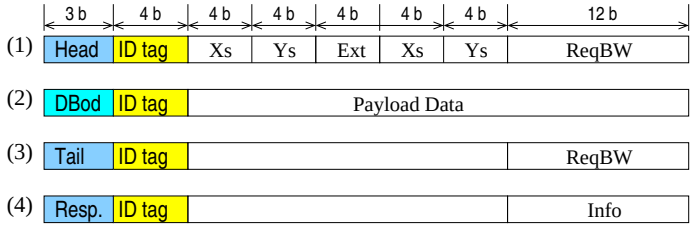
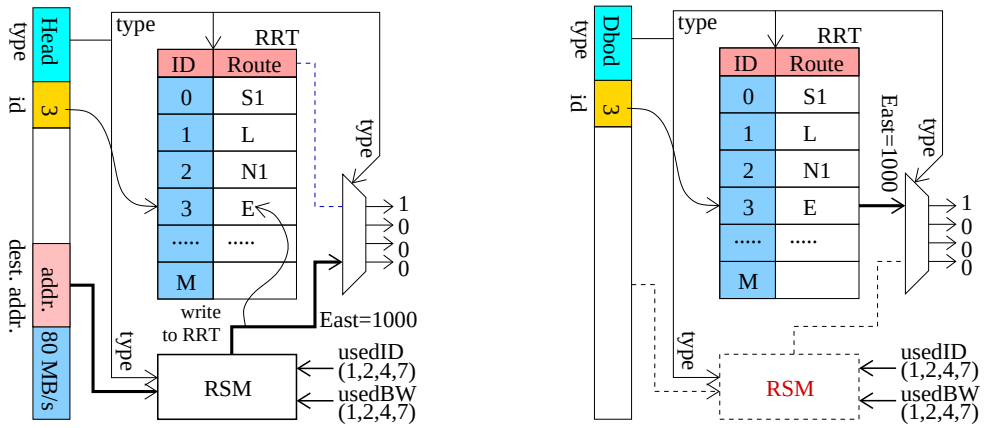


Fig. 6.6: Packet format for the CBWA and BWA adaptive routing selection strategy.



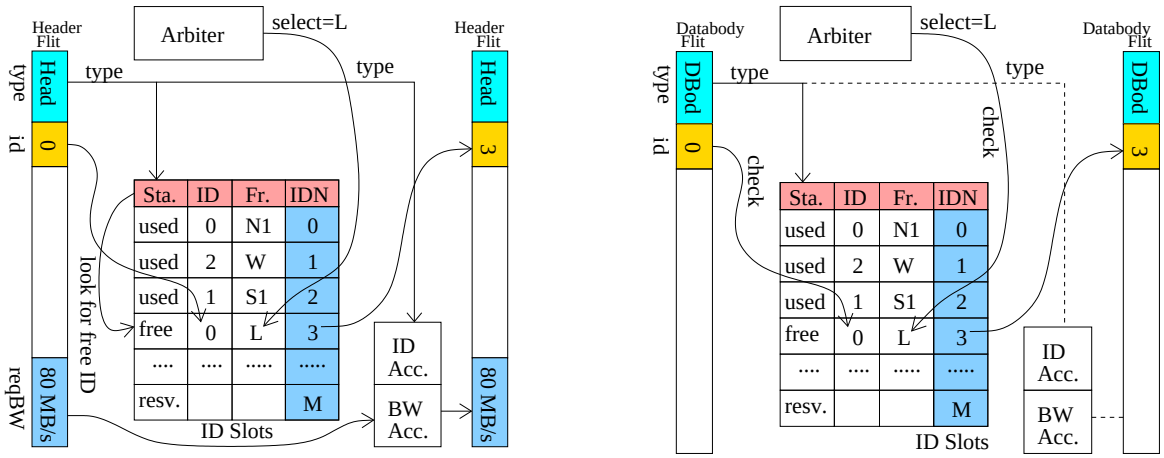
(a) Reservation by Header flit (b) Indexing by Databody flit

Fig. 6.7: ID-based routing table reservation and assignment.

bandwidth. When a header flit of a message flows through the crossbar multiplexor at an output port, the value in this field will be used to reserve BW for the message on the output port.

The width of the required Bandwidth ($ReqBW$) field determines the resolution of the BW space in each outgoing port. For q-bit field of the $ReqBW$, the resolution of BW space is 2^q. Hence, when $q = 12$ as set in the Fig. 6.6, then the number of BW variations that can be used by the messages to reserve BW space at the output port is $2^{12} = 4096$. When we use Mega-Byte per second (MB/s) as the unit of the required BW and the maximum capacity of the link were for instance $4096\ MB/s$, then if the required BW is $80\ MB/s$ for example, then the binary signal of the $ReqBW$ will be $[000001010000]$.

Fig. 6.7(a) shows us in detail how a header flit of a packet reserves a routing slot in the RRT unit at the West input port. The header with ID-tag 3 comes into West input port. The REB unit routes and buffers the header flit in its data register. The RSM unit computes the requested routing direction based on target address written in the header bit fields. The RSM unit selects one of the two alternative output ports based on two signals representing the number of used ID slots ($usedID$) and used BW spaces ($usedBW$) in the two alternative output port. Both signals are concatenated ($usedBW\&usedID$) by the RSM unit. The output port having less concatenated signal will be selected as the best output direction. The output routing made by the RSM unit is then stored in the slot number 3 of the RRT unit (in accordance with the ID-tag of the header flit). The routing



(a) ID Updating by Header flit

(b) ID Indexing by Databody flit

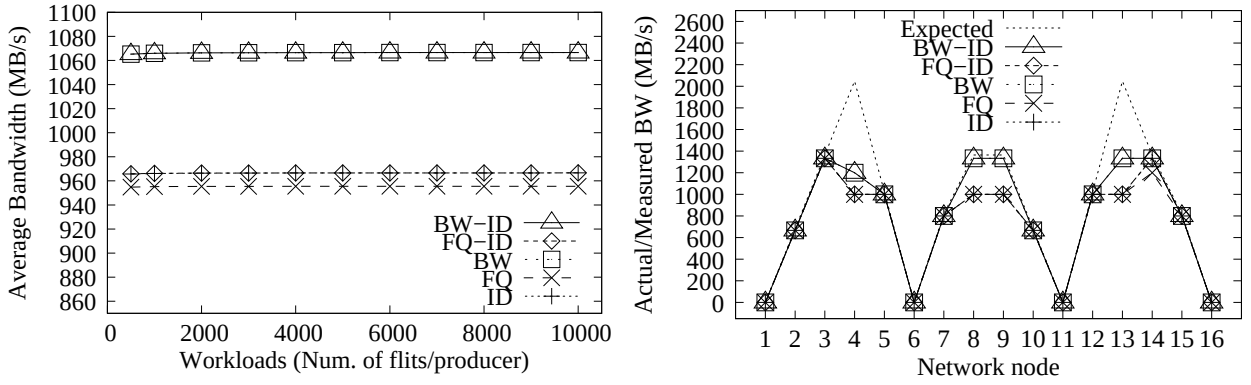
Fig. 6.8: Local ID slot reservation.

output decision is controlled by the type field of a flit via a two input multiplexor. If the flit type is a header, then routing decision is read from the RSM unit.

When a databody flit belonging to the same message flows through the REB component as presented in Fig. 6.7(b), the routing direction will be indexed by the databody flit using its ID-tag. The routing direction is thus read directly from the RRT. The header with ID-tag 3 belongs to the same message with the databody with ID-tag 3. Since they have the same ID-tag number. When a tail flit flows through the REB component, then the same mechanism takes place like the index operation made by the databody flit, but at the same cycle, the routing direction is removed from the RRT.

In the output port, there are two main components, i.e. an arbiter unit and a crossbar multiplexor with ID management unit (MIM). The ID management unit (IDM) consists of an ID slot table, bandwidth (BW) accumulator and ID accumulator units. Fig. 6.8(a) shows how the IDM unit works to allocate a header flit into a new local ID slot as its new ID tag. The ID tag of the header flit is 0 and required to perform a communication rate of $80\ MB/s$. Firstly, when a header flit type is detected, a free ID slot is looked for. As shown in the Fig. 6.8(a), it looks that new ID slot (IDN) 3 is found free, and then it is used as the new ID tag for the header flit. At the same cycle, the previous ID tag of the header and from which port the header flit comes is written in the slot number 3. The $select$ signal set by the arbiter unit will inform from which port the header flit comes. The BW accumulator unit adds the required BW of the header to the total BW spaces that has been reserved so far. Meanwhile the ID accumulator unit increments also the reserved ID slot from 2 to 3 ($usedID \Leftarrow usedID + 1$).

When a databody flit (ID-tag 0) belonging to the same message with the previously header flows through the MIM component as presented in Fig. 6.8(b), the IDM unit will check the current ID-tag of the databody flit and from which port it comes. As shown in the Fig. 6.8(b), the pair of both signals (ID-tag 0, from L port) is detected in the ID slot number 3, then the databody flit will also have ID-tag number 3, which is the same



(a) Average Bandwidth (b) Actual Bandwidth

Fig. 6.9: Average and actual bandwidth measurement per target node under transpose scenario in 4×4 mesh network.

as the previously header shown in Fig. 6.8(a). When a tail flit flows through the MIM component, then the same mechanism takes place like the index operation made by the databody flit, but at the same cycle, the signal pair will be removed from the ID slot table.

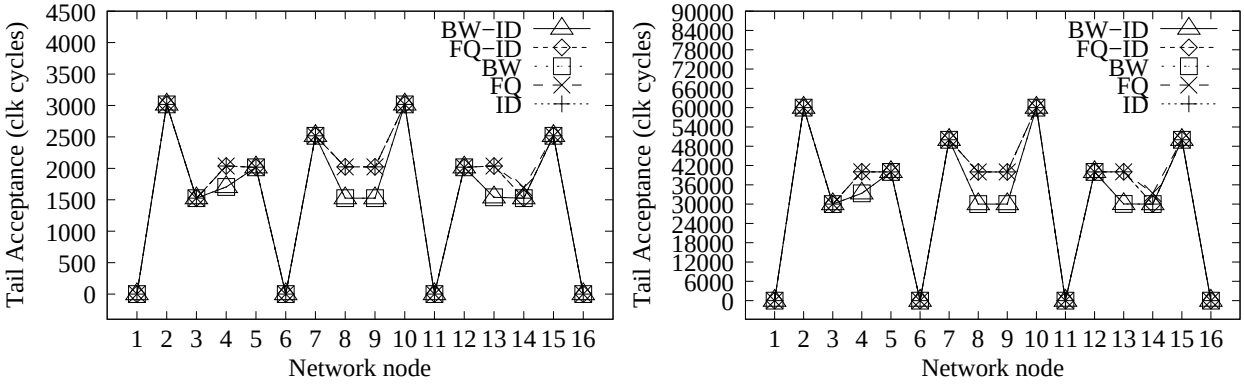
6.4 Experimental Results

In this section, the five NoC prototypes with different adaptive routing selection strategies are simulated. The adaptive routing for the five prototypes are minimal. It means that messages will not be routed away from their destination node. Thus, the message will have a maximum of two alternative routing direction on intermediate nodes.

Two simulation scenarios are made to verify the effectiveness of the bandwidth-aware adaptive routing algorithm. The first simulation is made under transpose traffic scenario in the 2D 4×4 mesh NoC, and the second simulation is run under bit complement traffic scenario in the 2D 8×8 mesh NoC. The experimental results are presented in the following subsections.

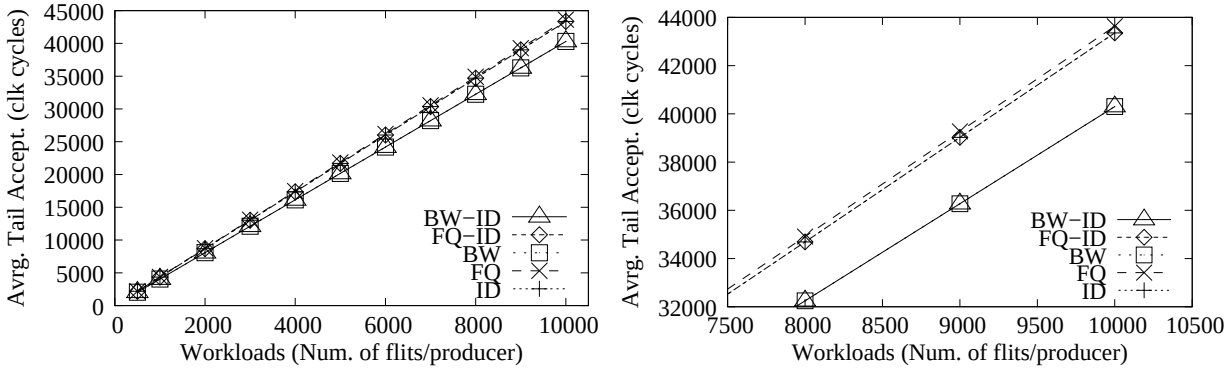
6.4.1 Transpose Scenario in 4x4 Mesh Network

The average bandwidth measurement under transpose scenario in 4x4 mesh network is presented in Fig. 6.9(a). The average bandwidth is measured for different workload sizes. It looks that the average bandwidth is constant, and the *BW-ID*, *BW* and *ID* prototypes give better performance than the *FQ-ID* and *FQ*. The injection rates of producer nodes in the transpose scenario are set randomly. The initial injection time on every data producer node is also set randomly. Fig. 6.9(b) presents the actual bandwidth measurement per target node under transpose scenario in 4x4 mesh network. The expected bandwidth depicted in Fig. 6.9(b) represents the set injection rates of the producer nodes. For example, messages injected from node 4 and node 13 are expected to communicate with maximum



(a) The acceptance of the 500^{th} flit (b) The acceptance of the 10000^{th} flit

Fig. 6.10: The tail flit acceptance measurement on every target node in clock cycle period under transpose scenario in 4×4 mesh network.



(a) 500-10000 workloads (b) 8000-10000 workloads

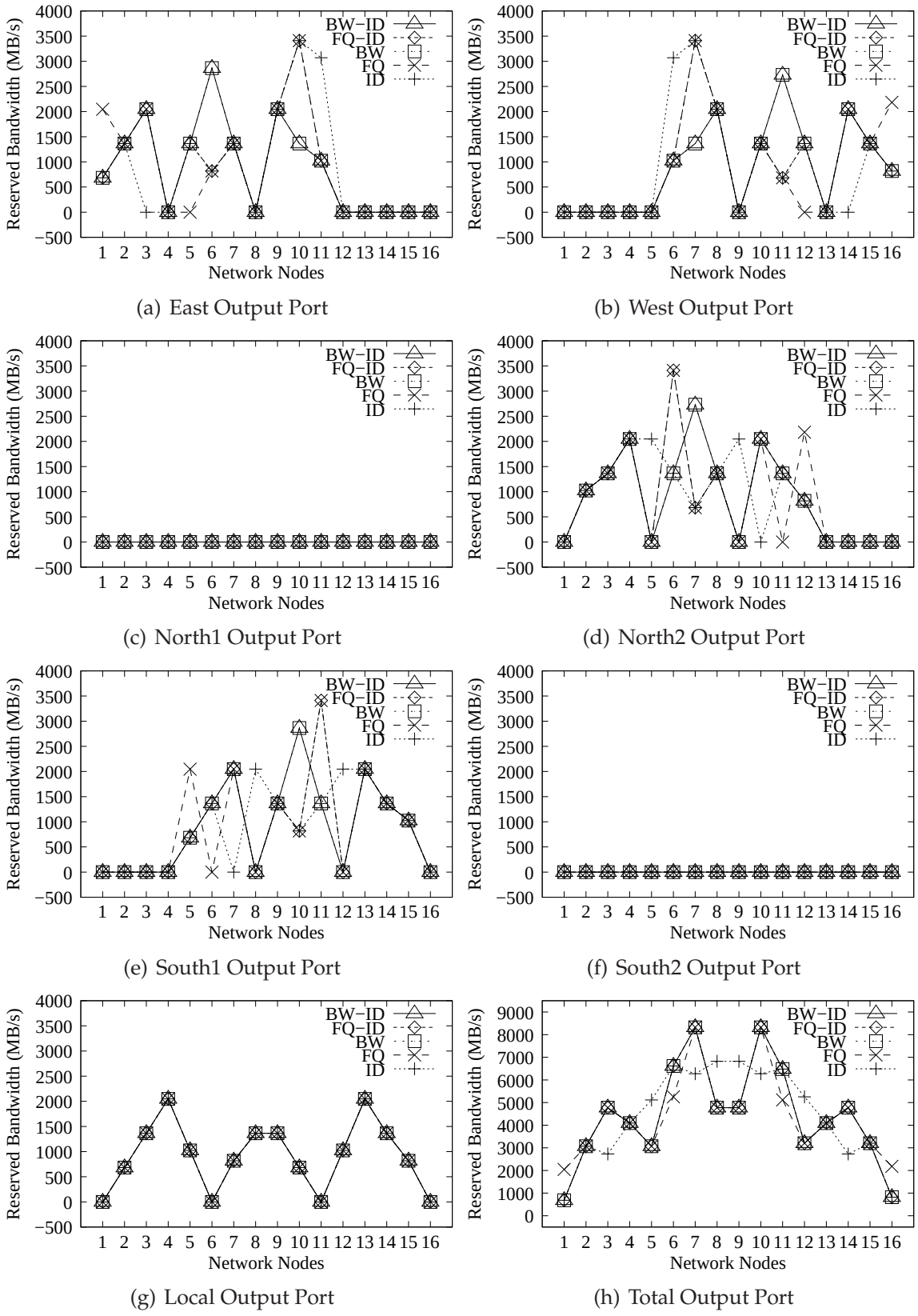
Fig. 6.11: Average tail flit acceptance latency under transpose scenario in 4×4 mesh NoC.

bandwidth rate of about 2000 MB/s.

Fig. 6.10 shows the tail flit acceptance measurement in clock cycle under transpose scenario in 4×4 mesh network. The tail flit acceptance or the 500^{th} flit is presented in Fig. 6.10(a) and the tail flit acceptance or the 10000^{th} flit is presented in Fig. 6.10(b). Node 1, node 6, node 11 and node 16 present zero acceptance latency because these nodes do not send and accept messages. The average tail flit acceptance latency under transpose scenario in 4x4 mesh NoC for different workload sizes is shown in Fig. 6.11(a). The enlarged view of the average tail flit acceptance latency to ilustrate clearly the difference between the adaptive router prototypes is presented in Fig. 6.11(b).

The bandwidth space and ID slots reservation for every output port of all network nodes are presented in Fig. 6.12 and Fig. 6.13, respectively. It looks that there is no bandwidth space and ID slots reservation at North 1 and South 2 output ports of all network nodes. The distribution of the bandwidth reservation and ID slots reservation are variant in the transpose traffic scenario for all adaptive router prototypes.

Meanwhile, Fig. 6.14 exhibits the occupancy of four selected FIFO buffers at input



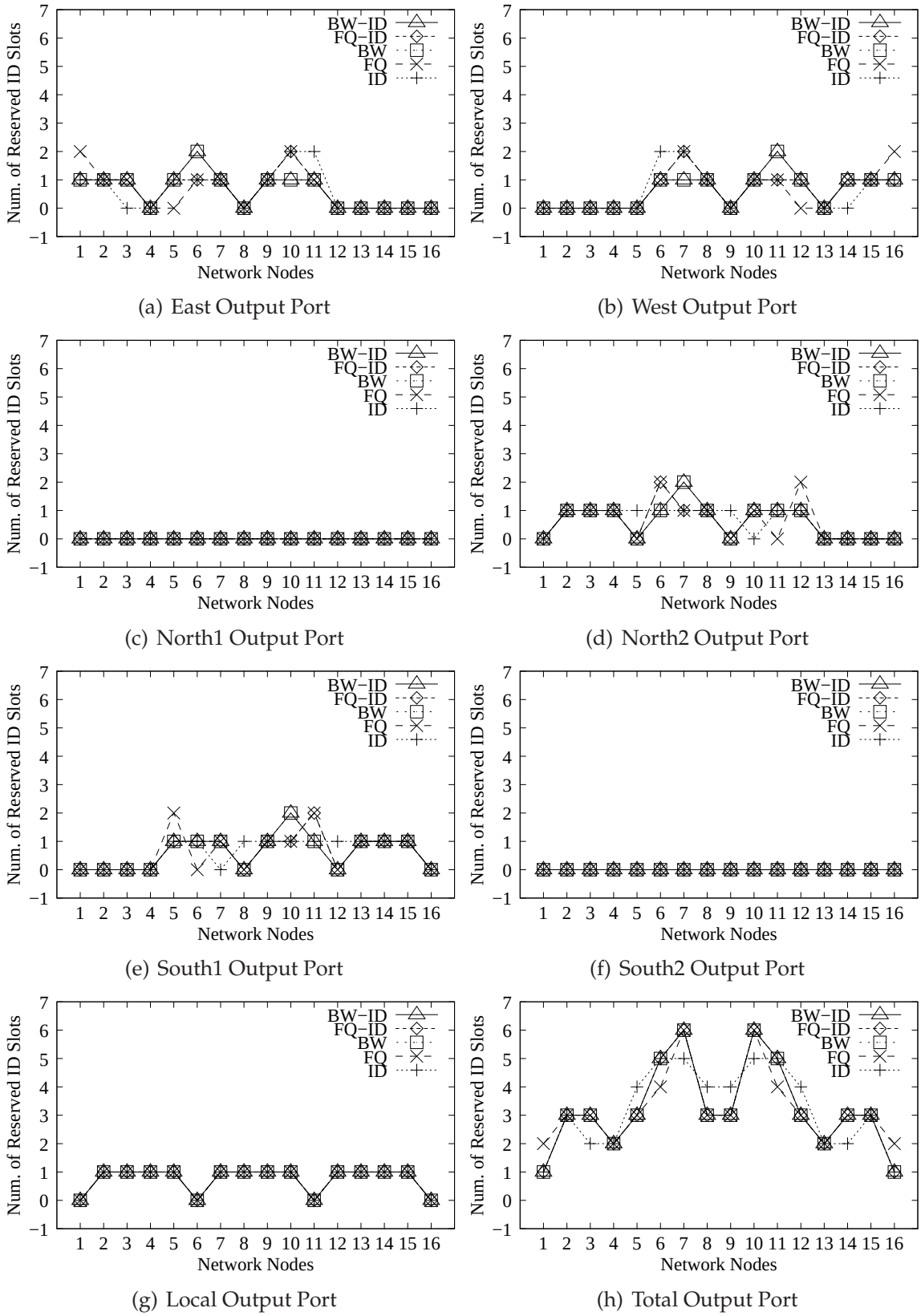
(a) East Output Port (b) West Output Port

(c) North1 Output Port (d) North2 Output Port

(e) South1 Output Port (f) South2 Output Port

(g) Local Output Port (h) Total Output Port

Fig. 6.12: Bandwidth space reservation at each output port under transpose scenario in 4×4 mesh NoC.



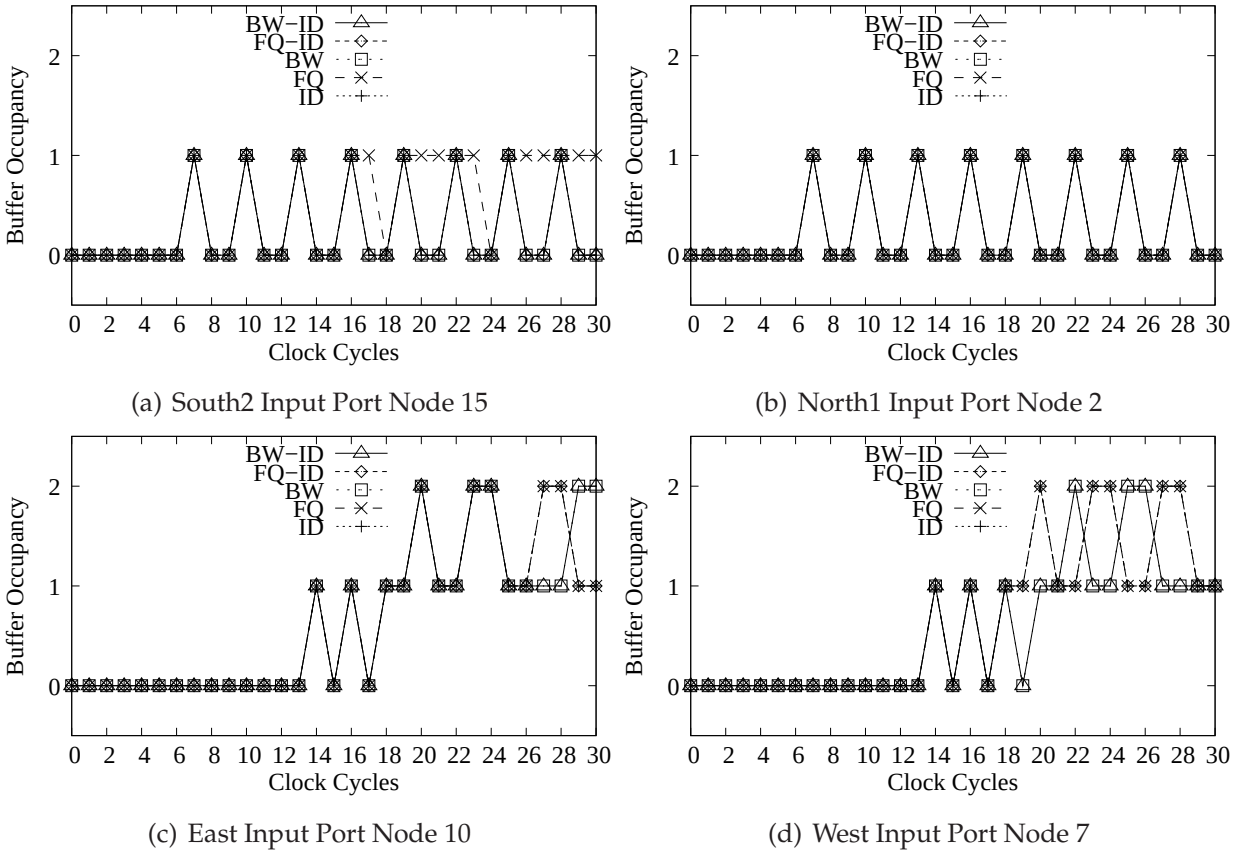
(a) East Output Port (b) West Output Port

(c) North1 Output Port (d) North2 Output Port

(e) South1 Output Port (f) South2 Output Port

(g) Local Output Port (h) Total Output Port

Fig. 6.13: ID slots reservation at each output port using transpose scenario in 4×4 mesh NoC.



(a) South2 Input Port Node 15

(b) North1 Input Port Node 2

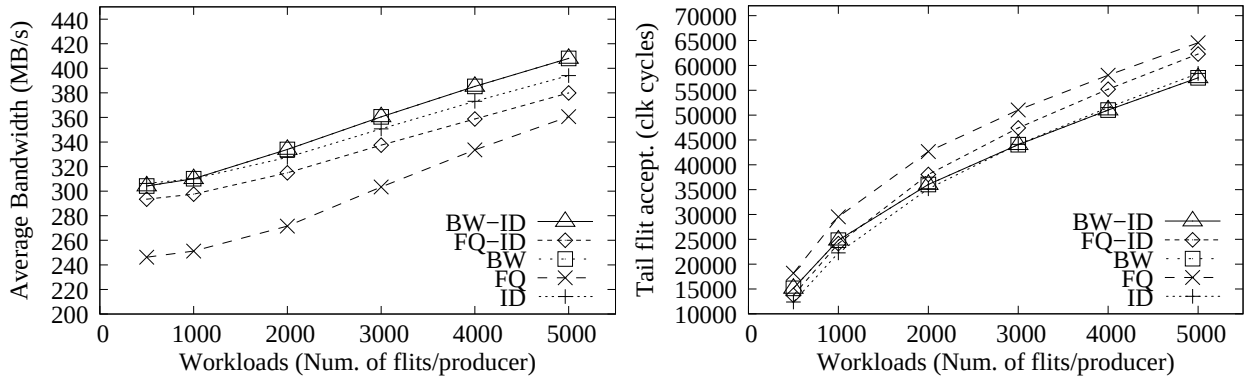
(c) East Input Port Node 10

(d) West Input Port Node 7

Fig. 6.14: FIFO Queue occupancy at selected output ports and network nodes for transpose scenario in 4x4 mesh NoC.

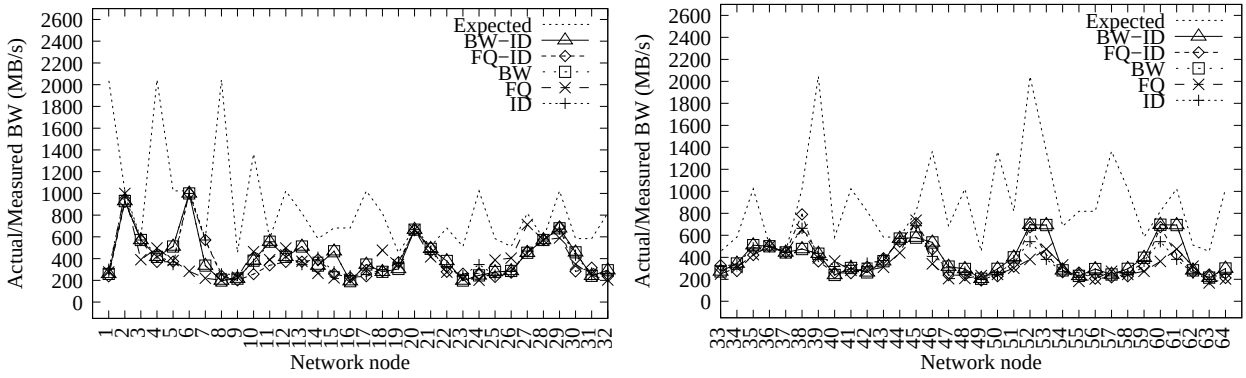
ports. The measurement of the number of data in the FIFO queues is made in every clock cycle period, i.e from initial time of the simulation until the 30^{th} cycle period. The four figures of the selected FIFO buffers show that there are several situation at any clock cycle periods where the FIFO buffers are emptied from data. This case will be more frequent when the injection rates of the messages are slower. Therefore, in line with the previous explanations in Section 6.2, the *FQ-ID* and *ID* prototypes could probably make a less optimal routing direction because of such situation compared with the bandwidth-aware adaptive routing algorithms (*BW-ID* and *BW*), which consider directly the available bandwidth spaces on the alternative outgoing ports.

Different initial injection times could give different performance evaluation results because correct decisions to make an optimal routing direction are strongly dependent on the dynamic neighbor states of the FIFO buffer occupancy, bandwidth space and ID slots reservation of the link at certain instant time as explained in Section 6.2. The experimental result presented in this subsection is only one of many simulations that could be run to test the performance of the five selected adaptive router prototypes. The following subsection will describe another simulation result in a larger network size with bit complement data distribution scenario.



(a) Average Bandwidth (b) Average Tail Acceptance Latency

Fig. 6.15: Average bandwidth measurement and tail acceptance delay for bit-complement scenario in 8×8 mesh network.



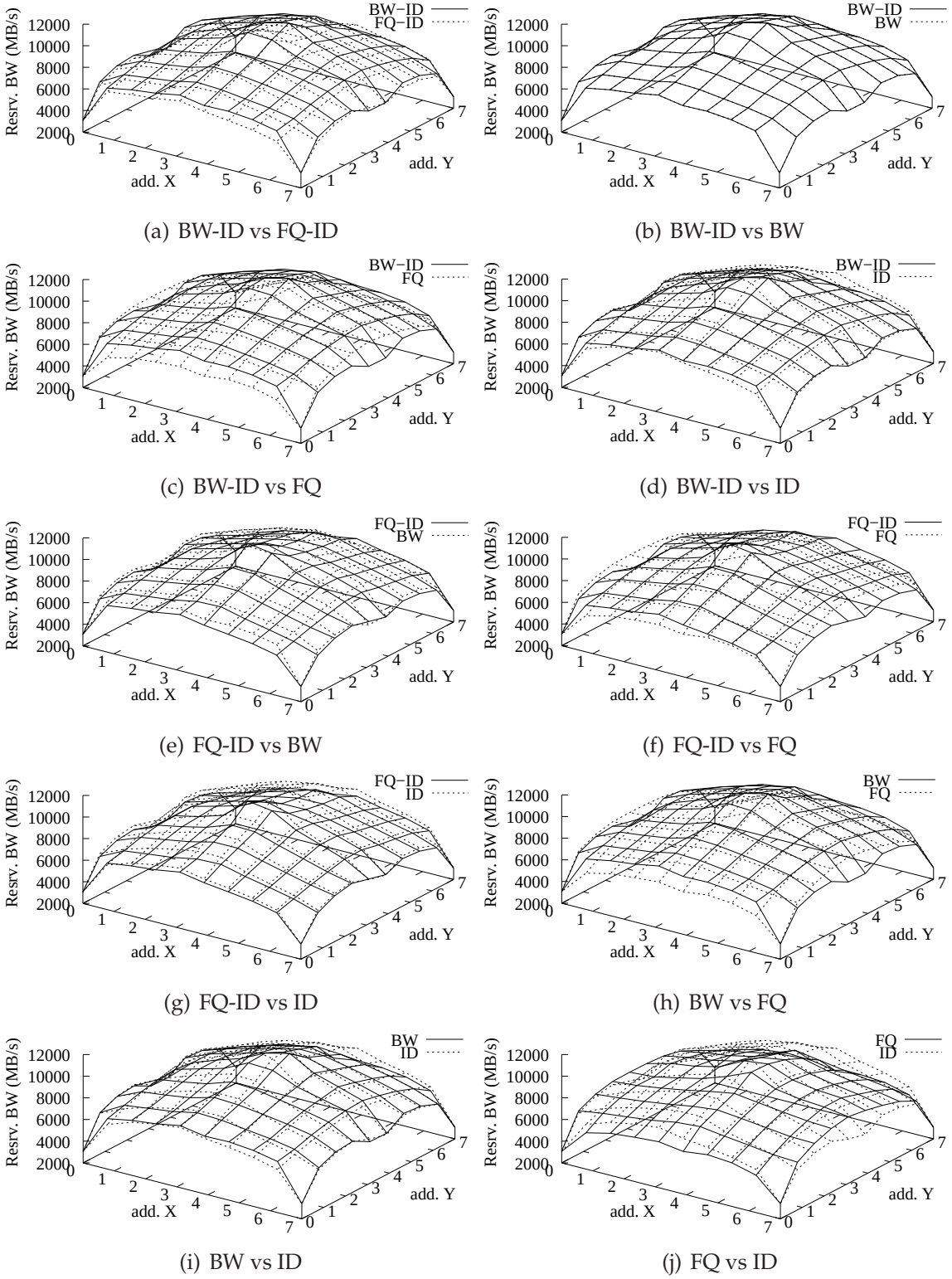
(a) Node 1–32 (b) Node 33–64

Fig. 6.16: Actual bandwidth measurement per network node for bit complement scenario in 8×8 mesh network.

6.4.2 Bit Complement Scenario in 8x8 Mesh Network

This subsection presents the performance evaluations of the five adaptive NoC router prototypes under bit complement traffic scenario in the 2D 8×8 mesh network. The injection rates of producer nodes in the bit complement scenario are set randomly. Fig. 6.15(a) shows the average bandwidth measurement in MB/s of all communication pairs in the bit-complement traffic scenario. Fig. 6.15(b) presents the average tail acceptance delay in clock cycle period. In both simulations, the measurements are made by varying the number of workloads per data producer node, i.e. the number of flits sent to the NoC. The workloads is varied between 500, 1000, 2000, 3000, 4000 and 5000 flits per data producer node. It looks like the *BW-ID* and *BW* NoC prototypes present the best performance followed by the *ID*, *FQ-ID* and *FQ* NoC prototypes.

The actual bandwidth measurement for every node-to-node communication pair (per network node) in MB/s for bit complement scenario in the 8×8 mesh network is presented in Fig. 6.16. The actual bandwidth measurements are made at destination nodes.



(a) BW-ID vs FQ-ID (b) BW-ID vs BW

(c) BW-ID vs FQ (d) BW-ID vs ID

(e) FQ-ID vs BW (f) FQ-ID vs FQ

(g) FQ-ID vs ID (h) BW vs FQ

(i) BW vs ID (j) FQ vs ID

Fig. 6.17: Distribution of the total bandwidth reservation on every network node for bit-complement scenario in 8×8 mesh NoC.

Fig. 6.16(a) shows the measurement for node 1–32, while Fig. 6.16(b) exhibits the actual bandwidth measurement for node 33–64. Some data producer nodes inject data with high expected communication rate. Therefore, some communication pairs cannot meet the expected data communication bandwidth because the contention of the high-throughput rate traffics to acquire the same outgoing link leads to a performance bottleneck situation. Only a few communication pairs can meet the expected communication bandwidth.

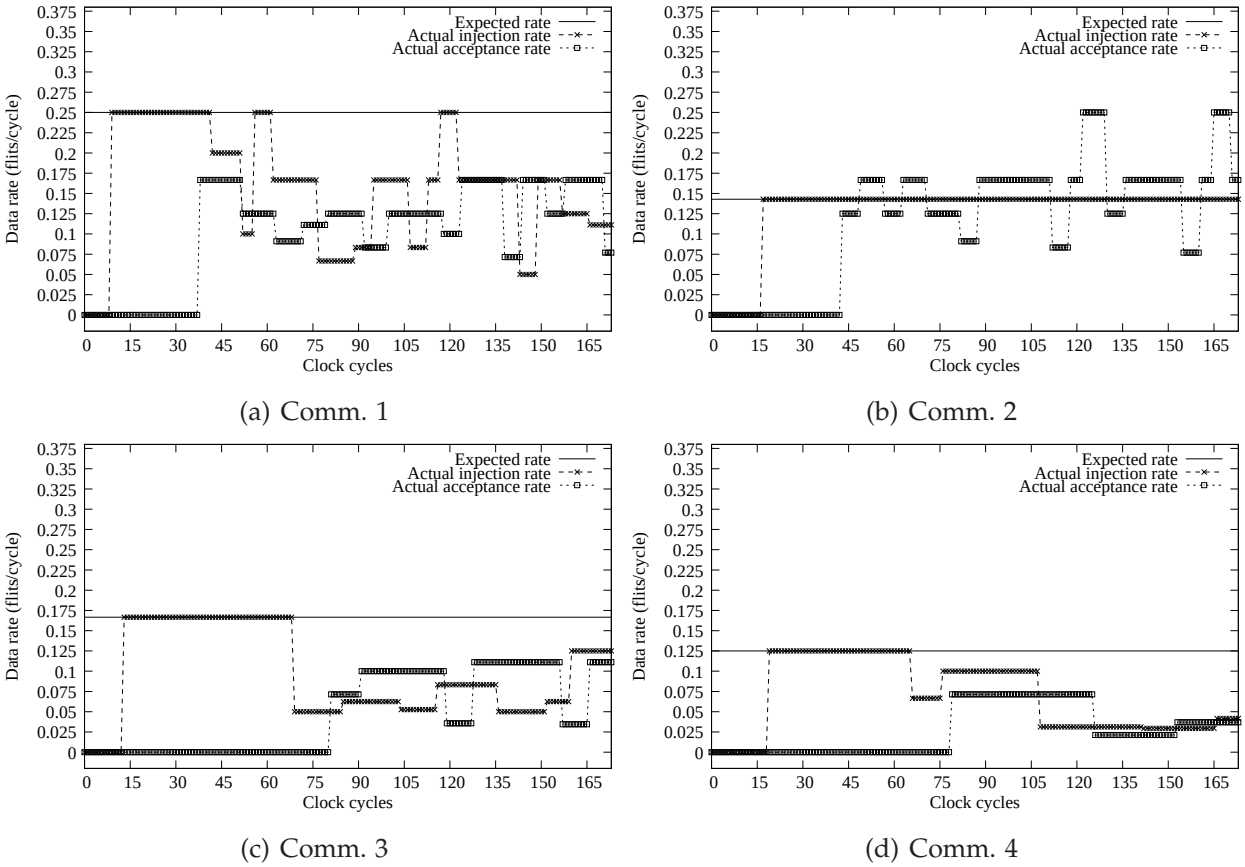
Fig. 6.17 presents the 3D surface of the distribution of the total bandwidth reservation on every network node (router) for the five adaptive NoCs under bit-complement scenario in 8×8 mesh NoC. Fig. 6.17(a) for example shows the comparison of the total bandwidth reservation between NoC router prototypes with *BW-ID* and *FQ-ID* selection parameters. In the Fig. 6.17(b), we can see that the bandwidth space occupancy between the *BW-ID* and *BW* methods are the same. In general, the proposed bandwidth-aware adaptive routing selection strategy with different selection parameters tends to move the traffic in the network center nodes. Thus, the bandwidth occupancy is concentrated in the network center.

Fig. 6.18 presents the transient responses of the actual injection and acceptance rates of 4 selected communication pairs for bit-complement traffic scenario using *BW-ID* method. The transient measurements in both figures are made from initial simulation time until the 173^{th} clock cycle period. As presented in Fig. 6.18(b), the acceptance rate of the $Comm.\ 2$ experiences some overshots at several clock cycle period and swings around the expected communication rate, while the injection rate can follow the expected communication rate. Meanwhile, the actual measured injection and acceptance rates of the $Comm.\ 1$, $Comm.\ 3$ and $Comm.\ 4$ are lower than the expected rate and fluctuate around at certain steady state points.

Fig. 6.19 presents the transient responses of the actual injection and acceptance rates of 4 selected communication pairs for bit-complement traffic scenario using *FQ-ID* method. As presented in Fig. 6.19(a), the acceptance rate of the $Comm.\ 1$ experiences overshots two times during a few clock cycle periods. The injection rate can follow the expected communication rate only in some short periods of clock cycle. Afterwards, the actual injection rates swings dynamically around a certain steady state point above the expected data rate.

6.5 Synthesis Results

The synthesis results of the five adaptive NoC routers with different routing selection function are presented in Table 6.1. The NoC routers are synthesized using 130-nm CMOS standard-cell library from *Faraday Technology* under *Design Vision* tool from *Synopsys*. The target data frequency for the five adaptive NoC router prototypes is 1 GHz. The table presents the total logic cell area, estimated dynamic power (net switching and cell internal power) and static leakage power of the NoC routers. We can see in the table that the BW-



(a) Comm. 1 (b) Comm. 2

(c) Comm. 3 (d) Comm. 4

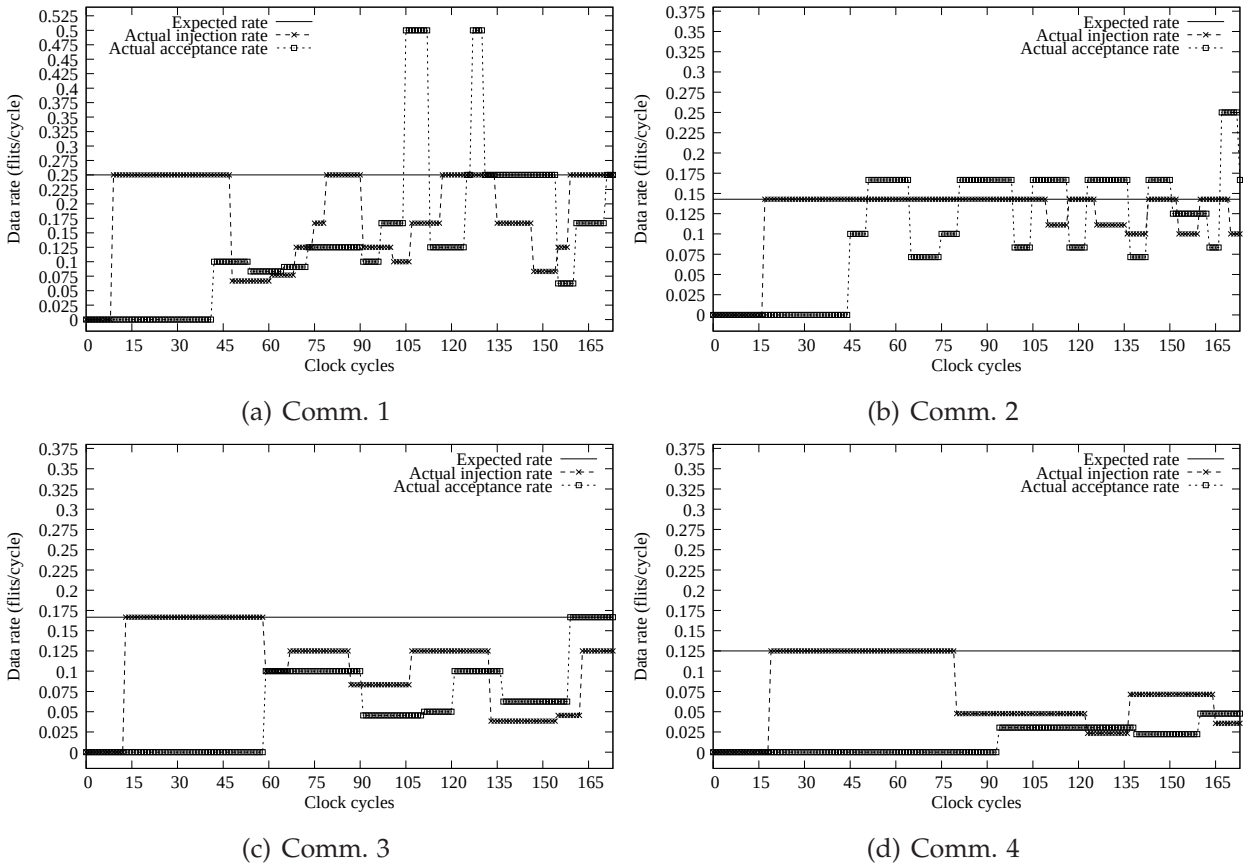
Fig. 6.18: Transient responses of the actual injection and acceptance rates of 4 selected communication pairs for bit-complement traffic scenario using BW-ID method.

ID version of the BWA adaptive routers consumes more logic cells and more powers than the other prototypes.

In Table 6.1, we can also see that the BW-version of the BWA adaptive router has larger logic cell area than the FQ-version. The area overhead is due to the area overhead of the bandwidth accumulator unit that is integrated in each crossbar multiplexor component of the router together with the ID Management unit. As presented in the table, the ID-version of the adaptive NoC router has the least logic cell area compared to the other adaptive NoC prototypes.

6.6 Summary

The contention- and bandwidth-aware (CBWA) adaptive NoC routers, which select the best outgoing port at runtime based on the bandwidth occupancy and the number of the free ID reservable ID slots were presented in this chapter. The orientation of the routing engine components of the NoC router to the number of free bandwidth spaces at alternative outgoing ports is aimed at avoiding congestion situations, in which the bandwidth capacity of communication channels is overloaded. In any case, the BWA



(a) Comm. 1 (b) Comm. 2

(c) Comm. 3 (d) Comm. 4

Fig. 6.19: Transient responses of the actual injection and acceptance rates of 4 selected communication pairs for bit-complement traffic scenario using FQ-ID method.

adaptive routing selection strategy will help to balance the bandwidth utilization of the total NoC bandwidth capacity provided by the overall communication channels.

The implementation of the BWA adaptive routing selection strategy is reasonable in heterogeneous NoC-based multiprocessor systems especially in embedded MPSoCs running a parallel task application, where several processing element cores may inject data to the NoC with different injection rates. The differences of the data injection rates between the cores are due to the application requirements, or due to hardware/software constraints and the complexity of the task executed in a certain node of the NoC. In multimedia applications, some audio/video streaming's request require different end-to-end data rates to meet the constraints of the proper application execution and scheduling. Therefore, constant bandwidth rates, which may be different for each data stream must be guaranteed. The hardware/software constraints is limited for example by the maximum data frequency of the ASIC/FPGA cores as hardware parts and the working frequency of the CPU cores as software parts, which run the executable code of a software/computer program.

As exhibited in the simulation results of the limited number of the selected traffic scenarios, the performances of the bandwidth-aware (BWA) adaptive routing and the contention- and bandwidth-aware (CBWA) adaptive routing selection strategy are almost

Tab. 6.1: Synthesis results of the adaptive routers using 130-nm CMOS technology library.

	BW-ID	FQ-ID	BW	FQ	ID
Target frequency	1 GHz	1 GHz	1 GHz	1 GHz	1 GHz
Total logic cell area	0.1504 mm^2	0.1322 mm^2	0.1436 mm^2	0.1304 mm^2	0.1300 mm^2
Est. net switch. power	22.238 mW	21.926 mW	21.989 mW	22.024 mW	21.134 mW
Est. cell intern. power	57.081 mW	55.145 mW	56.627 mW	55.216 mW	54.441 mW
Est. cell leakage power	29.50 μW	24.30 μW	27.10 μW	24.20 μW	24.30 μW

similar, and show better performance compared to the congestion-aware, contention-aware and contention- and congestion-aware (CCA) adaptive routing selection strategy. However, since the CBWA adaptive routing method considers not only the bandwidth space occupancy but also the number of messages contenting to acquires the alternative output ports, then the CBWA adaptive routing method theoretically would make efforts to balance the distribution of traffics on the NoC links.

Chapter 7

Connection-Oriented Guaranteed-Bandwidth for Quality of Service

Contents

Several multimedia applications for MPSoCs consist of some communication edges that must be performed with a certain communication bandwidth. Video/audio stream-

ing data transmission from a core to one or more than one core needs an average constant transmission rate. Hence, a specific service of the network is required to guarantee the throughput of the data streaming. Guaranteed-bandwidth or guaranteed-throughput service can be implemented based on an end-to-end connection setup method, where a stream header reserves the expected bandwidth on every communication resource during connection establishment phase before sending the video/audio streaming. By further applying a policy where every network link cannot be consumed by considered traffic exceeding its maximum bandwidth capacity, then a long-term saturated network condition can be avoided (non-blocking traffics flow is guaranteed).

Guaranteed-service can be implemented by allowing a link to be shared by multiple packets using a data multiplexing technique. The shared link configuration is also commonly called *switched virtual circuit (SVC) configuration*. The following section will present four recently published data multiplexing techniques for networks-on-chip (NoCs). One of them is our proposed data multiplexing based on local identity (ID) division with an ID management procedure.

7.1 State-of-the-art in Data Multiplexing Techniques For NoCs

7.1.1 NoCs with TDMA Technique

A commonly used method to provide guaranteed-service for NoCs is a pipeline circuit switching based on the Time-Division Multiple Access (TDMA) method. Æthereal [188] and Nostrum [157] are NoC examples that use such methodology. Fig. 7.1(a) presents the conceptional view of the TDMA method. The link connecting the input and output port of the routers is shared by four packets, i.e. packet A, B, C and D. Each packet establishes a virtual circuit configuration based on time slots allocation on the outgoing port. In the figure, we assume that the link has 8 time slots. The more time slots are allocated for a packet, the more bandwidth (BW) it reserves. Thus, the packets A, D, B and C reserve 50%, 25%, 12.5% and 12.5% of the maximum link BW capacity, respectively. A packet allocated at time slot S_t on a link must be allocated to time slot S_{t+1} on the next link. Based on Fig. 7.1(a) for instance, packet D allocated to time slots S_1 and S_2 on the link must be allocated to time slots S_2 and S_3 on the next link.

7.1.2 NoCs with SDMA Technique

The work in [131] has implemented the concepts of spatial division multiple Access (SDMA) for guaranteed throughput NoCs. The conceptional view of the SDMA method is presented in Fig. 7.1(b). The multiplexing concepts is based on the fact that NoC links are physically designed with a set of wires. The SDMA method allocates a subset of wires

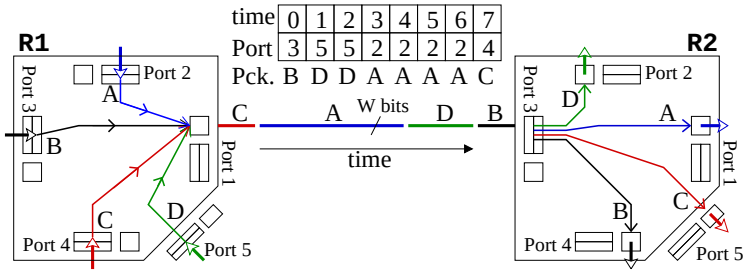
to a given virtual circuit. The more wires (the larger the subset of wires) are allocated for a packet, the more the bandwidth (BW) it reserves. Based on the Fig. 7.1(b), if the data width is $W = 32$ bits, and we assume that all considered packets consume the maximum link bandwidth, then packets A, D, B and C are allocated to a number of 16, 8, 4 and 4 wires, respectively, which means that every packet reserves 50%, 25%, 12.5% and 12.5% of the maximum link BW capacity, respectively.

7.1.3 NoCs with CDMA Technique

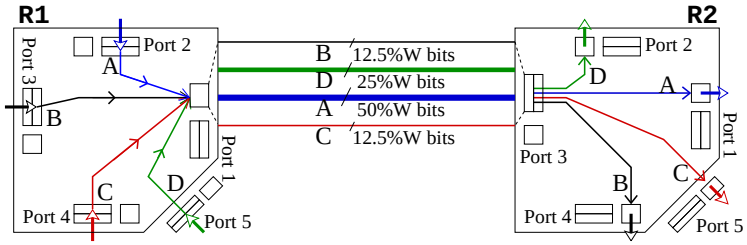
Another method for multiplexing data on NoC links based on code-division is presented in Fig. 7.1(c). An example prototype of the CDMA NoC method is introduced in [209]. The concept is implemented by introducing orthogonal spreading codes. The link can be shared by conflicting packets in which every bit of the packets are encoded and accumulated by a CDMA Transmitter and carried by the spreading codes to the next router. In the next router, the accumulated encoded packets are decoded by the CDMA receiver and then routed to the correct path. The number of available spreading codes indicates directly the maximum number of packets that can be accumulated at the same time.

7.1.4 NoCs with IDMA Technique

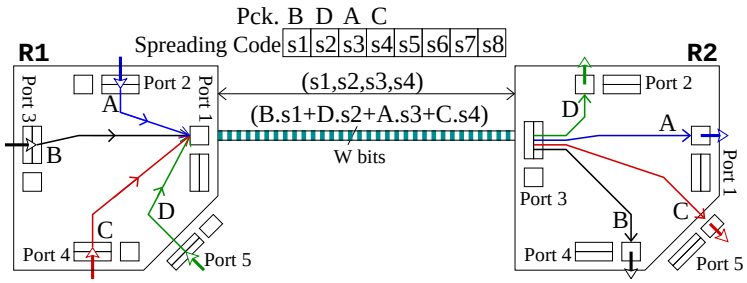
Among the aforementioned data multiplexing technique, we introduce another concept based on local identity (ID) division technique. Fig. 7.1(d) presents the concept, in which local ID slots can be reserved by single data stream as its ID-tag. The local ID tag appears on every flit and is updated every time the data stream acquires the next link. Flits belonging to the same stream will always have the same local ID. In order to guarantee a correct routing function, an *ID Management Unit* must index every reserved ID slot by identifying the previous ID tag of the stream/message which reserves one ID slot and from which port the stream/message comes. Based on the Fig. 7.1(d), for instance, packet D is allocated to local ID slot 1 (its new ID-tag), and is identified by the ID slot table as a packet from input port 5 having previous ID tag 0 in the router $R1$. In the next router, the stream/messages are routed based on their current/new ID-tags. Thus, the packet D with current ID-tag 1 is routed to Port 1 (East) in the next router $R2$. The number of available ID slots reflects the maximum number of stream/messages allowed to form switched virtual circuit configurations on the link. The bandwidth can be guaranteed by further implementing a connection-oriented communication protocol, where the requested BW attached on a header flit bit fields is used to reserve the expected end-to-end communication bandwidth over the network links.



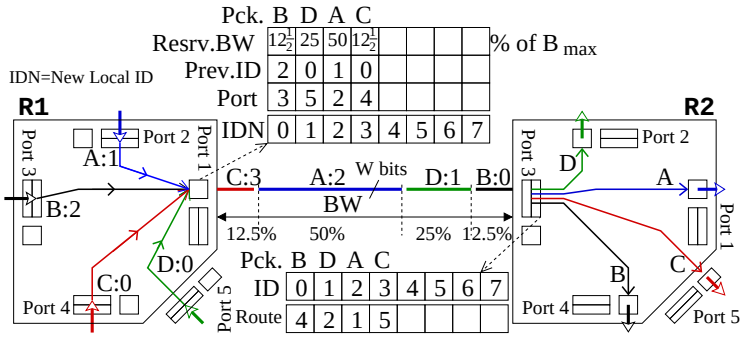
(a) TDMA-based Data Multiplexing



(b) SDMA-based Data Multiplexing

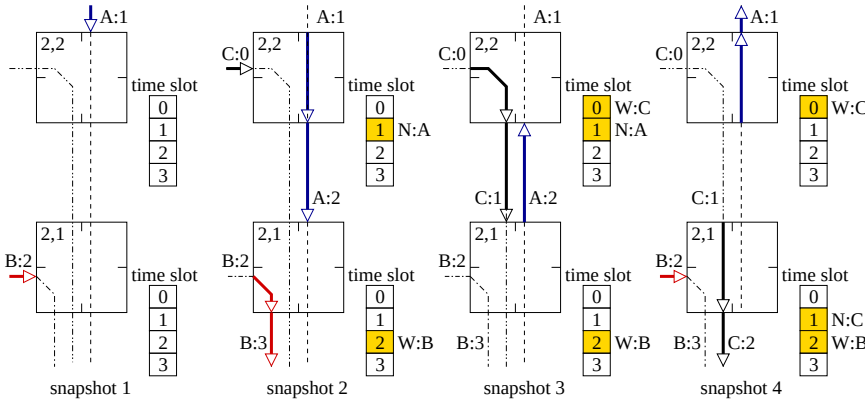


(c) CDMA-based Data Multiplexing

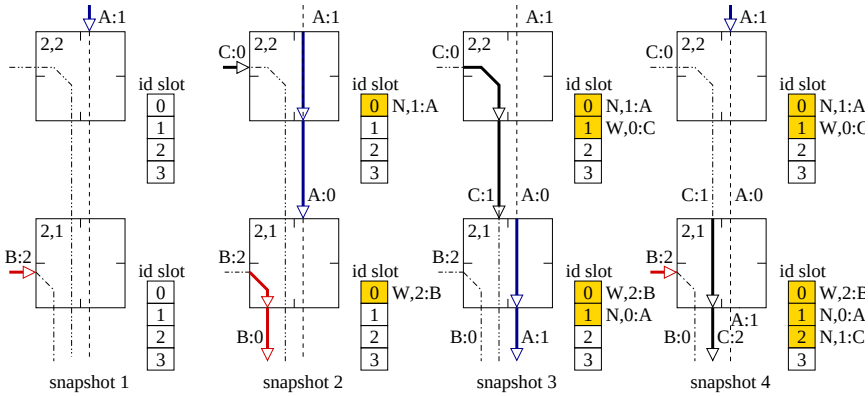


(d) IDMA-based Data Multiplexing

Fig. 7.1: State-of-the-Art of the data multiplexing techniques for NoCs.



(a) Time slot TDMA-based connection setup.



(b) Our proposed IDMA-based connection setup.

Fig. 7.2: Connection setup method using time slot TDMA-based and the IDMA-based methods.

7.1.5 Comparisons of the SVC Configuration Methods

The TDMA-based switching requires a very complex time-slot allocation algorithm to achieve a conflict-free routing and scheduling as presented in [93] ($UMARS+$), [145] (Virtual Circuit Configuration (VCC) Method), and by the time-slot allocation algorithm made for μSpidergon NoC [70]. Our proposed IDMA-based method does not need such complex time-slot allocation algorithms because the local ID slot on each outgoing link is reserved and allocated autonomously by header flits of a streaming data during application execution time (flexible runtime autonomous switched virtual circuit reconfiguration). The same technique could be certainly applied to the TDMA-based method, but the probability in which the header flit fails to establish connection is very high especially in high traffic situation as visually depicted in

In Fig. 7.2(a), three packets (*A*, *B* and *C*) are attempting to set up connections. Four snapshots of the network at successive times are presented. The Setup packet *A* enters node (2,2) from North (*N*) input port, and Setup packet *B* enters node (2,1) from West (*W*) input port as shown in Fig. 7.2(a)(a). The script (**A:1**) means that packet *A* will be programmed in time-slot *1* in the next router, and (**B:2**) has the same meaning as well.

As shown in Fig. 7.2(a)(b), packet *A* has been forwarded to South output port of node (2,2), and the time-slot *1* is allocated for packet *A* coming from *N*. While packet *B* has also been in South output port of node (2,1), and the time-slot *2* is allocated for packet B coming from *W* input port. A bold line shows the progress of the connection setup over time. In every snapshot, the Setup packets are routed to their next link and the slot table is incremented by one. Thus, in the next router, packets *A* and *B* will be programmed in time-slot *2* (**A:2**) and *3* (**B:3**) respectively.

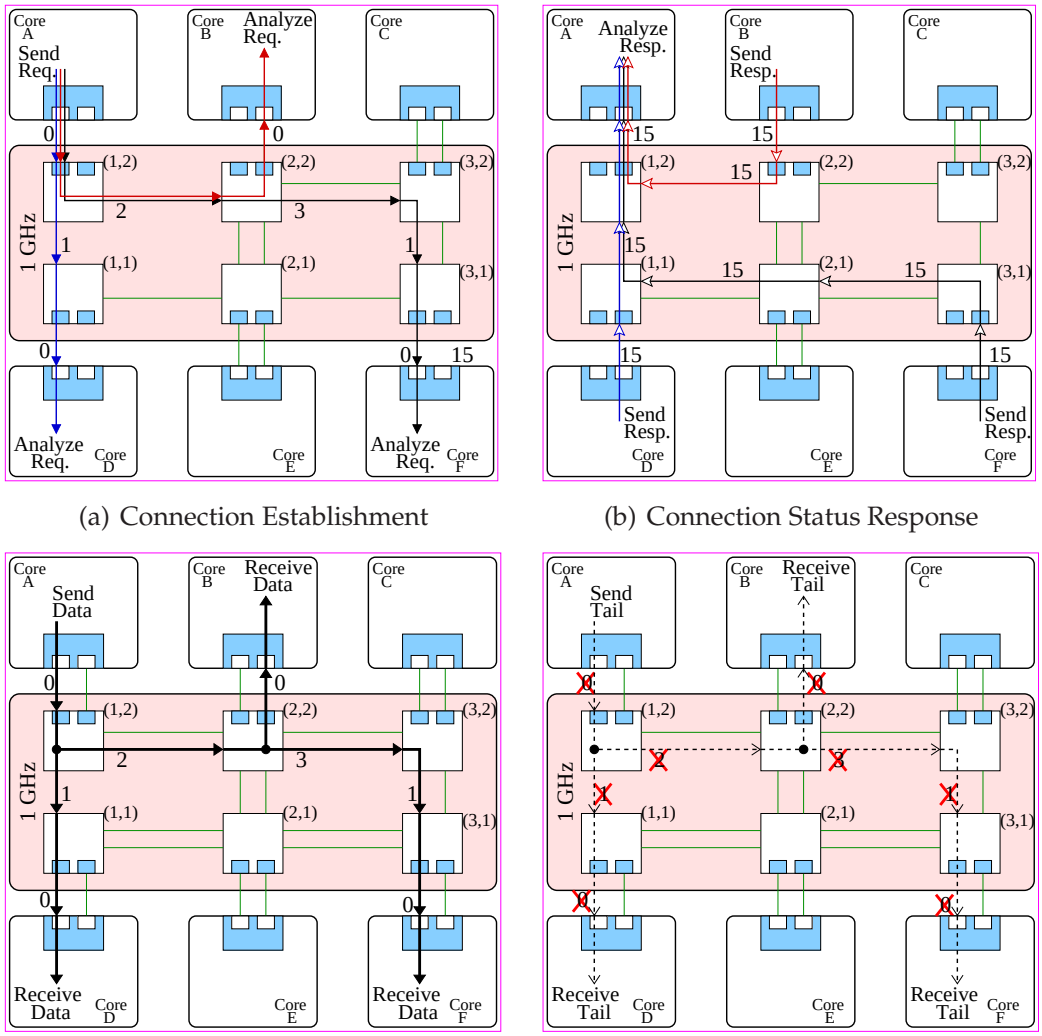
In Fig. 7.2(a)(c), packet *A* cannot reserve slot *2* for South output port (*S*) of node (2,1) because it has been reserved for the connection of packet *C*, thus the connection setup of packet *A* fails. Therefore, packet *A* is routed back along its path to remove the reservations made so far (see Fig. 7.2(a)(c)). In Fig.7.2(a)(d), packet *A* has removed the reservation of slot *1* that has been made in Fig. 7.2(a)(b).

The previous explanation has presented the disadvantage of using *Time-Division Multiplexing* (TDM) switching method by Æthereal NoC. In Fig. 7.2(a)(c), there is still three free time-slots in South output port of node (2,1) i.e., time-slot *0*, *1*, and *3*. However, Packet *A* cannot use that free time-slots, because Packet *A* has been programmed to reserve time-slot *2* that has been reserved by Packet *B* from West (*W*) input port. Therefore, we propose a more optimistic approach by introducing *ID-Tag Mapping Management* (*IDM*) unit to optimize dynamically the link bandwidth utilization. The *IDM* also consists of an *ID-Slot Table* and has the same functionality as *Slot Table* used by Æthereal.

The communication link setup by XHiNoC is presented in Fig. 7.2(b). Once again, four snapshots of the network at successive times are depicted in the figure. For the sake of simplicity, only the ID slot table of the *IDM* in South output port is presented for both mesh nodes. In Fig. 7.2(b)(a), a packet header *A* with ID-tag *1* (**A(1)**, numerical value in the bracket represents ID-tag) enters node (2,2) from North (*N*) input port, while a packet header *B* with ID-tag 2 (**B(2)**) enters node (2,1) from West (*W*) input port.

In Fig. 7.2(b)(b), the packet header *A* is routed to South (*S*) output port and is allocated to ID slot *0*. Therefore, all payload flits having ID-tag *1* from *N* input port in node (2,2) will get new ID-tag *0*. In Node (2,1), packet header *B* is routed to *S* output port and is allocated by *IDM* to ID slot *0*. Thus, the South *IDM* unit in this node will map all flits having ID-tag *2* from West (*W*) input port to receive new ID-tag *0*. The bullets in the figure indicate that the ID slots are being used (not free).

In Fig.7.2(b)(c), a packet header *C* coming from *W* input port with ID-tag *0* is routed to the *S* output port. The South *IDM* unit maps packet *C* into ID slot *1*. Hence, each payload flit having ID-tag *0* from *W* input port in node (2,2) will get new ID-tag *1*. While packet *A* coming from *N* with ID-tag *0* is allocated by the South *IDM* unit in node (2,1) to ID slot *1*. Hence, it receives new ID-tag *1* before being forwarded from node (2,1) into the next node. Fig. 7.2(b)(d) describes also the same mechanism, where packet *C* in node (2,1), which is coming from *N* input port with ID-tag *1* will be mapped to receive the new ID-tag *2*.



(a) Connection Establishment (b) Connection Status Response

(c) Data Transmission (d) Connection Termination

Fig. 7.3: Connection-oriented multicast routing protocol.

If a header flit requests a new ID slot allocation to reserve link bandwidth, then the *IDM* unit in each output port will search for a free ID-tag. After finding a new free ID-tag, the *IDM* unit will identify and record the current ID of the header and from which port it comes. Therefore, each payload flit belonging to the same message (because of having the same ID-tag) will be mapped by the *IDM* unit to receive the new ID-tag by using ID-based look-up table mechanism. Therefore, the XHiNoC has a lower probability that the packet header fails to establish connection at runtime compared with the existing time slot TDM-based method.

The complexity of the time-slot allocation algorithm in the TDMA-based method is due to the conflict-free constraint. In the IDMA-based method, the multicast conflicts, which potentially lead to a deadlock configuration (multicast dependency), are allowed and well organized by using the similar multicast conflict management presented in [224] and [232]. The multicast flow control is based on the fact that a multicast data will not be released from FIFO buffer at input port if all of the set multicast routing requests

has not been granted to access the multiple output ports. If a subset of the requests is granted, then the granted requests will be reset to avoid improper multicast flit replications. However, the work in [224] and [232] presents a deadlock-free tree-based multicast routing with best-effort without providing a guaranteed-bandwidth service. This chapter will present the same multicast routing concept with additional connection-oriented guaranteed-bandwidth service.

The main drawback of the CDMA-based switching method is the larger latency, in which the next downstream router must send back a spreading code to an upstream router from where the packet flits will be encoded with the spreading code. Indeed, since one original data bit will be spread into sequential S bits after encoding, the degree of the data transfer parallelism between the CDMA Transmitter and Packet Sender/Receiver blocks affects the data transfer latency in the CDMA NoC largely [209].

As mentioned openly in [131], beside its interesting characteristics, the SDMA-based method has a few drawbacks. Best-Effort service implementation by using the SDMA method will increase the area overhead, because introducing some buffers to prevent contention would require deserializing data in the routers. Compared with traditional crossbar interconnects, the SDM networks rely on multistage switches that are more complex. End-to-end flow control is also difficult to implement since it would require either a dedicated low-bandwidth communication architecture for control or reserving a 1-wire circuit for flow control.

The work in [136] also presents a NoC design methodology based on three kernels, i.e. traffic classification, flit-based switching and path pre-assignment and link-BW setting. The traffics are classified into guaranteed-latency (GL), guaranteed-bandwidth (GB) and best-effort (BE) traffics. The GL traffics have stringent maximum delay requirement from data injection until data acceptance. The GB traffic requires constant end-to-end communication bandwidth, while the BE traffic does not have bandwidth requirement neither stringent data transfer latency. The link allocation (path assignments) in [136] for the GL and GB traffic is static or computed off-line at design time. For a new application, the path assignment must be done again at design time. Therefore, the proposed methodology is not suitable for application mapping, where the applications are known after chip-manufacturing. The work in [200] also presents a scheduling function for a time-constraint streaming communication on NoCs. The methodology considers communication scenario overlap during application exchanges. However, the scheduling functions together with routing path assignments are also computed at design time after application mapping phase.

By using the IDMA-based method, a link-level flit flow control can be easily implemented with a credit-based method between two NoC routers to allow at-instant-time conflict between flits of different data streams, and to control the flow of wormhole packets when our NoC would be designed for Best-Effort Service. By using a runtime connection establishment made autonomously by a header flit, where the requested bandwidth is attached on the header and tail flits, the average communication bandwidth of the data

stream can be guaranteed. Thus, it results in an easy implementable end-to-end flow control initiated by a compute element at a sender node, and is suitable for the post-chip-fabrication application mapping.

7.2 Connection-Oriented Communication Protocol

The connection-oriented multicast routing protocol implemented in our NoC consists of four main phases, which are explained in the following items.

1. *Connection establishment*. In this first phase, the data producer node will scatter header flits to multiple destination nodes (See Fig. 7.3(a)). If the data will be sent to a single destination node (unicast data communication), then the producer node will send only one header flit.

2. *Connection Status Response*. In this second phase, every destination analyzes the header flit to know if the header has successfully established the multicast connection and reserved bandwidth on each communication media to guarantee the required communication bandwidth. Therefore, every destination sends back a response flit to inform the data producer node about the status of the connection (See Fig. 7.3(b)).

3. *Data Transmission*. If the producer has known that the multicast connection is successfully set up and the bandwidth on each reserved communication media is guaranteed, then it will send the data stream into the NoC through the same path set up previously by the header flits. Fig. 7.3(c) shows one of many possible ID slot reservation results. If one of the multicast connections is not successfully established, then the producer node will firstly terminate the multicast connection by sending a tail flit, then send again new header flits.

4. *Connection Termination*. In the fourth phase as presented in Fig. 7.3(d), tail flit is injected to close the multicast connection at the end of the data sending phase.

In order to close the established connection, a tear down method is not used, in which a single-flit control message/packet is backtraced from target to the source node following the connection path to remove the reserved communication resources. Instead, a progressive approach for connection termination is used as described before and depicted in Fig. 7.3(d). The backtraced routing method can not only increase complexity of the routing engine structure because of the need for a special module to record the routing history of the message header, but also can cause a backtrace deadlock configuration as presented in Fig. 7.4. In the figure, it looks that Packet A injected from North2 (N2) port on router $R1$ and Packet B injected from East (E) port on router $R2$ make a backtrace routing. Hence, deadlock configuration occurs because channels requested by Packet A and B are full and being shared and used by other packets.

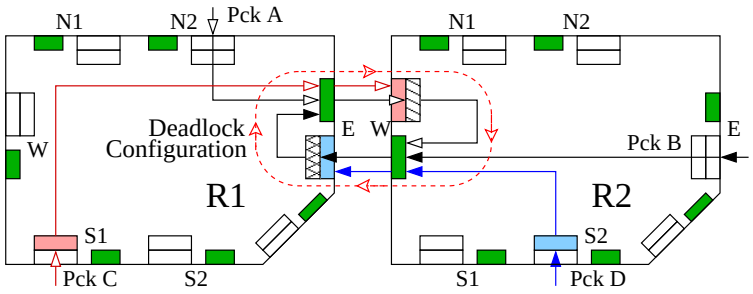


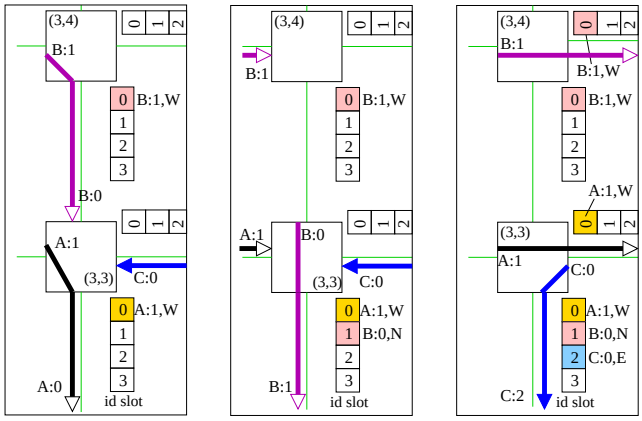
Fig. 7.4: Deadlock configuration when enabling backtrace.

7.2.1 Runtime Local ID Slot and Bandwidth Reservation

Local ID slot reservation in our NoC is made autonomously by the header flits at runtime during application execution. Hence, a specific slot allocation algorithm like time slot allocation in TDMA method is not needed. Conflicts between flits in the IDMA-based switching method is allowed. Fig. 7.5 presents 3 snapshots on how the conflict between header flits is solved autonomously by the header flits. At router node (1,1), we can see in Fig. 7.5(a) that header flit A from West input port with current ID tag 1 ($A:1$) reserves ID slot 0 and programs the reserved ID slot with its previous (old) ID and the port number from which it comes ($A:1,W$). The stream/message A will use the link connected to the South outgoing port of router node $(3,3)$ with new ID tag 0 ($A:0$).

In the same router node, the headers of stream $B:0$ from North and stream $C:0$ also arrive at the same stage. They will compete with each other to acquire the same South output port. We assume that the arbiter unit at the South output port firstly selects header flit B as shown in Fig. 7.5(b). Now, the header flit B reserves the next free ID slot 1, and header flit C must be held in the FIFO buffer while waiting for selection from the arbiter unit. Header flit C at the next stage can be switched afterwards to the South outgoing port and it reserves the next free ID slot 2 as presented in Fig. 7.5(c). Now, 3 ID slots have been reserved by the stream/message A, B and C. In the next time stages, when data payload flits of the data streams are switched out to the South port, then the IDM unit will assign every flit with the reserved id slots by identifying the old ID-tag of the flit and from which port the flit comes.

Fig. 7.6 shows four snapshots of three data streams A, B and C competing to acquire the same outgoing link, which is the extended view of the Fig. 7.5. On the right-side of the Fig. 7.6, we can see the current status of the reserved ID Slot and BW of the directional link from the South output port of the router node $(3,3)$ to the North input port of the router node $(3,2)$, and the input port number, the previous and new ID-tag of each flit flowing on the link in every time stage t. The streams A, B and C are injected to the NoC to consume 33.33% of the maximum BW capacity (B_{max}) of the NoC link (33.33% B_{max}). Thus in the figures, the consecutive flit is shown three hops behind the previously routed flit in non-blocking situation. Fig. 7.6(a) shows that flits $A1$, $B1$ and $C1$ compete each other to acquire the link connecting node $(3,3)$ and $(3,2)$, and it looks also for instance



(a) Snapshot 1 (b) Snapshot 2 (c) Snapshot 3

Fig. 7.5: Autonomous runtime local ID slot reservation allowing conflict of multicast headers.

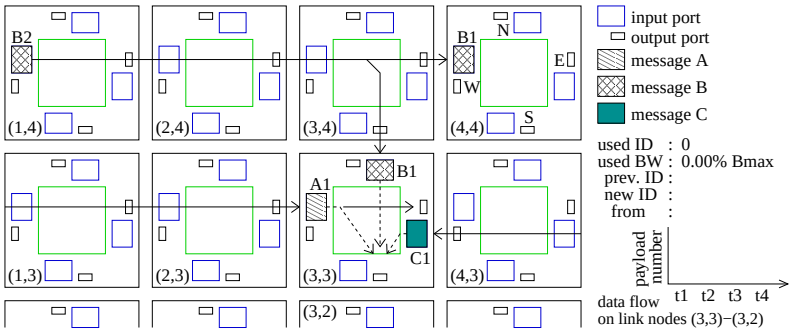
that flit $B2$ is three hops behind the flit $B1$.

Fig. 7.6(b), Fig. 7.6(c) and Fig. 7.6(d) show respectively the next snapshots when the flits $A1$, $B1$ and $C1$ are switched out to the link between node $(3,3)$ and $(3,2)$. It look in the figures, that the consecutive flits of the message A, B and C keep moving even if the previously routed flits are blocked due to contention. For instance, flit $B2$ moves from node $(1,4)$ to $(2,4)$, to $(3,4)$ until node $(2,3)$ as shown in Fig. 7.6(a) (Snapshot 1), Fig. 7.6(b) (Snapshot 2), Fig. 7.6(c) (Snapshot 3) and Fig. 7.6(d) (Snapshot 4), respectively. However, because each of the tree contenting flits requires only $\frac{1}{3}$ of the link BW capacity, the total BW consumption of the three messages does not exceed the maximum BW capacity of the link B_{max}, or it is exactly equal. Therefore, they can shares the link well with constant data throughputs. This situation will certainly have no impact on the data injection and acceptance rates in their source and destination nodes. Thus, the end-to-end communication BW of each message is guaranteed and performed automatically at runtime. Based on the data flow demonstration in the Fig. 7.6, our guaranteed-BW service with the IDMA method is tolerant to message contention, which is certainly difficult to achieve, when we use the TDMA method.

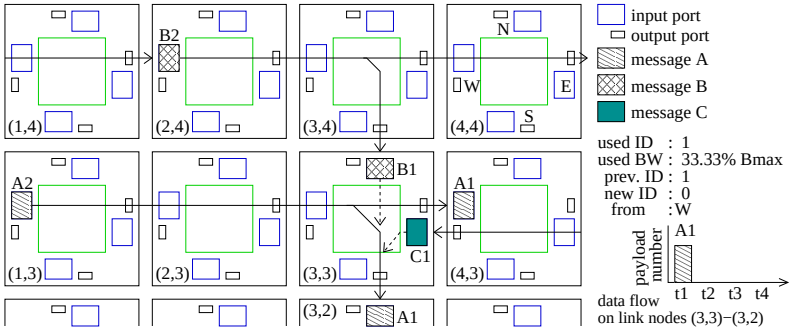
7.2.2 ID-based Routing Mechanism with Bandwidth Reservation

As explained in Section 7.2.1, every message or data stream reserves one ID slot as its ID-tag on each communication link, and flits belonging to the same message will have the same ID tag. Therefore, at every input port, we implement a routing engine, which consists of a *routing reservation table* (RRT) and a *routing state machine* (RSM). Fig. 7.7 shows conceptional view of the ID-based routing organization by presenting a link connecting the output and input ports of two adjacent routers ($R1$ and $R2$).

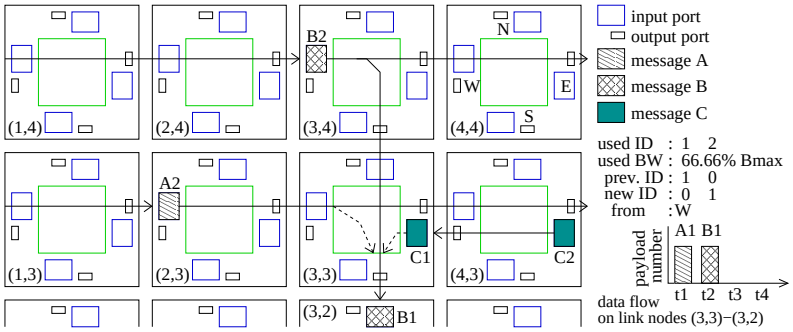
In Fig. 7.7(a), we can see that a header flit with local ID-tag 1 from West (W) input port is switched out to an output port of the router $R1$. The header flit will search for a



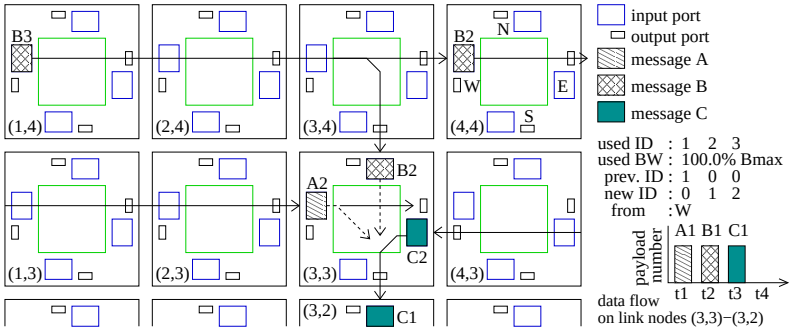
(a) Snapshot 1



(b) Snapshot 2



(c) Snapshot 3



(d) Snapshot 4

Fig. 7.6: Conflict management and link sharing for contenting multicast payload flits.

free ID slot that can be used as its new ID-tag and also check the BW availability of the required link. If the freely available BW on the link is less than its required bandwidth ($AvBW < ReqBW$ or $UsedBW + ReqBW > MaxBW$), or there is no more free available ID slot on the link, then the header flit will be assigned to a status control ID slot M. In our current NoC with the ID-tag field is 4 bits resulting in 16 available local ID slots per link. Hence, M is set to "`1111`" or "`0xF`". Once a header flit is assigned with ID tag `0xF`, it will be always assigned with the same ID tag when entering the next links until it reaches its target node. The destination node will then send back a response flit to the source node to inform that the header fails to reserve bandwidth or ID slot on a certain link in the NoC.

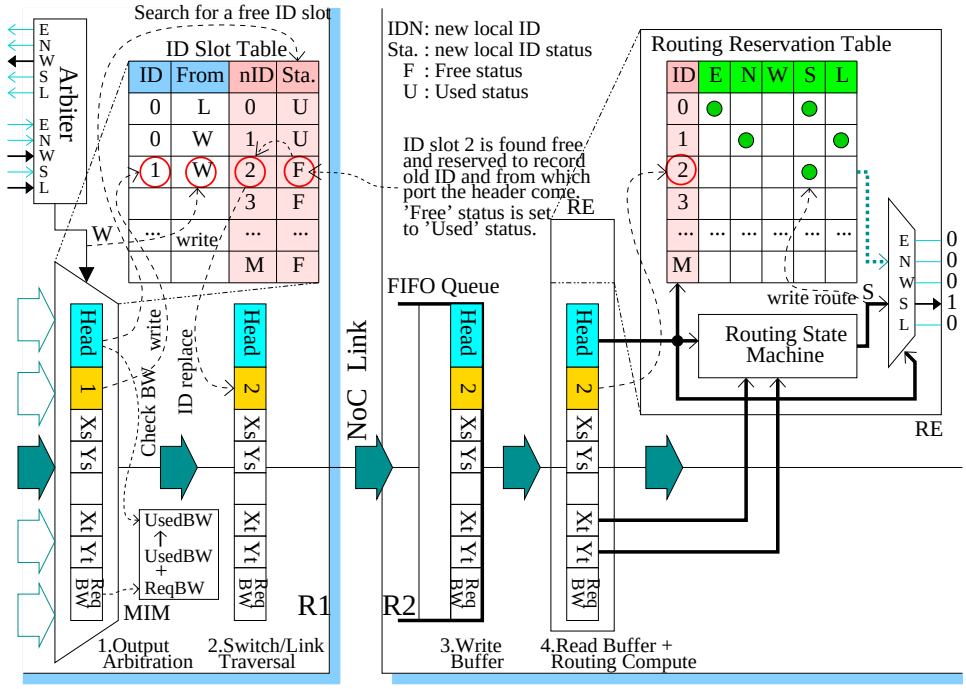
In Fig. 7.7(a), we assume that BW space and free local ID slots are still available. It looks that the ID slots 0 and 1 have been used ("U") by other stream/messages, and the local ID slot 2 is still free ("F"). Hence, the header flit reserves the ID slot 2 as its new ID tag, and set the status of the local ID slot 2 from "free" to "used" state. The BW accumulator unit in the MIM module will then add up the current used BW space with the required BW of the flit ($UsedBW \Leftarrow UsedBW + ReqBW$). Thus, the number of available BW spaces on the link is now reduced ($AvBW \Leftarrow MaxBW - UsedBW$).

As long as the FIFO buffer at an input port of the router $R2$ is not full, the header flit is buffered in the FIFO queue. In the next cycle, the header flit is buffered in the single buffer of the RE unit and at the same time, the RSM unit computes the routing direction (South S in this example) based on the target address appear in the Xt and Yt bit fields of the header . This routing information is also assigned in one slot of RRT unit based on its local/current ID tag. In this case, the current local ID tag is 2, thus the South routing slot is written in the register (slot) number 2 of the RRT unit.

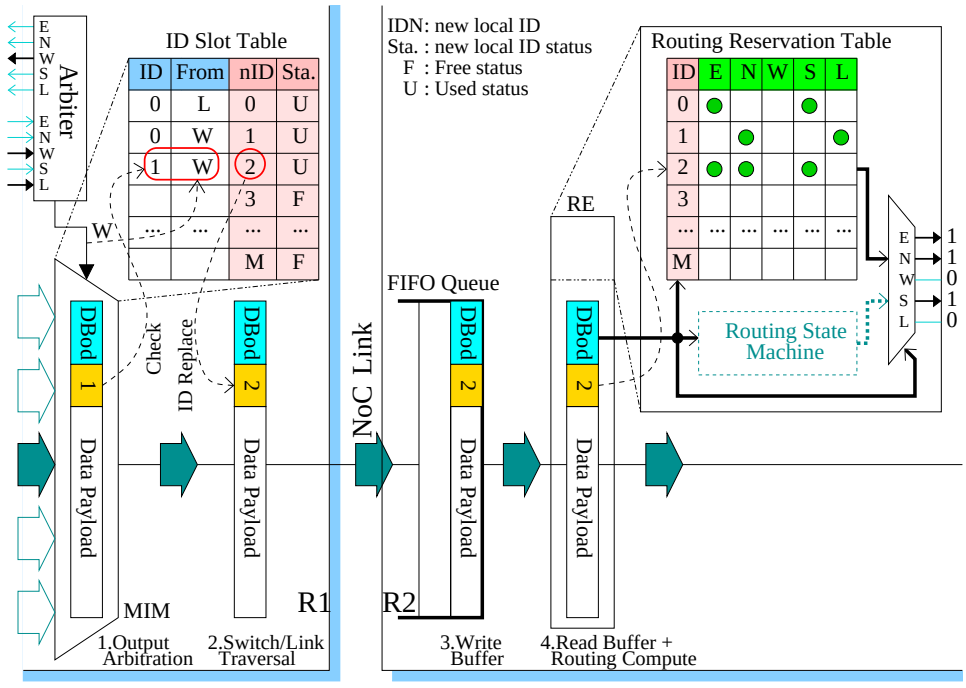
Other headers belonging to the same stream and flowing to the same link as well as the data payloads will not make a local ID slot reservation. The ID management unit will give them a new ID-tag by checking their ID-tag and the port number from where they come, and then find a match information the ID Slot Table as shown in Fig. 7.7(b). In the same manner, the RE routes them by fetching a routing direction from the RRT slot number according to their ID-tag. In Fig. 7.7(b), we can see a payload flit ($DBod$) with ID-tag 1 coming from input port W (West) is switched out from an output port of router $R1$ to the NoC link, and is assigned with the new ID-tag 2 according to the match information found in the ID Slot Table. Thus, the payload is in-flight on the link with local ID-tag number 2 and flows to the input port of the next router. The payload flit is then routed by the RE unit at an input port of the router $R2$ by fetching a routing direction from the slot number 2 (according to its local/current ID-tag) in the Routing Reservation Table.

7.2.3 Experiment on Radio System with Multicast Traffics

In this section, the connection-oriented multicast method will be verified on a radio system application benchmark from Nokia [145]. The allocation of each task after application

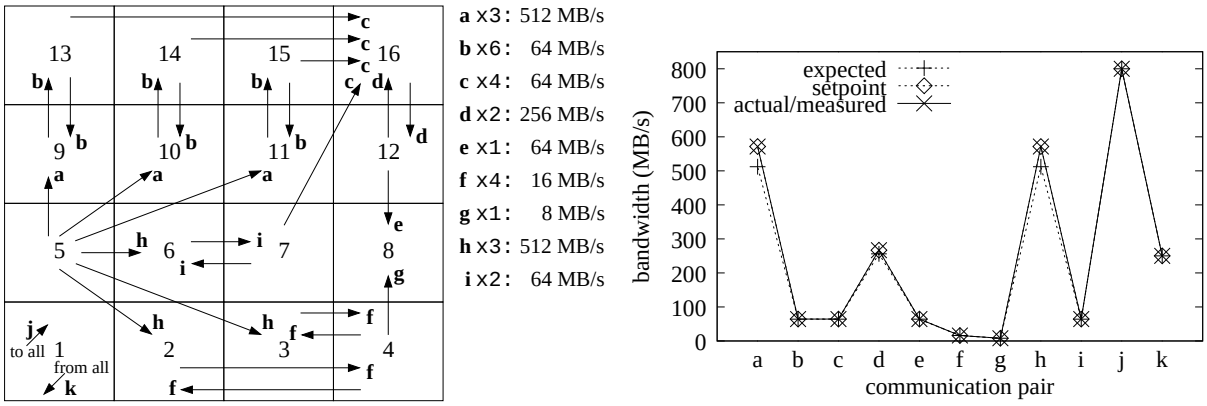


(a) ID-tag update and routing reservation by a header flit



(b) ID-tag and routing indexing by a databody flit

Fig. 7.7: Local ID slot reservation (indexing) and routing table slot reservation (indexing).



(a) Traffics on-chip radio system (b) Setpoint and actual communication BW

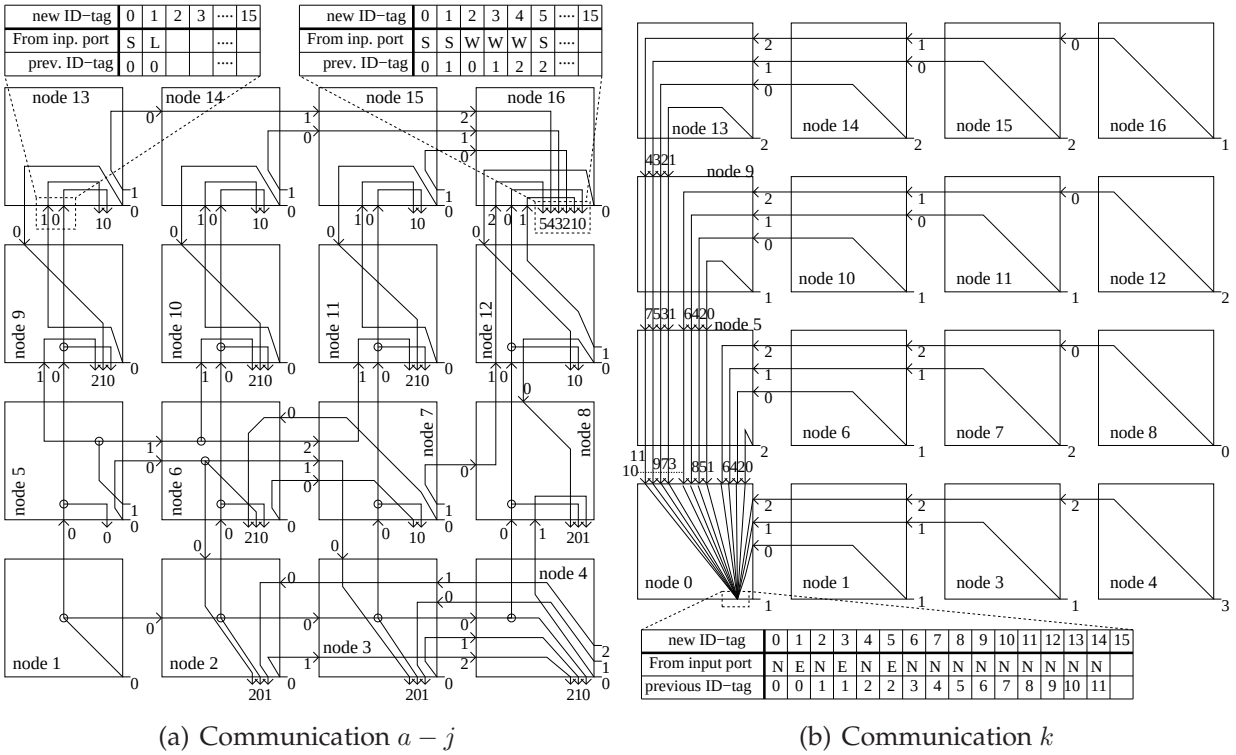
Fig. 7.8: Node-to-node traffic flow for an on-chip radio system and the bandwidth measurement results.

mapping on 2D 4×4 mesh topology is presented in Fig. 7.8(a). In general, the application consists of 11 communication edges. Two of them i.e. communication **a** and **h** are multicast data communication. Communication **j** is a broadcast (one-to-all) data communication, where the core at node (0,0) broadcasts data to all other cores. Communication **k** is all-to-one data communication, i.e. core at node (0,0) receives data from all cores.

The injection rate in our NoC is controlled by inserting jitters between two consecutive data flits. Jitter is a zero flit inserted in one cycle period. The more jitters inserted between two consecutive data flits, the lower the setpoint of the bandwidth (BW). If one data flits are injected with N_{jit} number of jitters in between, then the BW setpoint will be $I_{rate} = \frac{1}{N_{jit}+1}$ flit per cycle or word per cycle, or one data flit is injected in every $N_{jit}+1$ number of cycles. The maximum BW capacity of our NoC link is $1\ GHz \times 4 \times \frac{1}{2} = 2000\ MB/s$. Since our NoC router can perform 5 simultaneous IO connections, then the maximum BW capacity of the NoC router is $5 \times 2000\ MB/s = 10\ GB/s$. Thus, if we expect a BW of $512\ MB/s$, then we can set $N_{jit} = 6$ resulting in BW setpoint of $4000 \times \frac{1}{6+1} = 571.43\ MB/s$. If we set $N_{jit} = 7$, the the BW setpoint will be $4000 \times \frac{1}{7+1} = 500\ MB/s$. Finer BW granularity can be controlled by a compute element which actually produces data that will be sent to network interface with an expected BW or end-to-end data rate.

Fig. 7.8(b) shows the measurement of the expected, setpoint and actual measured BW for the on-chip radio system application. It looks like the minimum bandwidth requirements of all communication edge can be meet. The expected bandwidth is shown on the right side of the radio system traffic scenario in Fig. 7.8(a). The setpoint BW is set to each source node injection rate in the RTL (testbench) simulator based on the insertion of the number of jitters to set the BW requirement. The actual and average bandwidth communication of each communication pair is measured at the destination node.

For the sake of simplicity, the ID slot reservation results of the communication $a - j$ and communication k is separated. Fig. 7.9(a) presents one of many possible local ID slot reservations made autonomously by the header flits for communication edge a until j. If

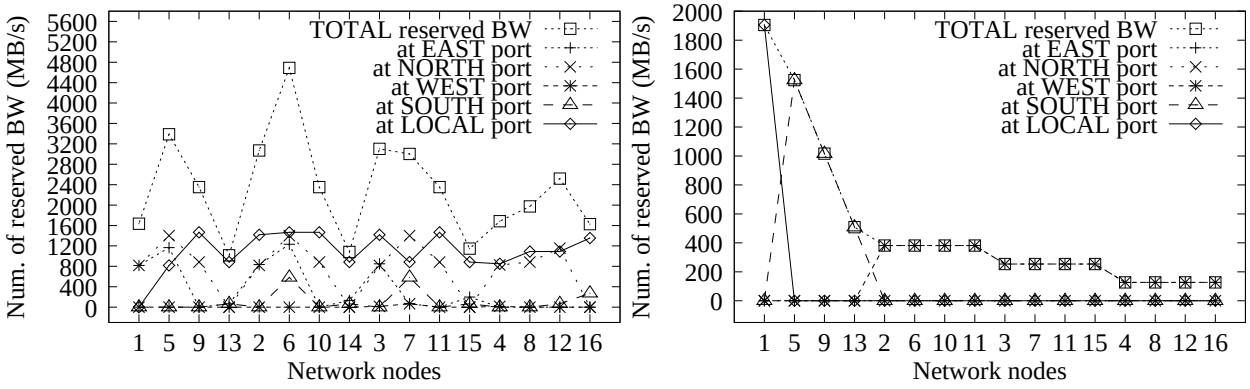


(a) Communication $a - j$ (b) Communication k

Fig. 7.9: One of many possible runtime local ID slot reservation configurations for Communication $a - j$ and Communication k.

the communication j is performed firstly, the multicast header flits will be sent flit-by-flit to all other nodes. Thus, this broadcasting communication edge will reserve the first local ID slots on every communication media (i.e. ID slot 0). The other communication edges (communication a–i), which arrive later to use the medium will then reserve the rest of the ID slots that have not reserved by other communication pairs including the communication j. Fig. 7.9(b) also depicts one of many possible combinations of the local ID slot reservations for communication edge k when this connection is set up after all other communication edges have established their connections. This communication edge will also reserve the local ID slots that have not been reserved by the communication a–j. The bottom part of the figure exhibits the ID slot reservation in the Local output port of the router node 1. The runtime of the local ID slot reservation is very flexible because the header flits, which reserve the ID slots autonomously, will check any available (free) ID slots that can be utilized as their ID-tag from the ID slot table.

Fig. 7.10 presents the reserved BW spaces on every output port of all 16 router nodes as well as the total BW consumption of all output ports on each node. Fig. 7.10(a) shows the BW reservation for communication edge a until j, while Fig. 7.10(b) presents the BW reservation for communication j. The figures represent a congestion situation on each router node and hotspot locations in the network. In Fig. 7.10(a), we can see that the hotspot occurs at node 6, where total BW consumption of all output port in the node is about $4100\ MB/s$, or about 41% of total maximum BW capacity ($10\ GB/s$) of the router.



(a) Communication $a - j$.

(b) Communication k.

Fig. 7.10: Number of bandwidth reservations at each outgoing port of all 16 network nodes.

In Fig. 7.10(b), we can see clearly that the hotspots are located in the local output port of the router node 1, in the south output port of the router node 5 and of the router node 9, respectively.

7.3 Combined Best-Effort and Guaranteed-Throughput Services

7.3.1 Microarchitecture for Combined GT-BE Services

The generic microarchitecture of the XHiNoC router is presented in Fig. 7.11. The router is designed with modular-oriented method, where each modular component is regularly instantiated for each input-output port. The XHiNoC in general, consists of three components in incoming port i.e., a FIFO buffer for Best-Effort (BE) messages (*QBE*), FIFO buffer for Guaranteed-Throughput (GT) messages (*QGT*)) and a *Routing Engine with multiplexed data buffering* (*REB*). In each outgoing port, there are two components, i.e. a *Multiplexor with ID-tag Management unit* (*MIM*) and an *Arbiter* (*A*) unit. In order to keep the router size small, the depth of each virtual channel is set to 2. As shown in Fig. 7.11, only single FIFO queue is allocated in the Local input port. Both the BE and GT messages can be buffered in the single FIFO queue component.

The ID management unit plays an important rule to interleave flits of different message in the same queues and to perform the flexible runtime communication resources reservation. The ID management unit is implemented in the *MIM* component at each output port. The detailed interconnected data and 1-bit control nets in the crossbar switch are presented in Fig. 7.12. The arbiter unit selects a data flits from input ports, which will be switched to the output port of the MIM module.

Because of using the wormhole cut-through switching with the ID-slot management, a contention between two data flits to acquire a similar outgoing channel can occur. There-

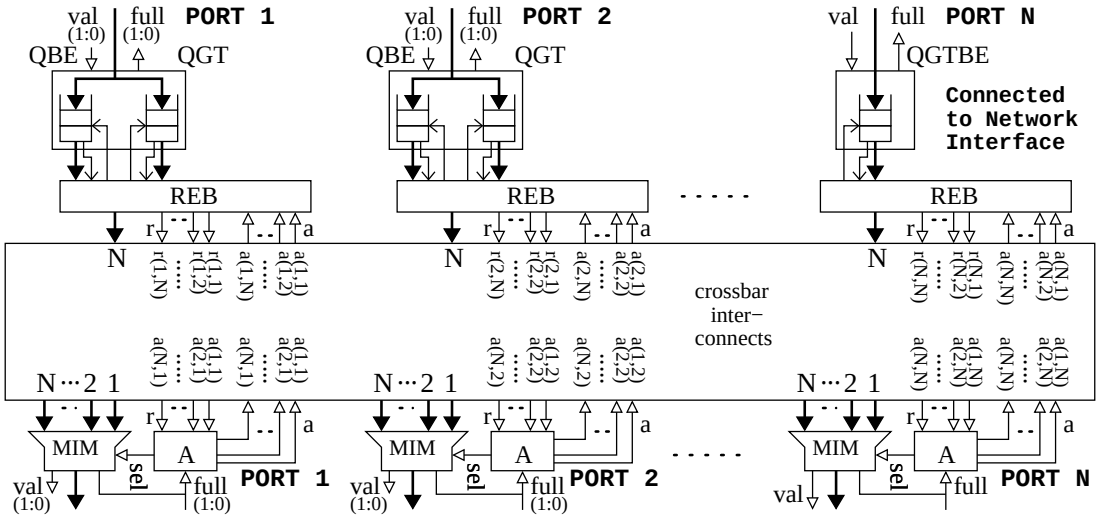


Fig. 7.11: Generic router architecture.

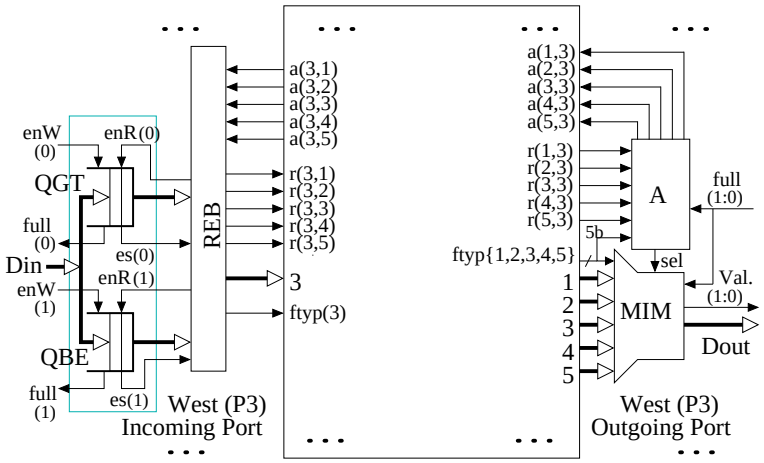


Fig. 7.12: Example of the general components in the West incoming and outgoing port.

fore, our NoC is also equipped with a link-level control to avoid data overflow in the NoC. When a contention happens, FIFO queues occupied by the contenting data flits at incoming ports will be busy or might be full. The congestion (full condition) signals are then traced back to the upstream nodes to avoid other data flits entering the congested FIFO queues. Fig. 7.12 presents the full flag (*ff*) signals from FIFO queues in one router to the modules A (arbiter) and modules MIM (multiplexor with ID-management unit) in the neighbor router.

Fig. 7.13 shows the detail structure of the modular units in the input port. The REB component consists of a *Dual-In Route Buffer*, a *Routing Engine*, a *Selector Unit (SU)*, and a *Grant Controller*. The *RE* component gives priority to route data from *QGT* buffer and stores the data in the Dual-In Route Buffer, if each buffered data in the *QBE* and the *QGT* request a routing service to the *RE* component. The *RE* will serve the routing request from the data in the *QBE* after the *QGT* is empty. However, in case that a BE flit is being

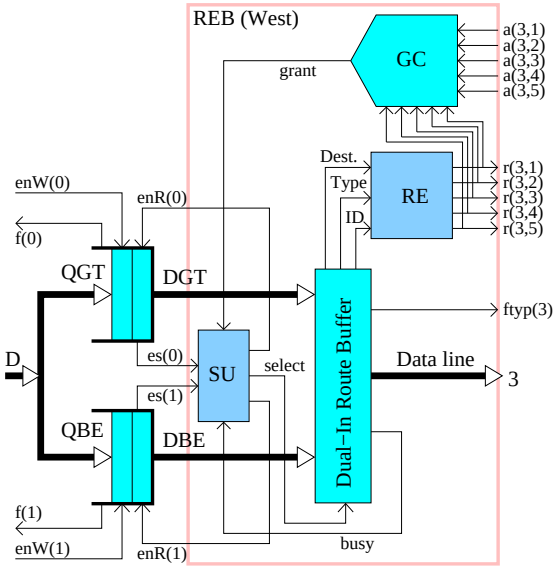


Fig. 7.13: The detail components in the incoming port.

routed and being stored in the Dual-In Route Buffer of the *RE* unit, while a GT flit is just being buffered in the *QGT*, then the *RE* unit will serve the GT flit after the BE flit has been switched out to its destinate output port. The aforementioned mechanism is controlled by the *Selector Unit* (*SU*) as presented in Fig. 7.13.

As shown in Fig. 7.13, the full flags are a two-bit signal. One bit is from the BE Queue (QBE) and the other bit is from the GT Queue (QGT). If an arbiter unit in an upstream node will select a BE flits which will be switched out to the link in the next cycle, and the full flag from the QBE in the downstream node is set, then the arbiter will reset its selection. The same mechanism is also valid for GT flits.

7.3.2 The Difference of the Connectionless and Connection-Oriented Routing Protocols

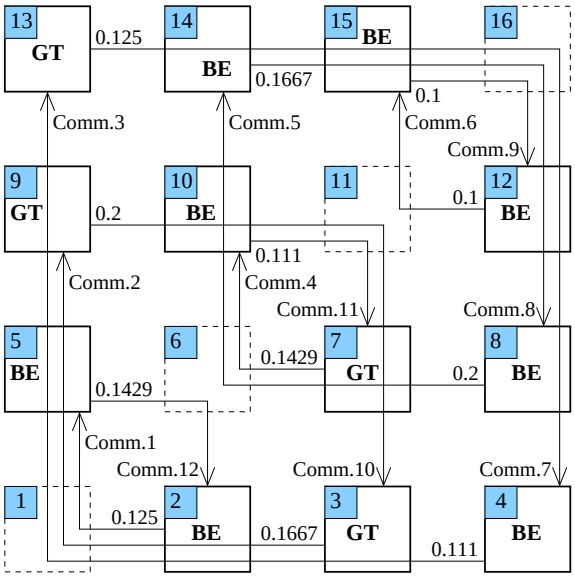
Table 7.1 shows the binary encoding of 8 flit types to differentiate packets used for the *connectionless best-effort (BE)* and the *connection-oriented guaranteed-throughput (GT)* routing protocols services. The flit types encoding of the BE and GT messages are set different to provide different services for the messages in the on-chip network and routing layer protocols.

7.3.3 Experiment with Combined GT-BE Traffics

In this subsection, an experimental simulation is run in which BE and GT messages are mixed in the matrix transpose traffic scenario (node (i, j) send a message to node (j, i)). As shown in Fig. 7.14, there are 12 communication pairs in the transpose traffic pattern,

Tab. 7.1: Flit types encoding for BE and GT packet services.

Hex	Binary	Flit type
0	"000"	not data
1	"001"	header flit for BE packets
2	"010"	databody for BE packets
3	"011"	tail flit for BE packets
4	"100"	header for GT packets
5	"101"	databody for GT packets
6	"110"	tail flit for GT packets
7	"111"	response/status flit

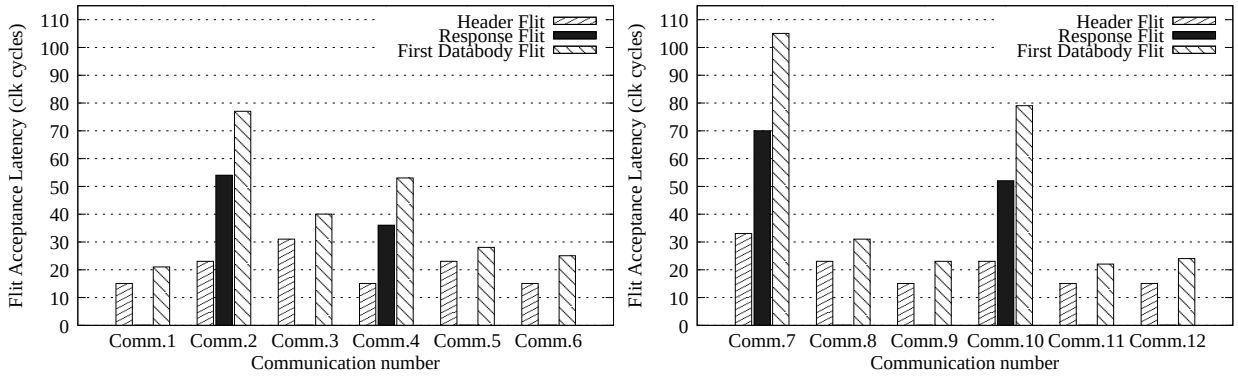


Comm.#	Type	Rate in fpc	Rate in MB/s
Comm.1	BE	0.125	400.00
Comm.2	GT	0.1667	533.33
Comm.3	BE	0.111	355.55
Comm.4	GT	0.1429	457.28
Comm.5	BE	0.2	640.00
Comm.6	BE	0.1	320.00
Comm.7	GT	0.125	400.00
Comm.8	BE	0.1667	533.33
Comm.9	BE	0.1	320.00
Comm.10	GT	0.2	640.00
Comm.11	BE	0.111	355.55
Comm.12	BE	0.1429	457.28

Fig. 7.14: Mixed GT-BE message data transmissions in the transpose traffic scenario.

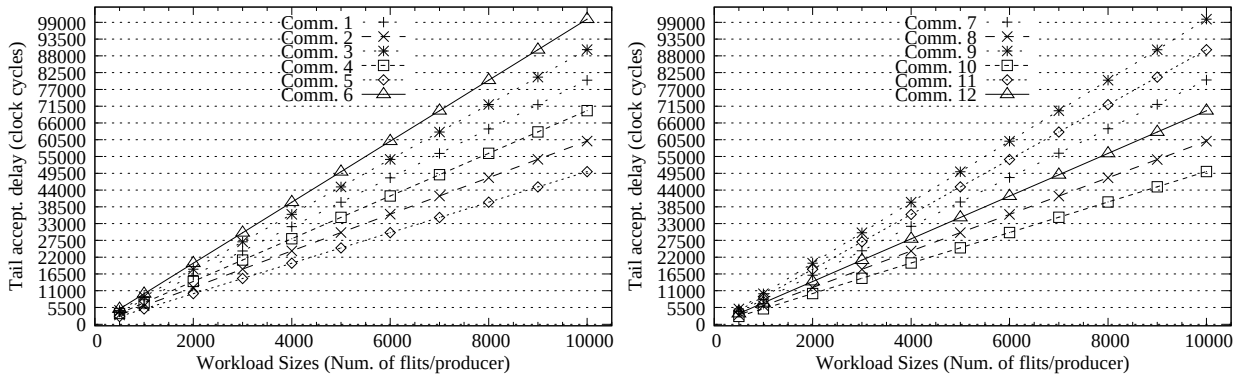
i.e. from $Comm.\ 1$ until $Comm.\ 12$. The $Comm.\ 2$, $Comm.\ 4$, $Comm.\ 7$ and $Comm.\ 10$ are set as GT-type injector-acceptor communication, while the remaining 8 communication pairs are set as BE-type injector-acceptor communication. A node symbolized with **BE** is a node sending a BE message, while a node symbolized with **GT** is a node sending a GT message.

In the simulation, the workload sizes (number of injected message per producer) are set to 500, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000 and 10000 flits. The injection rate per producer (in number of flits per cycle) is set randomly. The decimal values outside the source nodes (beside the paths of each communication pair) presented in the Fig. 7.14 are the expected data communication rates measured in an amount of flits per cycle (fpc). A table in the right-side of the figure presents in detail the 16 communication pairs together with their expected data injection rates in flits per cycle (fpc) and Megabyte per second (MB/s). For example, $Comm.\ 1$ with BE communication pro-



(a) Comm. 1–6 (b) Comm. 7–12

Fig. 7.15: The transfer latency (delay of acceptance) of the header, response and the first databody flits.

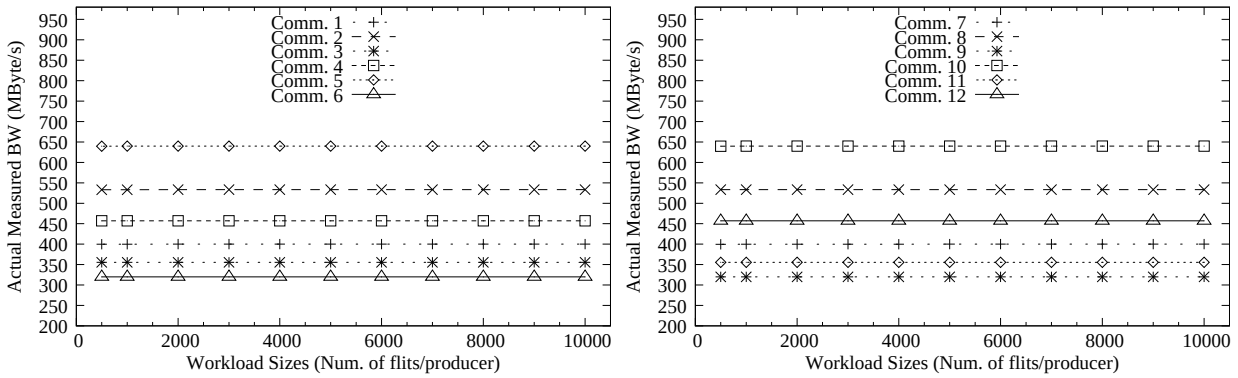


(a) Comm. 1–6 (b) Comm. 7–12

Fig. 7.16: The tail acceptance delays with different workload sizes for each communication pair.

tocol is expected to be injected from source nodes with $0.125\ fpc$, which is equivalent to $0.125 \times 4 \times 800 = 400\ MB/s$. The maximum link capacity of the XHiNoC prototype is $0.5\ fpc$, or with 800 MHz data clock cycle frequency of the XHiNoC router prototype with combined BE-GT services and 32-bit data word, the maximum link capacity is $0.5\ fpc \times 4\ Byte \times 800\ MHz = 1600\ MByte/s$ or $1.6\ GByte/s$. Therefore, the bisection bandwidth is $2 \times 1.6 = 3.2\ GByte/s$ and the router bandwidth capacity is $5 \times 1.6 = 8\ GByte/s$.

Fig. 7.15 presents the measurement of the delay (acceptance latency) of the header, the first databody flit and the response flits in clock cycle period of each communication pair. The response flits will exist only for the GT communication pairs, i.e. $Comm.\ 2$, $Comm.\ 4$, $Comm.\ 7$ and $Comm.\ 10$. Fig. 7.15(a) shows the latency measurement for $Comm.\ 1$ until $Comm.\ 6$, while Fig. 7.15(b) presents the latency measurement for $Comm.\ 7$ until $Comm.\ 12$. The transfer delay of the header flit is measured from its injection node until its destination node. While the transfer delay of the response flit is measured from the destination node until the injection node and is accumulated with the transfer delay of the header flit. The transfer delay of the first databody flit is measured from the injection node until the destination node and is accumulated with the previously measured



(a) Comm. 1–6 (b) Comm. 7–12

Fig. 7.17: The actual communication bandwidth measurement with different workload sizes for each communication pair.

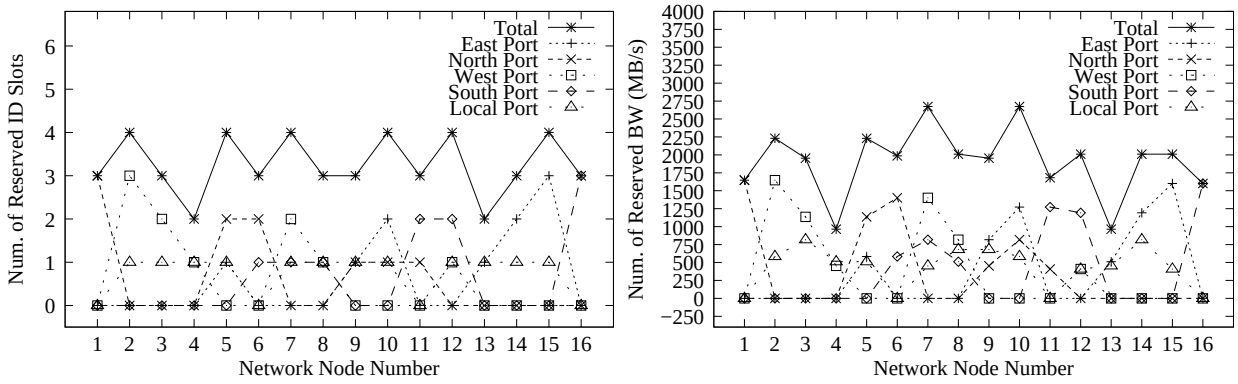
transfer delays of the header and response flits.

The measurement of the tail acceptance delays with different workload sizes for each communication pair is presented in Fig. 7.16. Fig. 7.16(a) shows the tail acceptance latency measurements for $Comm.\ 1$ until $Comm.\ 6$, while Fig. 7.16(b) shows the tail acceptance latency measurements for $Comm.\ 7$ until $Comm.\ 12$. In general, it looks that the tail flit acceptance latencies are increased linearly when the workload (data burst) sizes are incremented.

The measurement of the actual communication bandwidth with different workload sizes for each communication pair is presented in Fig. 7.17. Fig. 7.17(a) shows the actual communication bandwidth measurements for $Comm.\ 1$ until $Comm.\ 6$, while Fig. 7.17(b) shows the actual communication bandwidth measurements for $Comm.\ 7$ until $Comm.\ 12$. In general, it looks that the actual communication bandwidths are constant when the workload (data burst) sizes are incremented. The slopes of the tail flit transfer latencies of each communication pair presented in Fig. 7.16(a) and Fig. 7.16(b) have relationship with the communication bandwidth measurements presented in Fig. 7.17(a) and Fig. 7.17(b). The larger the slopes, the larger the communication bandwidth of the communication pairs.

Fig. 7.18(a) and Fig. 7.18(b) show the distribution of the ID slots and bandwidth spaces reservation at each communication link connected directly to an outgoing port in the NoC. The measurements are made at each outgoing port of every NoC router node (router node 1 until router node 16) in the 2D 4×4 mesh NoC. Fig. 7.19 presents the transient responses of the actual injection and acceptance rates as well as the expected constant data rate for $Comm.\ 1$ until $Comm.\ 6$, while Fig. 7.20 shows the transient responses for $Comm.\ 7$ until $Comm.\ 12$.

The simulations in this subsection have exhibited a very interesting characteristic of the XHiNoC that performs a very flexible runtime communication resource reservation to serve both the BE and GT messages. The BE and GT messages are switched through



(a) ID Slot Reservation (b) Bandwidth Reservation

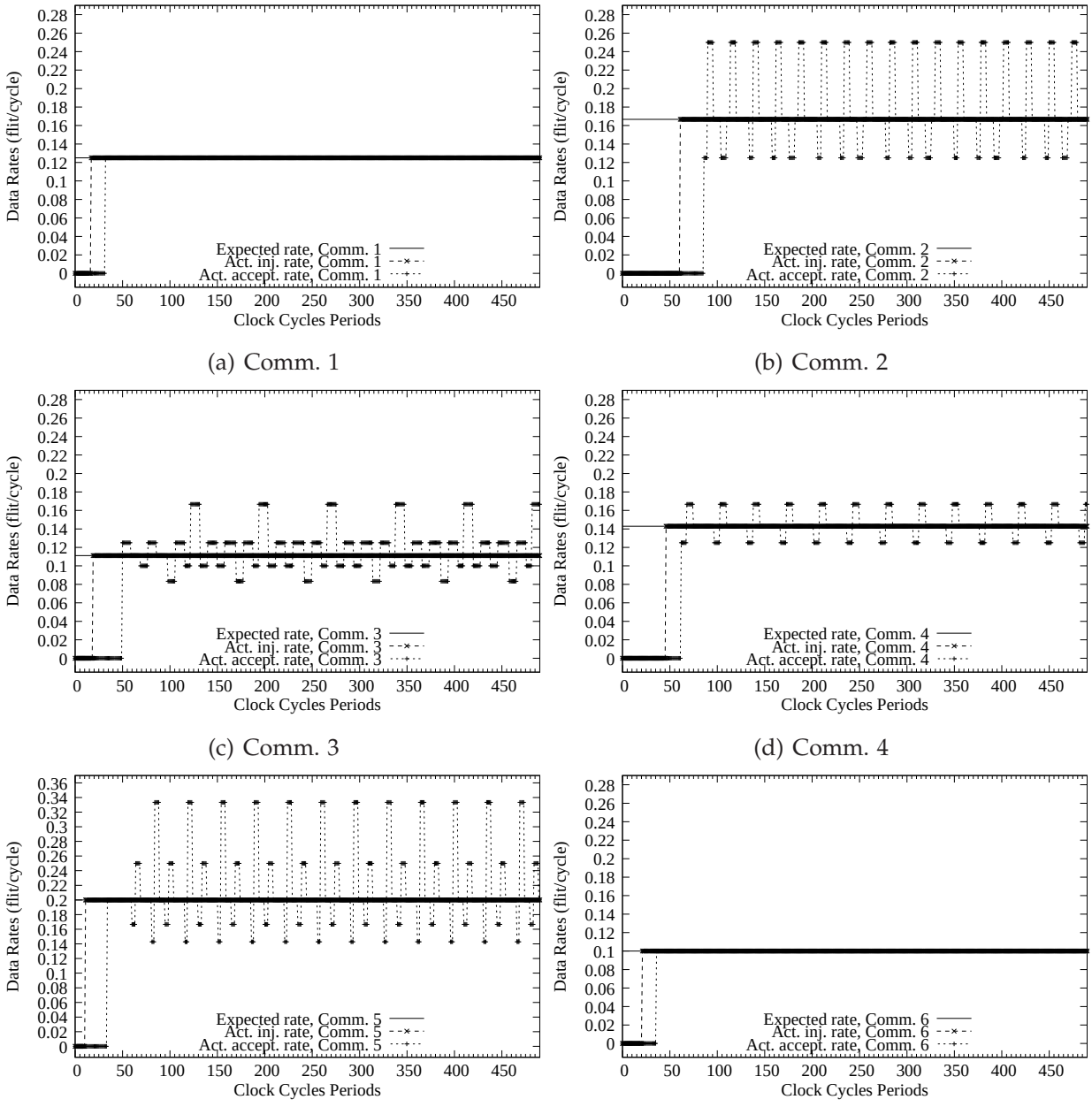
Fig. 7.18: The distribution of the ID slots and bandwidth reservation at each output port of the router nodes.

Tab. 7.2: Synthesis results of the connection-oriented guaranteed-bandwidth (GB) multicast routers using 130-nm CMOS technology library.

	GB-only	BE-GB
Target frequency	1 GHz	800 MHz
Total logic cell area	0.135 mm^2	0.179 mm^2
Est. net switch. power	15.982 mW	18.447 mW
Est. cell intern. power	43.139 mW	44.406 mW
Est. cell leakage power	29.6 μW	37.7 μW

virtual circuit configurations. The expected data communication rates in the simulation are set in such a way that the NoC is not saturated. Therefore, in line with the general performance characteristic of the XHiNoC that has been explained in Chap. 4, the communication latency is increased linearly with the workload size incrementation, and the communication bandwidth can be kept constant even if the workload sizes are incremented.

Due to the non-saturating condition, the expected bandwidth of every communication pair can be fulfilled. The data acceptance in the experimental results are also lossless, i.e. all injected flits in source nodes are accepted in the target nodes. Although some overshots of the actual measured data acceptance rates appear as shown in Fig. 7.19 and Fig. 7.20, the total average communication bandwidth of every end-to-end communication partner is guaranteed equal to the expected constant data rate. We can see that the acceptance rate of every communication partner fluctuates around the expected constant data rate, but the actual measured injection rate is always equal to the expected constant data rate. The overshots are due to contentions between messages to access the same link in the NoC. This situation has also been described visually in Fig. 7.6.



(a) Comm. 1

(b) Comm. 2

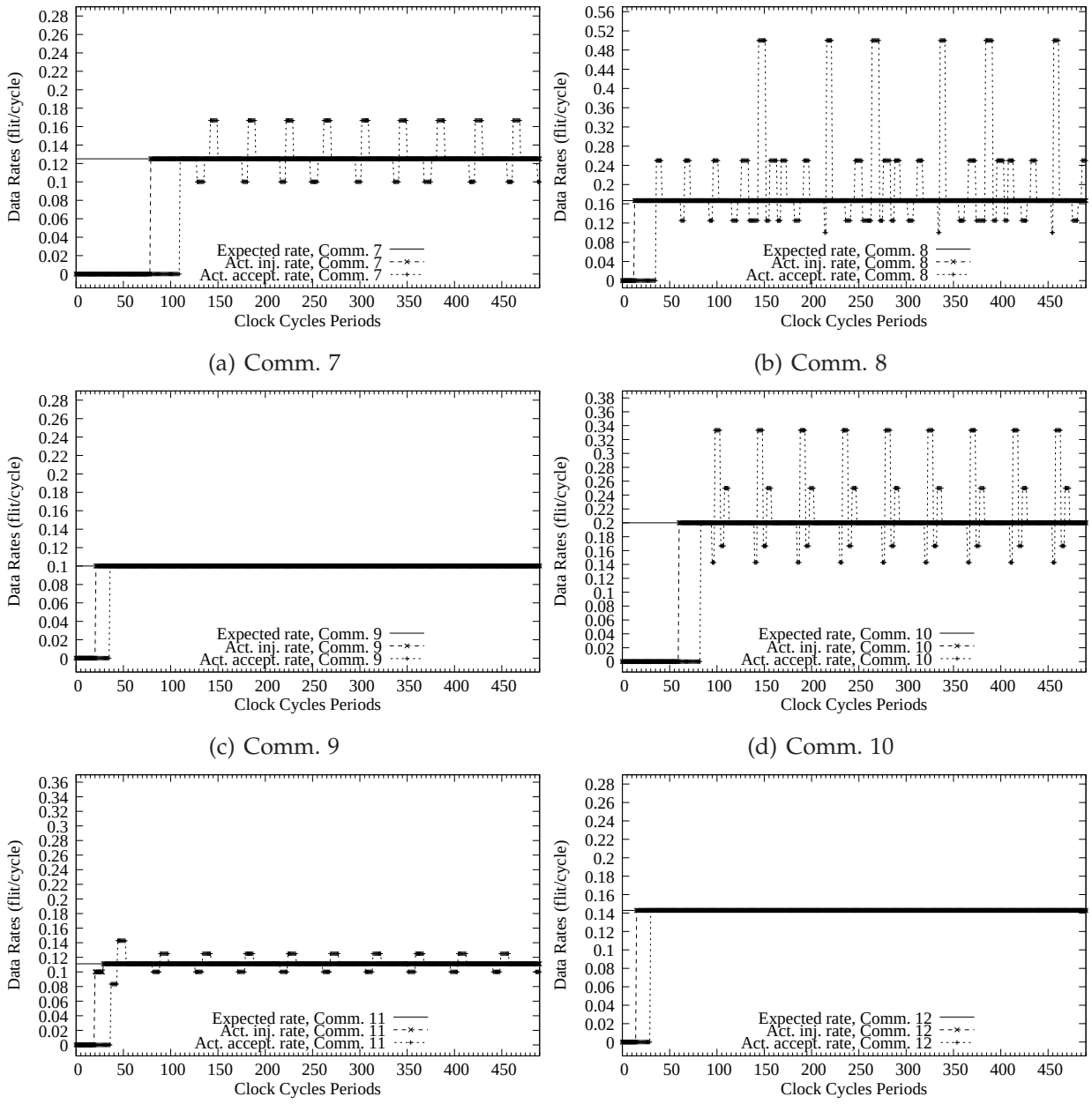
(c) Comm. 3

(d) Comm. 4

(e) Comm. 5

(f) Comm. 6

Fig. 7.19: Transient responses of the measured data injection and data acceptance rates for communication 1–6.



(a) Comm. 7

(b) Comm. 8

(c) Comm. 9

(d) Comm. 10

(e) Comm. 11

(f) Comm. 12

Fig. 7.20: Transient responses of the measured data injection and data acceptance rates for communication 7–12.

7.4 Synthesis Results

The synthesis results of the NoC prototypes with multicast guaranteed-bandwidth (GB) service and the NoC with combined multicast best-effort (BE) and guaranteed-bandwidth (GB) are presented in Table 7.2. The synthesis is made using 130-nm CMOS standard-cell technology library from *Faraday Technology*. We used *Design Vision* tool from *Synopsys* to synthesize the NoC routers. The NoC with GB-only service can be synthesized with target frequency of 1 GHz. While the NoC with combined BE-GB service can be clocked with 800 MHz. The reduction of the allowable data frequency of the BE-GB NoC is due an additional latency to the critical path of the synthesized NoC circuit.

As presented in Table 7.2, the area overhead if the multicast BE-GB NoC over the GB-only NoC is about 33 %. This area overhead is certainly due to the implementation of the double FIFO buffers at each input port of the BE-GB microarchitecture. The complexity of the arbitration control procedure to differentiate the services for both BE-type and GB-type messages has contributed to not only the area overhead, but also the longer critical path of the BE-GB NoC compared to the GB-only NoC.

Comparisons to other NoCs are given as follows. The maximum frequency to transfer data in the Æthereal NoC [187], which combines the BE and GT services with 32-bit word size, is $500\ MHz$ using 130-nm standard-cell technology, resulting in an aggregate bandwidth of $5 \times 500\ MHz \times 32bits = 80\ Gbit/s$. The aggregate bandwidth of the XHiNoC router (static routing, 2-depth FIFO, 32-bit word size) is $5 \times 800\ MHz \times 32bits \times \frac{1}{2} = 64\ Gbit/s$. The total logic area of the Æthereal NoC is $0.2600\ mm^2$. In general the maximum data frequency of the XHiNoC is better, but its aggregate bandwidth is divided by two because of two cycle delay between every flit during data link traversal pipeline stage. However, the logic area overhead of the XHiNoC is lower than the Æthereal NoC.

NOSTRUM NoC [157] has reported that its router consumes 13896 equivalent NAND gates (independent from the standard-cell technology). Without reporting the logic area, SoCBUS NoC [211] can be clocked at 1.2 GHz in a 180-nm technology process. The DSPIN NoC router [181] with 90-nm technology has a gate area of about $0.082mm^2$ after gate-level synthesis (4-depth GS (guaranteed-service) queue, 8-depth BE (best-effort) queue, 34-bit flit size). On a 500 MHz implementation, each GS channel in DSPIN has a bandwidth of $8\ Gbit/s$ ($40\ Gbit/s$ for 5 GS channels). By using a 180-nm standard-cell library, the number of equivalent gates of the CDMA NoC presented in [209] is 272806 gates with 32-bit word size. The postlayout area of the SDMA NoC presented in [131] by using 130-nm technology is $0.023737\ mm^2$ with critical path of $0.44\ ns$.

7.5 Summary

The connection-oriented multicast NoC routers with guaranteed-bandwidth service has been presented in this chapter. This chapter has also presented the efficiency of our pro-

posed concept and microarchitecture to combine both the best-effort and the guaranteed-throughput (GT) or guaranteed-bandwidth (GB) service. The interesting feature of the service-combination in the XHiNoC is that the flits of the BE-type and GB-type messages can be mixed and interleaved in the same NoC communication channel. The size or the depth of the FIFO buffer for BE and GB messages is the same, i.e. 2 register spaces. A soft guarantee is given to the GB-type messages in the virtual buffers placed at the input port. When GT-buffer and BE-buffer are occupied by the GT-type flits or BE-type flits at the same time, respectively, then the routing engine will route firstly the data flit in the GT-buffer until the GT-buffer is empty.

The main difference between the BE and GB communication is the data transport protocol phases used to transfer the messages into the NoC. The GB communication uses a three-phase communication protocol. Before sending the GB-type messages into the NoC, a header flit for a unicast message (multiple header flits for a multicast message) is injected to the NoC to reserve bandwidth (BW) and local ID slots on every intermediate communication channel. When the header flit has attained the destination node, then the destination node sends a response flit back to the source node to inform the status of the connection attempt. The source node will then analyze the response flit, whether the connection is successful, i.e. the requested BW is meet, and ID slot reservation per required links is available. If both requirements are meet, then the source starts injecting the message data. In the BE service, the message data flit (payloads) are sent to the NoC soon after the header flit is injected. Thus, there is no need to wait for a response from destination node.

The issue about BW share between the GB and BE messages is currently a very interesting topic in the NoC research area. Another version of the BE-GT NoC can be still extended from our current BE-GB NoC implementation to optimize the NoC area and the complexity of the NoC. In this way, the BW requirement of each message is implicitly represented in the ID slot reservation, and BW constraint for each message has been determined in the software application level. Thus, BW accumulator unit can be removed from the microarchitecture. If we set that the maximum BW of each communication channel is B_{max}, and the number of available local ID slot on each channel is H, then each message, which will reserve one ID slot on every intermediate communication channel, can be injected with a constrained BW B_{inj} where $B_{inj} \leq \frac{B_{max}}{H}$. Therefore, we will be able to guarantee that each communication link will not be overloaded, i.e. $B_{max} \leq \sum_{h=1}^{N_{msg}} B_{inj}(h)$, where N_{msg} is the number of messages sharing the communication link, and h is an individual message, such that $B_{inj}(h)$ is the BW requirement of the individual message.

Chapter 8

Concluding Remarks

Contents

A concept to design and to implement the VLSI microarchitecture of the XHiNoC routers together with advantageous services enabled due to the proposed concept has been presented so far in the previous chapters. The proposed concept allows a flexible support for the communication media share methodology by using the variable (dynamic) local ID management. This chapter will summarize the contributions of the work (Section 8.1) and some outlooks related to potential future research investigations in the area of NoC research field (Section 8.2).

8.1 Contributions of the Work

The new contributions of this thesis are summarized in the following points.

1. *Novel wormhole switching method* [223] [229], [237], where different wormhole messages can be interleaved among each other at flit-level in the same bufffer pool without using virtual channels. The main problem in the traditional wormhole switching method is the head-of-line blocking problem. The problem can be solved by using virtual channels. However, the solution based on the use of virtual channels will increase the logic gate consumption because of the additional virtual buffers and arbitration unit and control logics for the virtual buffers. Moreover, due to the additional arbitration and control logics, the critical paths of the NoC router with virtual channel will be increased. The critical paths can be reduced by introducing a new pipeline stage, but this solution will add up the NoC router latency because of the additional pipeline stage. This thesis proposed a new wormhole switching method

without using virtual channels. The proposed wormhole switching method solve the head-of-line blocking problem by allowing flits of different messages to be interleaved in the same communication. The flit-level message interleaving technique is enabled due to the concept of the variable (dynamic) local ID tag management presented in this thesis.

2. *Novel theory and VLSI architecture for deadlock-free multicast routing* [234], [224], [227], [232] suitable for NoCs. The main problem to implement a multicast routing service in network and routing protocol layer is the multicast dependency leading to deadlock configuration due to contention between multicast messages to acquire the same outgoing ports in the router. Some theories and methodology to design a deadlock-free multicast routing have been introduced in the literatures. Some of them solve the problem by using virtual channels that has drawbacks as mentioned in the previous point. Some of them uses a centralized routing (source routing) approach that run a preprocessing algorithm to compute a deadlock-free paths at source node before sending the multicast message into the network. This approach will introduce time-overhead due to the message preparation algorithm. This thesis proposed a distributed multicast routing approach without the preprocessing algorithm. The multicast contentions is solved by using a simple and smart technique called *hold-release multicast tagging mechanism*. This technique is enabled due to the concept of the variable (dynamic) local ID tag management presented in this thesis.

3. *Novel strategy to design runtime adaptive routing selection* [236] based on bandwidth space occupancy between alternative outgoing ports (*bandwidth-aware adaptive routing selection*) or based on its combination with the contention-information (*contention- and bandwidth-aware adaptive routing selection*). Each multiplexed message on an outgoing link will reserve one ID slot, thus every outgoing port can provide information to the routing engine about the number of messages in-flight on the link (contention information) by checking the used or freely available ID slots on the link. Information about the available bandwidths can also be provided by the outgoing port. When the header of each message brings information about the data communication bandwidth requirement and save the information into the outgoing port registers, then the amount of the consumed bandwidth space on the link can be measured. The routing engine at input port can then use both information, i.e. the number of free available bandwidth and the number of free ID slots of alternative outgoing ports, to make routing decision (output selection) at runtime. Some works in the literature use FIFO buffer occupancy (congestion information) to make routing decision at runtime, which has a few drawbacks. The performance of the contention- and bandwidth-oriented adaptive routing is better than the queue-length-based adaptive routing selection strategy.

4. *Novel efficient approach enable to combine connectionless best-effort and connection-oriented guaranteed-throughput services* [221]. By using the concept of the variable local ID-

division multiple access technique, service for the best-effort (BE) and guaranteed-throughput (GT) messages can be efficiently implemented on single NoC router with optimal communication resource utilization. In the case that network is not saturated, the BE and GT messages can share fairly the used communication channels in the NoC. A soft guaranteed-service is given to the GT-type messages. Hence, the GT-type messages will not interrupt the flow of the BE-type messages.

8.2 Directions for Future Works

On-chip networks are communication infrastructure dedicated for on-chip multiprocessor systems. The performance of an on-chip interconnection networks affects the performance of the networked multiprocessor systems because communication overheads due to task-level parallelism affecting the total computational time of a complex parallel computing. Therefore, a high performance network-on-chip should be a general requirement to develop a networked multiprocessor system in the future.

In accordance with the current status of the research results presented in this thesis, further investigations to improve the flexibility, to increase performance, to reduce power dissipation and to optimize the logic area of the proposed NoC router microarchitecture will be still open. Some potential improvements that can be made for the future investigations are described in the following points.

1. In the current XHiNoC implementation with combined BE and GT services, a virtual channel with two FIFO buffers per input port has been implemented. Further investigation can be made by replacing the virtual channel with a smart FIFO buffer having the same functionality with the virtual channel. This component replacement will help to reduce the area overhead of the NoC router due to the double-FIFO buffer implementation at the NoC input ports. The FIFO buffer will be equipped with smart algorithm to organized the contents of its registers. When the queue registers contain the GT-type and BE-type messages, the smart FIFO buffers will always firstly route the GT-type messages in order until there is no more GT-type message. In order to avoid a blocking situation for the GT-type messages, a BE-type flit will not be let to enter the FIFO buffer if the buffer is almost full (There is only one free space in the FIFO buffer).

2. The contention- and bandwidth-aware adaptive routing selection strategy implemented in the current XHiNoC microarchitecture has not covered the problem of NoC faults (Fault-Tolerance Routing algorithm). In the future, it is also interesting if the routing adaptivity is provided not only to avoid congestion situation, but also to handle a situation, where some post-chip-fabricated faults are found in some NoC router nodes and/or in NoC communication links.

3. Another challenging issue in the NoC area is the stacked 3D integration of NoC-based multiprocessor systems. The 3D integration can potentially increase the communication scalability and reduce network hops in massively parallel multiprocessor systems. Memory components can be potentially located near to the parallel processor cores on one or more stack layers. However, since the technology is not yet mature, there are still many research efforts to undertake, including supports from standard CAD tools.

Beside the aforementioned potential improvement in accordance with the current VLSI microarchitecture and the NoC services implementation results, there are still also some interesting topics to investigate, which are beyond the scope of this current work. The research about networks-on-chip topics has a strong relationship with researches in fields of multiprocessor systems and parallel computing systems. The directions of the future works related to the development of the XHiNoC router prototypes in line with the NoC-based multiprocessor systems realization are described in the following.

1. Network Interface (NI) is an important component to couple a NoC with a processing element (PE) core. This component is used as adapter to enable data transport and communication between the NoC routers and the PE cores. Therefore, the NI should be compatible with the NoC router and the PE core by separating the NI into two parts, i.e. a PE-independent module and a NoC-independent module. Since the architecture, packet format and IO signaling of a certain NoC router is specific compared with other NoCs, then the challenge to design a flexible NI with less design-effort is an interesting topic in the future.

2. A multicore chip integrates the switch cores to build a NoC communication infrastructure, NI cores together with other IP components or CPU/DSP/GPU (Graphics Processing unit) cores. The multicore chip fabrication will certainly be the most interesting work. After the chip manufacturing process, a cost-effective testing methodology for NoC-based multiprocessor systems will then be the next challenging research topic. Significant problems that could be challenging issue in the submicron interconnection circuits are e.g. crosstalk faults and single-event upsets (SEUs). The time and energy efficient test strategy to cover fabrication errors in the very complex NoC-based multiprocessor systems will be an interesting research field in the future.

3. Finally, efficient parallel programming models suitable for embedded MPSoCs and for chip-level multiprocessor (CMP) systems will be a key factor to the successfull of the NoC-based multiprocessor system release in markets. An easy-to-program NoC-based multicore system will be a very interesting feature for end-users which may have a limited knowledge about parallel programming method. For many years, there have been some parallel programming models used to develop parallel computers in high performance computing (HPC) area. The system platforms

can be called “on-rack” (off-chip) multiprocessor systems. Nevertheless, only a few people are familiar with the programming models, except for programmers who work in parallel computing societies. Ideally we really need to develop a parallel program compiler, which automatically parallelizes the implicitly-described parallel tasks from a sequential program to create multiple concurrent executable codes. It is really very hard to develop, but it is not impossible that one day such compiler can be found in the future.

References

[1] P. ABAD, V. PUENTE, and J.-A. GREGORIO. "MRR: Enabling fully adaptive multicast routing for CMP interconnection networks". In *Proc. the 15th IEEE Int'l Symp. on High Performance Computer Architecture (HPCA 2009)*, pages 355–366, Feb. 2009.

[2] A. AGARWAL, R. BIANCHINI, D. CHAIKEN, F. T. CHONG, K. L. JOHNSON, D. KRANZ, J. D. KUBIATOWICZ, B.-H. LIM, K. MACKENZIE, and D. YEUNG. "The MIT Alewife Machine". *Proc. of the IEEE, Special Issue on Distributed Shared Memory*, 87(3):430–444, March 1999.

[3] T. W. AINSWORTH and T. M. PINKSTON. "Characterizing The Cell EIB On-Chip Network". *IEEE Micro*, 27(5):6–14, Sep./Oct. 2007.

[4] M. F. AKAY and C. KATSINIS. "Contention Resolution on a Broadcast-based Distributed Shared Memory Multiprocessor". *IET Computers & Digital Techniques*, 2(1):45–55, Jan. 2008.

[5] C. AKTOUF. "A Complete Strategy for Testing an On-Chip Multiprocessor Architecture". *IEEE Design & Test of Computers*, 19:18–28, Jan–Feb. 2002.

[6] M. A. AL FARUQUE, T. EBI, and J. HENKEL. "Run-time adaptive on-chip communication scheme". In *Proc. the 2007 IEEE/ACM International Conf. on Computer-Aided Design (ICCAD'07)*, pages 26–31, Piscataway, NJ, USA, 2007. IEEE Press.

[7] J. D. ALLEN, P. T. GAUGHAN, D. E. SCHIMMEL, and S. YALAMANCHILI. "Ariadne – An Adaptive Router for Fault-Tolerant Multicomputers". *ACM SIGARCH Computer Architecture News*, 22(2):278–288, April 1994.

[8] M. AMDE, T. FELICIJAN, A. EFTHYMIOU, D. EDWARDS, and L. LAVAGNO. "Asynchronous on-chip networks". *IEE Proc. Computers and Digital Techniques*, 152(2):273–283, March 2005.

[9] A. M. AMORY, K. GOOSSENS, E. J. MARINISSEN, M. LUBASZEWSKI, and F. MORAES. "Wrapper design for the reuse of a bus, network-on-chip, or other functional interconnect as test access mechanism". *IET Computers & Digital Techniques*, 1(3):197–206, 2007.

[10] J. ANDREWS and N. BAKER. "Xbox 360 System Architecture". *IEEE MICRO*, 26(2):35–37, March/April 2006.

[11] F. ANGIOLINI, P. MELONI, S. M. CARTA, L. RAFFO, and L. BENINI. "A Layout-Aware Analysis of Networks-on-Chip and Traditional Interconnects for MPSoC". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 26(3):421–434, March 2007.

[12] K. AOYAMA and A. A. CHIEN. "The Cost of Adaptivity and Virtual Lanes in a Wormhole Router". *VLSI Design, Journal of Hindawi Pub.*, 2(4):315–333, 1995.

[13] ARVIND and R. S. NIKHIL. "Executing a Program on the MIT Tagged Token Dataflow Architecture". In *Lecture Notes in Computer Science, Springer-Verlag*, volume 259, pages 1–29, Berlin/Heidelberg, 1987.

[14] G. ASCIA, V. CATANIA, M. PALESI, and D. PATTI. "Implementation and Analysis of a New Selection Strategy for Adaptive Routing in Networks-on-Chip". *IEEE Trans. Computers*, 57(6):809–820, June 2008.

[15] E. AYGUADÉ, N. COPTY, A. DURAN, J. HOEFLINGER, Y. LIN, F. MASSAIOLI, X. TERUEL, P. UNNIKRISHNAN, and G. ZHANG. "The Design of OpenMP Tasks". *IEEE Transactions on Parallel and Distributed Systems*, 20(3):404–418, March 2009.

[16] J. BAINBRIDGE and S. FURBER. "Chain: A Delay-Insensitive Chip Area Interconnect". *IEEE Micro*, 22(5):16–23, Sep./Oct. 2002.

[17] A. O. BALKAN, Q. GANG, and U. VISHKIN. "Mesh-of-Trees and Alternative Interconnection Networks for Single-Chip Parallelism". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(10):1419–1432, Oct. 2009.

[18] A. BANERJEE, P. T. WOLKOTTE, R. D. MULLINS, S. W. MOORE, and G. J. M. SMIT. "An Energy and Performance Evaluation of Network-on-Chip Architectures". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(3):319–329, March 2009.

[19] M. BARNETT, D. G. PAYNE, R. A. VAN DE GEIJN, and J. WATTS. "Broadcasting on Meshes with Worm-Hole Routing". *Journal of Parallel and Distributed Computing*, 35(2):111–122, June 1996.

[20] B. BARNEY. "Introduction to Parallel Computing". Lawrence Livermore National Laboratory, http://computing.llnl.gov/tutorials/parallel_ comp/, available online 2009/2010.

[21] B. BARNEY. "Message Passing Interface (MPI)". Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/mpi/, available online 2009/2010.

[22] B. BARNEY. "OpenMP (Open Multi-Processing)". Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/openMP/, available online 2009/2010.

[23] B. BARNEY. "POSIX Threads Programming". Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/pthreads/, available online 2009/2010.

[24] T. A. BARTIC, D. DESMET, J.-Y. MIGNOLET, J. MILLER, and F. ROBERT. "Mapping concurrent applications on network-on-chip platforms". In *in IEEE Workshop on Signal Processing Systems Design and Implementation*, pages 154–159, Nov. 2005.

[25] T. A. BARTIC, J. Y. MIGNOLET, V. NOLLET, T. MARESCAUX, D. VERKEST, S. VERNALDE, and R. LAUWEREINS. "Topology adaptive network-on-chip design and implementation". *IEE Proc. Computers and Digital Techniques*, 152(4):467–472, July 2005.

[26] J. BEECROFT, M. HOMEWOOD, and M. MCLAREN. "Meiko CS-2 Interconnect Elan-Elite Design". *Parallel Computing*, 20(10–11):1627–1638, Nov. 1994.

[27] E. BEIGNÉ, F. CLERMIDY, S. MIERMONT, Y. THONNART, X.-T. TRAN, A. VALENTIAN, D. VARREAU, P. VIVET, X. POPON, and H. LEBRETON. "An Asynchronous Power Aware and Adaptive NoC Based Circuit". *IEEE Journal of Solid-State Circuits*, 44(4):1167–1177, April 2009.

[28] E. BEIGNÉ, F. CLERMIDY, P. VIVET, A. CLOUARD, and M. RENAUDIN. "An Asynchronous NOC Architecture Providing Low Latency Service and its Muti-level Design Framework". In *Proc. the 11th IEEE Int'l Symp. on Asynchronous Circuits and Systems*, pages 54–63, 2005.

[29] T. BENGTSSON, S. KUMAR, R.-J. UBAR, A. JUTMAN, and Z. PENG. "Test methods for crosstalk-induced delay and glitch faults in network-on-chip interconnects implementing asynchronous communication protocol". *IET Computers & Digital Techniques*, 2(6):445–460, 2007.

[30] L. BENINI and D. BERTOZZI. "Network-on-chip architectures and design methods". *IEE Proc. Computers and Digital Techniques*, 152(2):261–272, March 2005.

[31] L. BENINI and G. D. MICHELI. "Networks on Chips: A new SoC Paradigm". *Computer*, 35(1):70–78, Jan 2002.

[32] 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 Trans. Parallel and Distributed Systems*, 16(2):113–129, Feb. 2005.

[33] P. BHOJWANI and R. N. MAHAPATRA. "Robust Concurrent Online Testing of Network-on-Chip-Based SoCs". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 16(9):1199–1209, Sep. 2008.

[34] L. N. BHUYAN, R. R. IYER, T. ASKAR, A. K. NANDA, and M. KUMAR. "Performance of Multistage Bus Networks for a Distributed Shared Memory Multiprocessor". *IEEE Transactions on Parallel and Distributed Systems*, 8(1):82–95, Jan 1997.

[35] T. BJERREGAARD and S. MAHADEVAN. "A Survey of Research and Practices of Network-on-Chip". *ACM Computing Surveys*, 38(1):1, 2006.

[36] T. BJERREGAARD and J. SPARSØ. "Implementation of guaranteed services in the MANGO clockless network-on-chip". *IEE Proc. Computers and Digital Techniques*, 153(4):217–229, July 2006.

[37] F. BODIN, D. WINDHEISER, W. JALBY, D. ATAPATTU, M. LEE, and D. GANNON. "Performance Evaluation and Prediction for Parallel Algorithms on the BBN GP1000". *ACM SIGARCH Computer Architecture News*, 18(3b):401–413, Sep. 1990.

[38] P. BOGDAN and R. MARCULESCU. "Quantum-Like Effects in Network-on-Chip Buffers Behavior". In *Proc. ACM/IEEE Design Automation Conference (DAC'07)*, pages 266–267, 2007.

[39] R. V. BOPPANA, S. CHALASANI, and C. S. RAGHAVENDRA. "Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms". *IEEE Trans. Parallel and Distributed Systems*, 9(6):535–549, June 1998.

[40] M. J. BRIDGES, N. VACHHARAJANI, Y. ZHANG, T. JABLIN, and D. I. AUGUST. "Revisiting The Sequential Programming Model for Multicore Era". *IEEE Micro*, 25(2):12–20, Jan./Feb. 2008.

[41] M. BUTTS. "Synchronization through Communication in a Massively Parallel Processor Array". *IEEE Micro*, 27(5):32–40, Sep./Oct. 2007.

[42] G. T. BYRD and M. J. FLYNN. "Producer-Consumer Communication in Distributed Shared Memory Multiprocessor". *Proc. of The IEEE*, 87(3):456–466, March 1999.

[43] K.-C. CHANG and T.-F. CHEN. "Low-Power Algorithm for Application-Specific Networks on Chip". *IET Computers & Digital Techniques*, 2(3):239–249, 2008.

[44] B. CHAPMAN and L. HUANG. *Enhancing OpenMP and Its Implementation for Programming Multicore Systems*. C. Bischof and M. Bucker and P. Gibbon and G.R. Joubert and T. Lippert and B. Mohr and F. Peters (Eds.), NIC Series, vol. 38. John von Neumann Institute for Computing, Julich, 2007. Reprinted in: Advances in Parallel Computing, Volume 15, ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

[45] K. S. CHATHA, K. SRINIVASAN, and G. KONJEVOD. "Automated Techniques for Synthesis of Application-Specific Network-on-Chip Architectures". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1425–1438, Aug. 2008.

[46] A. A. CHIEN and J. H. KIM. "Planar Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors". In *Proc. The 19th Int'l Symposium on Computer Architecture*, pages 268–277, May 1992.

[47] A. A. CHIEN and J. H. KIM. "Planar Adaptive Routing: Low Cost Adaptive Networks for Multiprocesors". *Journal of the Association for Computing Machinery*, 42(1):91–123, Jan. 1995.

[48] G.-M. CHIU. "The Odd-Even Turn Model for Adaptive Routing". *IEEE Trans. Parallel and Distributed System*, 11(7):729–738, July 2000.

[49] C.-L. CHOU, U. Y. OGRAS, and R. MARCULESCU. "Energy- and Performance-Aware Incremental Mapping for Networks on Chip with Multiple Voltage Levels". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879, Oct. 2008.

[50] CISCO. *"Internetworking Technology Handbook"*. Chap. 1: Internetworking Basics. Cisco Systems, Inc. *http://www.cisco.com/en/US/docs/internetworking/technology/handbook/Intro-to-Internet.html*, 2009.

[51] CISCO. *"Internetworking Technology Handbook"*. Chap. 49: Quality of Service Networking. Cisco Systems, Inc. *http://www.cisco.com/en/US/docs/internetworking/technology/handbook/QoS.html*, 2009.

[52] M. COENEN, S. MURALI, A. RADULESCU, K. GOOSSENS, and G. D. MICHELI. "A Buffer-Sizing Algorithm for Networks on Chip using TDMA and Credit-based End-to-End Flow Control". In *Proc. the 4th Hardware/Software Codesign and System Synthesis (CODES+ISSS'06)*, pages 130–135, 2006.

[53] M. COLLETTE, B. COREY, and J. JOHNSON. *"High Performance Tools & Technologies"*. available online: https://computing.llnl.gov. Computing Applications and Research Department, Lawrence Livermore National Laboratory, Dec. 2004.

[54] M. COPPOLA, M. D. GRAMMATIKAKIS, R. LOCATELLI, G. MARUCCIA, and L. PIERALISI. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, Taylor & Francis Group, 2009.

[55] E. COTA, F. L. KASTENSMIDT, M. CASSEL, M. HERVÉ, P. ALMEIDA, P. MEIRELLES, A. AMORY, and M. LUBASZEWSKI. "Testing Network-on-Chip Communication Fabrics". *IEEE Trans. Computers*, 57(9):1202–1215, Sep. 2008.

[56] W. J. DALLY. "Performance Analysis of k-ary n-cube Interconnection Networks". *IEEE Trans. Computers*, C-39(6):775–785, June 1990.

[57] W. J. DALLY and C. L. SEITZ. "The Torus Routing Chip". *Journal of Distributed Computing*, 1(3):187–196, Oct. 1986.

[58] W. J. DALLY and C. L. SEITZ. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks". *IEEE Trans. Computers*, C-36(5):547–553, May 1987.

[59] W. J. DALLY and B. TOWLES. "Route Packets, Not Wires: On-Chip Interconnection Networks". In *The 38th ACM Design Automation Conf.*, pages 684–689, 2001.

[60] B. V. DAO, J. DUATO, and S. YALAMANCHILI. "Configurable Flow Control Mechanisms for Fault-Tolerant Routing". In *Proc. the 22nd International Symposium on Computer Architecture (ISCA'95)*, pages 220–229, June 1995.

[61] R. F. DEMARA and D. . MOLDOVAN. "Performance Indices for Parallel Marker-Propagation". In *Proc. Int'l Conf. Parallel Processing*, volume 1, pages 658–659, Aug. 1991.

[62] J. DUATO. "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks". *IEEE Trans. Parallel and Distributed Systems*, 4(12):1320–1331, Dec. 1993.

[63] J. DUATO. "A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks". *IEEE Trans. Parallel and Distributed Systems*, 6(9):976–987, Sep. 1995.

[64] J. DUATO, B. V. DAO, P. T. GAUGHAN, and S. YALAMANCHILI. "Scouting: Fully Adaptive Deadlock-free Routing in Faulty Pipelined Networks". In *Proc. the International Conference on Parallel and Distributed Systems*, pages 608–613, Dec. 1994.

[65] J. DUATO, S. YALAMANCHILI, and L. NI. *Interconnection Networks: An Engineering Approach*. Revised Printing. Murgan Kaufmann, 2003.

[66] T. H. DUNIGAN, J. S. VETTER, J. B. W. III, and P. H. WORLEY. "Performance Evaluation of The Cray X1 Distributed Shared-Memory Architecture". *IEEE Micro*, 25(1):30–40, Jan-Feb. 2005.

[67] J. S. K. (ED.). *Parallel MIMD Computation: HEP Supercomputer and Its Applications*. MIT Press, Cambridge, MA, 1985.

[68] A. EJLALI, B. M. AL-HASHIMI, P. ROSINGER, S. G. MIREMADI, and L. BENINI. "Performability/Energy Tradeoff in Error-Control Schemes for On-Chip Networks". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 18(1):1–14, Jan. 2010.

[69] ÉRIKA COTA and C. LIU. "Constraint-Driven Test Scheduling for NoC-Based Systems". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 25(11):2465–2478, Nov. 2006.

[70] S. EVAIN, J.-P. DIGUET, and D. HOUZET. "NoC Design Flow for TDMA and QoS Management in a GALS Context". *EURASIP Journal on Embedded Systems*, vol. 2006:1–12, 2006.

[71] S. FLOYD, V. JACOBSON, S. MCCANNE, C. LIU, and L. ZHANG. "A Reliable Multicast Framework for Light-Weight Sessions and Application-Level Framing". *IEEE/ACM Trans. Networking*, 5(6):784–803, Dec. 1997.

[72] M. FORSELL. "A Scalable High-Performance Computing Solution for Networks on Chip". *IEEE Micro*, 22(5):46–55, Sep./Oct. 2002.

[73] G. FOX, S. HIRANANDANI, K. KENNEDY, C. KOELBEL, U. KREMER, and ET. AL. "Fortran D Language Specification", *Technical Report CRPC-TR 90079*. Center for Research on Parallel Computation, Rice Univ., Houston, Texas, Dec. 1990.

[74] V. FRESSE, A. AUBERT, and N. BOCHARD. "A Predictive NoC Architecture for Vision Systems Dedicated to Image Analysis". *EURASIP Journal on Embedded Systems*, vol. 2007(Article ID 97929):1–13, 2007.

[75] A. GANGULY, P. P. PANDE, and B. BELZER. "Crosstalk-Aware Channel Coding Schemes for Energy Efficient and Reliable NOC Interconnects". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(11):1626–1639, Nov. 2009.

[76] D. GANNON and J. V. ROSENDALE. "On the Impact of Communication Complexity in the Design of Parallel Numerical Algorithms". *IEEE Trans. Computers*, C-33(12):1180–1194, Dec. 1984.

[77] P. T. GAUGHAN and S. YALAMANCHILI. "A Family of Fault-Tolerant Routing Protocols for Direct Multiprocessor Networks". *IEEE Trans. Parallel and Distributed Systems*, 6(5):482–497, May 1995.

[78] D. GEER. "Networks on Processors Improve On-Chip Communications". *Computer*, 42(3):17–20, March 2009.

[79] G. A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, and V. SUNDERAM. *"PVM: Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing"*. HTML version: http://www.csm.ornl.gov/pvm. MIT Press, 1994.

[80] G. A. GEIST, J. A. KOHL, and P. M. PAPADOPOULOS. "PVM and MPI: A Comparison of Features". *Calculateurs Paralleles*, 8(2), May 1996.

[81] N. GENKO, D. ATIENZA, G. D. MICHELI, and L. BENINI. "NoC Emulation: A Tool and Design Flow for MPSoC". *IEEE Circuits and Systems Magazine*, 7(4):42–51, Fourth Quarter 2007.

[82] M. GHONEIMA, Y. ISMAIL, M. KHELLAH, and V. DE. "Variation-Tolerant and Low Power Source-Synchronous Multicycle On-Chip Interconnect Scheme". *VLSI Design, Journal of Hindawi Publishing Corp.*, vol. 2007:1–12, 2007.

[83] C. J. GLASS and L. M. NI. "The Turn Model for Adaptive Routing". In *Proc. The 19th Int'l Symposium on Computer Architecture*, pages 278–287, 1992.

[84] C. J. GLASS and L. M. NI. "The Turn Model for Adaptive Routing". *Journal of the Association for Computing Machinery*, 41(5):874–902, Sep. 1994.

[85] C. GÓMEZ, M. GÓMEZ, P. LÓ, and J. DUATO. "Reducing Packet Dropping in a Bufferless NoC". In *Lecture Notes in Computer Science (LCNS) 5168, Euro-Par*, pages 899–909, Springer, 2008.

[86] T. F. GONZALES. "Multimessage Multicasting: Complexity and Approximation". UCSB, Dept. of Computer Science, *Technical Report TRCS-96-15*, July 1996.

[87] 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, Sep./Oct. 2007.

[88] C. GRECU, A. IVANOV, R. SALEH, and P. P. PANDE. "Testing Network-on-Chip Communication Fabrics". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 26(12):2201–2214, Dec. 2007.

[89] M. GSCHWIND, H. P. HOFSTEE, B. FLACHS, M. HOPKINS, Y. WATANABE, and T. YAMAZAKI. "Synergistic Processing in Cell's Multicore Architecture". *IEEE Micro*, 26(2):10–24, March/April 2006.

[90] P. GUERRIER and A. GREINER. "A Generic Architecture for On-Chip Packet-Switched Interconnection". In *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE'00)*, pages 250–256, 2000.

[91] C. GUNN. "CMOS Photonics for High-Speed Interconnects". *IEEE MICRO*, 26(2):58–66, March/April 2006.

[92] J. GURD, C. C. KIRKHAM, and I. WATSON. "The Manchester Prototype Dataflow Computer". *Communications of the ACM*, 28(1):34–52, Jan. 1985.

[93] A. HANSSON, K. GOOSSENS, and A. RĂDULESCU. "Unified Approach to Mapping and Routing on a Network-on-Chip for both Best-Effort and Guaranteed Service Traffic". *VLSI Design, Journal of Hindawi Publishing Corp.*, vol. 2007:1–16, 2007.

[94] J. R. HERRING, C. B. STUNKEL, and R. SIVARAM. "Multicasting Using A Wormhole Routing Switching Element". US Patent No. 6,542,502 B1, (Assignee: IBM Corp.), April 1, 2003.

[95] HIGH PERFORMANCE FORTRAN FORUM. "High Performance Fortran Language Specification, version 1.0". the material appeared in *Scientific Programming*, vol. 2, no. 1, june 1993. Rice Univ., Houston, Texas, May 1993.

[96] C. HILTON and B. NELSON. "PNOC: a flexible circuit-switched NoC for FPGA-based systems". *IEE Proc. Computers and Digital Techniques*, 153(3):181–188, May 2006.

[97] T. HOLLSTEIN and M. GLESNER. "Advanced hardware/software co-design on reconfigurable network-on-chip based hyperplatforms". *Computers and Electrical Engineering, an International Journal, Elsevier*, 33(4):310–319, July 2007.

[98] Y. HOSKOTE, S. VANGAL, A. SINGH, N. BORKAR, and S. BORKAR. "A 5-GHz Mesh Interconnects for A Teraflops Processor". *IEEE Micro*, 27(5):51–61, Sep./Oct. 2007.

[99] M. HOSSEINABADY, A. DALIRSANI, and Z. NAVABI. "Using the Inter- and Intra-Switch Regularity in NoC Switch Testing". In *Proc. Design Automation and Test in Europe*, pages 1–6, 2007.

[100] J. HU and R. MARCULESCU. "DyAD: Smart Routing for Networks-on-Chip". In *Proc. of the 41st annual Design Automation Conference (DAC'04)*, pages 260–263, New York, NY, USA, 2004. ACM.

[101] J. HU and R. MARCULESCU. "Energy- and Performance-Aware Mapping for Regular NoC Architectures". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 24(4):551–562, April 2005.

[102] J. HU, U. Y. OGRAS, and R. MARCULESCU. "System-Level Buffer Allocation for Application-Specific Network-on-Chip Router Design". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 25(12):2919–2933, Dec. 2006.

[103] D. A. ILITZKY, J. D. HOFFMAN, A. CHUN, and B. P. ESPARZA. "Architecture of the Scallable Communications Core's Network on Chip". *IEEE Micro*, 27(5):62–74, Sep./Oct. 2007.

[104] INTEL CORP. "Paragon XP/S Product Overview". Supercomputer Systems Division, Beaverton, OR, 1991.

[105] ITRS. "International Technology Road Map". http://www.itrs.net, 2009.

[106] A. JANTSCH and H. TENHUNEN. *Networks on Chip*. Kluwer Academic Publishers, 2003.

[107] N. E. JERGER, L.-S. PEH, and M. LIPASTI. "Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support". In *Proc. the 35th International Symp. on Computer Architecture (ISCA'08)*, pages 229–240, June 2008.

[108] C. R. JESSHOPE, P. R. MILLER, and J. T.YANTCHEV. "High Performance Communications in Processor Networks". In *Proc. the 16th International Symposium on Computer Architecture (ISCA'89)*, pages 150–157, May-June 1989.

[109] S. L. JOHNSSON. "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures". *Journal of Parallel and Distributed Computing*, 4:133–172, Apr. 1985.

[110] S. L. JOHNSSON and C.-T. HO. "Optimum Broadcasting and Personalized Communication in Hypercubes". *IEEE Trans. Computers*, 38(9):1249–1268, Sep. 1989.

[111] F. KARIM, A. NGUYEN, and S. DEY. "An Interconnect Architecture for Networking Systems on Chips". *IEEE Micro*, 22(5):36–45, Sep./Oct. 2002.

[112] S. K. KASERA, G. HJÁLMTÝSSON, D. F. TOWSLEY, and J. F. KUROSE. "Scalable Reliable Multicast Using Multiple Multicast Channels". *IEEE/ACM Trans. Networking*, 8(3):294–310, June 2000.

[113] P. KERMANI and L. KLEINROCK. "Virtual cut-through: A New Computer Communication Switching Technique". *Computer Networks*, 3:267–286, 1979.

[114] D. KIM, K. KIM, J.-Y. KIM, S. LEE, S.-J. LEE, and H.-J. YOO. "81.6 GOPS Object Recognition Processor Based on a Memory-Centric NoC". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(3):370–383, March 2009.

[115] K. KIM, S. LEE, J.-Y. KIM, M. KIM, and H.-J. YOO. "A 125 GOPS 583 mW Network-on-Chip Based Parallel Processor with Bio-Inspired Visual Attention Engine". *IEEE Journal of Solid-State Circuits*, 44(1):136–147, Jan. 2009.

[116] M. KISTLER, M. PERRONE, and F. PETRINI. "Cell Multiprocessor Communication Network: Built for Speed". *IEEE Micro*, 26(3):10–23, May/June 2006.

[117] A. K. KODI, A. SARATHY, and A. LOURI. "Adaptive Channel Buffers in On-Chip Interconnection Networks–A Power and Performance Analysis". *IEEE Trans. Computers*, 57(9):1169–1181, Sep. 2008.

[118] M. KOIBUCHI, K. ANJO, Y. YAMADA, A. JOURAKU, and H. AMANO. "A Simple Data Transfer Technique Using Local Address for Networks-on-Chip". *IEEE Trans. Parallel and Distributed Systems*, 17(12):1425–1437, Dec. 2006.

[119] P. KONGETIRA, K. AINGARAN, and K. OLUKOTUN. "Niagara: A 32-Way Multithreaded Sparc Processor". *IEEE Micro*, 25(2):21–29, March/April 2005.

[120] S. KONSTANTINIDOU and L. SNYDER. "Chaos Router: Architecture and Performance". In *Proc. the 18th International Symposium on Computer Architecture (ISCA'91)*, pages 79–88, June 1991.

[121] R. KOTA and R. OEHLER. "Horus: Large-Scale Symmetric Multiprocessing for Opteron Systems". *IEEE Micro*, 25(2):30–40, March/April 2005.

[122] O. J. KUIKEN, X. ZHANG, and H. G. KERKHOFF. "Built-In Self-Diagnostics for a NoC-Based Reconfigurable IC for Dependable Beamforming Applications". In *Proc. IEEE Int'l Symp. on Defect and Fault Tolerance of VLSI Systems*, pages 45–53, 2008.

[123] A. KUMAR, L.-S. PEH, P. KUNDU, and N. K. JHA. "Toward Ideal On-Chip Communication Using Express Virtual Channels". *IEEE Micro*, 28(1):80–90, Jan./Feb. 2008.

[124] D. R. KUMAR, W. A. NAJJAR, and P. K. SRIMANI. "A New Adaptive Hardware Tree-Based Multicast Routing in K-Ary N-Cubes". *IEEE Trans. Computers*, 50(7):647–659, July 2001.

[125] S. KUMAR, A. JANTSCH, J.-K. SOININEN, M. FORSELL, M. MILLBERG, J. ÖBERG, K. TIENSYRJA, and A. HEMANI. "A Network on Chip Architecture and Design Methodology". In *Proc. IEEE Computer Society Annual Symposium on VLSI*, pages 105–112, 2002.

[126] V. KUMAR and V. SINGH. "Scalability of Parallel Algorithms for the All-Pairs Shortest Path Problem". Proc. Int'l Conf. Parallel Processing, Aug. 1991.

[127] D. LATTARD, E. BEIGNÉ, F. CLERMIDY, Y. DURAND, R. LEMAIRE, P. VIVET, and F. BERENS. "A Reconfigurable Baseband Platform Based on an Asynchronous Network-on-Chip". *IEEE Journal of Solid-State Circuits*, 43(1):223–235, Jan. 2008.

[128] G. LEARY, K. SRINIVASAN, K. MEHTA, and K. S. CHATHA. "Design of Network-on-Chip Architectures with a Genetic Algorithm-Based Technique". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(5):674–687, May 2009.

[129] K. LEE, S.-J. LEE, and H.-J. YOO. "Low-Power Network-on-Chip for High-Performance SoC Design". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 14(2):148–160, Feb. 2006.

[130] D. LENOSKI, J. LAUDON, K. GHARACHORLOO, W.-D. WEBER, A. GUPTA, J. HENNESSY, M. HOROWITZ, and M. S. LAM. "The Stanford DASH Multiprocessor". *IEEE Computer*, 25(3):63–79, March 1992.

[131] A. LEROY, D. MILOJEVIC, D. VERKEST, F. ROBERT, and F. CATTHOOR. "Concepts and Implementation of Spatial Division Multiplexing for Guaranteed Throughput in Networks-on-Chip". *IEEE Trans. Computers*, 57(9):1182–1195, Sep. 2008.

[132] L.-F. LEUNG and C.-Y. TSUI. "Energy-aware synthesis of networks-on-chip implemented with voltage islands". In *Proc. the 44th annual Design Automation Conference (DAC'07)*, pages 128–131, New York, NY, USA, 2007. ACM.

[133] K. LI and R. SCHAEFER. "A Hypercube Shared Memory Virtual". In *Proc. Int'l Conf. Parallel Processing*, volume 1, pages 125–132, 1989.

[134] M. LI, Q.-A. ZENG, and W.-B. JONE. "DyXY: A Proximity Congestion-Aware Deadlock-Free Dynamic Routing Method for Network on Chip". In *Proc. the 43rd Annual Design Automation Conference (DAC'06)*, pages 849–852, New York, NY, USA, 2006. ACM.

[135] S.-Y. LI, C.-H. HUANG, C.-H. CHAO, K.-H. HUANG, and A.-Y. WU. "Traffic-Balanced Routing Algorithm for Irregular Mesh-Based On-Chip Networks". *IEEE Trans. Computers*, 57(9):1156–1168, Sep. 2008.

[136] S. LIN, L. SU, G. ZHOU, D. JIN, and L. ZENG. "Design Networks-on-Chip with Latency/Bandwidth Guarantees". *IET Computers & Digital Techniques*, 3(2):184–194, 2009.

[137] X. LIN, P. K. MCKINLEY, and L. M. NI. "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers". *IEEE Trans. Parallel and Distributed Systems*, 5(8):793–804, Aug. 1994.

[138] X. LIN and L. M. NI. "Multicast Communication in Multicomputer Networks". *IEEE Trans. Parallel and Distributed Systems*, 4(10):1105–1117, Oct. 1993.

[139] V. LINDENSTRUTH. "An Extreme Processor for an Extreme Experiment". *IEEE Micro*, 26(2):48–57, March/April 2006.

[140] D. H. LINDER and J. C. HARDEN. "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes". *IEEE Trans. Computers*, 40(1):2–12, Jan. 1991.

[141] A. LINES. "Asynchronous Interconnect for SoC Design". *IEEE MICRO*, 24(1):32–41, Jan./Feb. 2004.

[142] J. LIU, L.-R. ZHENG, and H. TENHUNEN. "Interconnect intellectual property for Network-on-Chip (NoC)". *Elsevier Journal of Systems Architecture*, 50(2–3):65–79, Feb. 2004.

[143] I. LOI, S. MITRA, T. H. LEE, S. FUJITA, and L. BENINI. "A low-overhead fault tolerance scheme for TSV-based 3D network on chip links". In *Proc. the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'08)*, pages 598–602, 2008.

[144] P. LOTFI-KAMRAN, M. DANESHTALAB, C. LUCAS, and Z. NAVABI. "BARP: A Dynamic Routing Protocol for Balanced Distribution of Traffic in NoCs". In *Proc. the Conf. on Design, Automation and Test in Europe (DATE '08)*, pages 1408–1413, New York, NY, USA, 2008. ACM.

[145] Z. LU and A. JANTSCH. "TDM Virtual-Circuit Configuration for Network-on-Chip". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 16(8):1021–1034, Aug. 2008.

[146] Z. LU, B. YI, and A. JANTSCH. "Connection-oriented Multicasting in Wormhole-switched Network-on-Chip". In *Proc. IEEE Comp. Society Annual Symposium on VLSI (ISVLSI'06)*, pages 1–6, 2006.

[147] J. D. M. P. MALUMBRES and J. TORRELAS. "An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors". In *Proc. of the 8th IEEE Symposium on Parallel and Distributed Processing*, pages 186–189, 1996.

[148] C. A. M. MARCON, E. I. MORENO, N. L. V. CALAZANS, and F. G. MORAES. "Comparisons of Network-on-Chip Mapping Algorithms Targeting Low Energy Consumption". *IET Computers & Digital Techniques*, 2(6):471–482, 2008.

[149] P. MARTIN. "Design of a Virtual Component Neutral Network-on-Chip Transaction Layer". In *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE'05)*, pages 336–337, 2005.

[150] P. K. MCKINLEY, H. XU, E. KALNS, and L. M. NI. "ComPaSS: Efficient Communication Services for Scalable Architecture". In *Proc. Supercomputing*, pages 478–487, Oct. 1992.

[151] C. MCNAIRY and R. BHATIA. "Montecito: A Dual-Core Dual-Thread Itanium Processor". *IEEE Micro*, 25(2):10–20, March/April 2005.

[152] S. MEDARDONI, D. BERTOZZI, L. BENINI, and E. MACII. "Control and Datapath Decoupling in the Design of a NoC Switch: Area, Power and Performance Implication". In *Proc. Int'l System-on-Chip Symposium*, pages 1–4, 2007.

[153] A. MEJIA, J. FLICH, and J. DUATO. "On the Potentials of Segment-based Routing for NoCs". In *Proc. The 37th Int'l Conf. on Parallel Processing*, pages 594–603, 2008.

[154] A. MEJIA, M. PALESI, J. FLICH, S. KUMAR, P. LÓPEZ, R. HOLSMARK, and J. DUATO. "Region-Based Routing: A Mechanism to Support Efficient Routing Algorithms in NoCs". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(3):356–369, March 2009.

[155] J. MERLIN. "Techniques for the Automatic Parallelization of Distributed Fortran 90", *Technical Report SNARC 92-02*. Southampton Univ., Southampton, Nov. 1991.

[156] MESSAGE PASSING INTERFACE FORUM. "MPI-2: Extensions to the Message-Passing Interface". *Technical Report*, Univ. of Tennessee, Knoxville, Nov. 2003.

[157] M. MILLBERG, E. NILSSON, R. THID, and A. JANTSCH. "Guaranteed-Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip". In *Proc. Design Automation and Test in Europe (DATE'04)*, pages 890–895, Feb. 2004.

[158] M. MONCHIERO, G. PALERMO, C. SILVANO, and O. VILLA. "Exploration of Distributed Shared Memory Architectures for NoC-based Multiprocesors". *Elsevier Journal of System Architecture*, 53(10):719–732, Oct. 2007.

[159] J. MONTRYM and H. MORETON. "The GeForce 6800". *IEEE Micro*, 25(2):41–51, March/April 2005.

[160] G. E. MOORE. "Cramming more components onto integrated circuits". *Electronics*, 38(8):114–117, April 19 1965.

[161] F. MORAES, N. CALAZANS, A. MELLO, L. MÖLLER, and L. OST. "Hermes: An Infrastructure for Low Area Overhead Packet Switching Networks on Chip". *Elsevier, Integration, The VLSI Journal*, 38(1):69–93, Oct. 2004.

[162] S. S. MUKHERJEE, P. BANNON, S. LANG, A. SPINK, and D. WEBB. "The Alpha 21364 Network Architecture". *IEEE Micro*, 22(1):26–35, Jan./Feb. 2001.

[163] S. MURALI, D. ATIENZA, L. BENINI, and G. D. MICHELI. "A Method for Routing Packets Across Multiple Paths in NoCs with In-Order Delivery and Fault-Tolerance Guarantees". *VLSI Design, Journal of Hindawi Pub.*, vol. 2007:1–11, 2007.

[164] M. D. NAVA, P. BLOUET, P. TENINGE, M. COPPOLA, T. BEN-ISMAIL, S. PICCHIOTTINO, and R. WILSON. "An Open Platform for Developing Multiprocessor SoCs". *Computer*, 38(7):60–67, July 2005.

[165] C. NEEB and N. WEHN. "Designing Efficient Irregular Networks for Heterogeneous Systems-on-Chip". In *Proc. the 9th EUROMICRO Conf. on Digital System Design: Architectures, Methods and Tools (DSD'06)*, pages 665–672, 2006.

[166] C. NEEB and N. WEHN. "Designing efficient irregular networks for heterogeneous systems-on-Chip". *Elsevier, Journal of System Architecture*, 54(3–4):384–396, Sep. 2007.

[167] C. NICOPOULOS, A. YANAMANDRA, S. SRINIVASAN, N. VIJAYKRISHNAN, and M. J. IRWIN. "Variation-Aware Low-Power Buffer Design". In *Proc. the 41st Asilomar Conf. on Signals, Systems and Computers*, pages 1402–1406, 2007.

[168] E. NILSSON, M. MILLBERG, J. OBERG, and A. JANTSCH. "Load Distribution with the Proximity Congestion Awareness in a Network on Chip". In *Proc. of Design and Test in Europe, Conference and Exhibition (DATE'03)*, pages 1126–1127, March 2003.

[169] B. NITZBERG and V. LO. "Distributed Shared Memory: A Survey of Issues and Algorithms". *IEEE Computer*, 24(8):52–60, Aug 1991.

[170] J. NONNENMACHER, E. BIERSACK, and D. TOWSLEY. "Parity-based Loss Recovery for Reliable Multicast Transmission". *IEEE/ACM Trans. Networking*, 6(4):349–361, Aug. 1998.

[171] S. NUGENT. "The iPSC/2 Direct Connect Communications Technology". In *Proc. the 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 51–59, Jan. 1988.

[172] J. L. NUNEZ-YANEZ, D. EDWARDS, and A. M. COPPOLA. "Adaptive Routing Strategies for Fault-Tolerant On-Chip Networks in Dynamically Reconfigurable Systems". *IET Computers and Digital Techniques*, 2(3):184–198, 2008.

[173] W. OED. "The Cray Research Massively Parallel Processing System: Cray T3D". Cray Research Inc., 1993.

[174] U. Y. OGRAS, R. MARCULESCU, D. MARCULESCU, and E. G. JUNG. "Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(3):330–341, March 2009.

[175] OPENMP REVIEW BOARD. "OpenMP Application Programming Interface Specification for Parallel Programming". http://openmp.org/wp/openmp-specifications/, May, version 3.0 2008.

[176] J. D. OWENS, W. J. DALLY, R. HO, D. N. JAYASIMHA, S. W. KECKLER, and L.-S. PEH. "Research Challenges for On-Chip Interconnection Networks". *IEEE Micro*, 27(5):96–108, Sep-Oct. 2007.

[177] G. PALERMO, C. SILVANO, G. MARIANI, R. LOCATELLI, and M. COPPOLA. "Application-Specific Topology Design Customization for STNoC". In *Proc. the 9th EUROMICRO Conf. on Digital System Design: Architectures, Methods and Tools (DSD'07)*, pages 547–550, 2007.

[178] M. PALESI, R. HOLSMARK, S. KUMAR, and V. CATANIA. "Application Specific Routing Algorithms for Networks on Chip". *IEEE Trans. Parallel and Distributed Systems*, 20(3):316–330, March 2009.

[179] M. PALESI, G. LONGO, S. SIGNORINO, R. HOLSMARK, S. KUMAR, and V. CATANIA. "Design of Bandwidth Aware and Congestion Avoiding Efficient Routing Algorithms for Networks-on-Chip Platforms". In *Proc. the 2nd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'08)*, pages 97–106, Washington, DC, USA, 2008. IEEE Computer Society.

[180] D. PAMUNUWA, J. ÖBERG, L.-R. ZHENG, M. MILLBERG, A. JANTSCH, and H. TENHUNEN. "A Study on the implementation of 2-D mesh-based networks-on-chip in the nanometre regime". *Integration, The VLSI Journal*, 38(1):3–17, Oct. 2004.

[181] I. M. PANADES, A. GREINER, and A. SHEIBANYRAD. "A Low Cost Network-on-Chip with Guaranteed Service Well Suited to the GALS Approach". In *Proc. the 1st Int'l Conf. and Workshop on Nano-Networks*, pages 1–5, 2006.

[182] P. P. PANDE, C. GRECU, M. JONES, A. IVANOV, and R. SALEH. "Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures". *IEEE Trans. Computers*, 54(8):1025–1040, Aug. 2005.

[183] D. PARK, S. EACHEMPATI, R. DAS, A. K. MISHRA, Y. XIE, N. VIJAYKRISHNAN, and C. R. DAS. "MIRA: A Multi-layered On-Chip Interconnect Router Architecture". In *Proc. the 35th International Symposium on Computer Architecture (ISCA'08)*, pages 251–261, 2008.

[184] A. PULLINI, F. ANGIOLINI, S. MURALI, D. ATIENZA, G. D. MICHELI, and L. BENINI. "Bringing NoCs to 65 nm". *IEEE Micro*, 27(5):75–85, Sep-Oct. 2007.

[185] B. R. QUINTON, M. R. GREENSTREET, and S. J. E. WILTON. "Practical Asynchronous Interconnect Network Design". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 16(5):579–588, May 2009.

[186] G. RESEARCH. *"IEEE Standard Testability Method for Embedded Core-based Integrated Circuits"*, 2005. IEEE Std 1500^{TM}.

[187] E. RIJPKEMA, K. GOOSSENS, A. RADULESCU, J. DIELISSEN, J. VAN MEERBERGEN, P. WIELAGE, and E. WATERLANDER. "Trade Offs in the Design of a Router with Both Guranteed and Best-Effort Services for Networks on Chip". *Proc. IEE Computers & Digital Techniques*, 150(5):294–302, Sep. 2003.

[188] E. RIJPKEMA, K. GOOSSENS, A. RĂDULESCU, J. DIELISSEN, J. VAN MEERBERGEN, P. WIELAGE, and E. WATERLANDER. "Trade Offs in the Design of a Router with Both Guranteed and Best-Effort Services for Networks on Chip". In *Proc. Design Automation and Test in Europe (DATE'03)*, pages 350–355, March 2003.

[189] S. RODRIGO, J. FLICH, J. DUATO, and M. HUMMEL. "Efficient unicast and multicast support for CMPs". In *Proc. the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO'08)*, pages 364–375, Nov. 2008.

[190] D. S.-TORTOSA, T. AHONEN, and J. NURMI. "Issues in the development of a practical NoC: the Proteo Concept". *Integration, the VLSI Journal, Elsevier*, 38(1):95–105, Oct. 2004.

[191] I. SAASTAMOINEN, D. S.-TORTOSA, and J. NURMI. "Interconnect IP Node for Future System-on-Chip Designs". In *Proc. The 1st IEEE Int'l Workshop on Electronic Design, Test and Applications (DELTA'02)*, pages 116–120, 2002.

[192] S. SAEIDI, A. KHADEMZADEH, and A. MEHRAN. "SMAP: An Intelligent Mapping tool for Network on Chip". In *Proc. Int'l Symp. Signals, Circuits and Systems*, pages 1–4, July 2007.

[193] S. M. SAIT and H. YOUSSEF. *Iterative Computer Algorithms with Applications in Engineering*. IEEE Computer Society, 1999.

[194] M. K.-F. SCHÄFER, T. HOLLSTEIN, H. ZIMMER, and M. GLESNER. "Deadlock-Free Routing and Component Placement for Irregular Mesh-based Network-on-Chip". In *Proc. IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD'05)*, pages 238–245, Washington, DC, USA, 2005. IEEE Computer Society.

[195] D. SCHINKEL, E. MENSINK, E. A. M. KLUMPERINK, E. VAN TUIJL, and B. NAUTA. "Low-Power, High-Speed Transceivers for Network-on-Chip Communication". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(1):12–21, Jan. 2009.

[196] M. SGROI, M. SHEETS, K. KEUTZER, S. MALIK, J. RABAEY, and A. S. VINCENTELLI. "Addressing the System-on-a-Chip Interconnect Woes Through Communication-Based Design". In *Proc. The 38th Design Automation Conf. (DAC'01)*, pages 667–672, 2001.

[197] A. SHACHAM, K. BERGMAN, and L. P. CARLONI. "Photonic Networks-on-Chip for Future Generations of Chip Multiprocessors". *IEEE Trans. Computers*, 57(9):1246–1260, Sep. 2008.

[198] A. SHEIBANYRAD, A. GREINER, and I. MIRO-PANADES. "Multisynchronous and Fully Asynchronous NoCs for GALS Architectures". *IEEE Design & Test of Computers*, 25(6):572–580, Nov/Dec. 2008.

[199] K. SRINIVASAN, K. S. CHATHA, and G. KONJEVOD. "Linear-Programming-Based Techniques for Synthesis of Network-on-Chip Architecture". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 14(4):407–420, April 2006.

[200] S. STUIJK, T. BASTEN, M. GEILEN, A. H. GHAMARIAN, and B. THEELEN. "Resource-Efficient Routing and Scheduling of Time-Constrained Streaming Communication on Networks-on-Chip". *Journal of System Architecture*, 54(3-4):411–426, 2008.

[201] C. B. STUNKEL, D. G. SHEA, D. G. GRICE, P. H. HOCHSCHILD, and M. TSAO. "The SP1 High-Performance Switch". In *Proc. the Scalable High Performance Computing Conference*, pages 150–157, May 1994.

[202] R. TAMHANKAR, S. MURALI, S. STERGIO, A. PULLINI, F. ANGIOLINI, L. BENINI, and G. D. MICHELI. "Timing-Error-Tolerant Network-on-Chip Design Methodology". *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 26(7):1297–1310, July 2007.

[203] M. B. TAYLOR, J. KIM, J. MILLER, D. WENTZLAFF, F. GHODRAT, B. GREENWALD, and E. A. H. HOFFMAN. "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs". *IEEE Micro*, 22(2):25–35, March/April 2002.

[204] A. S. VAIDYA, A. SIVASUBRAMANIAM, and C. R. DAS. "Impact of Virtual Channels and Adaptive Routing on Application Performance". *IEEE Trans. Parallel and Distributed Systems*, 12(2):223–237, Feb. 2001.

[205] S. R. VANGAL, J. HOWARD, G. RUHL, S. DIGHE, H. WILSON, J. TSCHANZ, D. FINAN, A. SINGH, T. JACOB, S. JAIN, V. ERRAGUNTLA, C. ROBERTS, Y. HOSKOTE, N. BORKAR, and S. BORKAR. "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS". *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan. 2008.

[206] A. VITKOVSKI, A. JANTSCH, R. LAUTER, R. HAUKILAHTI, and E. NILSON. "Low-power and error protection coding for network-on-chip traffic". *IET Computers & Digital Techniques*, 2(6):483–492, May 2007.

[207] F. VITULLO, N. E. L'INSALATA, E. PETRI, S. SAPONARA, L. FANUCCI, M. CASULA, R. LOCATELLI, and M. COPPOLA. "Low-Complexity Link Microarchitecture for Mesochronous Communication in Networks-on-Chip". *IEEE Trans. Computers*, 57(9):1196–1201, Sep. 2008.

[208] L. WANG, Y. JIN, H. KIM, and E. J. KIM. "Recursive partitioning multicast: A bandwidth-efficient routing for Networks-on-Chip". In *Proc. the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS'09)*, pages 64–73, May 2009.

[209] X. WANG, T. AHONEN, and J. NURMI. "Applying CDMA Technique to Network-on-Chip". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 15(10):1091–1100, Oct. 2007.

[210] D. WENTZLAFF, P. GRIFFIN, H. HOFFMANN, L. BAO, B. EDWARDS, C. RAMEY, and E. A. M. MATTINA. "Characterizing The Cell EIB On-Chip Network". *IEEE Micro*, 27(5):15–31, Sep./Oct. 2007.

[211] D. WIKLUND and D. LIU. "SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems". In *Proc. IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS'03)*, pages 1–8, 2003.

[212] D. WINGARD. "MicroNetwork-Based Integration for SOCs". In *Proc. The 38th Design Automation Conf. (DAC'01)*, pages 673–677, 2001.

[213] D. WU, B. M. AL-HASHIMI, and M. T. SCHMITZ. "Improving Routing Efficiency for Network-on-Chip Through Contention-Aware Input Selection". In *Proc. the 2006 Asia and South Pacific Design Automation Conference (ASP-DAC'06)*, pages 36–41, Piscataway, NJ, USA, 2006. IEEE Press.

[214] H. XU, P. K. MCKINLEY, and L. M. NI. "Efficient Implementation of Barrier Synchronization in Wormhole-Routed Hypercube Multicomputers". *Journal of Parallel and Distributed Computing*, 16:172–184, Oct. 1992.

[215] J. XU, W. WOLF, J. HENKEL, and S. CHAKRADHAR. "A Design Methodology for application-Specific Networks-on-Chip". *ACM Trans. on Embedded Computing Systems*, 5(2):263–280, May 2006.

[216] S. YAN and B. LIN. "Custom Networks-on-Chip Architectures with Multicast Routing". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 17(3):342–355, March 2009.

[217] T. T. YE, L. BENINI, and G. D. MICHELI. "Packetization and Routing Analysis of On-Chip Multiprocessor Networks". *Journal of System Architecture*, 50(2-3):81–104, 2004.

[218] D. ZHAO and Y. WANG. "SD-MAC: Design and Synthesis of a Hardware-Efficient Collision-Free QoS-Aware MAC Protocol for Wireless Network-on-Chip". *IEEE Trans. Computers*, 57(9):1182–1195, Sep. 2008.

[219] H. ZIMMER, S. ZINK, T. HOLLSTEIN, and M. GLESNER. "Buffer-Architecture Exploration for Routers in a Hierarchical Network-on-Chip". In *Proc. the 19th Int'l Parallel and Distributed Processing Symposium*, pages 1–4, 2005.

[220] A. ZITOUNI, M. ZID, S. BADROUCHI, and R. TOURKI. "A Generic and Extensible Spidergon NoC". *Int'l Journal of Electronics, Circuits and Systems*, 1(3):197–202, 2007.

List of Own Publications

Related Publications

[221] T. HOLLSTEIN, F. A. SAMMAN, and M. GLESNER. "Combined Best-Effort and Guaranteed-Throughput NoC with Locally Organized Variable Message Identity". *IEEE Trans. Computers*, Revised for new submission, 2010.

[222] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Flexible Infrastructure for Network-on-Chip Prototypes Development based on VHDL-Modular-Oriented Design". In *Proc. the 11th Indonesian Students Scientific Meeting*, pages 188–194, May 2008.

[223] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Flexible Parallel Pipeline Network-on-Chip based on Dynamic Packet Identity Management". In *Proc. the 22nd IEEE Int'l Parallel and Distributed Processing Symp. (in Reconfigurable Architecture Workshop)*, pages 1–8, April 2008.

[224] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Multicast Parallel Pipeline Router Architecture for Network-on-Chip". In *Proc. Design Automation and Test in Europe (DATE'08)*, pages 1396–1401, March 2008.

[225] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Network-on-Chips: Potential Synergy between Parallel Computing and Multicore Processor Systems". In *Proc. the 11th Indonesian Students Scientific Meeting*, pages 134–141, May 2008.

[226] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "NoCs with combined low and high priority wormhole packet delivery services". In *Proc. the 2nd Electronic Design Automation Workshop*, pages 19–24, May 2008.

[227] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Planar Adaptive Router Microarchitecture for Tree-based Multicast Network-on-Chip". In *Proc. the 41th IEEE/ACM Int'l Symp on Microarchitecture (in NoC Architecture Workshop)*, pages 6–13, Nov 2008.

[228] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Flexible Wormhole-Switched Network-on-Chip with Two-Level Priority Data Delivery Service". *International Journal of Computer Science and Engineering*, 3(1):1–11, 2009.

[229] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Networks-on-Chip based on Dynamic Wormhole Packet Identity Management". *VLSI Design, Journal of Hindawi Pub. Corp.*, vol. 2009(Article ID 941701):1–15, Jan 2009.

[230] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Pipeline Control-Path Effects on Area and Performance of a Wormhole Switched Network-on-Chip". *International Journal of Electronics, Communications and Computer Engineering*, 1(1):46–54, 2009.

[231] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Test Strategy for Network-on-Chip Supporting Unicast-Multicast Data Transport". In *Workshop on Diagnostic Services in Network-on-Chips, in conjunction with Design Automation and Test in Europe (DATE'09)*, April 2009.

[232] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Adaptive and Deadlock-Free Tree-based Multicast Routing for Networks-on-Chip". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, 18(7):1067–1080, July 2010.

[233] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Guaranteed-Bandwidth Multicast Routing Method for Network-on-Chip with Runtime Connection-Oriented Data Communication". *IET Computers & Digital Techniques*, Submitted, 2010.

[234] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "New Theory for Deadlock-Free Multicast Routing in Wormhole-Switched Virtual-Channelless Networks-on-Chip". *IEEE Trans. Parallel and Distributed Systems*, In-press(doi:10.1109/TPDS.2010.120):Accepted for publication April 2010, Appear online June 2010.

[235] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Planar Adaptive Network-on-Chip Supporting Deadlock-Free and Efficient Tree-Based Multicast Routing Method". *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, Submitted, 2010.

[236] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Runtime Contention- and Bandwidth-Aware Adaptive Routing Selection Strategy for Networks-on-Chip". *IEEE Trans. Parallel and Distributed Systems*, Revised for new submission, 2010.

[237] F. A. SAMMAN, T. HOLLSTEIN, and M. GLESNER. "Wormhole Cut-Through Switching: Flit-Level Messages Interleaving for Virtual-Channelless Network-on-Chip". *Microprocessor and Microsystems - Embedded Hardware Design, Elsevier Journal*, Revised for second round review, 2010.

Unrelated Publications

[238] N. HARUN and F. A. SAMMAN. "Fuzzy Logic System for Optimization of Thermal Generator Unit Operation on Peak Load Condition". *Sci&Tech, Jurnal Ilmiah Sains dan Teknologi (Scientific Journal of Science and Technology)*, 1(2):21–37, Dec. 2000.

[239] F. A. SAMMAN. "Defuzzification Circuit Standard-Cell and its layout on CMOS Analog Integrated Circuit". *Sci&Tech, Jurnal Ilmiah Sains dan Teknologi (Scientific Journal of Science and Technology)*, 4(1):1–10, April 2003.

[240] F. A. SAMMAN. "Design and Layout of Analog CMOS Standard-Cell of Fuzzy Membership Function Circuit". *Sci&Tech, Jurnal Ilmiah Sains dan Teknologi (Scientific Journal of Science and Technology)*, 4(1):47–57, April 2003.

[241] F. A. SAMMAN. "Digitally Programmable CMOS Analog Max-Min Fuzzy Inference Engine Circuit". *Elektrikal Enjiniring (Journal of Electrical Engineering)*, vol. 01:11–16, May-Aug. 2003.

[242] F. A. SAMMAN. "Perancangan Kendali Adaptif dengan Model Umpanbalik Input-Output (Adaptive Control System Design with Input-Output Feedback Model)". In *Proc. of the 5th Seminar on Intelligent Technology and Its Applications (SITIA 2004)*, pages 165–170, 2004.

[243] F. A. SAMMAN. "Model Reference Adaptive Control Design For Non Linear Plant with Parametric Uncertainty". In *Proc. of the Int'l Conference on Instrumentation, Communications and Information Technology (ICICI 2005)*, pages 289–295, Aug. 2005.

[244] F. A. SAMMAN. "Perancangan Kendali Adaptive Model Acuan untuk Kendalian Tak Linier (Design of Model Reference Adaptive Control for Non Linear Plant)". *MITE, Majalah Ilmiah Teknik Elektro (Scientific Journal of Electrical Engineering)*, Accepted for Publication, Appear online 2008, http://ejournal.itb.ac.id.

[245] F. A. SAMMAN and N. HARUN. "Optimal Regulator with Estimator Circuit for Reactive Power Control". In *Proc. of the 2nd Seminar on Electrical Power Systems (Seminar Sistem Tenaga Elektrik-SSTE II)*, 2001.

[246] F. A. SAMMAN and N. HARUN. "Optimal Generator Excitation Control using Linear Quadratic Regulator with State-Observer Circuit". *MITE, Majalah Ilmiah Teknik Elektro (Scientific Journal of Electrical Engineering)*, 8(3):1–10, Dec. 2002.

[247] F. A. SAMMAN and N. HARUN. "Kendali Struktur Beragam untuk Model Kendalian Tak Pasti (Variable Structure Control for Plant with Uncertainties)". In *Proc. of the 5th Seminar on Intelligent Technology and Its Applications (SITIA 2004)*, pages 177–182, 2004.

[248] F. A. SAMMAN, E. JAURY, C. ELVIRE, and R. S. SADJAD. "Design and Analysis of Model Reference Adaptive Control for Plant with Internal and External Disturbances". In *Proc. of the Int'l Conference on Instrumentation, Communications and Information Technology (ICICI 2005)*, pages 419–424, Aug. 2005.

[249] F. A. SAMMAN, KAMILINA, SUHARMA, and R. S. SADJAD. "Sistem Kendali Adaptif Model Acuan untuk Kendalian Pendulum Terbalik Tak Linier (Model Reference Adaptive Control System for Non Linear Inverted Pendulum Plant)". In *Proc. of the 5th Seminar on Intelligent Technology and Its Applications (SITIA 2004)*, pages 171–176, 2004.

[250] F. A. SAMMAN, I. RACHMANIAR, Y. AINUDDIN, and Y. U. SOMBOLAYUK. "Implementation of Reconfigurable Fuzzy Logic Controller on 8-bit Microcontroller using C-Generic Code". In *Proc. of the Int'l Conference on Instrumentation, Communications and Information Technology (ICICI 2005)*, pages 796–800, Aug. 2005.

[251] F. A. SAMMAN, RUSLAN, SYAFRIDA, and R. S. SADJAD. "Implementasi Kendali Adaptif Model Acuan Berbasis Metode Gradien dalam Rangkaian Elektronika Analog (Implementation of Model Reference Adaptive Control based on Gradient Method on Analog Electronic Circuit)". *INTEK-Informasi Teknologi, Jurnal Penelitian Teknologi (Journal of Technological Research)*, 10(2):108–120, June 2004.

[252] F. A. SAMMAN, C. R. SABIRIN, T. ROCHAELI, and J. MALTA. "Embedded Control Based on Programming and Reconfigurable Electronic Devices: Multidisciplinary and Theory-and-Practice Oriented Teaching Development". In *Proc. the 11th Indonesian Students Scientific Meeting*, pages 142–151, May 2008.

[253] F. A. SAMMAN and R. S. SADJAD. "Analog Fuzzy Controller Circuit Design for Control Applications". In *Proc. of the IFAC Asian Control Conference (ASCC 2002)*, Sep. 2002.

[254] F. A. SAMMAN and R. S. SADJAD. "Analog MOS Circuit Design for Reconfigurable Fuzzy Logic Controller". In *Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS 2002)*, volume 2, pages 151–156, 2002.

[255] F. A. SAMMAN and R. S. SADJAD. "Reconfigurable Analog Fuzzy Controller Design, Part 1: Fuzzifier Circuit". In *Proc. of the 3rd Seminar on Intelligent Technology and Its Applications (SITIA 2002)*, pages 274–278, 2002.

[256] F. A. SAMMAN and R. S. SADJAD. "Reconfigurable Analog Fuzzy Controller Design, Part 2: Fuzzy Inference Circuit". In *Proc. of the 3rd Seminar on Intelligent Technology and Its Applications (SITIA 2002)*, pages 269–273, 2002.

[257] F. A. SAMMAN and R. S. SADJAD. "Reconfigurable Analog Fuzzy Controller Design, Part 3: Defuzzifier Circuit". In *Proc. of the 3rd Seminar on Intelligent Technology and Its Applications (SITIA 2002)*, pages 279–283, 2002.

[258] F. A. SAMMAN and R. S. SADJAD. "The Membership Function Circuit of an Analog Fuzzy Controller using MOS Differential Amplifier Pairs". *Elektrikal Enjiniring (Journal of Electrical Engineering)*, vol. 01:1–5, Sep.-Dec. 2002.

[259] F. A. SAMMAN and R. S. SADJAD. "Reconfigurable Analog Fuzzy Logic Controller Design using CMOS Integrated Circuit Technology". *MITE, Majalah Ilmiah Teknik Elektro (Scientific Journal of Electrical Engineering)*, 9(1):19–28, April 2003.

[260] F. A. SAMMAN and R. S. SADJAD. "Kendali Adaptif Model Acuan untuk Kendalian Tak Linier Sistem Pengatur Suhu Ruang (Model Reference Adaptive Control for Non Linear Plant of a Chamber Temperatur Control System)". *Jurnal Penelitian Enjiniring (Journal of Engineering Research)*, 10(3):425–437, Sep.-Dec. 2004.

[261] F. A. SAMMAN, R. S. SADJAD, and E. Y. SYAMSUDDIN. "The Reconfigurable Membership Function Circuit using Analog Bipolar Electronic". In *Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS 2002)*, volume 2, pages 537–540, 2002.

[262] F. A. SAMMAN, I. SUMSULAWATY, RAHMI, and R. S. SADJAD. "Digital Control Signal Processor on FPGA Chip". In *Proc. of the Int'l Conference on Instrumentation, Communications and Information Technology (ICICI 2005)*, pages 151–156, Aug. 2005.

[263] F. A. SAMMAN and E. Y. SYAMSUDDIN. "Programmable Fuzzy Logic Controller Circuit Design using VHDL-based Synthesis". *MITE, Majalah Ilmiah Teknik Elektro (Scientific Journal of Electrical Engineering)*, 7(3):12–20, Dec. 2001.

[264] F. A. SAMMAN and E. Y. SYAMSUDDIN. "Fuzzy Controller using VHDL-Based Synthesis on CPLD Chip for Control Applications". In *Proc. of the IFAC Asian Control Conference (ASCC 2002)*, Sep. 2002.

[265] F. A. SAMMAN and E. Y. SYAMSUDDIN. "High-Speed Fuzzy Circuit on CPLD Chip for Fuzzy Information Processing". In *Proc. of the Asia-Pacific Conference on Communications (APCC 2002)*, Sep. 2002.

[266] F. A. SAMMAN and E. Y. SYAMSUDDIN. "Programmable Fuzzy Logic Controller Circuit on CPLD Chip". In *Proc. of the IEEE Asia-Pacific Conference on Circuits and Systems (APCCAS 2002)*, volume 2, pages 561–564, 2002.

[267] F. A. SAMMAN and E. Y. SYAMSUDDIN. "VHDL-Synthesis Model of Dedicated Fuzzy Controller Hardware". In *Proc. of the 3rd Seminar on Intelligent Technology and Its Applications (SITIA 2002)*, pages 263–268, 2002.

[268] F. A. SAMMAN and R. J. WIDODO. "Simulation on Modular Circuits for Parallel-Analogue Fuzzy Logic Controller". In *Proc. of the 2nd IFAC-CIGR Workshop on Intelligent Control for Agricultural Applications*, pages 265–270, 2001.

Supervised Theses

[269] J. ANTONI. *"Design of a Multicore Processing using NoC-Interconnect and a Low-Level Programming Interface"*. Master Thesis, TU Darmstadt, Nov. 2008. Co-Supervised by T. Hollstein.

[270] B. DOLLAK. *"Multiprozessor-System basierend auf einem Netzwerk-On-Chip und Multiprozessor-Programmierung"*. Studienarbeit, TU Darmstadt, Nov. 2009. Co-Supervised by T. Hollstein.

[271] S. LE. *"On-Chip Network Interface Design for Multicast and Guaranteed-Bandwidth Data Transportation Protocol"*. Studienarbeit, TU Darmstadt, Oct. 2009.

[272] F. B. LULEY. *"Parallel Task Application Mapping for Network-on-Chip-based Multiprocessor Systems"*. Studienarbeit, TU Darmstadt, Oct. 2009. Co-Supervised by T. Hollstein.

[273] S. RHAZI. *"Entwicklung einer Testumgebung für die Bewertung der Leistung der Datenübertragung des XHiNoC On-Chip Netzwerkes"*. Master Thesis, TU Darmstadt, Oct. 2009. Co-Supervised by T. Hollstein.

[274] Y. SENNANI. *"Tree-based Networks Topologies for Hierarchical Network-on-Chip"*. Diplomarbeit, TU Darmstadt, Aug. 2008. Co-Supervised by T. Hollstein.

[275] L. SHEN. *"Effiziente Zusammenstellung und Verarbeitung von Network-on-Chip Datenpaketen"*. Studienarbeit, TU Darmstadt, Sep. 2008. Co-Supervised by T. Hollstein.

Curriculum Vitae

Personal Data:

Surname	:	Samman
Given Name	:	Faizal Arya
Birth place	:	Makassar (Ujung Pandang), Indonesia
Birth date	:	June 5, 1975
Marital status	:	Married with 3 Children
Father's name	:	Djamaluddin Samman Daeng Mattarru'
Mother's name	:	Sitti Aisyah Ende Daeng Talele

School Education:

Year	*School level*	*School name*
1982–1988	Primary School	SD Negeri 6 Sungguminasa, Gowa, Indonesia
1988–1991	Secondary School	SMP Negeri 1 Sungguminasa, Gowa, Indonesia
1991–1994	High School	SMA Negeri Sungguminasa, Gowa, Indonesia

Higher Education:

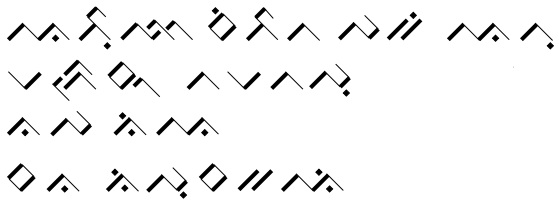
Year	*Degree*	*Institute*
Aug. 1994–Jan. 1999	Bachelor Degree	Universitas Gadjah Mada, Yogyakarta, Indonesia
Aug. 2000–Sep. 2002	Master Degree*	Institut Teknologi Bandung, Indonesia
Oct. 2006–Jan. 2010	Doctoral Degree**	Technische Universität Darmstadt (TUD), Germany

Work Experience:

Year	*Position*	*Institute*
2002–2006	Research & Teaching Staff*	Universitas Hasanuddin, Makassar, Indonesia
2006–2009	Research & Teaching Staff	Technische Universität Darmstadt (TUD), Germany
Since 2010	Post-Doctoral Fellow**	Fraunhofer Institute LBF and TUD, Germany

* Master degree program is supported by Scholarship Award from Indonesian Ministry of National Education.
* Research and Teaching Staff position is temporarily inactive.
** Doctoral research is supported by Scholarship Award from DAAD (*Deutscher Akademischer Austausch-Dienst*).
** Postdoctoral research in LOEWE–Zentrum AdRIA (Adaptronik–Research, Innovation, Application) Project funded by Hessian Ministry of Science and Arts.

PROVERBS FROM MAKASSAR:



TRANSLITERATION:

Abbulosibatang paki' antu
Ma'reso tamatappu'
Nampa nia'
Sannang nipusakai.

ENGLISH TRANSLATION:

If we are united and
we strive to work hard,
then
we will reap the benefits.

SPECIAL DEDICATION:

Disertasi ini kupersembahkan khusus
untuk Almarhumah Ibunda tercinta,
yang senantiasa mendidik kami putra-putrinya
dengan penuh kesabaran dan kasih sayang,
dan selalu berdoa kepada Allah Yang Maha Esa
untuk keselamatan dan kesuksesan hidup kami
di dunia dan di akhirat kelak.
Ibu, kami bangga menjadi putra-putrimu.

Index