

Advances in Deflection Routing based Network on Chips

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

Julius-Maximilians-
**UNIVERSITÄT
WÜRZBURG**



vorgelegt von

Armin Runge

aus
Frankfurt a. M.

Würzburg 2017

Advances in Deflection Routing based Network on Chips

Dissertation zur Erlangung des
naturwissenschaftlichen Doktorgrades
der Julius-Maximilians-Universität Würzburg

vorgelegt von

Armin Runge

aus
Frankfurt a. M.

Würzburg 2017

Eingereicht am: 13.02.2017
bei der Fakultät für Mathematik und Informatik
1. Gutachter: Prof. Dr. Reiner Kolla
2. Gutachter: Prof. Dr. Martin Radetzki

Abstract

The progress which has been made in semiconductor chip production in recent years enables a multitude of cores on a single die. However, due to further decreasing structure sizes, fault tolerance and energy consumption will represent key challenges. Furthermore, an efficient communication infrastructure is indispensable due to the high parallelism at those systems. The predominant communication system at such highly parallel systems is a Network on Chip (NoC). The focus of this thesis is on NoCs which are based on deflection routing. In this context, contributions are made to two domains, fault tolerance and dimensioning of the optimal link width. Both aspects are essential for the application of reliable, energy efficient, and deflection routing based NoCs.

It is expected that future semiconductor systems have to cope with high fault probabilities. The inherently given high connectivity of most NoC topologies can be exploited to tolerate the breakdown of links and other components. In this thesis, a fault-tolerant router architecture has been developed, which stands out for the deployed interconnection architecture and the method to overcome complex fault situations. The presented simulation results show, all data packets arrive at their destination, even at high fault probabilities. In contrast to routing table based architectures, the hardware costs of the herein presented architecture are lower and, in particular, independent of the number of components in the network.

Besides fault tolerance, hardware costs and energy efficiency are of great importance. The utilized link width has a decisive influence on these aspects. In particular, at deflection routing based NoCs, over- and under-sizing of the link width leads to unnecessary high hardware costs and bad performance, respectively. In the second part of this thesis, the optimal link width at deflection routing based NoCs is investigated. Additionally, a method to reduce the link width is introduced. Simulation and synthesis results show, the herein presented method allows a significant reduction of hardware costs at comparable performance.

Kurzfassung

Die Fortschritte der letzten Jahre bei der Fertigung von Halbleiterchips ermöglichen eine Vielzahl an Rechenkernen auf einem einzelnen Chip. Die in diesem Zusammenhang immer weiter sinkenden Strukturgrößen führen jedoch dazu, dass Fehlertoleranz und Energieverbrauch zentrale Herausforderungen darstellen werden. Aufgrund der hohen Parallelität in solchen Systemen, ist außerdem eine leistungsfähige Kommunikationsinfrastruktur unabdingbar. Das in diesen hochgradig parallelen Systemen überwiegend eingesetzte System zur Datenübertragung ist ein Netzwerk auf einem Chip (engl. Network on Chip (NoC)). Der Fokus dieser Dissertation liegt auf NoCs, die auf dem Prinzip des sog. Deflection Routing basieren. In diesem Kontext wurden Beiträge zu zwei Bereichen geleistet, der Fehlertoleranz und der Dimensionierung der optimalen Breite von Verbindungen. Beide Aspekte sind für den Einsatz zuverlässiger, energieeffizienter, Deflection Routing basierter NoCs essentiell.

Es ist davon auszugehen, dass zukünftige Halbleiter-Systeme mit einer hohen Fehlerwahrscheinlichkeit zurecht kommen müssen. Die hohe Konnektivität, die in den meisten NoC Topologien inhärent gegeben ist, kann ausgenutzt werden, um den Ausfall von Verbindungen und anderen Komponenten zu tolerieren. Im Rahmen dieser Arbeit wurde vor diesem Hintergrund eine fehlertolerante Router-Architektur entwickelt, die sich durch das eingesetzte Verbindungsnetzwerk und das Verfahren zur Überwindung komplexer Fehlersituationen auszeichnet. Die präsentierten Simulations-Ergebnisse zeigen, dass selbst bei sehr hohen Fehlerwahrscheinlichkeiten alle Datenpakete ihr Ziel erreichen. Im Vergleich zu Router-Architekturen die auf Routing-Tabellen basieren, sind die Hardware-Kosten der hier vorgestellten Router-Architektur gering und insbesondere unabhängig von der Anzahl an Komponenten im Netzwerk, was den Einsatz in sehr großen Netzen ermöglicht.

Neben der Fehlertoleranz sind die Hardware-Kosten sowie die Energieeffizienz von NoCs von großer Bedeutung. Einen entscheidenden Einfluss auf diese Aspekte hat die verwendete Breite der Verbindungen des NoCs. Insbesondere bei Deflection Routing basierten NoCs führt eine Über- bzw. Unterdimensionierung der Breite der Verbindungen zu unnötig hohen Hardware-Kosten bzw. schlechter Performanz. Im zweiten Teil dieser Arbeit wird die optimale Breite der Verbindungen eines Deflection Routing basierten NoCs untersucht. Außerdem wird ein Verfahren zur Reduzierung der Breite dieser Verbindungen vorgestellt. Simulations- und Synthese-Ergebnisse zeigen, dass dieses Verfahren eine erhebliche Reduzierung der Hardware-Kosten bei ähnlicher Performanz ermöglicht.

Danksagung

Mit der Abgabe dieser Dissertation endet ein bedeutender Abschnitt für mich und daher möchte ich mich bei einigen Menschen bedanken, die mich in den letzten Jahren unterstützt haben. Zu aller erst gilt mein besonderer Dank meinem Doktorvater Prof. Dr. Reiner Kolla, der mir schon in einer sehr frühen Phase meines Studiums die Möglichkeit gab als studentische Hilfskraft und später als wissenschaftlicher Mitarbeiter und Promotionsstudent im Team des Lehrstuhls für Technische Informatik zu arbeiten. Ebenfalls bedanken möchte ich mich bei Prof. Dr. Martin Radetzki für seine Anregungen und die Bereitschaft sich als Zweitgutachter für diese Dissertation zur Verfügung zu stellen.

Bedanken möchte ich mich außerdem bei allen aktuellen und ehemaligen Mitarbeiterinnen und Mitarbeitern des Lehrstuhls für Technische Informatik. Ich habe die tolle Arbeitsatmosphäre sowie die unzähligen Gespräche und fachlichen Diskussionen in der Elektronikwerkstatt, in der hin und wieder auch Kaffee gekocht wurde, sehr genossen. Insbesondere möchte ich mich bei Isabel Grimm und Johannes Mühr für das Lesen dieser Arbeit, sowie die hilfreichen Kommentare und Korrekturen bedanken.

Abschließend möchte ich mich noch bei meiner Familie und meinen Freunden bedanken. Ein besonderer Dank gilt meinen Eltern, die mir das Studium in Würzburg ermöglicht haben und dadurch den Grundstein für meine weitere wissenschaftliche Laufbahn gelegt haben. Ein ganz besonderer Dank gilt außerdem meiner Verlobten Isabel Grimm. Vielen Dank für deine Unterstützung und dein Verständnis in den letzten Monaten!

Contents

Abstract	iii
Kurzfassung	v
Danksagung	vii
List of Tables	xiii
List of Figures	xv
Abbreviations & Acronyms	xix
Symbols	xxi
1 Introduction	1
1.1 Motivation	1
1.2 Thesis Outline and Scientific Contributions	4
2 Network Basics	7
2.1 Building Blocks and Terminology	8
2.2 Topology	11
2.3 Routing	14
2.3.1 Classification	14
2.3.2 Routing Algorithms and Turn Model	15
2.4 Buffering and Flow Control	18
2.4.1 Bufferless Flow Control	18
2.4.2 Buffered Flow Control	19
2.4.3 Buffer Availability	24
2.5 Router Architecture	24
2.6 Performance	27
2.6.1 Performance Metrics	27
2.6.2 Evaluation Methodology	29
2.7 Conclusion and Existing Interconnection Networks	34
3 Deflection Routing based Router Architectures	37
3.1 Principle of Deflection Routing	37

3.2	Pros and Cons of Deflection Routing	39
3.3	Deflection Routing Implementations	42
3.3.1	Crossbar based Architectures	43
3.3.2	Permutation Network based Architectures	46
3.4	Basis Network on Chip and Router Architecture	53
3.4.1	Flit Prioritization Scheme	54
3.4.2	Routing Algorithm	55
3.4.3	Summary	65
4	Fault-tolerant and Deflection Routing based Router Architecture	67
4.1	Motivation and Scope	68
4.1.1	Fault Tolerance Methods	69
4.1.2	Fault-tolerant Routing	71
4.1.3	Conclusion	73
4.2	Related Work of Chapter 4	73
4.3	FaFNoC Router Architecture	77
4.3.1	Fault Tolerance and Banyan Networks	78
4.3.2	Substitute Benes Networks for Banyan Networks	86
4.3.3	Concept of Fault-aware Flits	94
4.3.4	Summary and Complete Overview of FaFNoC Router Architecture	99
4.4	Evaluation of FaFNoC Router Architecture	100
4.4.1	Non-fault-tolerant Architecture	101
4.4.2	Fault-tolerant Architecture	104
4.5	Summary and Conclusion of Chapter 4	111
5	Design of Deflection Routing based Network on Chips	113
5.1	Introduction and Motivation	113
5.1.1	Effect of the Link Width on Buffered, Packet Switched NoCs . . .	114
5.1.2	Effect of the Link Width on Deflection Routing based NoCs	115
5.2	Related Work	117
5.3	The optimal Link Width	119
5.3.1	Effect of the Flit Size on Hardware Costs	119
5.3.2	Effect of the Flit Size on Performance	123
5.4	<i>TwoPhases</i> - An Alternating Transmission Scheme	132
5.4.1	Methodology of <i>TwoPhases</i>	133
5.4.2	Transmission Methods	139
5.4.3	Evaluation	141
5.5	Summary and Conclusion of Chapter 5	149
6	Concluding Remarks	151
6.1	Contributions of this Thesis	151

6.2 Future Work	153
A Appendix	155
A.1 Fault Situations of Chapter 4	155
A.2 Lambert W function	159
Publications of the Author	161
Bibliography	163

List of Tables

2.1	Metrics for different Network on Chip (NoC) topologies.	14
2.2	Configuration of XST for synthesis.	29
2.3	Bit permutation traffic patterns.	30
2.4	Characteristics of the three herein evaluated PARSEC benchmarks.	32
2.5	Characteristics of the interconnection architectures of real processors.	35
4.1	Several methods and techniques to enhance the reliability of NoCs.	70
4.2	Possible and impossible routing decisions of different interconnection architectures.	90
5.1	Simulated link widths, and the consequential number of flits per large message as well as per small message.	131
5.2	PARSEC simulation results, for seven different link widths / flit sizes.	131
5.3	All Pareto optimal link widths for $ msg = 32$ and $ hd = 16$	142
5.4	PARSEC simulation results, for three different clock ratios.	146

List of Figures

1.1	Challenges of many-core processor design.	2
2.1	Three communication infrastructures used in System-on-Chips.	9
2.2	Data units at NoCs.	10
2.3	Flit structure of head, body, and tail flits.	11
2.4	Frequently used direct NoC topologies.	12
2.5	Routing dependent deadlock at a 2D mesh.	16
2.6	The turn model for a 2D mesh topology.	17
2.7	Classification of flow control mechanisms according to [DT04].	18
2.8	Time-space diagram of a transmission with store-and-forward flow control.	20
2.9	Time-space diagram of a transmission with virtual cut-through flow control.	21
2.10	Time-space diagram of a transmission with wormhole flow control.	22
2.11	Time-space diagram of a transmission with virtual channel flow control.	23
2.12	Block diagram of a typical input queued, virtual channel router, which uses credit-based flow control, according to [MWM04].	25
2.13	Three indirect network topologies: 6×6 crossbar, 4×4 Banyan network, and 4×4 Benes network.	26
2.14	Typical NoC performance metrics illustrated.	28
2.15	Injection probabilities during the first 50 million clock cycles of the ROI of three PARSEC benchmarks.	33
3.1	Crossbar based router architecture	43
3.2	CHIPPER router architecture [FCM10; FCM11]	47
3.3	A possible and an impossible permutation for a Banyan network.	49
3.4	Retransmit-once protocol scheme [FCM10, p. 12]	51
3.5	MinBD router architecture [Fal+11; Fal+12]	52
3.6	2D mesh topology with loop links at the edges of the mesh.	54
3.7	At AVOID_CENTER, the used routing algorithm depends on the routers' position.	59
3.8	Throughput for all evaluated routing algorithms and eight different injection probabilities.	62
3.9	Statistical values of the flits' hop count for all evaluated routing algorithms and eight different injection probabilities.	62

3.10	Statistical values of the flits' latency for all evaluated routing algorithms and eight different injection probabilities.	63
3.11	Link utilization at an injection probability of $\alpha = 20\%$	64
3.12	Statistical values of the flits' deflections for all evaluated routing algorithms and eight different injection probabilities.	65
4.1	Two fault shapes which are not tolerated by FoN.	75
4.2	Overview of the FaFNoC router architecture.	78
4.3	All one-direction fault situations of a Banyan network.	80
4.4	All three-directions fault situations of a Banyan network.	81
4.5	All unambiguous two-directions fault situations of a Banyan network.	82
4.6	All ambiguous two-directions fault situations of a Banyan network.	82
4.7	Two fault situations, the first situation requires a reflection of flits, whereas reflections create a risk of livelocks at the second situation.	84
4.8	Overview of the FaFNoC router architecture based on a Banyan network.	85
4.9	Return flag handling of FaFNoC-Banyan illustrated.	87
4.10	A 4×4 Benes network with configuration adopted from CHIPPER's Banyan network.	88
4.11	FaFNoC's Benes network, with a changed arrangement of input ports as well as a changed wiring between the first and the second stage of switching elements.	89
4.12	All one-direction fault situations of a Benes network.	92
4.13	All two-directions fault situations of a Benes network.	93
4.14	All three-directions fault situations of a Benes network.	94
4.15	Detailed illustration of switching element s_5 and fault-status-handler for north port.	97
4.16	Path of a flit which is routed from R_s to R_d . At R_a , the flit is turned to region evasion modus, and at R_c , back to normal mode.	98
4.17	FaFNoC's flit structure for an 8×8 NoC.	100
4.18	Simulation results for non-fault-tolerant router architectures with different interconnection architectures, uniform random traffic, and a variable injection probability.	102
4.19	Synthesis results for one non-fault-tolerant router, with different deployed interconnection architectures.	102
4.20	Frequency adjusted average latency for non-fault-tolerant router architectures.	103
4.21	Simulation results for random traffic and transpose traffic.	105
4.22	Average hop count and percentage of lost flits for full traffic.	106
4.23	Simulation results for two PARSEC benchmarks and a variable number of random link failures.	108

4.24 Hardware requirements for the herein compared fault-tolerant router architectures and four different NoC dimensions.	110
4.25 Achievable frequencies for the herein compared fault-tolerant router architectures.	110
4.26 Frequency adjusted average latency for fault-tolerant router architectures.	111
5.1 Livelock problem of deflection routing based NoCs illustrated.	116
5.2 Permutation network based router architecture	120
5.3 Synthesis results for one router and different flit sizes.	123
5.4 Number of transferred flits per message and a variable link width.	125
5.5 Link width for a variable number of flits per message.	126
5.6 Transmission of three flits between two two-hop neighbors.	126
5.7 Number of transferred flits and transferred messages for a variable flit size / link width.	128
5.8 Average latency l and average hop count hc for traffic with uniform length.	129
5.9 Average latency l and average hop count hc for traffic with non-uniform length.	130
5.10 Mesh topology with <i>TwoPhases</i> for two consecutive clock cycles.	134
5.11 Transmission of one packet, which is routed with <i>TwoPhases</i> from R_x to its two-hop neighbor R_z via R_y	136
5.12 Transmission of one packet, which is routed with <i>TwoPhases</i> and a pipelined router architecture with two stages from R_x to its two-hop neighbor R_z via R_y	137
5.13 Transmission of one message with Serialization between two-hop neighbors.	141
5.14 Average hop count hc and latency l for a variable injection probability α and different synthetic workloads.	144
5.15 Speedup for all evaluated PARSEC simulations, compared to standard deflection routing with links that are as wide as the largest packet.	147
5.16 Synthesis results for different router architectures.	148
A.1 First simulated fault situation	156
A.2 Second simulated fault situation.	157
A.3 Third simulated fault situation.	158

Abbreviations & Acronyms

ACK	ACKnowledgement
ARQ	Automatic Repeat reQuest
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BE	Best Effort
BLESS	BufferLESS routing algorithms
CHIPPER	CHeap-Interconnect Partially PERmuting Router
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
CMP	Chip MultiProcessor
CRC	Cyclic Redundancy Check
FaF	Fault-aware Flits
FaFNoC	Fault-aware Flits NoC
FEC	Forward Error Correction
fifo	first in - first out
flit	flow control digit
FPGA	Field-Programmable Gate Array
GT	Guaranteed Throughput
<i>HWR</i>	HardWare Requirement
I/O	Input/Output
<i>ID</i>	IDentifier
ILP	Instruction-Level Parallelism
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors
LUT	LookUp Table
MinBD	Minimally-Buffered Deflection
MPSoC	MultiProcessor System-on-Chip

Abbreviations & Acronyms

MSHR	Miss Status Holding Register
MUX	MUltipleXer
NACK	Negative-ACKnowledgement
NI	Network Interface
NoC	Network on Chip
OCP	Open Core Protocol
PARSEC	Princeton Application Repository for Shared-Memory Computers
PE	Processing Element
phit	physical transfer digit
REG	REGister
ROI	Region Of Interest
SEC code	Single-Error Correcting code
SEDED code	Single-Error Correcting and Double-Error Detecting code
SEU	Single Event Upset
SoC	System-on-Chip
TDM	Time-Division Multiplexing
TMR	Triple Modular Redundancy
TSV	Through-Silicon Via
VC	Virtual Channel
VCI	Virtual Component Interface
VHDL	Very high speed integrated circuit Hardware Description Language
VLSI	Very-Large-Scale Integration

Symbols

α	injection probability [flits/clock cycle/node]
λ	fault rate, probability that a specific link is defect
θ	throughput [flits/clock cycle/node]
b_i	i-th body flit of a packet
c_i	i-th network clock cycle
$d_1(\cdot)$	Manhattan distance of \cdot to destination [hops]
dst	destination address of a flit, field in the flit structure
E	port to the east
f_i	fault information, routers' input vector with a width of P bit, f_{ij} gives the state of link j
f_i	flit with ID i
$\#f$	number of flits per message
$ f $	size of a flit [bit]
fs	fault status, field in the flit structure, which consists of $tDir$ and $tDst$
ft	flit type, to distinguish between head, body, and tail flits
h	head flit of a packet
hc	hop count of a flit, field in the flit structure
$ hc $	size of the hc field [bit]
hd	header information of a flit, contains i.a.: flit's destination address, flit's hop count
$ hd $	size of the header information [bit]
i_i	i-th input
id	id of a flit, field in the flit structure
$ id $	size of the id field [bit]
L	locale port
l	latency [clock cycle], time from when a flit enters the network to when the flit departs the network

Symbols

$ LI $	link width [bit]
msg_i	message with ID i
$ msg $	size of a message msg [bit]
N	port to the north
o_i	i -th output
P	number of ports per router
p_i	packet with ID i
$ p $	size of a packet [bit]
ph_i	phit with ID i
$\#ph$	number of phits per flit
$ ph $	size of a phit [bit]
pl	payload of a flit, field in the flit structure
$ pl $	size of the pl field [bit]
pos	address of current router
R_i	Router with ID i
S	port to the south
s_i	switching element / permute block with ID i
src	source address of a flit, field in the flit structure
t	tail flit of a packet
$t_{ejec}(\cdot)$	ejection time of \cdot
$t_{injec}(\cdot)$	injection time of \cdot
$tDir$	turn direction, $tDir \in \{R, L, 0\}$, field in the flit structure for fault tolerance
$tDst$	turn distance, field in the flit structure for fault tolerance
W	port to the west
$z(\cdot)$	state of switching element \cdot , $z \in \{0, 1\}$

Introduction

Contents

1.1 Motivation	1
1.2 Thesis Outline and Scientific Contributions	4

1.1 Motivation

Technology scaling has been following the self-fulfilling Moore’s law [Moo98] for several generations of semiconductor chips. According to Gordon Moore’s prediction, the number of transistors per unit area on integrated circuits doubled every two years. However, an end of traditional transistor scaling was predicted in the latest International Technology Roadmap for Semiconductors (ITRS) report [ITR15] for the 2020s, and also Gordon Moore expects an end of his own prediction, as technology scaling gets close to the atomic limitation [Cou15]. Nevertheless, chips with billions of transistors are already commercially available [Saw+11]. Recently, Gordon Moore stated in an interview:

“But we will be able to make several billion transistors on an integrated circuit at that time. And the room this allows for creativity is phenomenal.”
[Cou15]

In addition to great opportunities, technology scaling implicates several challenges. For several decades, Dennard scaling [Den+74], which essentially suggests that power requirements of a transistor are proportional to its area, has allowed manufactures to raise clock frequencies without significantly increasing overall circuit power consumption. Since around 2005-2007, Dennard scaling no longer seems to apply. Current leakage poses major challenges and a heat up of the chips, which further increases energy consumption. This leads to an inability to increase clock frequencies significantly and created a so called *power wall*.

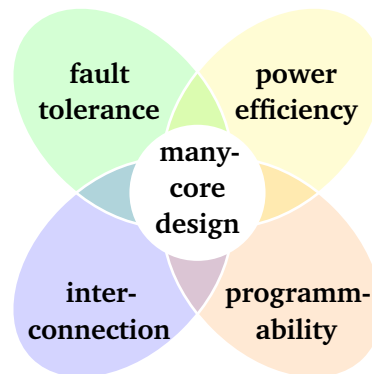


Figure 1.1: Challenges of many-core processor design.

Until recently, the utilization of Instruction-Level Parallelism (ILP) techniques was another key factor to improve processor performance. However, it became increasingly difficult to find enough parallelism in a single instruction stream to utilize the capacity of high-performance single-core processors. This lack of available instruction-level parallelism is called the *ILP wall*. Therefore, chip manufactures have focused on techniques to exploit higher levels of parallelism, as multithreading, and finally multi- and many-core processors.

The trend towards many-core processors is also confirmed by Pollack's Rule [Bor07]. This rule states that performance increase is roughly proportional to the square root of the increase in complexity, i.e. area. In contrast, the power consumption increase is roughly linearly proportional to the increase in complexity. Consequently, this leads to chips with hundreds or thousands of low-complexity cores, as those chips can provide more performance than a single complex core¹. Indeed, chips with hundreds of cores on a single die exist and are commercially available nowadays. These chips range from Chip MultiProcessors (CMPs), which are usually homogeneous, and frequently tile-based, to heavily customized MultiProcessor System-on-Chips (MPSoCs), which are usually very heterogeneous.

The design of many-core processors also entails several challenges, as programmability, power efficiency, fault tolerance and interconnection (cf. Figure 1.1). Parallel programming paradigms are required to be able to utilize the processing power provided by these systems. Moreover, technology scaling and shrinking manufacturing processes have several undesirable side-effects, like an increasing variability in performance and reliability as well as a significant energy consumption [Bor05]. The increasing number of transistors per chip leads to a power challenge. It is projected that the amount of

¹Obviously, the workload has to contain enough parallelism to utilize the performance of all available cores.

dark silicon, i.e. the part of a chip which has to be powered-off to comply with power constraints, may reach up to 50% – 80% at 8 nm technology [Esm+11]. In addition, those systems have to cope with highly varying component lifetimes. Hence, fault tolerance concepts will be required to achieve reasonable operating periods and to improve yield rate. Furthermore, the increasing number of components on a single die leads to growing communication requirements. Traditionally deployed communication infrastructures, like buses, rings, and crossbars, can not provide the required high bandwidth, low latency, and energy efficiency, as they do not scale with the high number of communication participants.

For some of the above mentioned challenges solutions have yet to be found, but Network on Chips (NoCs) provide a scalable, parallel communication infrastructure. Thus, they are considered as the dominant interconnection architecture of many-core processors. They are also complex systems, with a multitude of design parameters. Indeed, today's NoCs consume a significant proportion of die area and system power [Bor07; Bor10]. The NoC of Intel's 80-Core Teraflops Research Chip, for example, consumes 17% of die area [Van+08] and 28% of system power [Hos+07]. Most utilized NoCs are buffered, packet switched networks and a very large part of the area and the energy is consumed by the networks' buffers. For instance, 60% die area of the iMesh NoC of Tilera's TILE64 many-core system is dedicated to buffering [Wen+07] and for the TRIPS chip, it is reported that the router's input buffers even occupy 75% of the router area [Gra+06]. At Intel's Teraflops Research Chip, buffer queues consume 22% of the communication power [Van+08]. Consequently, avoiding these buffers can contribute to reduce the power requirements of NoCs. The main focus of this thesis is on bufferless deflection routing, which has already been proposed in 1964 by Baran [Bar64]. At deflection routing, the data units of an NoC are not buffered by routers of the network, i.e. they can not wait at a router for resource allocation. Instead, if two data units try to allocate the same path, one data unit is routed along a potentially non-shortest path, i.e. it is deflected. Comparisons to buffered, packet switched NoCs demonstrated that bufferless, deflection routing based networks require over 30% less power and area [FCM10; CMM15].

Besides power efficiency, fault tolerance is an important aspect of many-core systems and, as such, also of NoCs. Generally, fault tolerance is achieved by adding some kind of redundancy. Most NoC topologies possess an inherent path redundancy. Thus, it suggests itself to exploit this redundancy to tolerate failures at the NoC level.

To summarize, power efficient, fault-tolerant NoCs are an important research topic and key components of future many-core systems. To quote Avinash Sodani, former Senior Principal Engineer at Intel and Chief Architect of Intel's Knights Landing Xeon-Phi processor:

“The developers of the ENIAC, [the earliest electronic general-purpose computer,] were faced with two challenges, and we are faced with the same

challenges today: reliability and interconnection.” [Sod16]

This thesis contributes to the development of reliable and power efficient NoCs by:

1. presenting a fault-tolerant NoC based on a permutation network and additionally on deflection routing. Moreover, a new concept to overcome complex fault situations is introduced.
2. investigating the effect of the link width on permutation network based router architectures. In addition, a new concept to reduce the routing overhead of a specific link width is presented.

1.2 Thesis Outline and Scientific Contributions

This thesis focuses on NoCs which utilize deflection routing and a permutation network based router architecture. An overview of NoCs, including main building blocks and terminology used in this context, is given in Chapter 2. The general principle of bufferless deflection routing is introduced in Chapter 3. Moreover, several existing implementations of deflection routing based NoCs and the basis router architecture used in this thesis are presented in Section 3.3 and Section 3.4, respectively. The main scientific contributions presented in this thesis are divided into two chapters.

Chapter 4 focuses on fault-tolerant and deflection routing based router architectures. This chapter starts with a brief introduction to fault tolerance at NoCs in Section 4.1. Existing fault-tolerant router architectures are presented in Section 4.2. Following this introduction, the FaFNoC router architecture, which has been developed as part of this thesis, is presented in Section 4.3. This architecture stands out for the utilized interconnection architecture and the method to tolerate complex fault situations. Most router architectures use either a crossbar or a 4×4 Banyan network as interconnection architecture. Instead, a 4×4 Benes network is utilized at the FaFNoC architecture, which requires fewer hardware resources than a crossbar and provides significant benefits in terms of fault tolerance compared to a Banyan network. Moreover, every fault-tolerant router architecture requires some kind of global fault information to be able to overcome complex fault situations, as they can arise if several components fail. Existing approaches are based either on costly routing-tables or fault information of adjacent nodes. These approaches suffer from poor scalability or limited fault tolerance. Thus, the concept of Fault-aware Flits (FaF) has been developed, at which the transferred data units are aware of their encountered fault situation.

Chapter 5 addresses the design of deflection routing based NoCs, and specifically, the utilized link width. As an introduction and motivation, the effect of the link width on buffered, packet switched NoCs and bufferless, deflection routing based NoCs is compared in Section 5.1. The optimal link width and its effect on performance and

hardware requirements is considered in Section 5.3. Subsequently, *TwoPhases*, an alternating transmission scheme for deflection routing is presented in Section 5.4. Compared to standard deflection routing, *TwoPhases* allows a reduction of the routing overhead by more than half.

Both chapters close with a brief summary and conclusion. Finally, concluding remarks and an outlook to future work are provided in Chapter 6.

This thesis is based on the following publications:

- [Run12a] Armin Runge. **Determination of the Optimum Degree of Redundancy for Fault-prone Many-Core Systems**. In: *Zuverlässigkeit und Entwurf - 6. GMM/GI/ITG-Fachtagung*. VDE VERLAG GmbH, 2012
- [Run12b] Armin Runge. **Reliability Enhancement of Fault-prone Many-core Systems Combining Spatial and Temporal Redundancy**. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. Institute of Electrical & Electronics Engineers (IEEE), June 2012. DOI: 10.1109/hpcc.2012.233
- [Run15a] Armin Runge. **FaFNoC: A Fault-tolerant and Bufferless Network-on-chip**. In: *Procedia Computer Science* 56 (2015), pp. 397–402. DOI: 10.1016/j.procs.2015.07.226
- [Run15b] Armin Runge. **Fault-tolerant Network-on-Chip based on Fault-aware Flits and Deflection Routing**. In: *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM. Association for Computing Machinery (ACM), 2015, p. 9. DOI: 10.1145/2786572.2786585
- [RK16a] Armin Runge and Reiner Kolla. **An Alternating Transmission Scheme for Deflection Routing Based Network-on-Chips**. In: *Architecture of Computing Systems – ARCS 2016*. Springer International Publishing. Springer Nature, 2016, pp. 48–59. DOI: 10.1007/978-3-319-30695-7_4
- [RK16b] Armin Runge and Reiner Kolla. **Consideration of the Flit Size for Deflection Routing based Network-on-Chips**. In: *1st International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems (AISTECS)*. Association for Computing Machinery (ACM), 2016, 5:1–5:6. DOI: 10.1145/2857058.2857060
- [RK16c] Armin Runge and Reiner Kolla. **TwoPhases: A Transmission Scheme to Reduce the Link Width at Deflection Routing based Network-on-Chips**. In: *Journal of Systems Architecture* (Dec. 2016). DOI: 10.1016/j.sysarc.2016.12.001

- [RK16d] Armin Runge and Reiner Kolla. **Using Benes Networks at Fault Tolerant and Deflection Routing based NoCs**. In: *Proceedings of the 10th International Symposium on Networks-on-Chip*. Institute of Electrical and Electronics Engineers (IEEE), Sept. 2016. DOI: 10.1109/NOCS.2016.7579325

Network Basics

Contents

2.1 Building Blocks and Terminology	8
2.2 Topology	11
2.3 Routing	14
2.3.1 Classification	14
2.3.2 Routing Algorithms and Turn Model	15
2.4 Buffering and Flow Control	18
2.4.1 Bufferless Flow Control	18
2.4.2 Buffered Flow Control	19
2.4.3 Buffer Availability	24
2.5 Router Architecture	24
2.6 Performance	27
2.6.1 Performance Metrics	27
2.6.2 Evaluation Methodology	29
2.7 Conclusion and Existing Interconnection Networks	34

This chapter aims to give an overview of NoCs and introduce typical evaluation criteria in this field. Towards this end, the main building blocks, as well as the terminology used in this context, are introduced in Section 2.1. Subsequently, three important aspects of NoCs are presented in more detail, namely topology in Section 2.2, routing in Section 2.3, and flow control in Section 2.4. An overview of a typical virtual channel router architecture is provided in Section 2.5. Network performance metrics and the herein used evaluation methodologies are presented in Section 2.6. Finally, this chapter closes with a conclusion in Section 2.7, which includes an overview of several interconnection networks of existing processors.

2.1 Building Blocks and Terminology

In general, NoCs provide a communication infrastructure for large systems, which consist of a huge number of communication participants or components. For such a high number of components, traditionally deployed communication infrastructures, e.g. buses and dedicated point-to-point links, do not scale or can not provide the required area, power, and timing requirements. An illustration of buses, point-to-point links, as well as an overview of NoCs, is depicted in Figure 2.1. This figure also shows the main building blocks of NoCs, namely *network interfaces* (NI), *routers* (*R*), and *links*.

Network interfaces² connect Intellectual Property (IP) cores, as processors, memories, or caches, to the network. They implement the conversion of messages, which are generated and processed by the IP cores, into packets, which can be routed by the network. Thus, network interfaces decouple computation from communication. A network interface consists of a front end and a back end. The front end is unaware of the NoC and usually adheres to a System-on-Chip (SoC) socket standard [BM06, p. 12], which is supported by the connected IP core. Sockets which already have been implemented in network interfaces include the Open Core Protocol (OCP) [Acc16], the Advanced eXtensible Interface (AXI) [ARM16], and the Virtual Component Interface (VCI) of the dissolved Virtual Socket Interface Alliance. The back end handles the network protocol, which involves, among others, end-to-end flow control handling as well as assembling and disassembling of packets.

The components of a network are connected by point-to-point links, which are composed of a set of wires. Depending on the definition, a link provides either a full-duplex or a simplex connection between the two components it connects. In this thesis, it is assumed that every link provides a simplex connection. However, components are connected by pairs of links. Thus, two links together enable a full-duplex communication between the two connected components.

One link can consist of several virtual or physical channels, which allows a parallel communication over this single link. Herein, every link consists of exactly one physical channel and potentially several virtual channels. More information about virtual channels is given in Section 2.4.2.

At most NoC topologies, the number of wires per link, also referred to as link width, is uniform throughout the NoC. However, network topologies with non-uniform link widths exist as well, e.g. the fat tree topology. Compared to off-chip networks, wires are cheap but the packet latency is critical. Therefore, links of NoCs are usually much wider than links of off-chip networks. Typical link widths of NoCs range from 16 bit to 512 bit.

The core elements of a NoC are routers. They consist of buffers, arbiters, and several Input/Output (I/O) ports. Routers receive packets from their input ports and forward them to their output ports. They implement a routing algorithm (cf. Section 2.3), as well

²Network interfaces are also referred to as network adapters [CML12, p. 16].

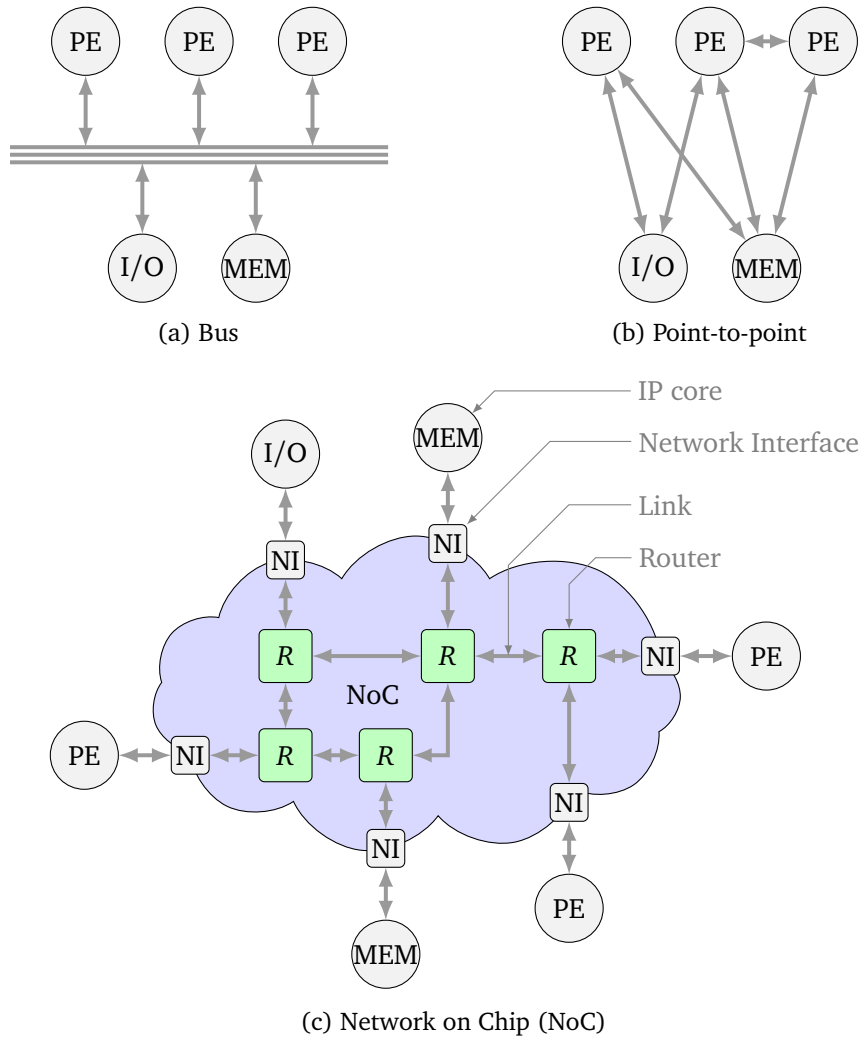


Figure 2.1: Three communication infrastructures used in System-on-Chips.

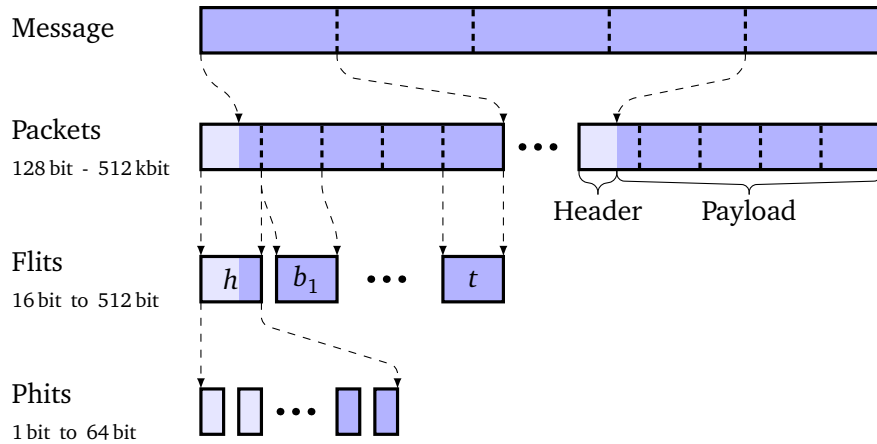


Figure 2.2: Data units at NoCs. The numbers underneath the left labels give ranges of typical sizes, according to [DT04, p. 225].

as a flow control scheme (cf. Section 2.4). Compared to off-chip networks, resources on a chip are constrained and routers share area and power budgets with other components. Consequently, NoC routers typically have limited buffering capacities. An overview of a typical router architecture is provided in Section 2.5.

Data units at NoCs

As mentioned before, the components of a NoC operate with different data units. An overview of the data units of NoCs is depicted in Figure 2.2. IP cores send and receive *messages*, e.g. cache lines, which can be arbitrarily long. The network interfaces create packets out of these messages. *Packets* usually have a restricted length, thus one message is divided into one or more packets. Additionally, a header is appended to each packet, which includes routing information. Packets are the basic unit of routing and sequencing, i.e. channels are allocated to packets. Depending on the switching technique, a packet may be further divided into flow control digits (flits). A flit is the basic unit of buffer and bandwidth allocation. The first flit of a packet is referred to as the *head flit* (herein denoted by h), which contains the packet's routing information. Depending on the packet's length, a head flit can be followed by several *body flits* (herein denoted by b_i , $i \in \mathbb{N}$) and at most one *tail flit* (herein denoted by t). As those flits carry no routing information, they have to follow the route of their head flit. Body and tail flits differ in that a tail flit deallocates the channel of the packet, which has been allocated by the head flit. In the case of a very short packet which consist of a single flit the head flit is also the tail flit. Figure 2.3 shows an example flit structure. Flits can be further divided into physical transfer digits (phits). A phit is the data unit that can be transmitted across a link in one clock cycle. Thus, if a flit is transferred as several phits, the flit is serialized

Head flit:	$ft=h$	dst	src	hc	id	pl
Body flit:	$ft=b$	pl				
Tail flit:	$ft=t$	pl				

Figure 2.3: Flit structure of head, body, and tail flits. All flits contain a flit type field ft to distinguish between head, body, and tail flits, as well as a payload field pl . Head flits additionally contain a destination and source address field, dst and src , a hop count field hc , and an ID field id .

and deserialized at each transmission. In most cases, a flit corresponds to a phit and hence the flit size equals the link width.

In the following three sections, three essential aspects of NoCs are considered in more detail.

2.2 Topology

The topology of a NoC defines the network's structure, i.e. it specifies the way how routers are connected to each other. Therefore, the deployed topology affects many other aspects, as routing, achievable performance, reliability, and ease of layout. Topologies can be classified in different ways, and most of them are orthogonal to each other.

Directness: At direct topologies, the communication participants, referred to as components, sit inside the network. Every component is connected to a router and vice versa. The combination of a router and its connected component is referred to as a node. Thus, such networks are also known as router-based networks. At indirect topologies, the components sit outside the network. There, the components are connected by switches or switching elements, whereby some switches just forward messages, but they are not connected to any sender or receiver.

This section concentrates on direct topologies, which are most frequently utilized at NoCs. An overview of several direct topologies is depicted in Figure 2.4. The interconnection architecture inside a router architecture itself is usually based on an indirect topology. Thus, several indirect topologies are introduced in Section 2.5.

Regularity: A regular topology is defined in terms of a regular graph structure, such as a ring or a mesh. In some textbooks, a topology is defined to be regular if all nodes are identical in terms of their number of ports, e.g. in [DYL02, p. 14] and [QS13, p. 335]. Regular topologies are easier to analyze, whereas irregular topologies can be customized for a specific application. Already utilized irregular

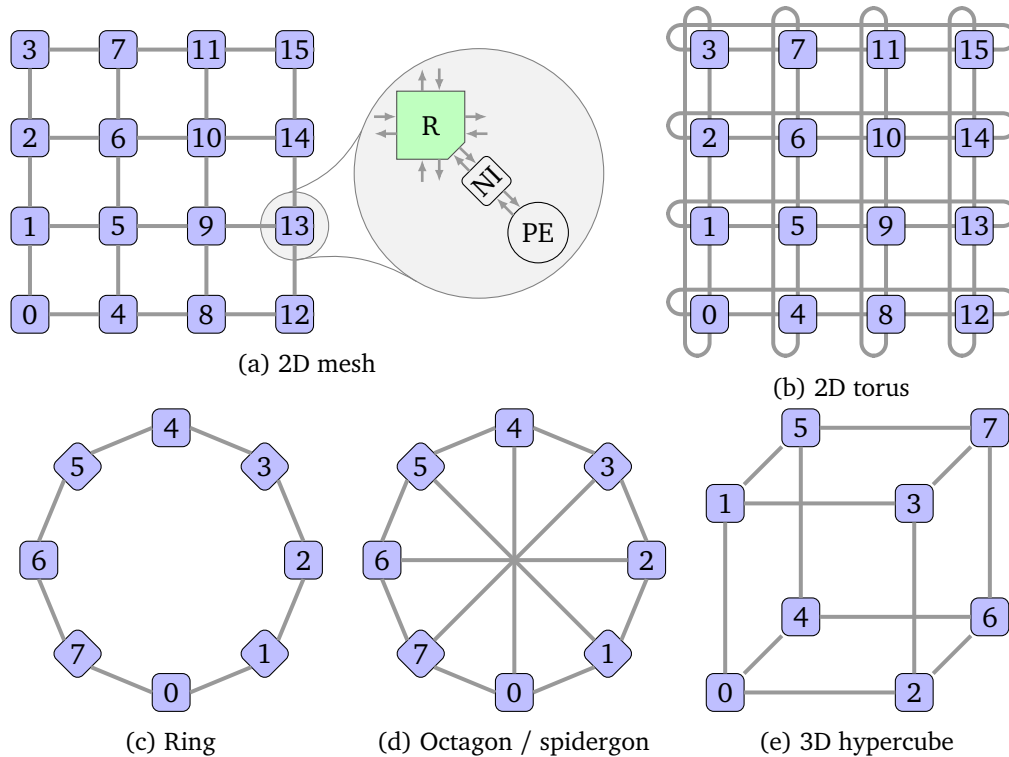


Figure 2.4: Frequently used direct NoC topologies.

topologies are frequently composed of two regular topologies, e.g. a mesh of rings. Moreover, an irregular topology can arise if parts of the network fail.

Symmetry: At symmetric topologies, the network looks alike from every node.

Additionally to these fundamental attributes of topologies, they can also be compared by several theoretical metrics. Some important metrics are listed below.

Number of links: The number of links is usually given as a function of N , whereas N denotes the number of nodes or routers in the network. Links provide bandwidth and hence, a higher number of links can increase the throughput and decrease the latency. On the other hand, a higher number of links increases the costs and can complicate the layout on-chip.

Radix: The radix denotes a router's degree, i.e. its number of I/O ports. The costs of a router increase with its radix, as the number of buffers and arbiters, as well as the size of the router's interconnection architecture, increase with the radix.

Diameter: The diameter of a topology gives the maximum distance between two nodes. Hence, the diameter is a lower bound of the maximum hop count³.

Average distance: The average distance between two nodes influences the average hop count.

Bisection bandwidth: The bisection bandwidth gives the number of links which have to break before the network is cut into two halves. At a higher bisection bandwidth, the blocking probability inside the network is lower.

Some popular and frequently utilized NoC topologies are briefly introduced below and pictured in Figure 2.4. The *ring* topology (cf. Figure 2.4c) requires only as many links as nodes are present in the network. Due to the small number of links, the average latency is $\mathcal{O}(N)$ and the bisection bandwidth is constant. Thus, rings do not scale with a high number of nodes. However, hierarchical designs, as well as multi-ring designs, have been used to address the scalability of rings. The *octagon* topology (cf. Figure 2.4d) is another low-cost topology. It is based on a ring with eight nodes and additional links from each node to its diagonally opposite neighbor node. The *spidergon* topology is a generalization to an arbitrary but even number of nodes, which has been developed by STMicroelectronics [Cop+08]. The *2D mesh* topology (cf. Figure 2.4a) is frequently utilized in commercial NoCs, e.g. in Tiler’s Tile Processor family [Wen+07], as well as assumed in most scientific NoCs. It is a regular topology, easy to layout on-chip, and has links of equal length. Furthermore, it maps to the tiled CMP architecture naturally [BD06]. The path diversity and low average latency of $\mathcal{O}(\sqrt{N})$ are additional advantages of this topology. Meshes are not symmetric on their edges, and thus the center of a mesh tends to be more congested. As a result, the performance of a node depends on its position in the network. The *2D torus* topology (cf. Figure 2.4b) corresponds to a 2D mesh topology with additional connections at the edge nodes, which leads to a donut shape. Thus, a torus provides a higher path diversity and higher bisection bandwidth than a mesh topology. On the other hand, a torus has higher costs, is harder to layout on-chip, and has unequal link lengths. However, the latter point can be solved by a folded torus, at which all links have the same length. The *n-dimensional hypercube* topology connects $2^n = N$ nodes and provides a latency of $\mathcal{O}(\text{ld } N)$. There, two processors are connected if and only if the binary representation of their labels differ at exactly one bit. Hypercubes were used in the past, e.g. in the Cosmic Cube [Sei85], as links were more expensive than routers. Values for the presented metrics and topologies are shown in Table 2.1.

³The hop count of a packet denotes the number of nodes the packet has traversed.

	Ring	Spidergon ^a	2D $n \times n$ mesh	2D $n \times n$ torus	n dim. hypercube
Number of nodes	N	$N = 4n$	$N = n^2$	$N = n^2$	$N = 2^n$
Number of links	N	$1.5N$	$2N - 2n$	$2N$	$2^{n-1}n$
Router degree / radix	2	3	4	4	$\text{ld}(N)$
Diameter	$\frac{N}{2}$	$\frac{N}{4}$	$2\sqrt{N}$	\sqrt{N}	$\text{ld}(N)$
Avg. distance	$\frac{N}{4}$	$\frac{2n^2+2n-1}{N}$	$\frac{2}{3}\sqrt{N}$	$\frac{1}{2}\sqrt{N}$	$\frac{\text{ld}(N)}{2}$
Bisection bandwidth	2	8	\sqrt{N}	$2\sqrt{N}$	$\frac{N}{2}$

^aThe depicted values correspond to $N = 4n$. Average distance and bisection bandwidth for $N = 4n + 2$ differ slightly from the depicted values. Exact values can be found in [Cop+08, p. 102].

Table 2.1: Metrics for different topologies. Values are given as a function of N , whereas N denotes the number of nodes in the network.

2.3 Routing

At almost all topologies, several paths between two nodes are available. The routing algorithm specifies how packets get from their source node to their destination node, i.e. which path is taken. Therefore, the utilized routing algorithm is a fundamental characteristic of a NoC, which affects the network's performance and correctness.

2.3.1 Classification

As topologies, routing algorithms can be classified in different ways and most of them are orthogonal to each other.

Mechanism: A routing mechanism which allows very simple routers, at the expense of larger headers, is *source based* routing. There, a source node specifies a packet's complete route throughout the network, which is stored in the packet's header.

At *lookup table based* routing, it is the other way round. A routing table is used to determine the output port for a given destination. This leads to larger routers but allows small packet headers.

Arithmetic based routing algorithms exploit the regularity of most NoC topologies. A simple arithmetic is used to determine the route throughout the network, e.g. approach the destination in x direction first.

Type: If always the same path between a source node and a destination node is chosen, a routing algorithm is referred to as *deterministic routing algorithm*. Such algorithms

are usually very simple and deadlock free, i.e. cycles at resource allocation are avoided. On the other hand, deterministic algorithms do not exploit the path diversity efficiently, which could lead to contention.

The goal of an *oblivious routing algorithm* is to balance the network load. However, the network state is not considered towards this end. An example is Valiant's algorithm [Val82], which randomly chooses an intermediate destination node, to which a packet is routed first. Thus, Valiant's algorithm routes packets along non-minimal paths to balance the network load.

If the algorithm adapts to the state of the network, it is referred to as an *adaptive routing algorithm*. Adaptive routing algorithms can be further classified as minimal or non-minimal algorithms⁴. *Minimal adaptive algorithms* use the network state, e.g. the capacity or utilization of downstream buffers, to select one productive output port⁵. *Non-minimal adaptive algorithms* can additionally misroute packets to non-productive output ports. The utilization of non-productive output ports can improve load balancing and network utilization. However, this also necessitates precautions against livelocks, as otherwise non-productive output ports could be selected infinitely. For instance, deflection routing (cf. Chapter 3) belongs to the group of non-minimal adaptive algorithms.

Deadlock avoidance: In general, deadlocks are caused by a circular dependency on resources. In particular, wormhole flow control (cf. Section 2.4) is susceptible to routing dependent deadlocks, at which a set of packets obstruct each other. Every packet of the set waits for a buffer or a link which is already occupied by another packet of the set. Figure 2.5 shows such a deadlock situation. One solution to avoid circular dependencies is to restrict the turns a packet can take. Glass and Ni developed the well-known turn model [GN92], which is briefly introduced below. Another solution to avoid routing dependent deadlocks is to use Virtual Channels (VCs). Finally, it is also possible to detect and break deadlocks, however, this requires buffer preemption.

2.3.2 Routing Algorithms and Turn Model

One of the most frequently deployed NoC routing algorithms, which is very simple, deterministic, minimal, and deadlock⁶ free, is *dimension order routing*. There, packets are routed to their destination address along one dimension before they are routed along

⁴More precisely, both deterministic and oblivious algorithms can be minimal as well as non-minimal. However, all herein considered deterministic routing algorithms are minimal, and all oblivious algorithms are non-minimal.

⁵An output port or direction is referred to as *productive* for a packet if it brings the packet one hop closer to its destination node.

⁶If not stated otherwise, the term deadlock hereinafter refers to a routing dependent deadlock.

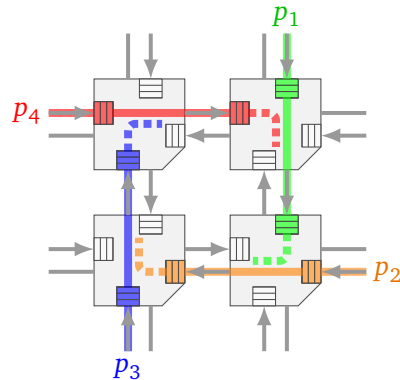


Figure 2.5: Routing dependent deadlock at a 2D mesh with input buffered routers. Packet p_1 occupies right link (upper right router to lower right router) and needs bottom link. Packet p_2 occupies bottom link and needs left link. Packet p_3 occupies left link and needs top link. Packet p_4 occupies top link and needs right link.

the next dimension. At a 2D mesh topology, packets are either routed along the x or the y direction before they are routed along the remaining direction. Depending on the direction which is chosen first, such a routing algorithm is referred to as X -first routing or Y -first routing⁷. Dimension order routing is deadlock free as the turns which a packet can take are restricted in a way that cyclic dependencies can not occur. Figure 2.6a shows all eight possible turns for a 2D mesh topology. These eight turns create two cycles, one in clockwise direction and one in counterclockwise direction. As shown in Figure 2.6b and Figure 2.6c, at both X -first routing and Y -first routing only four out of the eight possible turns are allowed, and thereby, both cycles are broken.

Restricting the number of turns is a simple and cost-effective solution to guarantee the abstinence of deadlocks. However, prohibiting more turns than necessary restricts the potential adaptiveness of a routing algorithm. Glass and Ni presented in [GN92] the well-known *turn model*. This model is herein explained for a 2D mesh, but the general concept can be extended to higher dimensional meshes as well as other topologies. Glass and Ni showed that a deadlock free routing algorithm for a 2D mesh can be created if exactly one turn in each of both cycles is prohibited. Furthermore, they found that only 12 ways, of the 16 possible ways to prohibit one turn in each cycle, prevent deadlocks. If symmetries are taken into account, three unique ways exist, each creating a deadlock free routing algorithm, as well as one way at which deadlocks can occur. The algorithms presented by Glass and Ni have in common that the north to west turn of the counterclockwise cycle is eliminated, as well as one turn in the clockwise cycle. However, any other turn instead of the north to west would be equally possible. The

⁷They are also referred to as XY / YX routing as well as XY / YX dimension order routing.

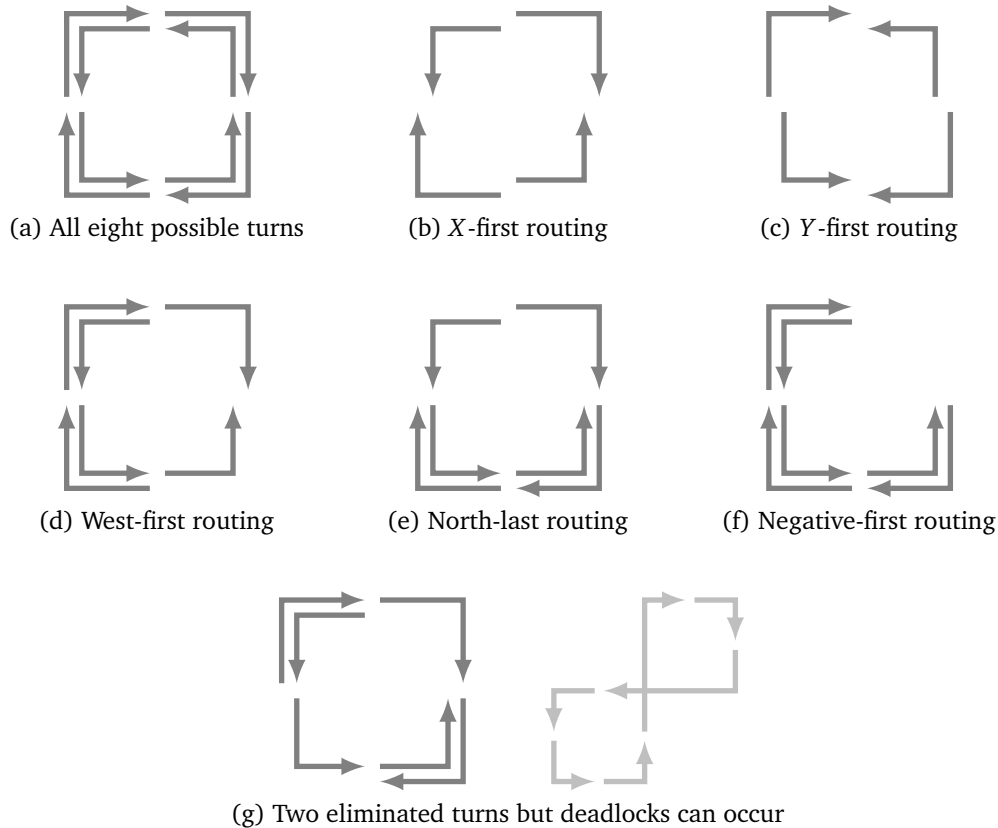


Figure 2.6: The turn model for a 2D mesh topology. Figure 2.6a shows all possible turns of a 2D mesh, Figures 2.6b to 2.6f show the allowed turns of the associated routing algorithms, and Figure 2.6g shows that not every combination of six turns results in a deadlock free routing algorithm.

first algorithm of the turn model is *west-first routing* (cf. Figure 2.6d). There, the south to west turn is prohibited in addition to the north to west turn. Hence, every packet has to be routed to the west direction first, as it can not turn to the west after it has turned to any other direction. The second algorithm is *north-last routing* (cf. Figure 2.6e), at which the north to east turn is prohibited in addition to the north to west turn. As a result, packets can not change their direction after heading to the north, and every packet has to be routed to any other direction before it is routed to the north. The third algorithm is *negative-first routing* (cf. Figure 2.6f). There, the east to south turn is prohibited in addition to the north to west turn. At negative-first routing, packets must be routed to the negative directions (the south and the west) first before they can turn to the positive directions (the north and the east). The fourth turn which can be eliminated in the clockwise cycle is the west to north turn (cf. Figure 2.6g). However, as

depicted on the right of Figure 2.6g, eliminating this turn does not result in a deadlock free routing algorithm.

Chiu presented the odd-even turn model in [Chi00], which is an extension of Glass and Ni’s turn-model. Depending on a router’s position, different turns are prohibited to break the two cycles. In odd columns, the south to west turn and the north to west turn are prohibited. In even columns, the east to south turn and the east to north turn are not allowed. Compared to the original turn-model, the odd-even turn model provides a more even adaptiveness for different source-destination pairs.

2.4 Buffering and Flow Control

Besides the topology, which specifies the physical interconnection structure, and the routing algorithm, which specifies the set of paths a packet can follow, buffering and flow control are two further important aspects of NoCs. In general, buffering and flow control describes where data is stored within the network, how resources are allocated to packets traversing the network, and what happens if resources can not be allocated. Thus, flow control can be regarded as a problem of resource allocation as well as contention resolution [DT04, p. 221]. The resources to allocate include channel bandwidth, buffer capacity, and control state. A router’s control state tracks the resource allocation within the router itself. A classification of NoC flow control mechanisms, presented by Dally and Towles in [DT04], is shown in Figure 2.7.

2.4.1 Bufferless Flow Control

Bufferless flow control is the simplest form of flow control. As no buffers exist, packets only have to allocate channel bandwidth and control state, but no buffer capacity. Consequently, packets can not be buffered in routers, i.e. they can not wait if an allocation

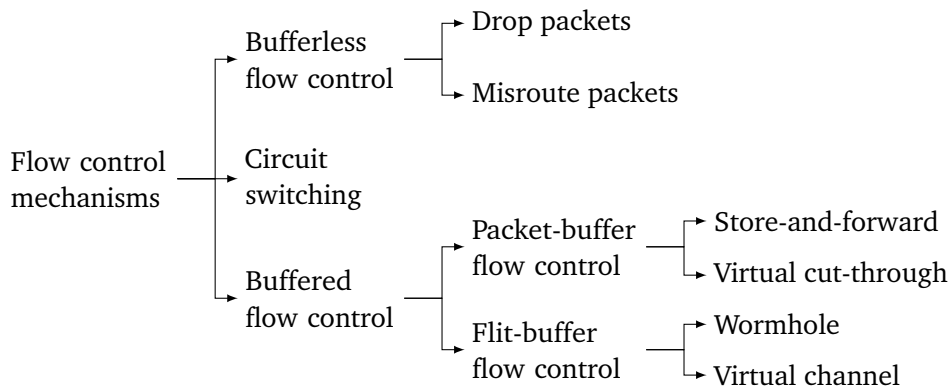


Figure 2.7: Classification of flow control mechanisms according to [DT04].

fails. There are two options to deal with a packet whose attempt to allocate its required resources failed. Such a packet can either be *dropped* or *misrouted*. In the case of packet dropping, packets which are dropped later waste bandwidth. Furthermore, a retransmission scheme is required. Misrouting does not drop packets, but sends them to non-productive directions instead. Thus, misrouting is susceptible to livelocks and a guarantee that every packet gets delivered eventually is needed. Misrouting is also referred to as *deflection routing* or *hot-potato routing*, and has been proposed by Baran in [Bar64]. Deflection routing is introduced in more detail in Chapter 3.

Circuit switching is considered as a form of bufferless flow control in several textbooks, yet the packets' headers are buffered while they allocate channels. This flow control scheme consists of four phases. In the first phase, a request, i.e. a packet's head flit, propagates from its source router to its destination router and allocates the channels it traverses along the taken route. During this phase, the head flit might be buffered in routers if the next channel allocation fails. In the second phase, an acknowledge is transmitted back to the source router, signaling that the circuit is established. In the third phase, the channels are reserved and an arbitrary number of data packets can be transmitted without further control. Finally, in the fourth phase, the circuit is deallocated by a tail flit. Circuit switching has its origins in the analog telephone network. If primarily short messages are transferred, the latency with circuit switching can be significant, as the path between the source router and the destination router has to be traversed three times to deliver a packet. Thus, circuit switching is more suitable for large messages and steady traffic flows than for short packets.

2.4.2 Buffered Flow Control

At *buffered flow control*, packets are buffered within the network while they allocate the required resources in order to advance. Thus, buffered flow control schemes decouple the allocation of adjacent channels in time. Buffers are either allocated in units of packets, in case of *packet-buffer flow control*, or in units of flits, in case of *flit-buffer flow control*. At *store-and-forward* flow control, every router along a packet's path waits until the complete packet has been received (stored) and then forwards the packet to the next router. Thus, a packet-sized buffer, as well as exclusive access to the channel, has to be allocated to a packet, before this packet can make forward progress. One drawback of store-and-forward flow control is the high average latency. As every packet has to be received at one router completely before it can be forwarded to the next router, a serialization latency occurs at each hop. Figure 2.8 shows a time-space diagram of the transmission of a packet, which consists of four flits, from router R_0 to router R_2 with store-and-forward flow control. During clock cycles c_0 to c_3 , router R_0 receives the four flits of the packet. Only after the complete packet has been received, and the required resources at the next router are successfully allocated, the packet can be forwarded to the next router. In the depicted example, a successful allocation is assumed and the

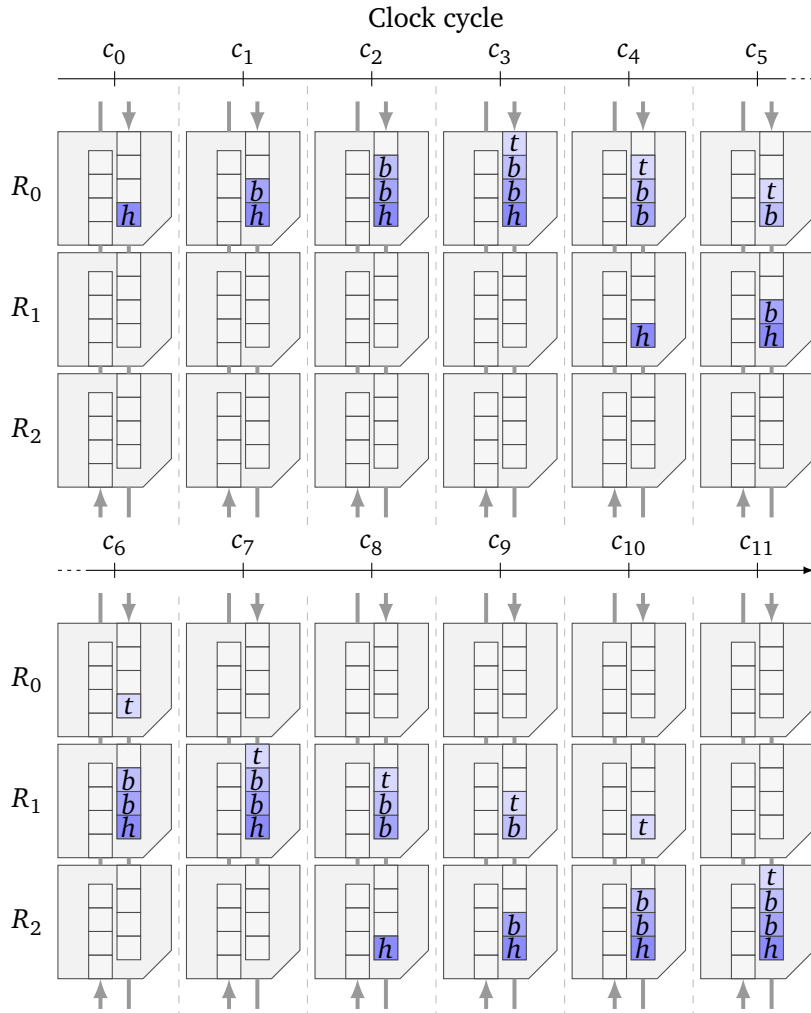


Figure 2.8: Time-space diagram of the transmission of a packet, which consists of four flits, from router R_0 to R_2 with **store-and-forward** flow control. For reasons of clarity, the links, ports, and buffers are depicted only for the north and the south.

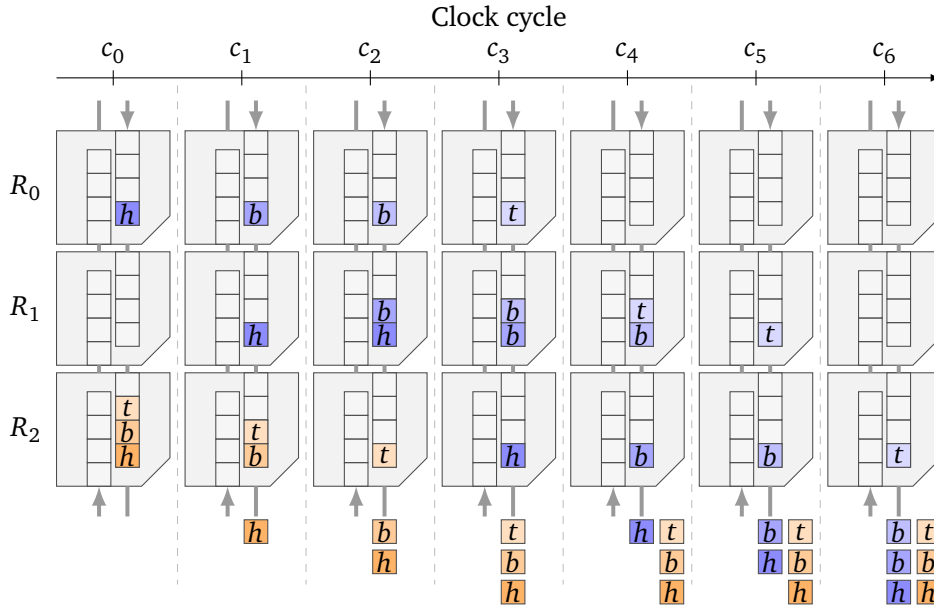


Figure 2.9: Time-space diagram of the transmission of a blue-colored packet, which consists of four flits, from router R_0 to the south of R_2 with **virtual cut-through** flow control. Additionally, an orange-colored packet is en route to the south of R_2 .

packet is forwarded from R_0 to R_1 during clock cycles c_4 to c_7 . Finally, the packet is transmitted from R_1 to R_2 during clock cycles c_8 to c_{11} . Consequently, it takes 12 clock cycles to transmit the complete packet to router R_2 .

At *virtual cut-through* flow control, routers start to forward packets as soon as their header has been received and the required resources are allocated, without waiting for the entire packet to be received. Buffer and channel bandwidth are allocated in units of packets, as the case at store-and-forward. Thus, packet-sized buffers are still required, but the average latency is significantly reduced compared to store-and-forward flow control. Without taking contention into account, the latency consists of a single serialization latency as well as a constant number of cycles per node. A time-space diagram of the packet transmission with virtual cut-through flow control is depicted in Figure 2.9. In this example, it is assumed that another (orange-colored) packet is in transit and buffered at R_2 , while the new (blue-colored) packet arrives at R_0 in clock cycle c_0 . In clock cycle c_1 , the orange and the blue head flit are forwarded, i.e. it is assumed that they successfully allocated their resources. In clock cycle c_2 , the orange body flit is forwarded, but the blue head flit is buffered in R_1 as the north input buffer of R_2 is still allocated to the orange packet. In clock cycle c_3 and the subsequent clock cycles, the blue packet can be forwarded to the south of R_2 .

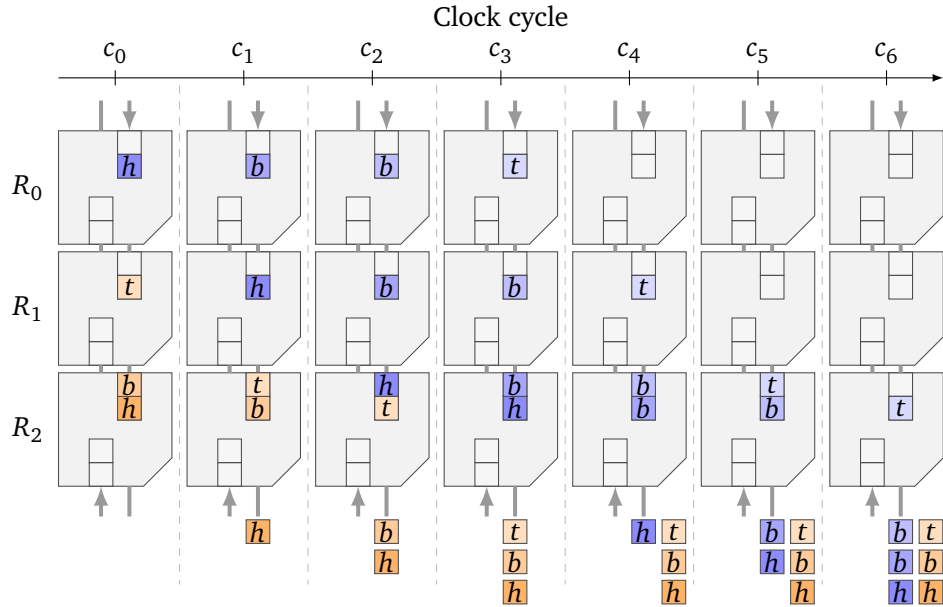


Figure 2.10: Time-space diagram of the transmission of a blue-colored packet, which consists of four flits, from router R_0 to the south of R_2 with **wormhole** flow control. Additionally, an orange-colored packet is en route to the south of R_2 .

In summary, virtual cut-through overcomes the latency problem of store-and-forward, but still requires packet-sized buffers. The reason for this is that the flow control unit of packet-buffer flow control schemes is the entire packet. Thus, there are two conflicting objectives. Packets should be large, to minimize the routing and sequencing overhead, i.e. the ratio of this overhead to the transmitted payload. On the other hand, large packets involve large buffering requirements at every router. Thus, small packets are preferable to enable fine-grained resource allocation and to minimize blocking latency.

Flit-buffer flow control schemes overcome this drawback of packet-buffer flow control. There, channel bandwidth and buffer space are allocated on the granularity of flits rather than packets. Consequently, the buffering requirements are a magnitude of order lower than for packet-buffer flow control schemes. *Wormhole* flow control operates very similar to virtual cut-through flow control, but channel bandwidth and buffer space are allocated on a flit-by-flit basis. Every flit allocates just one flit buffer and one flit of channel bandwidth, which yields in a more efficient buffer utilization. Additionally, if a head flit arrives at a router, it allocates a control state for the entire packet. Thus, the remaining flits of a packet, which follow their head flit, use the already allocated control states. An allocated control state is released by a packet's tail flit. Figure 2.10 shows a time-space diagram of the packet transmission with wormhole flow control.

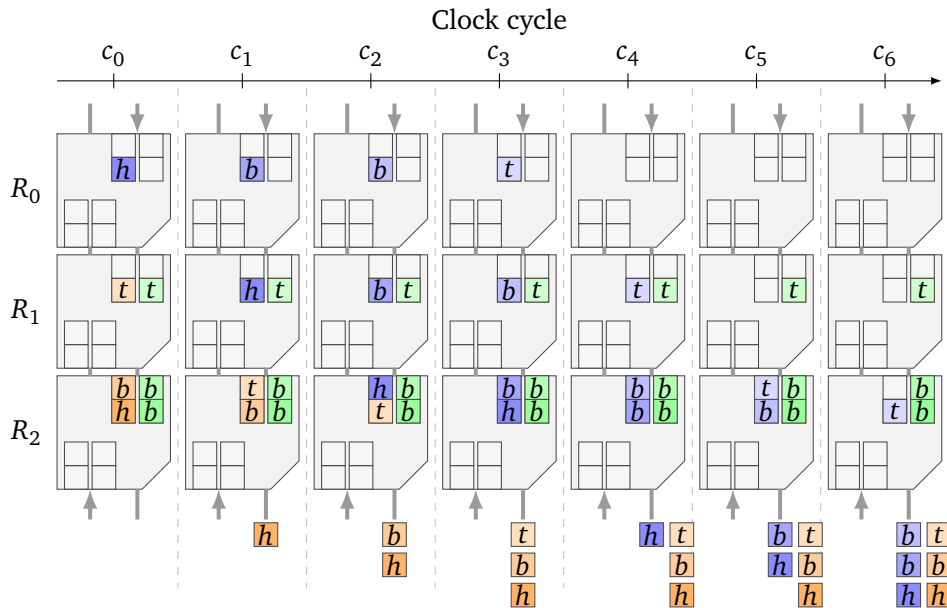


Figure 2.11: Time-space diagram of the transmission of a blue-colored packet, which consists of four flits, from router R_0 to the south of R_2 with **virtual channel** flow control. Additionally, an orange-colored packet is en route to the south of R_2 , and a green packet, which is blocked, is buffered in R_1 and R_2 .

There, every flit can advance as soon as one flit buffer is available at the input port of the next router.

The reduced buffering requirements of flit-buffer flow control come at the expense of a reduced throughput. The reason for this is that buffers are allocated on a flit-by-flit basis, but channels (and the corresponding control states) are allocated to packets. If parts of a packet can not advance due to a failed buffer allocation, a channel can become idle. Other packets can not use this idle channel, even if they need a different output port, as the channel is allocated to the blocked packet. Thus, the buffers operate according to a first in - first out (fifo) method. An analogy to road traffic is a road without left-turn lane, at which left-turning vehicles can block other vehicles behind them. At wormhole flow control, this limitation is referred to as *head of line blocking*.

Virtual channel flow control overcomes the blocking problem of wormhole flow control by allowing to share the available channel bandwidth among multiple packets. To this end, several virtual channels are associated with every physical channel and each virtual channel includes flit buffers and a channel state. The general principle of virtual channel flow control is illustrated in the time-space diagram of Figure 2.11. In this depicted example, it is assumed that two virtual channels per physical channel exist. Furthermore, a third (green-colored) packet exists, which is buffered in the right virtual channels

of R_2 's and R_1 's north input port. It is assumed that this green packet, or rather the depicted flits of the green packet, can not advance for the entire depicted time period. Without virtual channels, the physical channel between R_1 and R_2 would become idle and no other packet could use this channel. In particular, the blue and orange packets would not be able to advance. However, as two virtual channels per physical channel exist, the blue and orange packets can advance and use the available channel bandwidth.

2.4.3 Buffer Availability

Buffered flow control techniques need a method to communicate buffer availability between adjacent routers. In general, an upstream router has to know if enough buffer space is available at a downstream router to store the next flit or packet. As the main focus of this thesis is on bufferless deflection routing, only a subset of these methods is very briefly introduced herein.

At *credit-based* flow control, the upstream router counts the number of free flit buffers in each downstream router. If the upstream router sends a flit to the downstream router, the corresponding counter is decremented. When the downstream router forwards a flit to the next router, it sends a credit back to the upstream router, to signal that the buffer is available again.

At *ON/OFF* flow control, the downstream router signals the upstream router if free buffer space is available (ON) or not available (OFF). Thus, the upstream signal is a single control bit, which is changed only when the available buffer space exceeds or falls below certain thresholds.

At *ACK/NACK* flow control, upstream routers send downstream optimistically. If the downstream router can accept the flit or packet, it sends an ACKnowledgement (ACK) to the upstream router. Otherwise, the downstream router drops the flit or packet and sends a Negative-ACKnowledgement (NACK). Thus, the upstream router has to keep the data in its own buffer until it receives an ACK.

2.5 Router Architecture

In general, routers are capable of receiving packets at their input ports, determining a destination per received packet, and forwarding the packets to an appropriate output port. A block diagram of a typical input queued, virtual channel router is depicted in Figure 2.12. The blocks can be partitioned into a datapath and a control plane. The data path includes several input ports and an interconnection architecture, e.g. a crossbar. The control plane includes the remaining blocks, i.e. a route computation block, a virtual channel allocator, and a switch allocator.

Most modern routers are pipelined and packets proceed through four stages: (1) route computation, (2) virtual channel allocation, (3) switch allocation, and (4) switch

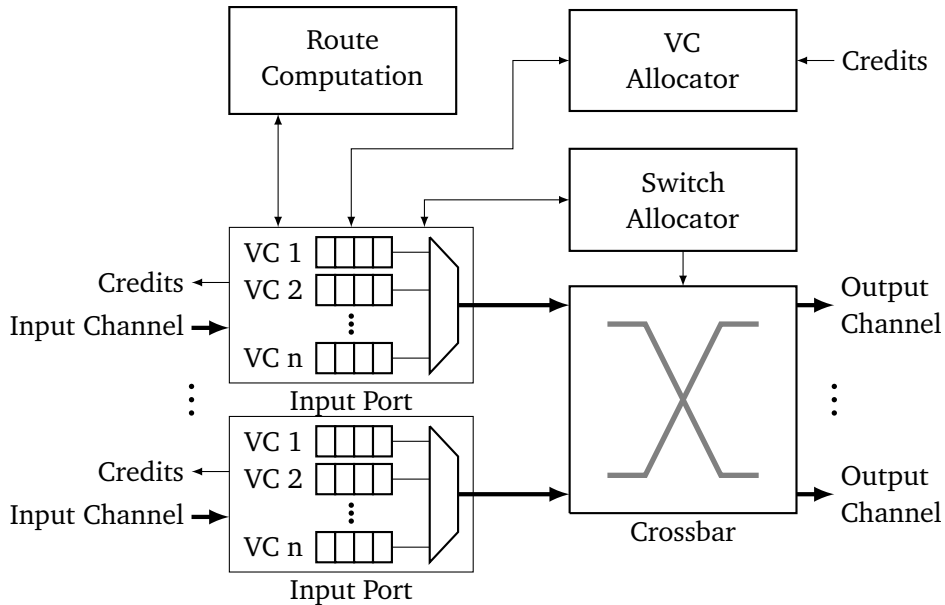


Figure 2.12: Block diagram of a typical input queued, virtual channel router, which uses credit-based flow control, according to [MWM04].

traversal. Route computation and virtual channel allocation has to be performed only once per packet. Thus, only head flits proceed through these two stages. At route computation, one or a set of output virtual channels is determined, to which the packet can be routed. At virtual channel allocation, an attempt is made to allocate one of the virtual channels, which has been determined by the route computation. If buffer space is available at the assigned virtual channel, an input virtual channel can request the necessary output channel from the switch allocator. The switch allocator also generates the required crossbar control signals. At switch traversal, flits that have been granted to traverse the crossbar are forwarded to their appropriate output channel.

Within a router, switching, i.e. establishing a connection between the router's input and output channels, is performed by an interconnection architecture. Most frequently, crossbars are deployed at NoC routers. However, some router architectures do not utilize a crossbar. In particular, the CHIPPER router architecture [FCM10; FCM11], that served as a basis of this thesis, uses a multistage logarithmic network instead of a crossbar. This architecture is introduced in more detail in Chapter 3.

As the utilized interconnection architecture has a huge influence on the performance of a router architecture, a brief overview and comparison of interconnection architectures is given in this section. Usually, the interconnection architecture of a router is based on an indirect topology, at which the components sit outside the network. The components, which are placed on the edges of a network, are connected by a series of switches

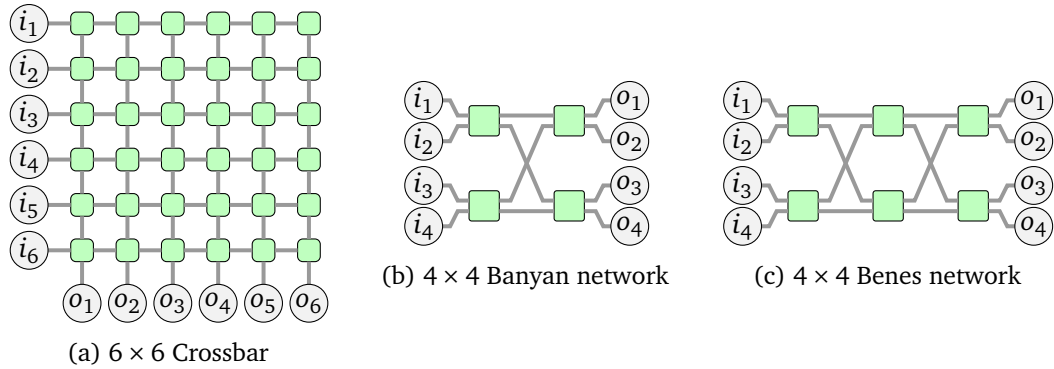


Figure 2.13: Three indirect network topologies: 6×6 crossbar, 4×4 Banyan network, and 4×4 Benes network.

or switching elements at the interior nodes. Data is transmitted from the sending components, through one or more switches, to the receiving components. Figure 2.13 shows three indirect network topologies, a 6×6 crossbar and two 4×4 multistage logarithmic networks, a Banyan network and a Benes network. An $N \times N$ crossbar consists of N^2 switches to connect the N inputs to the N outputs. Here, inputs and outputs are denoted by i_1, \dots, i_N and o_1, \dots, o_N , respectively. Crossbars are non-blocking, i.e. the connection of any permutation of inputs and outputs is supported, which enables a high throughput. Furthermore, crossbars feature a constant and very low latency of $\mathcal{O}(1)$. On the other hand, crossbars are expensive, scale poorly, and are difficult to arbitrate. However, as only a small number of inputs and outputs has to be connected within a router architecture, the desirable characteristics, as low latency and high throughput, dominate. Thus, crossbars are nevertheless frequently deployed as interconnection architectures within router architectures.

Another group of indirect topologies, which has also already been used as interconnection architectures within NoC routers, are multistage logarithmic networks, also referred to as permutation networks. Several variations of multistage logarithmic networks exist, as Omega networks, Butterfly networks, Banyan networks (cf. Figure 2.13b), and Benes networks (cf. Figure 2.13c). These networks are constructed using 2×2 switching elements, which are able to swap both inputs, i.e. a packet from the first input is forwarded to the second output, or pass both inputs to their corresponding outputs. In general, cost and latency of a multistage logarithmic network are $\mathcal{O}(N \log(N))$ and $\mathcal{O}(\log(N))$, respectively. If the switching elements use only local information, i.e. their address, and information of the received packets, the network is referred to as a self-routing network.

An $N \times N$ Banyan network consists of $\log(N)$ stages, each with $\frac{N}{2}$ switching elements. They provide exactly one path from any input to any output. However, not every permutation between all inputs and outputs is supported by a Banyan network. For

instance, at a 4×4 network as depicted in Figure 2.13b, input i_2 can only be connected to o_3 or o_4 if i_1 is already connected to o_1 or o_2 . Thus, Banyan networks are internal blocking. Banyan networks are explained in more detail in Chapter 3.

Benes networks, also referred to as dual-Banyan networks, consist of $2 \cdot \text{ld}(N) - 1$ stages, each with $\frac{N}{2}$ switching elements. Due to the increased number of switching elements, several paths from any input to any output exist at Benes networks. Furthermore, they are rearrangeable non-blocking, which means that the network can be rearranged once the permutation is known. The increased path diversity provides benefits in terms of fault-tolerance. Benes networks are explained in more detail in Chapter 4.

2.6 Performance

The performance of NoCs is usually described by performance metrics. Some metrics are primarily topology dependent, and in particular, traffic independent, as bisection bandwidth and maximum throughput. The focus of this section is on performance metrics which are influenced by the utilized routing algorithm, flow control mechanism, and in particular, by the traffic conditions. In Section 2.6.1, several performance metrics are introduced. In Section 2.6.2, the herein used general evaluation methodology is presented. This includes an introduction of synthetic traffic models as well as the used approach of application performance evaluation.

2.6.1 Performance Metrics

Two important performance metrics of NoCs are latency and throughput. They depend on NoC characteristics, as topology, routing, flow control, and router design. Additionally, both values are a function of the offered traffic. The traffic offered to a network, either by a single node or by all nodes of the network, during a certain period of time is given by the injection probability α . Herein, the injection probability is expressed as the percentage of the theoretically maximum possible number of injections, at which every node injects a packet into the network every clock cycle.

The traffic injected into the network is to be contrasted with the traffic ejected from the network. The rate at which packets are ejected, i.e. delivered by the network to the nodes, during a time interval is referred to as throughput and denoted by θ . In this thesis, the throughput is expressed as the number of ejected packets per node per clock cycle. Typically, throughput is plotted as a function of the offered traffic, i.e. the injection probability (cf. Figure 2.14a). If the network is not saturated, the number of ejected packets equals the number of injected packets, and the plot equals a straight line. If the offered traffic increases beyond the saturation point θ_s , the network is not able to deliver the packets as fast as they are created. This means, once the network is saturated, some packets can not be injected into the network directly. In this thesis, it is

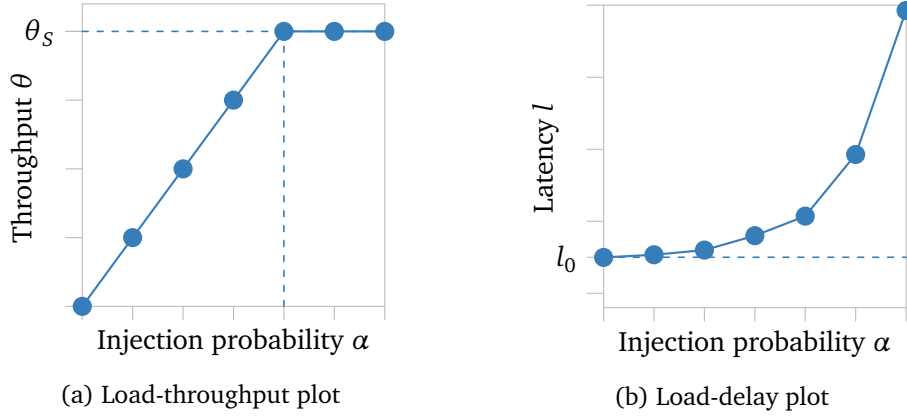


Figure 2.14: Typical NoC performance metrics illustrated.

assumed that every node includes an injection queue. If a router is not able to inject a packet, i.e. there is no idle output link and no empty buffer, the packet is buffered by the node's injection queue and injected as soon as possible.

Another important performance metric is the packet latency, denoted by l , which is the time a packet requires to traverse the network. If latency due to contention with other packets is ignored, this latency is also referred to as zero-load latency and denoted by l_0 . The zero-load latency consists of two components, the head latency and the serialization latency. The head latency is the time the packet's head flit requires to traverse the network. It is determined by the number of hops between the packet's source and destination node, as well as the time required to traverse a router and the corresponding link. If there is no contention, the number of hops between a packet's source and destination node correspond to the Manhattan distance between these two nodes, which is denoted by d_1 . The serialization latency is the time required for the tail flit to catch up with the head flit. This latency is defined by the quotient of the packet size $|p|$ and the channel bandwidth bw . Thus, the zero-load latency is given by:

$$\begin{aligned}
 l_0 &= l_{head} + l_{serialization} \\
 &= d_1 \cdot l_{router+link\ traversal} + \frac{|p|}{bw}
 \end{aligned} \tag{2.1}$$

The zero-load latency represents a lower bound on the packet latency. As throughput, latency can be plotted as a function of the offered traffic, also referred to as load-delay plot (cf. Figure 2.14b). Once the offered traffic increases, an increased contention causes that the average packet latency also increases, as packets must wait for resources. More precisely, the head latency increases, as the time for router and link traversal rises. If a non-minimal routing algorithm is used, the number of taken hops from a packet's source to destination does no longer correspond to the Manhattan distance of these two

Name	Value
Family:	Virtex-6
Device:	XC6VLX75T
Optimization Goal:	Area
Optimization Effort:	Normal
Keep Hierarchy:	Yes

Table 2.2: Configuration of XST for synthesis. All other parameters correspond to their default values.

nodes. Moreover, if the network gets saturated, some packets might have to stay in the injection queues before they can be injected into the network. Indeed, if the network is congested, the average time packets spend in an injection queue, which is denoted by l_{queue} , can be significantly larger than the average time they spend in the network itself. Thus, the packet latency is given by:

$$l = l_{head} + l_{serialization} + l_{queue} \quad (2.2)$$

2.6.2 Evaluation Methodology

All router architectures and general concepts presented in this thesis are implemented and evaluated in Very high speed integrated circuit Hardware Description Language (VHDL). This enables the evaluation of performance as well as hardware requirements with a single implementation. More precisely, the same source code can be used for simulation and synthesis. Dedicated NoC simulators, as BookSim [Jia+13], Noxim [Cat+15], and NIRGAM [Jai], allow higher simulation speeds at a higher level of abstraction. However, if those simulators are used, a separate implementation for synthesis is required.

All herein presented synthesis results are generated with Xilinx’s XST [Xil15]. Please note that XST synthesizes a design for Field-Programmable Gate Arrays (FPGAs) and does not use a standard cell library like Application-Specific Integrated Circuit (ASIC) synthesis software. In contrast to specifically FPGA tailored NoC designs, as Hoplite [KG15], however, all herein evaluated implementations are not optimized for FPGAs. Nevertheless, the synthesis results generated with XST allow a comparison of the hardware requirements. Table 2.2 shows the herein used configuration of XST. In general, FPGAs consist of an array of Configurable Logic Blocks (CLBs) and routing channels. The used Virtex-6 FPGA contains one pair of slices per CLB. Every slice contains four LookUp Tables (LUTs), eight storage elements (REGs), wide-function multiplexers, and carry logic. Some slices support additional functions, as storing and shifting data.

Name	Pattern
Bit complement:	$d_i = \overline{s_i}$
Bit reverse:	$d_i = s_{b-i-1}$
Shuffle:	$d_i = s_{i-1 \bmod b}$
Transpose:	$d_i = s_{i+\frac{b}{2} \bmod b}$

Table 2.3: Bit permutation traffic patterns. Here, s_i and d_i denote the i -th bit of the b bit source address and b bit destination address, respectively.

For more detailed information on Virtex-6 CLBs please refer to [Xil12]. XST reports the consumption of slices, LUTs, and REGs. In order to compare the hardware requirements of different designs, the main building blocks of slices are compared in this thesis, namely LUTs and REGs, as well as the estimated frequencies after synthesis.

The simulation based performance evaluations presented herein are generated with the open-source VHDL simulator GHDL [GHD15]. GHDL is also a compiler, as it directly translates VHDL code to machine code. For simulation, the synthesizable network implementations are extended by a traffic generator, which instantiates the unit under test, i.e. the NoC, and provides network traffic. Furthermore, every router is equipped with a wrapper component that logs the router's traffic for evaluation.

The network traffic used for evaluation significantly influences the generated outcomes. In general, traffic can be described by its spatial and temporal distribution. Two different kinds of network traffic, in terms of their spatial as well as temporal distribution, are used for performance evaluation in this thesis, namely synthetic traffic and application traffic. Both traffic classes, as well as the accompanying simulation methodologies, are described below.

Synthetic Traffic

The spatial distribution of synthetic traffic for NoC performance evaluation is frequently described by traffic patterns. Those patterns are usually based on particular applications, for instance parallel numerical algorithms or sorting algorithms. The herein used traffic patterns include *random traffic* and *bit permutation traffic*. At random traffic, the destination d , to which a source s sends a packet, is determined by a uniform random process. At permutation traffic, every source s sends all of its traffic to a single destination d . Consequently, the generated network load is not uniformly distributed over all links of the network. Instead, permutation traffic is used to stress the network. Bit permutation traffic is a subset of permutation traffic, at which a destination address is determined by a bit permutation of the binary representation of a source address. Four bit permutation traffic patterns are listed in Table 2.3. There, it is assumed that the

nodes are addressed according to their x and y coordinates in a 2D mesh, as depicted in Figure 2.4a. Furthermore, s_i and d_i denote the i -th bit of the binary representation of the source and destination addresses, respectively, and addresses are b bit wide.

Traffic patterns specify the spatial distribution of network traffic, but not its temporal distribution. In the herein presented evaluations for synthetic traffic, a uniform random injection process is used. Furthermore, these evaluations are based on open-loop simulations, which means that the traffic parameters are controlled independently of the NoC itself. The network traffic is generated according to a given injection probability α and is injected into the nodes' injection queues directly. Injection into these queues never fails as sufficiently dimensioned queues are assumed⁸. However, injections from the queues into the network are only possible if the corresponding routers are able to accept a packet, i.e. there is an idle output channel or an empty buffer.

Application Traffic

Synthetic traffic is frequently used in the NoC domain to compare networks or individual sub-aspects of them. However, even if application performance is affected by the deployed interconnection, it is not linearly correlated to the interconnection performance. Moreover, the temporal and spatial distribution of application traffic is usually non-uniformly distributed. Hence, evaluating application traffic allows to draw conclusions on application performance, i.e. application runtime, as well as the assessment of network performance under more realistic traffic conditions.

In contrast to simulations with synthetic traffic, application traffic requires closed-loop simulations, as the network influences the traffic. Real applications are usually self-throttling, which means that there is a sequential dependency between packets and new injections depend on previous ejections. For instance, processors usually send only a limited number of memory requests, and then wait for the responses, before they create new requests. Thus, to trace pure application traffic and to use this traced traffic for simulations afterwards allows no conclusions on the resulting runtimes, as traces do not contain packet dependencies.

In order to evaluate application performance, the herein used VHDL simulator was connected to the Netrace [HGK10; HK10] library. As Netrace is implemented in C, named pipes were used for inter-process communication between the simulator and Netrace. Netrace comes with network packet traces for several Princeton Application Repository for Shared-Memory Computers (PARSEC) [Bie11] benchmarks, which additionally include packet dependencies. PARSEC is a benchmark suite composed of multi-threaded programs. Compared to pure network trace files, Netrace traffic is self-throttling and enables more accurate performance results. Furthermore, the required simulation

⁸In fact, an injection queue size of 32 flits was sufficient in the vast majority of executed simulations.

Benchmark:	blackscholes	canneal	x264
Input:	simlarge	simmedium	simsmall
Clock cycles:	5.1×10^7	5.6×10^7	5.3×10^7
Packets:	6.0×10^6	4.5×10^6	1.0×10^6
Large packets:	48%	47%	49%
Small packets:	52%	53%	51%

Table 2.4: Characteristics of the three herein evaluated PARSEC benchmarks.

runtimes⁹ are considerably lower, compared to full-system simulation.

The PARSEC evaluations presented in this thesis are restricted to three benchmarks, namely blackscholes, canneal, and x264. The characteristics of these three PARSEC benchmarks are listed in Table 2.4. They are selected because of their different communication requirements. Due to the long simulation times, the presented results are based on approximately 5×10^7 clock cycles from the benchmarks' Region Of Interest (ROI). The chosen clock cycles correspond to exactly 6×10^6 packets, 4.5×10^6 packets, and 1×10^6 packets for blackscholes, canneal, and x264, respectively. The percentages of large packets (size of 72 B) and small packets (size of 8 B) are also depicted in Table 2.4. Figure 2.15 shows the injection probabilities of the three evaluated benchmarks as a function of execution time. In contrast to the uniformly distributed injection probability of synthetic traffic, all three benchmarks inject bursts of packets.

⁹Side note: The herein presented application performance simulations lasted between several hours and up to two weeks.

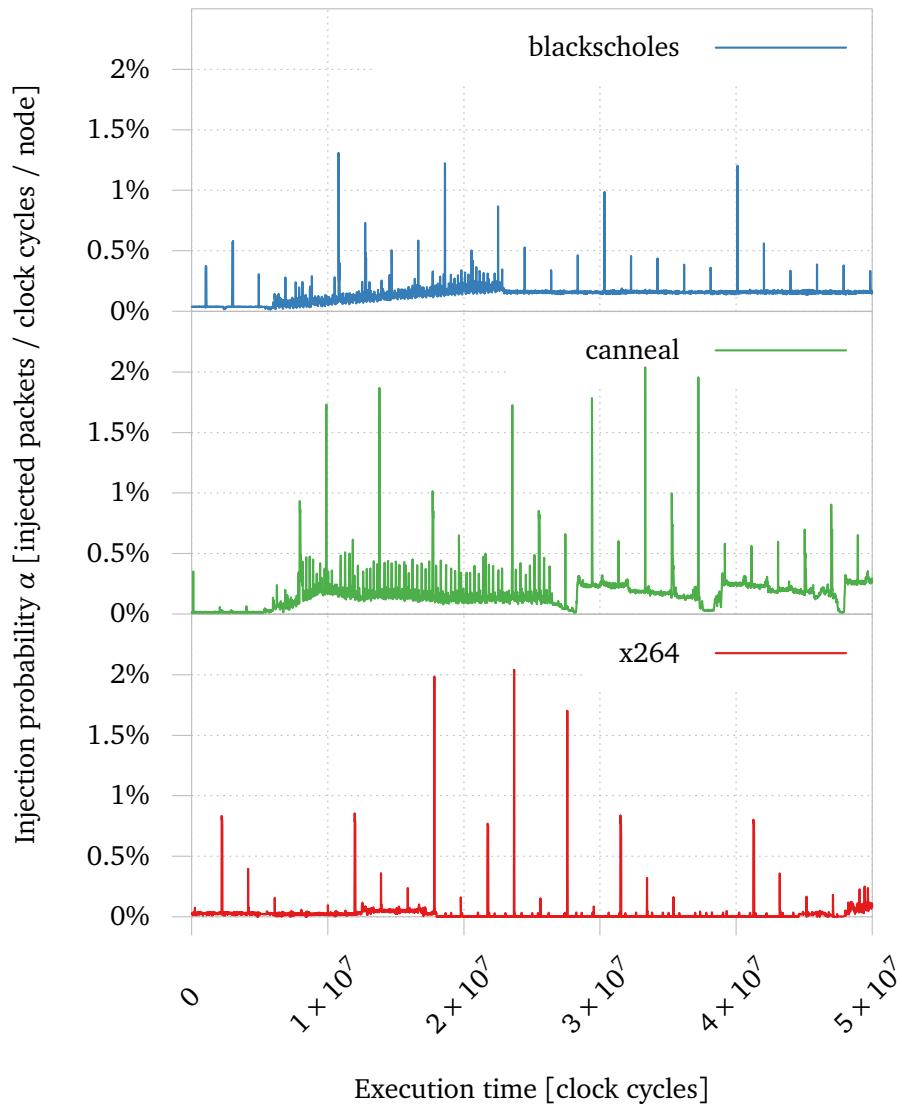


Figure 2.15: Injection probabilities during the first 50 million clock cycles of the ROI of three PARSEC benchmarks.

2.7 Conclusion and Existing Interconnection Networks

NoCs are considered as the prevalent interconnection infrastructure for future many-core systems, as NoCs can meet the demanding communication requirements of these systems. Due to their modular design and their scalability, NoCs are well suitable to connect this high number of components. However, NoCs are complex systems themselves and have a huge design space.

Table 2.5 shows the characteristics of several interconnection architectures of real processors. As frequently the case at commercial products, information about these NoCs is publicly available only to a certain extent. Nevertheless, it becomes apparent that these interconnection architectures are very diverse. Even their underlying data models vary. Some networks transfer data for the message passing paradigm, e.g. at Tiler's TILE64, while other networks transfer cache-coherent data for the shared memory paradigm, e.g. at Intel's Knights Corner. For instance, all herein introduced flow control methods have already been utilized at NoCs. Despite this diversity, there are also some similarities. For a small number of cores, the ring topology is appropriate. In contrast, the mesh topology is more scalable, and thus more suitable for a larger number of cores. Most real NoCs use dimension order routing if they are based on a 2D mesh, and shortest path routing if a ring topology is utilized. Both topologies are very simple, deadlock free, and require only few hardware resources.

Processor	Topology	Routing Algorithm	Flow Control	Additional Information
IBM, Sony, Toshiba Cell [KPP06]	four directional rings, 12 cores	shortest path	circuit switching	128 bit wide links
Tilera TILE64 [Wen+07]	five separate 8×8 meshes	<i>X</i> -first routing	DYN: wormhole, credit-based flow control / STN: circuit switching	32 bit links, four dynamic networks (DYN) and one static network (STN)
Intel Teraflops Research Chip [Van+08]	8×10 mesh, two cores / node	source routing	wormhole, ON-OFF flow control	39 bit wide links, two VCs
Intel SCC [How+10]	6×4 mesh, two cores / node	<i>X</i> -first routing	virtual cut-through, credit-based flow control	144 bit wide links, eight VCs
Intel Knights Corner [Rah13]	three bidirectional rings, up to 61 cores	shortest path	slotted ring	largest ring: 512 bit wide links
Intel Knights Landing [Sod+16]	6×6 mesh, two cores / node	<i>Y</i> -first routing	arbitrate at injection and at turn	four separate networks per chip

Table 2.5: Characteristics of the interconnection architectures of real processors.

Deflection Routing based Router Architectures

Contents

3.1 Principle of Deflection Routing	37
3.2 Pros and Cons of Deflection Routing	39
3.3 Deflection Routing Implementations	42
3.3.1 Crossbar based Architectures	43
3.3.2 Permutation Network based Architectures	46
3.4 Basis Network on Chip and Router Architecture	53
3.4.1 Flit Prioritization Scheme	54
3.4.2 Routing Algorithm	55
3.4.3 Summary	65

In this chapter, the general principle of deflection routing is introduced in Section 3.1. Afterwards, the pros and cons of deflection routing are illustrated in Section 3.2. Optimal operating conditions of deflection routing are shown, by reviewing several comparisons of bufferless and buffered router architectures presented in literature. Existing implementations of deflection routing based NoCs are presented in Section 3.3. In Section 3.4, the herein used router architecture is introduced, which serves as a basis for Chapters 4 and 5. Finally, as deflection routing is not related to any specific routing algorithm, several algorithms are evaluated for this basis router architecture.

3.1 Principle of Deflection Routing

Most industrial NoCs, as well as NoCs developed in academia, are packet switched networks. In such networks, flit buffers are fundamental components, which highly influence the performance of the interconnection. On the other hand, the energy

consumption of NoCs of today's generation is substantial and will be a barrier in the future [Bor07; Bor10]. The NoC of Intel's 80-Core Teraflops Research Chip, for example, consumes 28% of system power [Hos+07]. A significant proportion of static and dynamic energy used for the network is consumed by buffers. Additionally, buffers require significant chip area. At the Tiler TILE64 processor [Wen+07], the buffer size is reduced to an absolute minimum required for efficient flow control and no virtual channels are deployed. Nevertheless, more than 60% of a tile's die area is consumed by buffers. For TRIPS-OCN, the interconnection of the TRIPS processor, it is reported that the input buffers consume 75% of the router area and 10.2% of the tile area [Gra+06]. They used a buffer depth of just two flits and a flit size of 128 bit. This is more than three times of the area required for the second largest component of a TRIPS-OCN router, which is the crossbar. Moreover, buffers add complexity and latency to the router architecture as buffer management logic is required. This includes flow control logic, e.g. credit management in case of credit based flow control, as well as virtual channel logic if virtual channels are deployed. To summarize, buffers increase the NoC throughput, but at the same time, buffers consume energy, require a significant amount of chip area, and add complexity, as well as latency to the router architecture.

Due to these evident drawbacks of flit buffers, several approaches exist to get completely rid of buffers, or at least reduce the amount of buffers. Bufferless NoC routers can either drop [HJL09; Góm+11] or deflect [MM09; Mil02; FCM10; FCM11; Fal+11; Fal+12] messages in case of collision. Dropped messages have to be retransmitted, which necessitates ACKnowledgement (ACK) or Negative-ACKnowledgement (NACK) logic. At deflection routing, retransmissions are avoided by deflecting messages, i.e. those messages are routed along non-shortest paths.

For the sake of completeness, NoCs which utilize circuit switching or virtual circuits also require no flit buffers at the routers. The best-known circuit-switched network is the analog telephone network. At circuit switching, network resources are dedicated to single communication flows at a time. Hence, circuit switching is well suited for steady traffic flows, but can not cope with frequently changing and interleaving traffic as present in many CMPs and MPSoCs. At virtual circuits, network resources are shared by several traffic flows, but are still reserved for the required time span by using Time-Division Multiplexing (TDM). The network traffic's periodicity is exploited to prevent packet collisions, and hence, also the need for flit buffers [Shp+15]. This also enables guaranteed service traffic and the compliance of real time requirements. Real-time NoCs which utilize virtual circuits include Argo [Kas+16], aelite [HSG09], and dAEIite [SMG14].

Deflection routing has already been proposed in 1964 by Baran, who entitled this communication scheme *hot-potato routing* [Bar64]. Deflection routing requires no flit buffers at all, and hence, flits can not be stored in routers. To avoid collisions and packet dropping, every flit has to be forwarded to a neighbored router as soon as new flits can arrive. That means, at a single cycle router architecture, flits have to leave the current

router exactly one network clock cycle after they arrived. At any clock cycle, every output port can be assigned only to a single flit. If several flits try to arbitrate the same output port simultaneously, this port is assigned only to the highest prioritized flit. As the name indicates, all other flits are deflected to idle output ports. A basic prerequisite of deflection routing is that the number of output ports is greater or equal to the number of input ports. Hence, it is guaranteed that there is always an idle output port per flit. The output port, to which a flit is deflected to, does not have to be a productive port. In case of a non-productive port, the deflected flit takes a detour and is routed along a non-shortest path between its source and destination. Thus, the links of a deflection routing based NoC constitute a sort of flit buffers. Due to the costs of optical buffering, deflection routing has been frequently used in optical NoCs [HLH02].

3.2 Pros and Cons of Deflection Routing

Deflection routing enables to omit flit buffers, and thus, allows area and energy savings. Besides this, deflection routing has several further desirable characteristics of a NoC. On the downside, the energy and area efficiency does not necessarily have to hold for the entire NoC as the saved area and energy can be consumed by other parts of the router architecture. In this section, the pros (+) and cons (–) of deflection routing are considered first. Afterwards, it is shown why deflection routing is a viable solution for NoCs. Lastly, existing comparisons of deflection routing and buffered NoCs from literature are presented.

+ **Local flow control:** Omitting flit buffers leads to a simplified router design and purely local flow control, i.e. no explicit buffer arbitration is required. In contrast, additional logic as well as wires between adjacent routers are required for the flow control scheme of buffered NoCs.

+ **Adaptivity:** Deflection routing is also adaptive, independently from the used routing algorithm. Flits are routed around highly congested areas as they are deflected away from their destination if all productive directions are already allocated to higher prioritized flits. At buffered, packet switched networks, packets can only progress if buffer space can be allocated in the adjacent router. Otherwise, packets stay in their current buffer.

+ **Abstinence of deadlocks**¹⁰: In case of wormhole flow control, one packet can even be distributed over several routers, and hence among several flit buffers. Body and tail flits can only progress if their head flit can be routed. As flit buffers are a shared resource, deadlocks can occur if circular dependencies arise. Deflection routing based router architectures basically calculate a new permutation every network clock cycle. Flits can not stay in a router, and thus deflection routing is inherently deadlock free.

¹⁰If not stated otherwise, the term *deadlock* refers to a routing dependent deadlock in this thesis.

– **Topology restriction:** Omitting flit buffers also involves several challenges and potential drawbacks, of course. First of all, deflection routing requires a topology at which every router has at least as many output ports as input ports. This prerequisite is necessary as the routers act as repeaters and all receiving flits have to be able to be forwarded. For most practical NoC topologies (e.g. ring, torus, mesh) this prerequisite is fulfilled anyway.

– **Livelocks:** Even if deadlocks can not occur at deflection routing based NoCs, precautions against livelocks have to be taken. In other words, a mechanism is required which ensures that no flit is deflected infinitely. One simple solution, which is used i.a. in [MM09; Mil02], is the prioritization by age. In this case, a hop count field in the flit structure is required, which is incremented at every router. If the hop count field of a flit overflows, the flit is assumed to be undeliverable.

Further solutions are the golden flit scheme [FCM10; FCM11] and silver flit scheme [Fal+11; Fal+12]. There, at least one golden flit exists, which is prioritized over non-golden flits. All routers determine the golden flits independently by other routers, based on the flit's sender and transaction Identifier (*ID*). Further, both the golden flit scheme and the silver flit scheme ensure that every packet will be golden eventually. Both prioritization schemes are introduced in more detail in Section 3.3.2.

– **Routing overhead:** The basic data units at deflection routing are flits, instead of packets, as deflection routing can not be combined with packet switching. Furthermore, it can not be guaranteed that two flits which belong to each other, follow the same path from their source to their destination. This is because of (i), flits can not be stored in routers and (ii), the highest prioritized flit has to be routed one hop closer to its destination at every routing decision. Due to (i) and (ii), a new packet, which arrives at a router at clock cycle c , can cause the separation of head and body flits of another packet, which arrived before clock cycle c . Hence, every flit has to carry routing information, or in other words, every flit has to be a head flit. Consequently, at deflection routing, messages are divided into flits, instead of packets. This problem is illustrated in more detail in Chapter 5, Figure 5.1. If every flit contains routing information, the routing overhead can be significant. The routing overhead denotes the non-payload information, transferred by the NoC. The ratio of routing overhead and payload depends on the flits' size. In Chapter 5, methods to reduce this ratio are presented and the optimal flit size for permutation network based router architectures is considered.

– **Reassembly buffers:** If a message's size exceeds the link width or flit size, the message can only be transferred by decomposition into several sub-messages. Each sub-message must not be larger than a flit's payload size. As all flits which belong to one message are routed independently, they can arrive out of order at their destination. A destination node has to receive all flits of a message to be able to reassemble it. Further, flits of several messages can be received in an interleaved way, which leads to large buffering requirements at the receiver side. Too small reassembly buffers can lead to *reassembly buffer deadlocks* if a buffer is filled with partial messages. Due to the lack

of backpressure¹¹ at deflection routing, reassembly buffers have to be dimensioned for the worst case. Fallin, Craik, and Mutlu proposed to use Miss Status Holding Registers (MSHRs) as reassembly buffers, which are present anyway in the memory systems [FCM10]. By this, additional buffers can be avoided. A detailed explanation of this approach is given in Section 3.3.2, page 50.

– **Router latency:** Due to the sequential dependency of flits at prioritization and port allocation, the router latency can even increase compared to buffered, virtual channel routers. Permutation network based router architectures, like [FCM10; FCM11; Fal+11; Fal+12] and the herein used basis router architecture, provide a possible solution.

– **Reduced bandwidth:** Potential energy and area savings of deflection routing are gained by omitting flit buffers. These buffers, however, also enhance the maximum bandwidth and throughput of a NoC. Several approaches exist [Fal+12; OWB12a; Jos+13; Jon+14], which try to reduce the deflection rate by utilizing a small number of flit buffers.

The presented pros and cons of deflection routing raise the question of the conditions under which deflection routing is a viable solution for NoCs. Performance wise, the router latency as well as the deflection rate are two major factors. Permutation network based router architectures enable a critical path which is comparable to buffered routers [FCM10], and thus provide a comparable router latency. The deflection rate is highly correlated to the network load, as deflections can only occur if at least two flits compete for the same port. Hence, this suggest that deflection routing is suitable for NoCs with a low network load. Real applications are usually self-throttling, which means that at some point, new messages are only injected if the responses for previous requests have been received. Further, it is reported that the typical load of NoCs is below their peak throughput most of the time [MM09]. Moscibroda and Mutlu compared the deflection routing based architecture BLESS [MM09] to a baseline buffered router architecture. They observed a performance degradation of only 0.5% (3.2%) on average (worst-case) for a set of multiprogrammed SPEC CPU2006 and Windows Desktop applications. Several drawbacks of BLESS are addressed at CHIPPER [FCM10; FCM11], i.a. the costly sequential port allocation and the crossbar are replaced by a permutation network. In [FCM10], CHIPPER is evaluated with five SPLASH-2 applications. They reported that the average performance with CHIPPER is only 1.8% lower compared to the baseline buffered router architecture. They noticed very little performance degradation relative to the buffered router architecture at workloads that are not network intensive. Hence, they concluded that bufferless routing is an attractive option for low to medium load cases.

Concerning hardware costs, it has to be ensured that the saved area and energy of

¹¹At buffered NoCs, backpressure is provided by the flow control scheme and avoids packet dropping. Upstream nodes are informed when they have to stop sending new flits, as all downstream buffers are occupied [DT04, p. 245].

flit buffers is not consumed by other parts of the router architecture. In [MM09], the authors observed an energy reduction of $\approx 40\%$ for BLESS compared to a baseline buffered router architecture. Fallin, Craik, and Mutlu reported in [FCM10] that BLESS requires $\approx 35\%$ less area than the buffered router architecture. On the other hand, the critical path of BLESS is $\approx 43\%$ longer than that of the baseline architecture due to the sequential dependency at port allocation. In [Mic+10], an independent comparison of BLESS and a buffered router architecture, with empty buffer bypassing, is performed. They found that BLESS is only up to 1.5% more energy efficient at very light network loads. Further, they reported that the buffered network provides a lower latency and a higher throughput per unit power. On the other hand, in [FCM10], it is reported that CHIPPER, which is a developed version of BLESS, has $\approx 36\%$ lower area requirements and only an 1.1% longer critical path than the buffered baseline router architecture.

Craik and Mutlu investigated the impact of both BLESS and a buffered router architecture as one component of the memory hierarchy [CM11]. They found, the performance of the bufferless network can get very close to that of a buffered network if a locality aware mapping of data to cache slices is used. Such a locality aware data mapping algorithm can reduce the network utilization, and hence, makes a lower throughput network potentially more effective. Further, the power advantage of a bufferless design compared to a buffered one increases if the locality is exploited efficiently.

Nychis et al. showed that BLESS performs similar to a buffered router architecture at a small number of cores. However, they also observed that the average latency with BLESS increases significantly, compared to a buffered network, as the size of the CMP increases. They proposed a source throttling congestion control mechanism, which enables linear throughput scaling.

Recently, Cai, Mai, and Mutlu compared FPGA and ASIC implementations of bufferless and buffered router architectures [CMM15]. Their evaluation showed that deflection routing enables reductions of required area (38%), consumed power (30%), and also router cycle time (8%).

To summarize, buffers can enhance the performance of NoCs, in particular, if the NoC is highly utilized. On the downside, buffered routers consume significantly more area and energy than deflection routing based NoCs. Several enhancements exist for both bufferless and buffered router architectures. Unfortunately, they can not be compared completely, due to the large quantity of enhancements. However, the herein presented comparisons of bufferless and buffered router architectures indicate that deflection routing is a viable solution for small, fast, and energy efficient router architectures.

3.3 Deflection Routing Implementations

In this section, several existing router architectures which are based on deflection routing are introduced. Thereby, the design parameters of such a router architecture

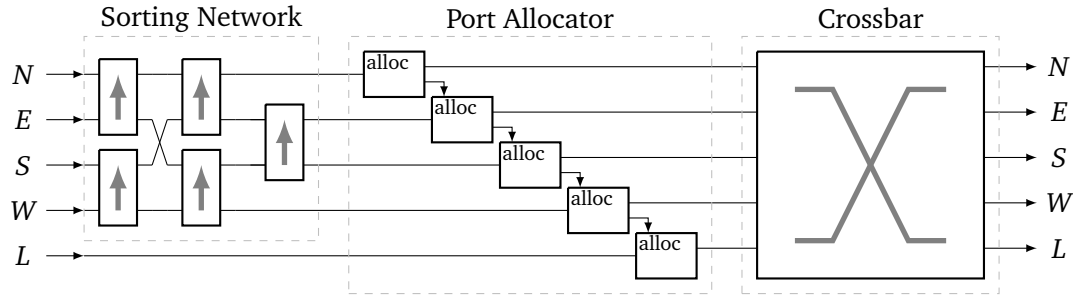


Figure 3.1: Crossbar based router architecture

are illustrated.

Every NoC router consists of several input ports and output ports (I/O ports), which provide links to neighbored routers as well as local resources. One important design parameter of NoCs is the number of ports P that a router possesses. This number is referred to as the radix of a router¹². Another crucial design parameter is the topology of a NoC. All router architectures presented in this section are designed - or at least support - the 2D mesh topology. This topology allows low radix routers with four I/O ports and at least one local port.

The physical interconnection of a router's input ports and output ports is enabled by an interconnection architecture. At most deflection routing based NoCs, either a crossbar or a permutation network is deployed. The used interconnection architecture is a fundamental design decision, in particular, for deflection routing based NoCs. Hence, the herein presented router architectures are grouped according to their interconnection architecture.

3.3.1 Crossbar based Architectures

Crossbar based router architectures usually share the same basic structure. They consist of three major parts: a sorting network, a port allocator, and a crossbar (cf. Figure 3.1). The sorting network determines the highest prioritized flit, which has to be routed to a productive direction, in order to avoid livelocks. The sorting network sorts all incoming flits by their priority. Typical implementations of such networks are odd-even merging networks or bitonic sort networks [Bat68]. The building block of these networks are two-input comparison elements. At four inputs, both network types consist of three stages of comparison elements. The sorting network depicted in Figure 3.1 is an odd-even merging network. The local port L is not sorted by the sorting network as it is assumed that a flit from the local port always has the lowest priority. This is the case for

¹²A port of a router consists of one input port and one output port. Hence, at a 2D mesh topology and a router architecture with one local port, the radix is $P = 5$.

several prioritization schemes, e.g. if the flits are prioritized by age. Please note that a sorting network with two stages of comparison elements is sufficient to identify the highest prioritized flit out of four flits. However, if a network with just two stages is used, the other flits might be unsorted. Therefore, a complete sorting network enhances the quality of the routing decision by taking all flits' priorities into account.

Even though only low radix topologies, as a 2D mesh, are considered herein, one might ask how the sorting networks scale with higher radices. Both the odd-even merging network and the bitonic sort network have a complexity of $\mathcal{O}(n \lg(n)^2)$ and a depth of $\mathcal{O}(\lg(n)^2)$. The benefit of odd-even merging networks is a slightly lower number of comparison elements, whereas the benefit of the bitonic sort network is its modularity [Bat68].

The second major block of a crossbar based router architecture is the port allocator. The port allocator assigns an output port to every flit, from the highest prioritized flit to the lowest prioritized flit. The preferred output ports are determined by the flits' destination addresses and the routing algorithm. For the highest prioritized flit, an arbitrary output port, out of the five available output ports, is selectable. Then, the second highest prioritized flit can be assigned to one out of the four unallocated ports, and so forth. Hence, there is a sequential dependency between each allocation step. If flits arrive at all input ports, the output port of the lowest prioritized flit is predefined by the higher prioritized flits.

Even if this greedy port allocation is frequently used in deflection routing based router architectures, it is not optimal in terms of performance. Let us consider a situation at which two flits, f_1 and f_2 , arrive at a router. It is assumed that the first flit, f_1 , has a higher priority than the second flit, f_2 . Further, f_1 has two productive directions, the north (N) and the east (E), whereas f_2 has only one productive direction, which is also the north (N). If the first allocator assigns the N output port to f_1 , the second flit f_2 has to be deflected, as the only productive port is already allocated. However, in this situation, a productive port could have been assigned to both flits if the N output port had not been assigned to flit f_1 .

To overcome this drawback of greedy port allocation, the allocator can first identify all productive ports based on the flits' destination addresses. In a second step, the actual routing decision can be performed. The allocator assigns output ports to every flit, e.g. by minimizing a global function based on all flits preferred output ports. However, also at this approach, it has to be ensured that the highest prioritized flit gets assigned to one of its productive directions.

After an output port is allocated for every flit which has arrived, the flits can traverse the crossbar. The crossbar physically interconnects the I/O ports and enables direction changes of the flits. As the crossbar does not distinguish from a crossbar of a buffered router, please refer to Section 2.5 for further details about crossbars.

Nostrum

Nostrum [Mil02; Mil+04b; Mil+04a; Lu+05; PJ06; MJ07; Mil11] is not only a NoC, it is a communication platform, and it was a research project at KTH Royal Institute of Technology. The research project lasted from 2001 to 2008, and was headed by Professor Axel Jantsch. A broad variety of NoC related research topics have already been investigated on basis of this platform. In [Lu+05], a simulation environment for Nostrum, called NNSE, is introduced. The Nostrum Backbone, which basically defines a communication protocol stack, is proposed in [Mil+04a]. Nostrum is also the basis of several fault-tolerant router architectures (e.g. [Fen+10b; Fen+10a]), which are introduced in more detail in Chapter 4. The concepts and results presented in this section are restricted to the basic hardware structure and characteristics to enable a comparison to other deflection routing based architectures.

The used topology at Nostrum is a 2D mesh [Mil+04a; MJ07]. Routing is split into three phases. First, a priority is assigned to every flit on basis of the flit's hop count, i.e. its age. In the second phase, the flits select their favored output ports. Two different strategies, *uniform* and *proportional*, are implemented for the case that two productive output ports exist. At *uniform*, the priorities for both directions are selected uniformly. At *proportional*, the priorities are weighted by the distances to the respective destination. Finally, the actual routing permutation is selected on basis of the weighted priorities.

In contrast to all other deflection routing based NoCs, Nostrum also supports Guaranteed Throughput (GT) traffic, besides Best Effort (BE) traffic. The basic idea is to use container flits, which are routed along predefined routes from source to destination and back in a closed loop fashion [Mil+04b]. Container flits get loaded at the sender side and unloaded at the receiver side. To avoid deflections of container flits, their routing decision is stored in all corresponding routers. Further, the involved routers reserve an empty slot for these flits.

BLESS

BufferLESS routing algorithms (BLESS) is a set of routing algorithms developed by the SAFARI research group of Professor Onur Mutlu at Carnegie Mellon University. However, Moscibroda and Mutlu also proposed a bufferless router architecture in [MM09]. It is a two cycle router architecture, which consists of a route computation stage and a switch traversal stage. Their deflection routing based algorithm is called FLIT-BLESS. For FLIT-BLESS, the router architecture closely resembles the architecture depicted in Figure 3.1. Flits are prioritized by their age (*oldest flit first*). They evaluated four more prioritization schemes, which are *closest flit first*, *most deflections first*, *round robin*, and *mixed policy*, at which *oldest flit first* and *round robin* are alternated. However, their evaluation showed that *oldest flit first* performs best in terms of average latency, maximum latency, and average number of deflections.

A second algorithm is WORM-BLESS, which combines bufferless routing with ideas from wormhole routing. There, a message is divided into a packet, or rather a worm, which consists of one head flit, several body flits, and one tail flit. The authors identified two major problems which usually prohibit a combination of these two concepts. The so-called *livelock problem* occurs as port arbitration is only performed for head flits. If a head flit of a worm arrives at a router, all productive directions of the worm could already be allocated to other worms which are currently in transit. However, as mentioned before, deflecting the highest prioritized flit can lead to livelocks. The second problem is the so-called *injection problem*. New flits can be injected into the NoC if at least one input port is idle, and hence, also one output port. Unfortunately, it is difficult to predict the state of the input ports. If flits arrive at all input ports during the injection of a worm from the local port, this injection has to be aborted.

Moscibroda and Mutlu solved both problems by worm truncation. If, for instance, a higher prioritized worm arrives at a router, the lower prioritized worm might be deflected. In case of worm truncation, some flits of a worm are separated from their head flit. As the header contains routing relevant information, every router has to be able to create new head flits out of body flits. Thus, every router stores the header information of all worms being in transit, as well as a mapping of output ports to allocated worms. At WORM-BLESS, header information is transferred by dedicated wires, which are unused in case of body flits and tail flits. Moscibroda and Mutlu compared the costs of these additional wires and the costs of additional buffers for packet switching. As mentioned before, the results showed that bufferless routing enables significant energy and area savings, while providing similar performance at low network utilization.

3.3.2 Permutation Network based Architectures

One major drawback of router architectures which are based on a crossbar and additionally on deflection routing is the long critical path. This path is dominated by the sorting network and the port allocator stage. A permutation network can substitute all three major components of crossbar based router architectures, which are the sorting network, the port allocator, as well as the crossbar. Compared to crossbar based designs, permutation networks enable a shorter router latency as well as a more area and energy efficient router architecture, as the following results show. On the downside, permutation networks can be internal blocking, depending on the permutation network's depth. Internal blocking means, some permutations between the I/O ports are not supported.

Due to the area and energy efficiency of permutation networks, they complement bufferless deflection routing, whose main advantages are also area and energy savings. Furthermore, if it can be guaranteed that the highest prioritized flit can reach every output port, even the internal blocking characteristic is not a major issue. If a flit can not be transferred to a certain output port, it is just deflected to an other direction.

In this section, two prominent permutation network based NoCs are introduced in

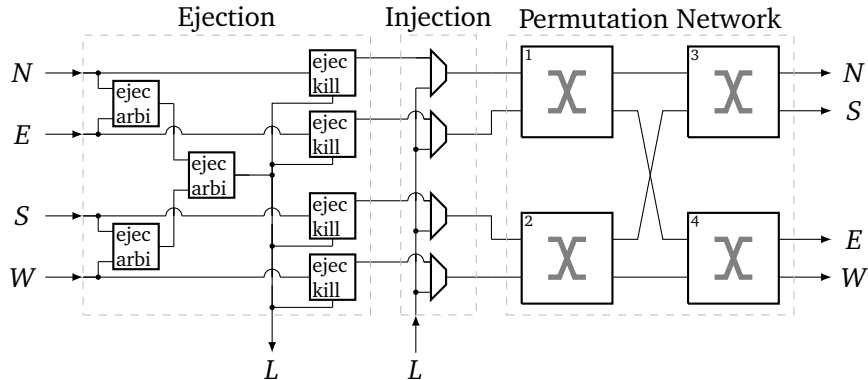


Figure 3.2: CHIPPER router architecture [FCM10; FCM11]

more detail: CHeap-Interconnect Partially PErmuting Router (CHIPPER) and Minimally-Buffered Deflection (MinBD). However, apart from these two architectures, several further architectures exist which use deflection routing and additionally a permutation network (e.g. [Jos+13; Jon+14; Lee+13b]).

CHIPPER

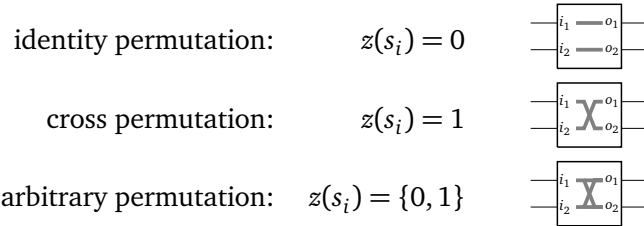
To the best of my knowledge, CHIPPER [FCM10; FCM11] is the first router architecture which was based on deflection routing and additionally utilized a permutation network. It is an enhanced version of the BLESS router architecture, and like the BLESS architecture, it is also developed by researchers of the SAFARI research group (Fallin, Craik, and Mutlu) at Carnegie Mellon University. CHIPPER consists of three blocks, an ejection stage, an injection stage, and a permutation network. An overview of this router architecture is depicted in Figure 3.2.

At CHIPPER, the first two stages handle the local port separately from the other ports. The ejection stage selects at most one flit which is ejected from the network. This stage consists of three ejection arbiters, each with two input ports and one output port. These arbiters compare the destination addresses of both inputs to the router's own address. If both flits have reached their destination, the higher prioritized flit is selected for ejection. Finally, the selected flit is ejected by one of the four ejection kill elements, which also clears the corresponding input port.

The injection stage allows the injection of a flit from the local port into the network if at least one port is idle. An injection element is basically just a MUltipleXer (*MUX*), which selects either the flit from the local port *L* or from the corresponding input port. The *MUX*s' control logic is placed in the ejection stage to a large extent (e.g. comparators to identify an idle input port).

The third stage is a permutation network, which is a 4×4 Banyan network at CHIPPER.

It is build of 2×2 switching elements, or rather permute blocks, hereinafter abbreviated as s_1, \dots, s_4 . Each switching element s_i , with $i \in \{1, 2, 3, 4\}$, is always in one of the two possible states. At state $z(s_i) = 0$, also referred to as the identity permutation, the element s_i passes the first input i_1 to the first output o_1 , and the second input i_2 to the second output o_2 . At state $z(s_i) = 1$, also referred to as the cross permutation, s_i swaps both inputs and outputs, and hence, i_1 is transferred to o_2 , and i_2 to o_1 . Hereinafter, the following symbols are used to illustrate the switching elements' states:



Please note, the most frequently used symbol for a switching element, as well as for a crossbar, equals the herein used symbol to denote a cross permutation. To indicate that a switching element can be in an arbitrary state, an overlay of identity and cross permutation will be depicted in the following.

The switching elements' states are defined by the used routing algorithm and the two received flits. As the routing algorithm is implemented in a distributed manner, the entire permutation network is self routing. This means, each switching element routes the two received flits independently from the other switching elements, using only local information. CHIPPER's exact routing algorithm, i.e. to which direction the highest prioritized flit is transmitted if two productive directions exist for this flit, is not mentioned in [FCM10; FCM11]. A Banyan network with four I/O ports is a single path network, which means that there is exactly one path between an input port and an output port. Further, it is internal blocking, which means that some permutations of the Banyan network's I/O ports are not possible. Figure 3.3 shows one possible and one impossible permutation. In both situations, the highest prioritized flit arrives at the first input of the upper left switching element s_1 . As the destination of this flit is in the south, the state of this element has to be $z(s_1) = 0$. As a consequence, the second flit, which arrived at the second input port of s_1 , is transmitted to the lower right switching element s_4 , independently from the second flit's preferred output direction. Switching element s_4 is connected to the east and to the west. Hence, both flits can be transmitted to their preferred output direction in the first example. However, in the second example, the second flit does not reach its destination N , but is deflected as s_1 is in state $z(s_1) = 0$. There is no configuration of the switching elements to transmit both flits of the second example to their preferred direction. Please note, even if the Banyan network is internal blocking, it is guaranteed that the highest prioritized flit can always be transferred to its preferred output port.

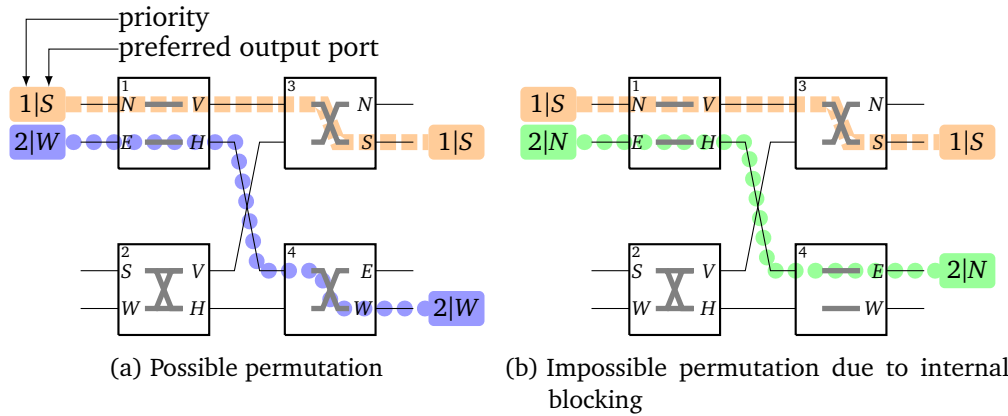


Figure 3.3: A possible and an impossible permutation for a Banyan network. The number in the upper left corner of the switching elements denotes their ID.

Due to the internal blocking property of a Banyan network, the arrangement of the I/O ports highly influences the performance. At CHIPPER, output ports are grouped by vertical ports and horizontal ports, whereas input ports are connected in compass direction, i.e. N , E , S , W . The vertical output ports N and S are connected to the upper right switching element s_3 . The horizontal output ports E and W are connected to the lower right switching element s_4 . This ensures, first, the most frequently routing decision can be realized by the Banyan network, and second, an efficient implementation of the routing algorithm. As a Banyan network does not support every permutation between its I/O ports, it is particularly important to support the most common permutations. At a frequently used 2D mesh topology, most routing algorithms route the majority of all packets along the horizontal axis, i.e. $E \leftrightarrow W$, or along the vertical axis, i.e. $N \leftrightarrow S$, and the direction of a packet is switched only once [Kim09; FCM10]. The utilized arrangement of the I/O ports allows that this permutation is supported without any flit being blocked. Furthermore, this allows an efficient implementation of dimension order routing. For instance, if x -first routing is used, the switching elements of the first stage (s_1 and s_2) just have to compare the highest prioritized flit's x coordinate with the router's x coordinate. If both x coordinates are equal, the flit has reached its desired x coordinate and is transmitted to s_3 , which then routes the flit along the y axis. Otherwise, the flit is transmitted to s_4 , which routes the flit further towards the desired x coordinate along the x axis.

Besides the long critical path of prior router architectures based on deflection routing, the developers of CHIPPER identified two more problems of deflection routing in [FCM10; FCM11]. Here, the term prior router architectures refers, in particular, to the BLESS architecture. The first mentioned problem is the expensive flit prioritization scheme *oldest flit first*, which requires a large hop count field in the header, and

additionally large comparators in the arbiters. The authors' solution for this problem is the *golden packet* prioritization scheme. The second drawback is the need for large reassembly buffers at the receiver side due to the lack of feedback to senders. This problem is solved by the *retransmit-once* flow control scheme, as well as the use of the MSHRs as reassembly buffers.

Golden Packet: The basic idea behind the *golden packet* prioritization scheme is that the abstinence of livelocks can be ensured by:

- (i) prioritizing a single packet over all other packets for a period which is long enough to ensure delivery,
- (ii) ensuring that every packet gets this status eventually.

This globally prioritized packet is called the golden packet. A flit, which belongs to the golden packet, is always prioritized over any non-golden flit. However, most of the time, only non-golden flits arrive at the routers. In this case, every router selects a prioritized flit pseudo-randomly. In the rare event that two flits arrive at a router which both belong to the golden packet, the flit with the lower sequence number is prioritized. The authors indicated that the field of this sequence number is usually only two or three bits large.

The golden packet is selected by an implicit function of time, which is computed by each router independently. With this function, the *ID* of the golden packet is specified, whereas the same packet gets the golden status for a time span, called golden epoch. The golden epoch encompasses the time period to ensure the delivery of the entire golden packet. The function rotates through all possible packet *IDs*, and thus, every packet receives this status eventually.

Retransmit-Once: As mentioned in Section 3.2, deflection routing is inherently free of routing dependent deadlocks, but reassembly buffer deadlocks can occur if the reassembly buffers are too small. The retransmit-once protocol provides deadlock-free packet reassembly. This protocol is illustrated in Figure 3.4, according to [FCM10, p. 12]. Initially, the requester sends a request flit to the receiver ①. If no request buffer is available, as assumed in the depicted example, the receiver is forced to drop this flit ②. As soon as buffer space becomes available, the receiver sends a flit back to the requester to initiate the retransmission. Further, the receiver also reserves the available buffer space for this transaction ③. At ④, the requester retransmits the request. As the receiver's buffer space is reserved until the writeback flit is received ⑤, this transaction is guaranteed to be non-blocked and no further retransmission is required. Finally, the requester sends this writeback flit ⑥, and the buffer reservation is released.

To avoid special reassembly buffers, the authors further proposed to use the already existing MSHRs as reassembly buffers. The main purpose of MSHRs is to keep track of

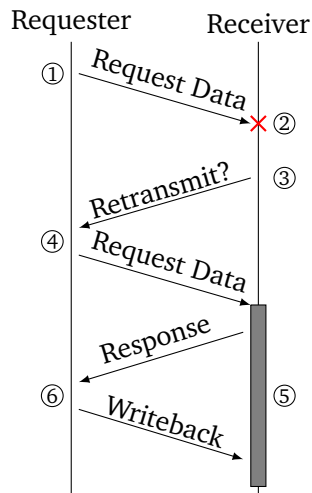


Figure 3.4: Retransmit-once protocol scheme [FCM10, p. 12]

outstanding cache misses at non-blocking caches [Kro81]. Using the existing buffers and data-steering logic of the MSHRs allows reassembling of packets, or rather cache blocks, in-place.

MinBD

The MinBD router architecture [Fal+11; Fal+12] is an enhanced version of CHIPPER and as such also an enhanced version of the BLESS architecture. As the latter two router architectures, MinBD is developed by researchers of the SAFARI research group (Fallin et al.) at Carnegie Mellon University. The main objective of CHIPPER is to make deflection routing efficiently implementable in hardware. MinBD improves the performance of CHIPPER, or any router architecture based on a permutation network and on deflection routing. In order to achieve this improvement, three performance bottlenecks are identified and eliminated. More precisely, Fallin et al. found that many deflections occur as two or more flits reach their destination at the same clock cycle, but only one can be ejected. Further, the golden flit prioritization scheme causes unnecessary deflections, as only a single packet is globally prioritized in the system. Finally, they propose a very small side buffer, which temporally stores deflected flits, to reduce the deflection rate even further. An overview of this router architecture is depicted in Figure 3.5. Additionally to the components of CHIPPER, MinBD consists of a second ejection stage as well as a side buffer. The side buffer itself consists of a redirection stage, a buffer injection stage, a buffer ejection stage, and the actual side buffer.

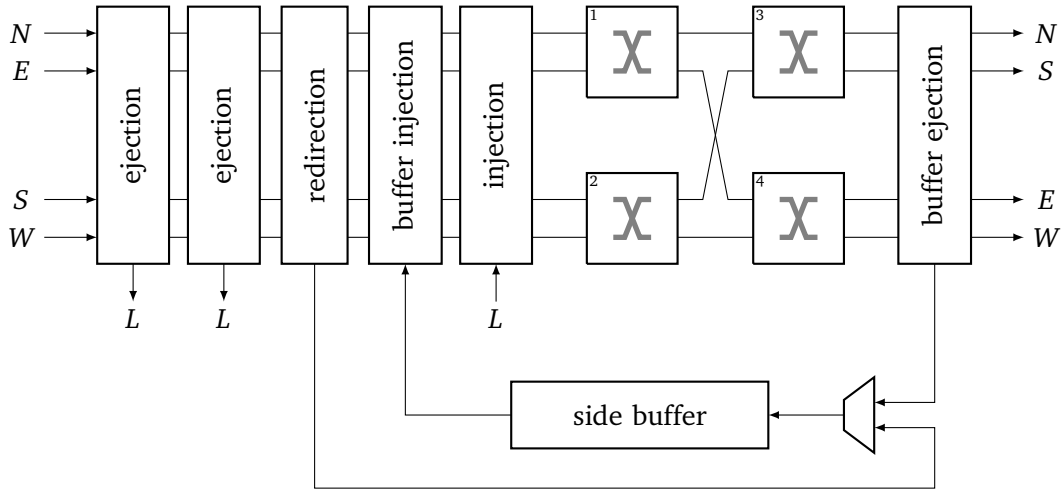


Figure 3.5: MinBD router architecture [Fal+11; Fal+12]

Ejection Bottleneck: Fallin et al. observed that in 8% of all clock cycles, at any given router, two or more flits reach their destination. Further, the traffic arrives in a bursty manner. At CHIPPER, only one flit can be ejected at any clock cycle. Hence, the other flits, which also reached their destination, have to be deflected. This leads to a higher latency of such flits, and also to a higher traffic volume in the NoC. Therefore, MinBD consists of two ejection stages, each allowing to eject one flit per node per cycle.

Prioritization Scheme: The golden flit prioritization scheme of CHIPPER globally prioritizes at most one packet, which is referred to as the golden packet, in the system. Even if this is sufficient to guarantee the abstinence of livelocks, this leads to unnecessary deflections. In the common case, a switching element processes only non-golden flits, which are prioritized pseudo-randomly at CHIPPER. Every flit passes exactly two switching elements at every router. As the Banyan network is self-routing, both switching elements operate independently of each other and they both choose a prioritized flit independently. Hence, their prioritization decisions do not have to be consistent, which may lead to more deflections than necessary.

To avoid deflections caused by inconsistent prioritization, the *silver flit* prioritization scheme is proposed. Every router picks a silver flit pseudo-randomly, which is prioritized throughout the entire permutation network. The silver flit is indicated by a single control bit, which exists only within the router itself.

Side Buffer: The key idea of side buffering is to store flits which would be deflected otherwise. At a later time, the stored flits, hopefully, can be re-injected and routed to a productive direction.

If a specific flit is deflected or not is known after the flit passed the permutation network. Hence, the buffer ejection stage, which ejects at most one deflected flit, is placed after the permutation network. It picks one eligible flit pseudo-randomly, which is then stored in the side buffer, if there is free buffer-space and no flit from the redirection stage accesses the side buffer. A flit is buffer-eligible if it is not golden and not addressed to the current node. Flits which have arrived at their destination are not stored in the side buffer as they are usually deflected back to their destination in the next clock cycle. Golden flits are not side-buffered as they may not be delivered to their destination during the golden epoch. However, golden flits could only be deflected if at least two golden flits arrive at a router, which occurs very rarely.

The buffer injection stage allows to eject a flit from the side buffer and inject it into the router pipeline if at least one input port is idle. It is placed before the injection stage to prioritize injections from the side buffer over the local port. However, if all input ports are occupied, the flits stored in the side buffer can make no forward progress. The redirection stage is required to avoid side buffer starvation and provide livelock free delivery for flits stored in the side buffer. If the side buffer has not re-injected any flit for a given time threshold, the redirection stage ejects an arriving flit pseudo-randomly and stores it in the side buffer. Thereby, a free slot in the router pipeline for one side-buffered flit is created.

3.4 Basis Network on Chip and Router Architecture

In this section, the herein used NoC router architecture is described. For the deployed topology, the widely adopted 2D mesh is assumed. The influence of various topologies on the performance of deflection routing based NoCs is investigated in [LZJ06b; Fen+11a]. However, the majority of the concepts and results, presented in this work, are not restricted to 2D meshes. Certain conditions concerning the used topology are given in the following chapters when necessary. Here, the output ports at the edges of the 2D mesh are connected to their corresponding input ports by loop links (cf. Figure 3.6). This enables a homogeneous router architecture, independently from the routers' positions, with an equal number of I/O ports. Moreover, links act as a kind of buffer space at deflection routing based NoC, and hence, the buffer capacity is enhanced.

The local port L of a router is connected to a Network Interface (NI), which in turn is connected to an IP core or PE. The NIs convert transactions from the IP cores into flits and packets, which can be routed in the NoC, and vice versa. In all herein presented simulation results, it is assumed that each NI contains a sufficiently dimensioned injection queue. Implemented in hardware, the injection queues might be placed in the processors' MSHRs, as proposed at CHIPPER [FCM10; FCM11]. The term sufficiently dimensioned injection queues means that injections from the PEs into these queues are never blocked. As the evaluated injection probabilities are only increased up to the saturation point of

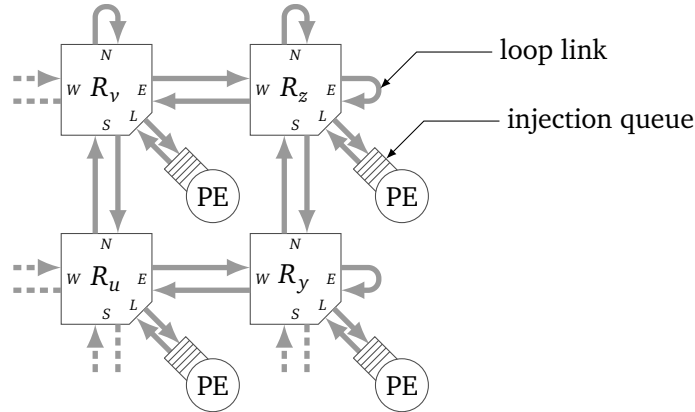


Figure 3.6: 2D mesh topology with loop links at the edges of the mesh. Each Processing Element (PE) is connected to an injection queue.

the NoC, an injection queue size of 32 flits was sufficient in the vast majority of executed simulations. Furthermore, it is also assumed that ejected flits can always be received by PEs or the corresponding NIs.

The used router architecture is based on CHIPPER to a large extent. CHIPPER is selected because of its simplicity as well as its area and energy efficiency. These benefits are primarily achieved by the deployed permutation network, which complements deflection routing very well. CHIPPER's clear structure and modularity allows a simple extension as well as a quick evaluation of new concepts. Hence, the basic composition is adopted, which comprises the ejection stage, the injection stage, and the permutation network. At CHIPPER, the MSHRs are used for message reassembly. However, as MSHRs are part of the memory hierarchy, this requires full-system simulation, or at least large parts of the memory hierarchy have to be simulated. This would significantly increase the simulation time, and thus, the reassembly problem is not considered in this work. As mentioned before, a sufficiently sized injection queue is assumed for the results presented herein.

3.4.1 Flit Prioritization Scheme

A further contribution of CHIPPER is the *golden flit* prioritization scheme, which selects the globally highest prioritized flit. In contrast to CHIPPER, the *oldest flit first* scheme is used in this work, due to following reasons:

- Even for an 8×8 NoC, a hop count field of 8 bit is sufficient. The extra costs of 8 bit comparators, compared to 3 bit comparators plus additional logic to determine the golden flit *ID*, should be acceptable.

- Prioritization by age achieves a better performance than the golden flit scheme. The golden flit scheme determines just a single highest prioritized flit, whereas 255 different flit priorities¹³ exist at *oldest flit first* with a hop count field of 8 bit. Furthermore, all existing router architectures compared in Chapter 4 use the *oldest flit first* prioritization scheme. Thus, the evaluation results are not influenced by the chosen prioritization scheme.
- The most important reason is that *oldest flit first* enables the detection of undeliverable flits. In Chapter 4, link and router failures are considered. Some of the evaluated routing algorithms can not deliver all flits to their destinations, even if physical paths between all nodes exist. Thus, some kind of livelock resolution is required. Prioritization by age allows to identify and discard undeliverable flits efficiently.

Oldest flit first selects a highest prioritized flit at every router. Furthermore, as the implementation of the prioritization scheme is distributed over all four switching elements of a router, one higher prioritized flit, denoted by f_{hp} , and one lower prioritized flit, denoted by f_{lp} , exist at every switching element. In the case that two flits arrive at a switching element and both flits have the same age, the switching element deterministically prioritizes one of its two input ports.

Even if prioritization by age is frequently applied at deflection routing based architectures, several further prioritization schemes have been proposed in literature. Lu, Zhong, and Jantsch investigated three different priority policies in [LZJ06b]. Two of them (*non-priority* and *straight-through*) can not guarantee the abstinence of livelocks. The third policy, referred to as *weighted priority*, is based on the flits' age, distance, and number of deflections. *Weighted priority* performed best, however, they did not mention how livelocks can be resolved. The *multipath* flit prioritization scheme, introduced in [OWB12b], is also a multi-criteria scheme. It is based on the flits' age and number of productive output ports. Flits with a single productive output port are temporarily prioritized over flits with multiple productive directions.

3.4.2 Routing Algorithm

Once a switching element has determined the locally highest prioritized flit f_{hp} , it transmits this flit to its desired output port, i.e. to its productive direction. If two flits have arrived at a switching element, the lower prioritized flit is forwarded to the remaining direction. The desired direction of f_{hp} is defined by f_{hp} 's destination node, abbreviated as $f_{hp}(dst)$, and by the utilized routing algorithm. CHIPPER's routing algorithm is not specified in [FCM10; FCM11]. However, independently from the routing algorithm, three different situations can be distinguished:

¹³A hop count of 0 indicates an idle link. New flits are injected into the NoC with a hop count of 1. Thus, only $2^{|hc|} - 1$ different priorities are possible, whereas $|hc|$ denotes the length of the hop count field.

- (i) There is no productive direction for f_{hp} . This means, f_{hp} has reached its destination and it is not ejected.
- (ii) A single productive direction for f_{hp} exists. In this case, every minimal routing algorithm¹⁴ selects this productive direction as the desired direction.
- (iii) Two productive directions exist for f_{hp} .

The latter case is the most interesting one. Two productive directions are always neighbored, this means one direction is along the vertical axis (N or S) and the other productive direction is along the horizontal axis (E or W). At dimension order routing, one of the two axis is preferred deterministically. Hence, the highest prioritized flit is routed to the correct position along one dimension first, and afterwards along the remaining dimension. For instance, at Y_FIRST , f_{hp} is routed along the vertical axis first. Dimension order routing is frequently deployed at packet switched networks, because of its simplicity and its deadlock freeness. As deflection routing is inherently routing-dependent deadlock free, the deployed routing algorithm can select the desired output ports without the risk of cyclic dependencies. For instance, one of the two productive directions can be picked randomly, which is referred to as $RANDOM_FIRST$. This algorithm is also minimal, but non-deterministic, as it randomly switches between X_FIRST and Y_FIRST . A variety of routing algorithms is imaginable. However, due to the utilized permutation network, the routing algorithm has to be efficiently implementable in a distributed manner, as all four switching elements operate independently of each other.

Obviously, the used routing algorithm influences the performance of the NoC significantly. As $CHIPPER$'s routing algorithm is not specified and routing algorithms of crossbar based architectures are not implemented in a distributed manner, several different routing algorithms are evaluated as a basis for the analysis in the following chapters. In this section, six different routing algorithms are introduced and examined. In general, a deflection routing algorithm can be considered as a function which calculates a new permutation between all input ports and all output ports. More formal, it is a bijection $RT_ALG : \{N, E, S, W\} \rightarrow \{N, E, S, W\}$. However, as the algorithms are implemented in a distributed manner, they actually consist of three different routing algorithm parts:

$$\begin{aligned} RT_ALG_{1,2} &: \{i_1, i_2\} \rightarrow \{V, H\}, \\ RT_ALG_3 &: \{i_1, i_2\} \rightarrow \{N, S\}, \\ RT_ALG_4 &: \{i_1, i_2\} \rightarrow \{E, W\}, \end{aligned}$$

whereas RT_ALG_i denotes the routing algorithm implemented in switching element s_i , with $i \in \{1, 2, 3, 4\}$ (cf. Figure 3.2). Furthermore, i_1 and i_2 denote the first and

¹⁴A routing algorithm is denoted as minimal if flits are routed only along shortest paths, without considering deflections.

Algorithm 1 Centralized Y_FIRST routing

```

1: procedure Y_FIRST( $f_1, f_2, f_3, f_4$ )                                //  $f_i, i \in \{1, 2, 3, 4\}$  denotes the input flits
2:    $f_{hp} \leftarrow \text{PRIO}(f_1, f_2, f_3, f_4)$                         // find highest prioritized flit
3:    $d_x \leftarrow x - f_{hp}(dst_x)$                                     // hops / distance from router to  $f_{hp}$ 's destination along  $x$ 
4:    $d_y \leftarrow y - f_{hp}(dst_y)$                                     // hops / distance from router to  $f_{hp}$ 's destination along  $y$ 
5:   if  $d_y < 0$  then
6:      $S \leftarrow f_{hp}$                                              //  $f_{hp}$  is routed to the south
7:   else if  $d_y > 0$  then
8:      $N \leftarrow f_{hp}$                                              //  $f_{hp}$  is routed to the north
9:   else if  $d_x < 0$  then                                          //  $d_y = 0 \Rightarrow$  route  $f_{hp}$  along  $x$ 
10:     $W \leftarrow f_{hp}$                                              //  $f_{hp}$  is routed to the west
11:   else                                                            //  $f_{hp}$  is routed to the east if  $f_{hp}$ 's destination is located in the east ...
12:     $E \leftarrow f_{hp}$                                              // ... or if  $f_{hp}$  reached its destination and had not been ejected
13:   end if
14: end procedure

```

second input port, respectively, of the corresponding switching element. The switching elements of the first stage, s_1 and s_2 , use the same routing algorithm. RT_ALG_{1,2} selects between the vertical axis V , i.e. switching element s_3 , and the horizontal axis H , i.e. switching element s_4 . The second stage switching elements, s_3 and s_4 , are connected only to one axis, either the vertical axis or the horizontal axis. Hence, their routing algorithm, RT_ALG₃ and RT_ALG₄, takes only the connected axis into account.

The first herein considered routing algorithm is Y_FIRST. As described before, Y_FIRST can be listed as one algorithm for the entire permutation network, as depicted in Algorithm 1.

Furthermore, it can be listed as three separate routing algorithms, Y_FIRST_{1,2} for s_1 and s_2 , Y_FIRST₃ for s_3 , and Y_FIRST₄ for s_4 . The same algorithm, implemented in this distributed manner, is depicted in Algorithm 2. As the routing algorithm operates in a distributed manner in hardware, all herein considered routing algorithms are listed as three separate routing algorithms. This is not only the more natural way to describe a routing algorithm of a permutation network, it further simplifies the assessment of the routing algorithms' complexity. Besides Y_FIRST (listed in Algorithm 2), the algorithms considered in this section are:

- RANDOM_FIRST (listed in Algorithm 3),
- KEEP_DIST (listed in Algorithm 4),
- AVOID_CENTER (listed in Algorithm 5),
- FLITID_DEPEND (listed in Algorithm 6), and
- STRESS_VALUE (listed in Algorithm 7).

At RANDOM_FIRST, the first stage switching elements, s_1 and s_2 , randomly select a

Algorithm 2 Distributed Y_FIRST routing

<p>// s_1, s_2 select between V and H</p> <p>1: procedure Y_FIRST_{1,2}(f_1, f_2)</p> <p>2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2)</p> <p>3: $d_y \leftarrow y - f_{hp}(dst_y)$</p> <p>4: if $d_y \neq 0$ then</p> <p>5: $o_1 \leftarrow f_{hp}$ // $f_{hp} \rightarrow s_3$</p> <p>6: $o_2 \leftarrow f_{lp}$ // $f_{lp} \rightarrow s_4$</p> <p>7: else</p> <p>8: $o_2 \leftarrow f_{hp}$ // $f_{hp} \rightarrow s_4$</p> <p>9: $o_1 \leftarrow f_{lp}$ // $f_{lp} \rightarrow s_3$</p> <p>10: end if</p> <p>11: end procedure</p>	<p>// s_3 selects between N and S</p> <p>1: procedure Y_FIRST₃(f_1, f_2)</p> <p>2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2)</p> <p>3: $d_y \leftarrow y - f_{hp}(dst_y)$</p> <p>4: if $d_y > 0$ then</p> <p>5: $o_1 \leftarrow f_{hp}$ // $f_{hp} \rightarrow N$</p> <p>6: $o_2 \leftarrow f_{lp}$ // $f_{lp} \rightarrow S$</p> <p>7: else</p> <p>8: $o_2 \leftarrow f_{hp}$ // $f_{hp} \rightarrow S$</p> <p>9: $o_1 \leftarrow f_{lp}$ // $f_{lp} \rightarrow N$</p> <p>10: end if</p> <p>11: end procedure</p>	<p>// s_4 selects between E and W</p> <p>1: procedure Y_FIRST₄(f_1, f_2)</p> <p>2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2)</p> <p>3: $d_x \leftarrow x - f_{hp}(dst_x)$</p> <p>4: if $d_x > 0$ then</p> <p>5: $o_1 \leftarrow f_{hp}$ // $f_{hp} \rightarrow E$</p> <p>6: $o_2 \leftarrow f_{lp}$ // $f_{lp} \rightarrow W$</p> <p>7: else</p> <p>8: $o_2 \leftarrow f_{hp}$ // $f_{hp} \rightarrow W$</p> <p>9: $o_1 \leftarrow f_{lp}$ // $f_{lp} \rightarrow E$</p> <p>10: end if</p> <p>11: end procedure</p>
--	--	--

Algorithm 3 RANDOM_FIRST routing

<p>1: procedure RANDOM_FIRST_{1,2}(f_1, f_2)</p> <p>2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2)</p> <p>3: $d_y \leftarrow y - f_{hp}(dst_y)$</p> <p>4: $d_x \leftarrow x - f_{hp}(dst_x)$</p> <p>5: if $d_y \neq 0 \wedge d_x \neq 0$ then</p> <p>6: RANDOMROUTING(f_1, f_2)</p> <p>7: else</p> <p>8: Y_FIRST_{1,2}(f_1, f_2)</p> <p>9: end if</p> <p>10: end procedure</p>	<p>1: procedure RANDOMROUTING(f_1, f_2)</p> <p> // rand() creates new random value $\in (0, 1)$</p> <p>2: if rand() > 0.5 then</p> <p>3: $o_1 \leftarrow f_1$</p> <p>4: $o_2 \leftarrow f_2$</p> <p>5: else</p> <p>6: $o_2 \leftarrow f_1$</p> <p>7: $o_1 \leftarrow f_2$</p> <p>8: end if</p> <p>9: end procedure</p>
<p>1: procedure RANDOM_FIRST₃(f_1, f_2)</p> <p>2: Y_FIRST₃(f_1, f_2)</p> <p>3: end procedure</p>	<p>1: procedure RANDOM_FIRST₄(f_1, f_2)</p> <p>2: Y_FIRST₄(f_1, f_2)</p> <p>3: end procedure</p>

productive output port if exactly two productive output ports exist for the current f_{hp} flit. If there is only one or even no productive output port for this flit, RANDOM_FIRST does not distinguish from Y_FIRST. Hence, in particular, RANDOM_FIRST₃ equals Y_FIRST₃ and RANDOM_FIRST₄ equals Y_FIRST₄, as at most one productive output port exists at the second stage switching elements. In contrast to Y_FIRST, it is expected that flits are less frequently deflected, as two productive output ports exist for a longer period of time. However, it can also be expected that the hot spot in the center of the network is intensified with RANDOM_FIRST.

KEEP_DIST, tries to keep both the absolute horizontal distance and the absolute vertical distance to the flit's destination as large as possible. This also attempts to minimize the deflections by keeping two productive directions as long as possible. If less than two productive directions exist, also KEEP_DIST resembles simple Y_FIRST.

Algorithm 4 KEEP_DIST routing

```

1: procedure KEEP_DIST1,2( $f_1, f_2$ )
2:   ( $f_{hp}, f_{lp}$ )  $\leftarrow$  PRIO( $f_1, f_2$ )
3:    $d_y \leftarrow |y - f_{hp}(dst_y)|$  // absolute y dist.
4:    $d_x \leftarrow |x - f_{hp}(dst_x)|$  // absolute x dist.
5:   if  $d_y > d_x$  then
6:      $o_1 \leftarrow f_{hp}$  //  $f_{hp} \rightarrow s_3$ 
7:      $o_2 \leftarrow f_{lp}$  //  $f_{lp} \rightarrow s_4$ 
8:   else
9:      $o_2 \leftarrow f_{hp}$ 
10:     $o_1 \leftarrow f_{lp}$ 
11:   end if
12: end procedure

```

```

1: procedure KEEP_DIST3( $f_1, f_2$ )
2:   Y_FIRST3( $f_1, f_2$ )
3: end procedure

```

```

1: procedure KEEP_DIST4( $f_1, f_2$ )
2:   Y_FIRST4( $f_1, f_2$ )
3: end procedure

```

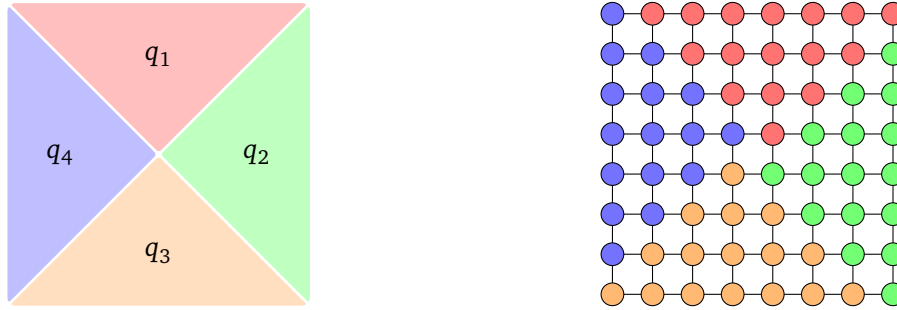


Figure 3.7: At AVOID_CENTER (cf. Algorithm 5), the used routing algorithm depends on the routers' position. The routers in the first and third quadrant (q_1 and q_3) use X_FIRST, and the routers in the quadrants q_2 and q_4 use Y_FIRST. The right graph shows this division for an 8×8 mesh.

For all routing algorithms introduced so far, the center of the NoC is a hot spot. As the name suggests, **AVOID_CENTER** tries to avoid this hot spot. Therefore, the router plane is divided into four quadrants q_1, q_2, q_3, q_4 . Figure 3.7 shows this division for the simulated 8×8 2D mesh topology. The routers which belong to the quadrants q_1 and q_3 use X_FIRST. The other routers, which belong to the quadrants q_2 or q_4 use Y_FIRST. Hence, all routers transfer flits to the center of the mesh if and only if the flits' destination address also lies in the center of the mesh.

FLITID_DEPEND uses another method to spread traffic across the network. Depending on f_{hp} 's ID, which is denoted by $f_{hp}(id)$, either X_FIRST or Y_FIRST is used. Thus, also for this algorithm, the routing function of the switching elements s_3 and s_4 does not distinguish from Algorithm 2.

STRESS_VALUE prefers the less utilized direction, in case of two productive directions. To achieve this, every router counts the utilization of its four outgoing links during the

Algorithm 5 AVOID_CENTER routing

<pre> 1: procedure AVOID_CENTER_{1,2}(f_1, f_2) 2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2) 3: $q \leftarrow$ QUADRANT() // Get router's quadrant 4: if $q = q_2 \vee q = q_4$ then 5: Y_FIRST₃(f_1, f_2) 6: else 7: X_FIRST₃(f_1, f_2) 8: end if 9: end procedure </pre>	<pre> 1: procedure AVOID_CENTER₃(f_1, f_2) 2: Y_FIRST₃(f_1, f_2) 3: end procedure 1: procedure AVOID_CENTER₄(f_1, f_2) 2: Y_FIRST₄(f_1, f_2) 3: end procedure </pre>
---	---

Algorithm 6 FLITID_DEPEND routing

<pre> 1: procedure FLITID_DEPEND_{1,2}(f_1, f_2) 2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2) 3: if $f_{hp}(id) \bmod 2 = 1$ then 4: Y_FIRST₃(f_1, f_2) 5: else 6: X_FIRST₃(f_1, f_2) 7: end if 8: end procedure </pre>	<pre> 1: procedure FLITID_DEPEND₃(f_1, f_2) 2: Y_FIRST₃(f_1, f_2) 3: end procedure 1: procedure FLITID_DEPEND₄(f_1, f_2) 2: Y_FIRST₄(f_1, f_2) 3: end procedure </pre>
---	---

last four clock cycles. This value is referred to as the stress value, and the idea is adopted from the Nostrum router architecture. Routers exchange their four stress values with all neighbored routers. This allows to identify and avoid highly congested areas during runtime.

Please note that this is only a subset of the evaluated routing algorithms. For instance, all presented algorithms only consider the higher prioritized flit f_{hp} . Hence, they all could be extended by considering the lower prioritized flit if no reasonable routing decision is possible for the f_{hp} flit.

In order to assess the performance of the presented routing algorithms, they are simulated using our in-house cycle accurate simulator implemented in VHDL. All simulation results are based on an 8×8 2D mesh topology and 100.000 clock cycles of simulation time. The local port of each router is connected to a traffic generator, which is able to generate different traffic classes. If the traffic generator tries to inject a flit, but the router has no idle output ports, the flit is stored in an injection queue. For all simulations presented in this chapter, the injection queues' size is 16 slots. For injection probabilities higher than the saturation point of the NoC, the injection queues overflowed. In this case, the last injected flit which caused the overflow is discarded and never injected again.

All routing algorithms are evaluated with a multiplicity of synthetic traffic patterns as well as with different injection probabilities. As this evaluation should only show the impact of the routing algorithm and justify the algorithm used in Chapter 4 and

Algorithm 7 STRESS_VALUE routing

1: procedure STRESS_VALUE _{1,2} (f_1, f_2) 2: (f_{hp}, f_{lp}) \leftarrow PRIO(f_1, f_2) 3: $d_y \leftarrow y - f_{hp}(dst_y)$ 4: $d_x \leftarrow x - f_{hp}(dst_x)$ 5: if $d_y \neq 0 \wedge d_x \neq 0$ then // Determine stress value of f_{hp} 's vertical // and horizontal productive direction 6: $\tau_V \leftarrow$ STRESS(PROD_V_DIR(f_{hp})) 7: $\tau_H \leftarrow$ STRESS(PROD_H_DIR(f_{hp})) // Route along direction with lower stress value 8: if $\tau_V < \tau_H$ then 9: Y_FIRST _{1,2} (f_1, f_2) 10: else 11: X_FIRST _{1,2} (f_1, f_2) 12: end if 13: else 14: Y_FIRST _{1,2} (f_1, f_2) 15: end if 16: end procedure	1: procedure STRESS_VALUE ₃ (f_1, f_2) 2: Y_FIRST ₃ (f_1, f_2) 3: end procedure 1: procedure STRESS_VALUE ₄ (f_1, f_2) 2: Y_FIRST ₄ (f_1, f_2) 3: end procedure
---	--

Chapter 5, solely the results for uniform random traffic are shown here. However, the results are very similar for all evaluated traffic patterns.

Figure 3.8 shows the throughput θ , which is the number of ejected flits per clock cycle and per node, for eight different injection probabilities α . Up to an injection probability of 25%, the NoC stays below the saturation point θ_s . Hence, the injection queues have to buffer injected flits only for a very short period. Furthermore, every injected flit is ejected eventually, and therefore, the same throughput is achieved with all routing algorithms. At injection probabilities higher than 30%, the network is at least occasionally saturated. For these high injection probabilities, the routing algorithms affect the maximum throughput. The highest throughput is achieved with AVOID_CENTER, but also Y_FIRST performs surprisingly well.

The box-and-whisker plot depicted in Figure 3.9 shows statistical values of the flits' hop count. The flits' latency, which is based on the hop count, but further includes the queue time, is depicted in Figure 3.10. The boxes of the box-and-whisker plots indicate the lower quartiles and the upper quartiles. The whiskers indicate the minimum values and maximum values. Further, the medians are plotted as black lines inside the boxes.

Figure 3.9 shows that the highest maximum hop count is achieved with Y_FIRST, whereas the lowest maximum hop count is achieved with RANDOM_FIRST. However, this does not hold for the median. The median with RANDOM_FIRST and with KEEP_DIST is considerably higher than for the rest of the evaluated algorithms, in particular, for high injection probabilities. The common characteristic of these two algorithms are

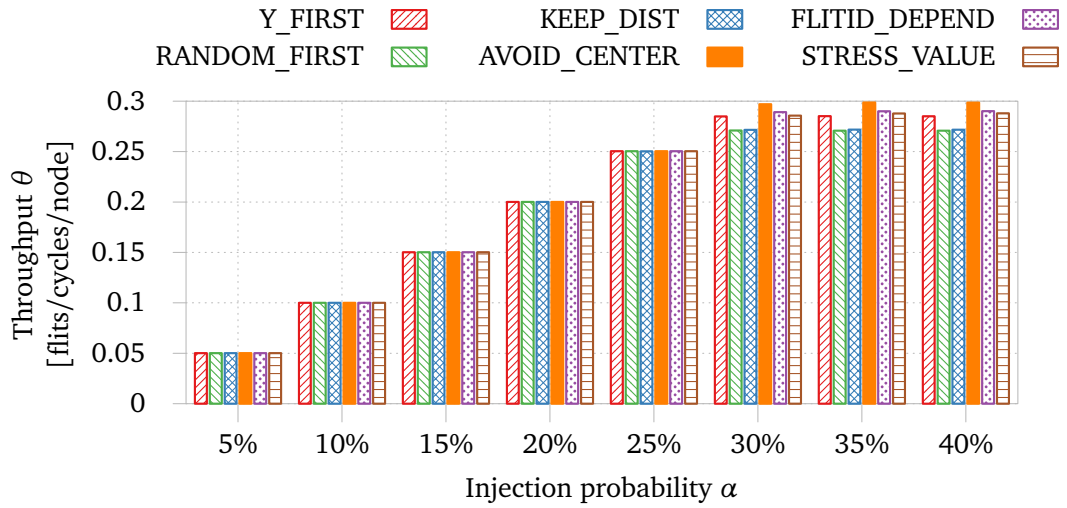


Figure 3.8: Throughput for all evaluated routing algorithms and eight different injection probabilities.

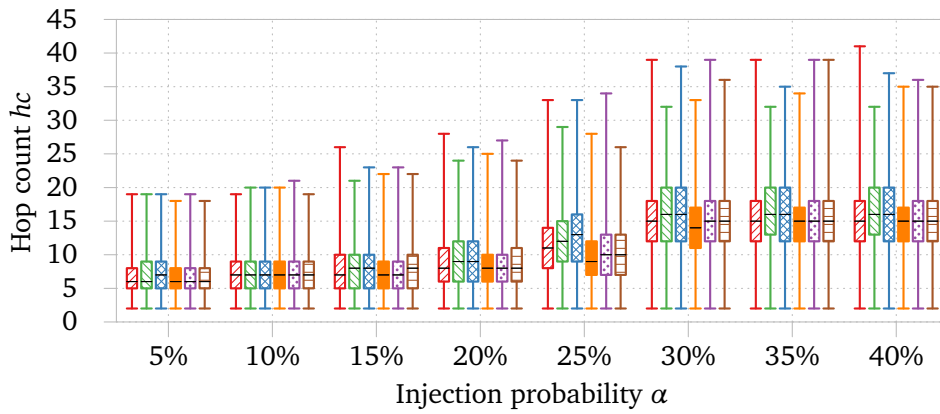


Figure 3.9: Statistical values of the flits' hop count for all evaluated routing algorithms and eight different injection probabilities. Please confer the legend of Figure 3.8.

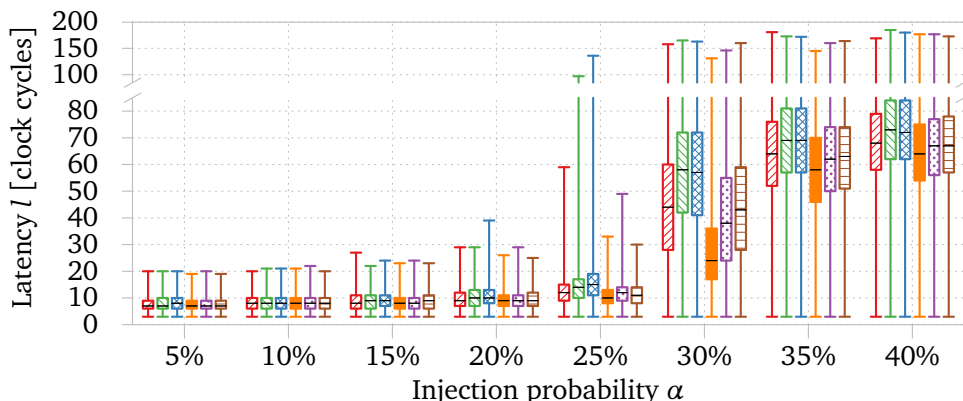


Figure 3.10: Statistical values of the flits' **latency** for all evaluated routing algorithms and eight different injection probabilities. Please confer the legend of Figure 3.8.

frequent changes in direction. This means, flits are continuously routed from the vertical axis to the horizontal axis, and vice versa. However, the permutation network does not allow every possible permutation, and it is designed for traffic without direction changes. Y_FIRST, and also AVOID_CENTER and FLITID_DEPEND, minimize direction changes by transmitting the flit along one axis until the destination is reached for this axis.

Furthermore, RANDOM_FIRST and KEEP_DIST try to preserve two productive output ports as long as possible. However, this also stresses the center of the 2D mesh, at least for random traffic. Figure 3.11 shows the utilization of all output links of the simulated 8×8 network, an injection probability of $\alpha = 20\%$, for all six routing algorithms.

Another evaluation criterion is the number of deflections, which is depicted in Figure 3.12. The number of deflections of a flit is defined as the difference between the flit's final hop count and the Manhattan distance of the flit's source and destination node. As energy is consumed every time a flit is routed, this is an important metric for energy efficiency. As expected, the number of deflections increases with the network load. Since the results are based on the same simulations, the statistical values of the number of deflections and the statistical values of the hop count are highly correlated. Frequent direction changes lead to more deflections, due to the deployed permutation network.

To conclude, expensive and hardware-consuming algorithms do not necessarily pay off. STRESS_VALUE, for instance, requires additional wires and four counters per router, or FLITID_DEPEND requires comparators for the flits' IDs. However, both algorithms do not perform better than AVOID_CENTER, at which each router deterministically uses either Y_FIRST or X_FIRST, based on its coordinates in the mesh. Even simple

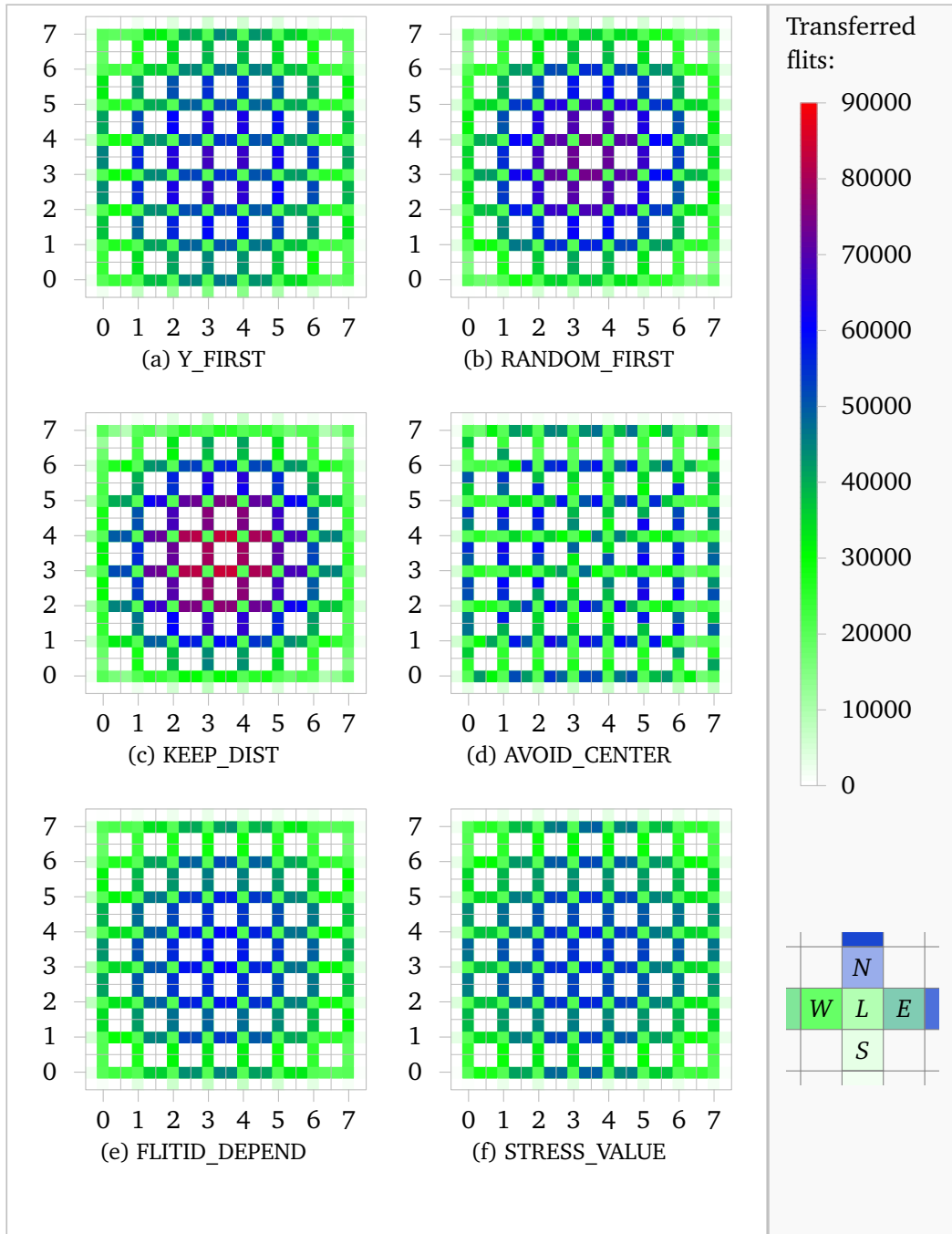


Figure 3.11: Link utilization at an injection probability of $\alpha = 20\%$.

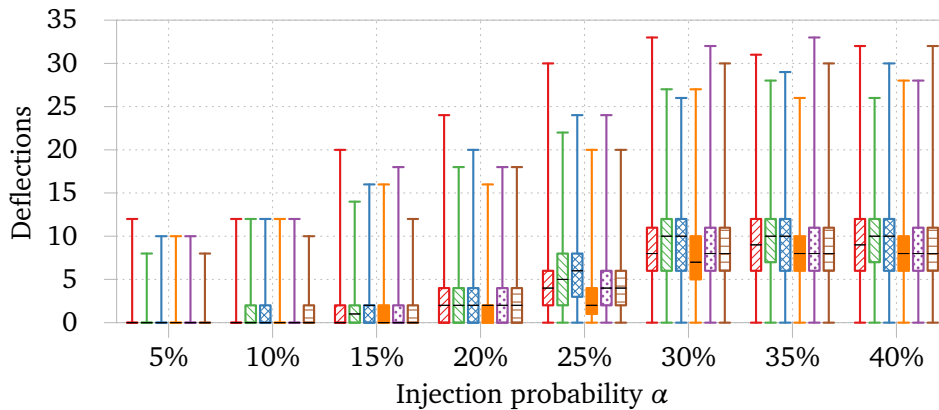


Figure 3.12: Statistical values of the flits' **deflections** for all evaluated routing algorithms and eight different injection probabilities. Please confer the legend of Figure 3.8.

Y_FIRST performs surprisingly well, and in particular, better than RANDOM_FIRST and KEEP_DIST. The reasons for this are on the one hand the permutation network, which does not allow every permutation, and on the other hand the congested center of the network, which leads to more deflections in this area.

3.4.3 Summary

In this chapter, the general principle of deflection routing as well as its strengths and weaknesses have been presented. Furthermore, four well-known router architectures which are based on deflection routing have been introduced. In particular, these architectures differ in their deployed interconnection architectures. Crossbar based router architectures, as Nostrum and BLESS, sort all arriving flits according to their priority at the port allocation. This yields in a longer critical path due to the sequential dependence of the flit sorting. Permutation network based router architectures, as CHIPPER and MinBD, avoid the crossbar and the corresponding arbitration logic. Every switching element acts independently from the other switching elements and requires only local information, which enables a fast and simple router architecture. Thus, a permutation network is deployed at the basis router architecture used in this thesis. If not stated otherwise, all herein presented results are based on the following configuration:

- an 8×8 2D mesh topology
- loop links at the edges of the mesh
- NIs with sufficiently sized injection queues

- permutation network based router architecture
- flit prioritization scheme *oldest flit first*
- flit structure contains 8 bit hop count field
- flits are discarded if hop count field overflows
- routing algorithm AVOID_CENTER

The foundation of the basis router architecture is CHIPPER. However, the incremental improvements of MinBD and most other enhanced versions can be easily integrated into our architecture.

Fault-tolerant and Deflection Routing based Router Architecture

Contents

4.1 Motivation and Scope	68
4.1.1 Fault Tolerance Methods	69
4.1.2 Fault-tolerant Routing	71
4.1.3 Conclusion	73
4.2 Related Work of Chapter 4	73
4.3 FaFNoC Router Architecture	77
4.3.1 Fault Tolerance and Banyan Networks	78
4.3.2 Substitute Benes Networks for Banyan Networks	86
4.3.3 Concept of Fault-aware Flits	94
4.3.4 Summary and Complete Overview of FaFNoC Router Architecture	99
4.4 Evaluation of FaFNoC Router Architecture	100
4.4.1 Non-fault-tolerant Architecture	101
4.4.2 Fault-tolerant Architecture	104
4.5 Summary and Conclusion of Chapter 4	111

Continued technology scaling enables an ever growing number of resources per chip, but not without any disadvantages. Shrinking manufacturing processes also lead to an increasing variability in performance and reliability. Future systems have to be able to cope with different types of failures, as it is predicted that they will be ubiquitous in these systems. Hence, the trends, which give rise to NoCs as the dominant communication infrastructure of VLSI systems, also necessitate fault tolerance concepts. In the field of NoCs, fault tolerance is a broad research topic, which is addressed in several dedicated textbooks [YA12; CML12]. Therefore, only a branch of this research topic is considered

in this chapter, namely fault-tolerant routing at deflection routing based NoCs. Fault-tolerant routing exploits the inherent path redundancy of most NoC topologies to tolerate certain fault classes. More precisely, such algorithms can contribute to tolerate link failures or even the breakdown of complete routers.

This chapter is organized as follows. In Section 4.1, a motivation and an introduction to fault tolerance and the required terminology is provided. A summary of existing router architectures, which are fault-tolerant and additionally based on deflection routing, is given in Section 4.2. In Section 4.3, the Fault-aware Flits NoC (FaFNoC) router architecture [Run15a; Run15b; RK16d] is introduced, which has been developed as part of this thesis. A FaFNoC router utilizes a 4×4 Benes network as interconnection architecture which consists of one additional stage of switching elements compared to a more frequently deployed 4×4 Banyan network. To demonstrate the benefits of a Benes network in terms of fault tolerance, the required modifications to avoid faulty links at the Banyan network based non-fault-tolerant basis router architecture are shown in Section 4.3.1. In Section 4.3.2, the required modifications to a Benes network based router architecture are presented. The modifications of Sections 4.3.1 and 4.3.2 ensure that faulty links are avoided, but they are not sufficient to tolerate complex fault situations, which may arise at higher fault rates. To achieve this, the concept of Fault-aware Flits (FaF) has been developed, which is introduced in Section 4.3.3. An evaluation and comparison of the herein presented FaFNoC router architecture to existing fault-tolerant and deflection routing based architectures is given in Section 4.4. Finally, this chapter closes with a summary and conclusion in Section 4.5.

4.1 Motivation and Scope

First of all, it is necessary to clarify the terminology used in this chapter. The definitions of faults, errors, and failures used in this thesis are adopted from [Avi+04]. A *failure* is a deviation of the system's external state from the correct state. In the context of digital systems, or in particular NoCs, the external state corresponds to the system's output ports. The deviation itself is called an *error*. However, not every error propagates to the system's external state and causes a failure. The adjudged cause of an error is called a *fault*. Likewise, not every fault causes an error, e.g. a stuck-at-0 fault causes an error only if the correct logical value was 1. Hence, a fault can be active or dormant, i.e. the fault causes an error or it does not cause an error, respectively.

Faults can be classified by their temporal occurrence. *Transient faults* appear only for a very short period of time, i.e. one or a few clock cycles. They are mainly caused by environmental conditions such as radiation, and thus, occur more or less randomly. *Intermittent faults* last for several clock cycles [YA12, p. 11]. They can occur in bursts and repeatedly appear at the same location. Such faults can be caused i.a. by voltage fluctuations or temperature variations, and they can be aggravated by aging hardware.

In case of wear out, intermittent faults can precede *permanent faults*, which do not vanish and reflect irreversible damages of the circuit.

Also the physical causes of faults are manifold. Radiation, like neutrons or alpha particles, can cause soft errors (e.g. Single Event Upsets (SEUs)). Electromagnetic interference can lead to crosstalk between wires. Permanent faults, which are the focus of this chapter, are mainly engendered by aging and process variabilities (e.g. material impurities). Aging of CMOS devices can be caused by electromigration, hot carrier injection, or dielectric breakdowns [Rad+13]. A more detailed overview of physical fault mechanisms can be found in [Rad+13]. Several of these physical causes gain in importance with technology scaling. The ITRS from 2011 states:

“One of the key problems that designers face due to further shrinking of feature sizes is the increasing variability of design-related parameters, resulting either from variations of fabrication parameters or from the intrinsic atomistic nature which affects, e.g., channel doping.” [ITR11, p. 39]

Besides increasing probabilities of faults, future systems will have to cope with highly varying component lifetimes. Hence, it is expected that fault tolerance concepts will be required for future VLSI systems to achieve reasonable operating periods and improve yield rate.

4.1.1 Fault Tolerance Methods

Generally, fault tolerance at NoCs is achieved by adding some kind of redundancy to the network at the appropriate level. Redundancy can be classified as *spatial redundancy*, *temporal redundancy*, and *information redundancy* [Run12a; Run12b]. The latter class is characterized by the fact that additional information is transferred, e.g. checksums or identical information over multiple links. Information redundancy can be used to handle all fault classes mentioned above. At temporal redundancy, transmissions or samplings are repeated. For instance, Automatic Repeat reQuest (ARQ) uses acknowledgments and timeouts to control retransmission. As the same resources could be used continually, temporal redundancy is applicable for transient as well as intermittent faults, but not for permanent faults. Spatial redundancy is best suited for permanent faults, as multiple components are used in parallel.

These three kinds of redundancy can be applied at different levels at NoCs, which can be classified according to the OSI layer model. Table 4.1 gives an overview over several fault tolerance methods, categorized according to their OSI layer as well as their form of redundancy. Some of these methods require dedicated hardware support (e.g. the techniques at the data link layer), others exploit the inherent path redundancy of NoCs. Fault-tolerant routing, which is the main focus of this chapter, belongs to the latter category. As fault-tolerant routing is a broad research topic itself, a more detailed overview is given in Section 4.1.2.

OSI layer	spatial redundancy	temporal redundancy	information redundancy
Data link layer	<ul style="list-style-type: none"> • TMR^a to protect important control signals (e.g. NACK signals) • Spare wires to replace faulty wires 	<ul style="list-style-type: none"> • Multi-sampling (sampling twice to detect, or sampling ≥ 3 to even correct) • Hop-to-hop ARQ^b • Split-link transmission to use only intact wires 	<ul style="list-style-type: none"> • Codes (e.g. SEC codes^c) to protect communication between routers. Particularly suitable for header information
Network layer	<ul style="list-style-type: none"> • Fault-tolerant routing, exploits the inherent path redundancy of NoCs <p>Main focus of this chapter. More detailed overview is given in Section 4.1.2.</p>	Temporal redundancy techniques operate either on the data link layer or transport layer	<ul style="list-style-type: none"> • Probabilistic routing (e.g. flooding, random walk, ...), replicated packets are routed over different paths
Transport layer	<ul style="list-style-type: none"> • Source routing 	<ul style="list-style-type: none"> • End-to-end ARQ between sender NIs and receiver NIs 	<ul style="list-style-type: none"> • Codes (e.g. FEC^d) to protect end-to-end communication. Particularly suitable for the payload information

^aTriple Modular Redundancy (TMR)

^bAutomatic Repeat reQuest (ARQ)

^cSingle-Error Correcting code (SEC code)

^dForward Error Correction (FEC)

Table 4.1: Several methods and techniques to enhance the reliability of NoCs, according to [Rad+13]. They are grouped by their form of redundancy as well as their OSI layer.

4.1.2 Fault-tolerant Routing

Fault-tolerant routing, as part of the network layer, can be achieved by using either spatial redundancy or information redundancy. Spatial redundancy exploits the inherent path redundancy of most NoC topologies. Information redundancy based methods utilize several copies of the same packet. The latter approach can provide tolerance against transient, intermittent, and permanent faults, as well as low message latencies. However, depending on the degree of redundancy, both power consumption and network load increase. Several fault-tolerant routing algorithms for NoCs are based on flooding. At *probabilistic flooding* [BDM07; DKM03], received packets are transmitted to all adjacent routers with probability p , and dropped with $1 - p$. Hence, packets are forwarded regardless of their destination addresses. The factor p , also referred to as the *gossip rate*, defines the amount of redundant packets which flood the network. *Directed flooding* [Pir+04] exploits the regular structure of most NoC topologies, by favoring productive directions over non-productive directions. *Redundant random walk* [Pir+04] also favors productive directions, but furthermore limits the number of copies to a predetermined value. There, messages are not dropped and follow non-deterministic paths. Pirretti et al. compared these three algorithms in [Pir+04] and concluded that flooding algorithms are only feasible at very low injection probabilities, due to the significant communication overhead. They reported that the overhead with redundant random walk is an order of magnitude lower than for the compared flooding algorithms, which are more fault resistant instead.

Routing algorithms which utilize spatial redundancy exploit the inherent path redundancy of NoCs. These algorithms avoid defect links and routers by choosing an intact path out of all existing paths. Even if *fault-tolerant source routing* [KK07] can be ascribed to the transport layer, most fault-tolerant routing algorithms are part of the network layer. In the following sections, a fault-tolerant routing algorithm refers to an algorithm which utilizes spatial redundancy at the network layer.

The kind of fault situations which are tolerable by such an algorithm depend on the routers' fault-awareness. Routing algorithms which use global fault information provide good performance, i.e. good routing decisions and a smaller number of unusable healthy nodes. However, this is achieved at the expense of an increased complexity, as additional wires or packets, as well as routing tables, are required to propagate and store fault information. The other extreme is strictly local fault information, which is optimal in terms of scale, but allows only non-optimal routing decisions in many cases. Thus, several routing algorithms use fault information in between these two extremes. There, local fault information is exchanged between N -hop neighboring nodes. Usually, N is a small number ($N \leq 2$) due to the increasing complexity at higher N .

Another important aspect of routing algorithms, independently from fault tolerance, is deadlock avoidance. As shown in Section 2.3, most wormhole routing based algorithms either use VCs or prohibit some specific turns, to guarantee the abstinence of routing

dependent deadlocks. Fault-tolerant routing algorithms have to be non-minimal adaptive, i.e. packets may be routed along non-shortest paths and routing decisions are based on the network's (fault-)state. Obviously, this complicates deadlock avoidance, as defect links and routers have to be avoided. Further, those failures can cause that other links and routers become essential to reach some destinations. Generally, most fault-tolerant routing algorithms also use VCs or routing restrictions for deadlock avoidance. Depending on the tolerated fault situations and the present fault information, however, even fully functional routers and links may have to be deactivated to achieve this. Deflection routing is a special case, since it is inherently deadlock free. Instead, livelocks have to be avoided, which is also complicated in case of failures. Section 4.2 gives an overview of fault-tolerant routing algorithms for NoCs which are based on deflection routing. Due to the comparatively small number of these algorithms, this overview is complete, to the best of my knowledge. In contrast, the amount of fault-tolerant routing algorithms that are not based on deflection routing is voluminous, which makes a complete overview difficult. Nevertheless, some instances of this category are listed below.

Wu presented in [Wu03] the *extended X-Y routing* algorithm. In the fault free case, this algorithm uses *X-first routing*. In case of faults, packets are routed in abnormal mode around the faulty region. To allow this, 2-hop fault information is required. Deadlocks are avoided in the abnormal mode by prohibiting some turns, based on the odd-even turn model (cf. Section 2.3.2).

Another fault-tolerant routing algorithm, used in the Vicis NoC [Fic+09b], is introduced by Fick et al. in [Fic+09a]. It uses negative first routing and is based on routing tables, which are updated in an offline phase. This approach exploits that some turns, which have been disabled at negative first due to deadlock avoidance, can be reactivated without engendering cyclic dependencies, i.e. routing dependent deadlocks, if other links are defect.

NS-FTR [PZ11] is based on the north-last and south-last turn models. At low fault rates, a single packet is transmitted with north-last. At high fault rates, two copies of a packet are routed, the original packet with north-last, and the copy with south-last. To avoid deadlocks, two separate virtual channels are required for the two packet types. Packets which can not propagate in a valid direction because of faults are dropped. To compensate this, a fault-free ACK network is assumed. Therefore, only local fault information is sufficient for this approach.

FT_DyXY [Val+10] tries to surround faulty regions and also requires two virtual channels for deadlock prevention. It is based on the non-fault-tolerant *DyXY* algorithm [LZJ06a], which uses stress values of adjacent nodes. Additionally, *FT_DyXY* utilizes local fault information.

4.1.3 Conclusion

Due to the variety of faults which can occur, an holistic approach is required to create fault-tolerant NoCs. This means, a combination of several fault tolerance methods with various kinds of redundancy and at different layers should be deployed. However, the work presented in this chapter is restricted to fault-tolerant routing, with the aim of tolerating permanent faults at link and router level. This means, fault detection, non-permanent faults, and faults at a finer granularity than link and router breakdowns are not considered herein, despite of the importance of these aspects. Nevertheless, the ideas and concepts presented in this chapter could be integrated in other router architectures, and existing fault tolerance and fault detection techniques could be integrated into the herein presented router architecture. A few of these existing methods are introduced in Section 4.2.

4.2 Related Work of Chapter 4

This section gives an overview of existing fault-tolerant and additionally deflection routing based router architectures, as well as the therein used routing algorithms. All these router architectures have in common that Nostrum (cf. Page 45) was chosen as basis router architecture. Thus, they use a crossbar as interconnection architecture, or at least they are not specifically designed for permutation networks.

Cost-based [KSR10; KR09; SRK10]: Kohler, Schley, and Radetzki developed a router architecture which utilizes the remaining functionality of partly defective routers. This is achieved by a fine grained fault model, an online fault diagnosis method, and a fault-adaptive routing algorithm. To detect errors, packets are equipped with an 8 bit Cyclic Redundancy Check (CRC) checksum. The fault diagnosis hardware uses this checksum to detect faults inside the crossbar, the routing logic, and the links. Initially, detected faults are considered as transient faults, which are handled by router-to-router retransmission. If the faults do not disappear, the faulty component is tested with specific test patterns to identify permanent faults. The fault states of the crossbar and the adjacent links are used by a fault-tolerant routing algorithm to avoid permanent faulty components. The routing algorithm itself distinguishes from other deflection routing algorithms by applying a cost function to find a cost-optimal routing decision [RK11]. This cost function penalizes deflections by one, reflections by two, and faulty directions by infinity. In contrast, most routing algorithms based on deflection routing allocate output ports from the highest prioritized flit to the lowest prioritized flit greedily. Hence, the existence of multiple productive output directions is not exploited by these algorithms, resulting in potentially non-optimal routing decisions. Finding cost-optimal routing decisions requires additional hardware, but also improves the routing decisions'

quality.

To summarize, in contrast to all other deflection routing based approaches, this is the only holistic approach, which includes fault diagnosis. The fault model on a finer granularity enables graceful performance degradation and avoids the immediate shutdown of defect routers. Furthermore, the cost-based routing improves the routing decisions' quality.

FoN [Fen+10b]: Feng et al. presented the Fault-on-Neighbor (FoN) aware deflection routing algorithm, which uses 2-hop fault information. Compared to the cost-based approach, the primary focus of FoN is on tolerating more complex fault situations, which can arise at higher fault rates. Toward this end, adjacent routers exchange their local fault information. A router's local fault information consists of 4 bit, representing the states of the router's links to the four compass directions. Every router transmits its own local fault information to every adjacent router and receives their local fault information in return. Thus, two adjacent routers exchange 8 bit of fault information. The 2-hop fault information is a compromise between strictly local and global fault information. FoN's fault model only considers completely broken or fully functional links and routers. The basic idea of FoN is to assign packets to a productive output direction, from the highest prioritized packet to the lowest prioritized packet. If two productive directions exist, the direction with the lower stress value, i.e. the lower utilized link, is chosen. If faults prevent a routing decision, the 2-hop fault information is used to determine the best alternative path towards the destination.

With 2-hop fault information, FoN is capable of tolerating faulty regions which do not contain two sequential concave points. Here, concave points correspond to 270° corners on the boundary of a faulty region. Two fault shapes which both contain such concave points are depicted in Figure 4.1.

Feng et al. reported that the cost-based approach achieves a slightly higher saturation throughput than FoN in the fault free case. However, in the case of link failures, the throughput with FoN is 13% higher on average, due to expanded fault information. The authors further reported that the maximum frequency of FoN is approx. three times higher than that of the cost-based approach, whereas the required area or a router is only $\approx 45\%$ of a cost-based router.

FTDR / FTDR-H [Fen+10a]: The fault-tolerant deflection routing (FTDR) algorithm, also developed by Feng et al., is based on Q-learning. In contrast to FoN, FTDR is not limited to special fault patterns. Every router is equipped with a routing table which stores estimated distances, referred to as Q-values, to all other routers in the network, for all four output directions. The table entries are initialized with Manhattan distances and infinity for broken links. Further, routers response to received packets with their own minimum Q-value for the received packet's destination address. These

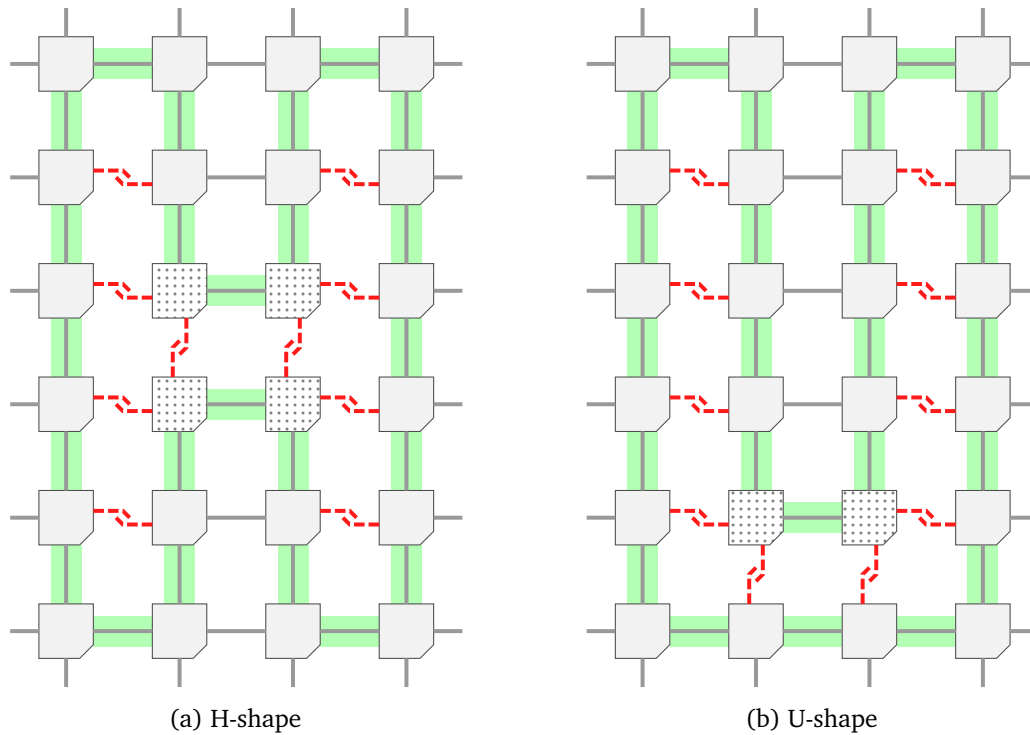


Figure 4.1: Two fault shapes which are not tolerated by FoN. Boundaries of the faulty regions are highlighted in green. Router which correspond to concave points are filled with a dotted pattern.

Q-values are transmitted over dedicated wires. Thereby, routers receive a Q-value for each sent packet and gradually learn the best route to every destination. Additionally to the routing tables, 2-hop fault information is used, which reduces the average hop count, compared to just local fault information.

The authors assumed a size of 6 bit for a Q-value, independent of the used NoC dimension. Thus, the size of each routing table is $N \cdot 4 \cdot 6$ bit, whereas N denotes the number of routers in the network. Additionally, 12 bit are required to exchange the Q-values between a router and one of its neighbored routers. As the routing tables' size increases with the NoC dimension, FTDR is only suitable for small and medium-sized networks. To mitigate the large overhead of the routing tables, the same authors also developed FTDR-H [Fen+10a], which is an hierarchical version of FTDR. At FTDR-H, the mesh is divided into several regions of fixed sizes, and each router contains a local as well as a region routing table. The local routing table is used if a packet's destination router is located in the same region as the current router. If this is not the case, the region routing table is used. The application of a hierarchical routing table reduces

the required overhead, in particular, for large NoC dimensions, but also impairs the fault tolerance qualities. FTDR-H can not deal with complex fault situations that span multiple regions of the region routing tables.

An improved version of FTDR-H, based on differential Q-routing, is presented by Radetzki in [Rad11a; Rad11b]. Differential Q-routing exploits that the expected minimal distances (Q-values) correspond to Manhattan distances in the fault free case and that a 2D mesh topology is deployed. Therefore, it is sufficient to store and exchange only ΔQ -values, which give the deviation of Q-values and Manhattan distances. Furthermore, the structure of the global and local routing tables is improved. Instead of fixed global routing tables, the network is individually segmented for each router. Every router is equipped with a routing table, which contains entries for neighbored routers with a maximum distance of two hops, as well as entries for the eight compass directions *N*, *NE*, *E*, *SE*, *S*, *SW*, *W*, and *NW*. These improvements enable better performance in case of faults and additionally 40% of area savings compared to the original FTDR-H algorithm.

FTDR for a 3D mesh based version of Nostrum is presented in [Fen+11b]. Besides the four ports related to the 2D mesh, every router has two vertical Through-Silicon Via (TSV) ports to connect adjacent routers in the upper and lower layer. Packets are routed on the same layer towards their destination first, using the 2D FTDR algorithm. If the destination's *x* and *y* coordinates are reached, the packet is routed across the layers. In the case of a faulty TSV link, the current router sets a temporary address field in the flit structure to route the packet to an intermediate router with a healthy vertical link. In order to find such an intermediate router, every routers is equipped with two TSV state vectors, which record the fault state of the up and down links of the corresponding layer. These two vectors are exchanged between adjacent routers of the same layer, and thereby, every router learns the fault state of all vertical links of its own layer.

In [Fen+13], Feng et al. extended their FTDR algorithm and the corresponding router architecture by (1) a link-level error control scheme, (2) an online fault diagnosis mechanism, and (3) a test process. The fault diagnosis mechanism is based on Single-Error Correcting and Double-Error Detecting codes (SECDED codes) to distinguish between transient and permanent faults. Segments of the packets' head and payload information are decoded with two and five different SECDED codes, respectively. Hence, the link-level error control scheme can correct between one and seven single-bit errors and detect between two and 14 faulty bits. Double-bit errors are solved by retransmission. If a double-bit error holds for more than one cycle, the link is checked by a test process to detect permanent faults and deactivated if necessary.

In this section, several fault-tolerant and additionally deflection routing based router architectures have been introduced. These architectures are designed to tolerate different fault types. Furthermore, the complexity of tolerable fault situations varies from single faults to very complex fault situations, which are caused by a combination of several

faults. The main focus of the FaFNoC router architecture, which is presented in the next section, is on tolerating permanent faults at link and router level, as well as potentially arising complex fault situations. The router architectures FON, FTDR, and FTDR-H are selected for comparisons herein, as these architectures are able to tolerate complex fault situations.

4.3 FaFNoC Router Architecture

In this section, the fault-tolerant FaFNoC router architecture is introduced, which has been developed as part of this thesis. An overview of this architecture is depicted in Figure 4.2. As mentioned in Section 3.4, it is partially based on the non-fault-tolerant CHIPPER architecture. As such, deflection routing is used and a permutation network is deployed. In contrast to CHIPPER and the non-fault-tolerant router architecture used in the rest of this thesis, a Benes network is used instead of a Banyan network and an additional component for fault tolerance, the fault-status-handler, is deployed. The utilized 4×4 Benes network consists of one additional stage of switching elements, which provides benefits in terms of fault tolerance. More information on the deployed permutation network can be found in Section 4.3.2.

The herein used fault model takes permanent, bidirectional link failures into account, which can be caused, for example, by wearout mechanisms. It is assumed that every router has a 4 bit wide fault information input f_i , representing the states of the router's links to the four compass directions N , E , S , and W (see Figure 4.2). Even if fault detection is not considered herein, existing fault detection mechanisms, such as the mechanisms presented in [KSR10; Fen+13], can be used to obtain this information. Link failures are considered as bidirectional, as a basic assumption of deflection routing is an equal number of input and output ports. If a complete router fails, all links connected to a router are regarded as faulty.

To be able to tolerate permanent link and router failures, two key aspects have to be ensured:

1. Faulty links have to be avoided.
2. A solution to overcome more complex fault situations and avoid livelocks is required.

The required adjustments to a permutation network to avoid faulty links are discussed in the subsequent Sections 4.3.1 and 4.3.2. To overcome complex fault situation, FaFNoC uses the concept of FaF, which is introduced in Section 4.3.3.

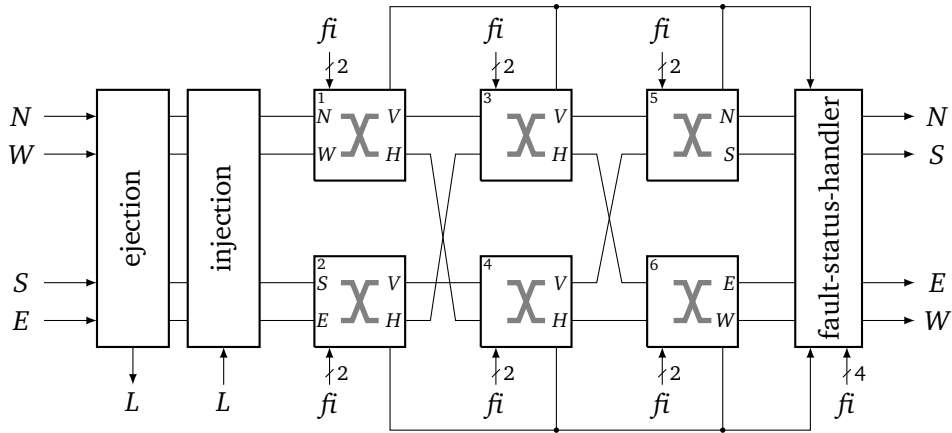


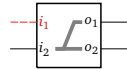
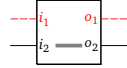
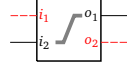
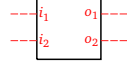
Figure 4.2: Overview of the FaFNoC router architecture.

4.3.1 Fault Tolerance and Banyan Networks

The non-fault-tolerant router architecture used in the rest of this thesis utilizes a 4×4 Banyan network as interconnection architecture. In contrast, the fault-tolerant FaFNoC architecture uses a 4×4 Benes network, as depicted in Figure 4.2, which consists of one additional stage of switching elements compared to a 4×4 Banyan network. As a motivation for this design decision, the drawbacks of Banyan networks in terms of fault tolerance are emphasized in this section. To be able to demonstrate the resulting consequences for a router architecture, the required adjustments to avoid faulty links at Banyan network must be known. Thus, the required adjustments are shown first, and afterwards the resulting consequences are presented.

Compared to a crossbar based router architecture, avoiding faulty links at permutation networks is more complicated, as prioritization, routing, and allocation are performed individually by the switching elements. At permutation networks, there is no central component, like a central switch allocator (cf. Figure 2.12), which allocates the output ports. Hence, a decentralized implementation of the fault tolerance concept is desirable.

To illustrate the state $z(s_i)$ of a switching element s_i which is connected to a faulty link, the following symbols are used, in addition to the symbols depicted on Page 48:

	Required permutation	Cause / defect ports	State	Symbol
1)	arbitrary permutation	one input or one output is defect	$z(s_i) = \{0, 1\}$	
2)	identity permutation	i_j and o_j are defect, $j \in \{1, 2\}$	$z(s_i) = 0$	
3)	cross permutation	i_j and $o_{j \bmod 2+1}$ are defect, $j \in \{1, 2\}$	$z(s_i) = 1$	
4)	arbitrary permutation	both inputs or both outputs are defect	$z(s_i) = \{0, 1\}$	

The first symbol is used if only a single input or a single output of s_i is defect. In this case, an arbitrary permutation can be performed, but it has to be ensured that the faulty input or output is not used. At the second and the third symbol, an identity permutation and a cross permutation are required to avoid the usage of the faulty input and output, respectively. The fourth symbol is used if both inputs and/or both outputs are faulty, and hence, the switching element's state is irrelevant, as no flits can arrive at this element. These symbols are used horizontally and/or vertically flipped if i_2 is faulty instead of i_1 or the outputs are defect instead of the inputs. Hereinafter, a switching element with no faulty input or output, at least one faulty input or faulty output, and two faulty inputs or two faulty outputs is denoted as *fully functional switching element*, *partially functional switching element*, and *faulty switching element*, respectively.

The fault situations which can occur at a 4×4 permutation network can be categorized by the number of faulty links. As link failures are considered as bidirectional, a faulty link from router R_a to router R_b always entails an unusable link from R_b to R_a , regardless of this link's status. Consequently, links are deactivated pairwise, or rather, the links of a compass direction are deactivated. In the following, an n -direction(s) fault situation denotes a situation at which exactly n directions of a router are faulty, and $n \in \{0, 1, 2, 3, 4\}$. Hence, a zero-direction and a four-directions fault situation correspond to a fully functional and a faulty router. Both situations are easy to handle, as either the router operates in normal mode or is completely deactivated. The required adjustments for the other fault situations are described in the following. Figure 4.3 shows all $\binom{4}{1} = 4$ existing one-direction fault situations of a Banyan network. All these situations have in common that the state of the two partially functional switching elements has to correspond to an identity permutation, as a faulty link might be used otherwise. Furthermore, the two remaining switching elements can perform an arbitrary permutation. The reason for this characteristic will be explained on the example of a faulty north direction (cf. Figure 4.3a), but the same reasons apply for all one-direction fault situations. If the north direction is faulty, the switching element s_3 , i.e. the upper

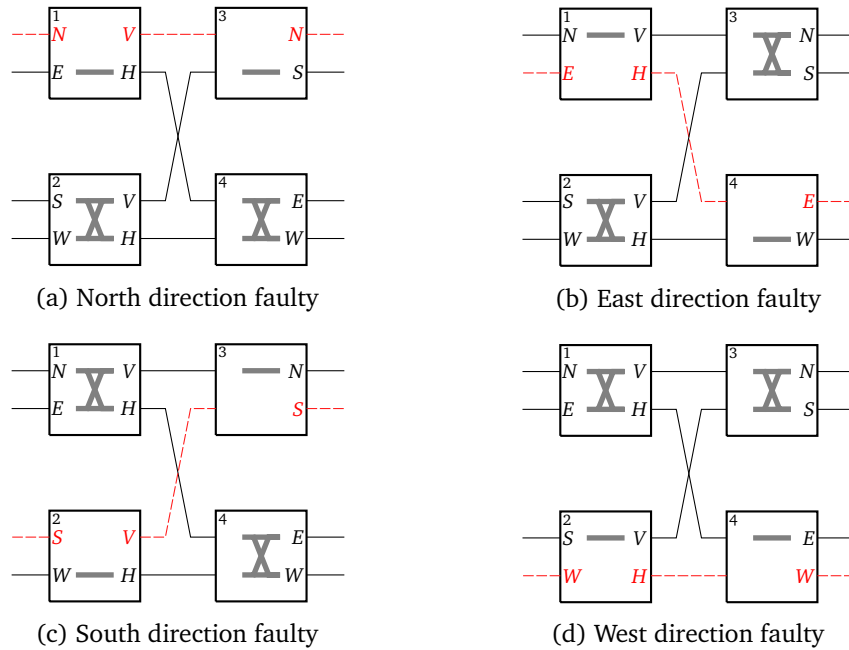


Figure 4.3: All one-direction fault situations of a Banyan network.

right element, has only one functional output port, the south. Thus, at any time, at most one flit is allowed to arrive at s_3 . However, it has to be expected that one flit arrives from the fully functional switching element s_2 , as two flits can arrive at s_2 and in this case one flit has to be forwarded to s_3 . Hence, the partially functional switching element s_1 is not allowed to transmit a flit to s_3 . From this follows that the state of s_1 has to be 0, i.e. an identity permutation. Furthermore, the state of s_3 also has to correspond to an identity permutation, as otherwise an arriving flit, which only can be received from s_2 , would be transferred to the faulty north direction. In conclusion, at every one-direction fault situation, the state of both partially functional switching elements has to correspond to an identity permutation, as otherwise a flit could be transferred to a faulty link.

The $\binom{4}{3} = 4$ three-directions fault situations of a Banyan network are depicted in Figure 4.4. If three out of four directions are faulty, only a single routing decision remains: the only functional input port has to be connected to the associated output port. At the herein used input and output port order of the permutation network, which has been adopted from CHIPPER, an input port is connected to its associated output port if and only if the state of all involved switching elements corresponds to an identity permutation. Hence, all three-directions fault situations have in common, that the state of the two partially functional switching elements has to be 0, i.e. correspond to an identity permutation. Furthermore, two faulty switching elements exist which can be in

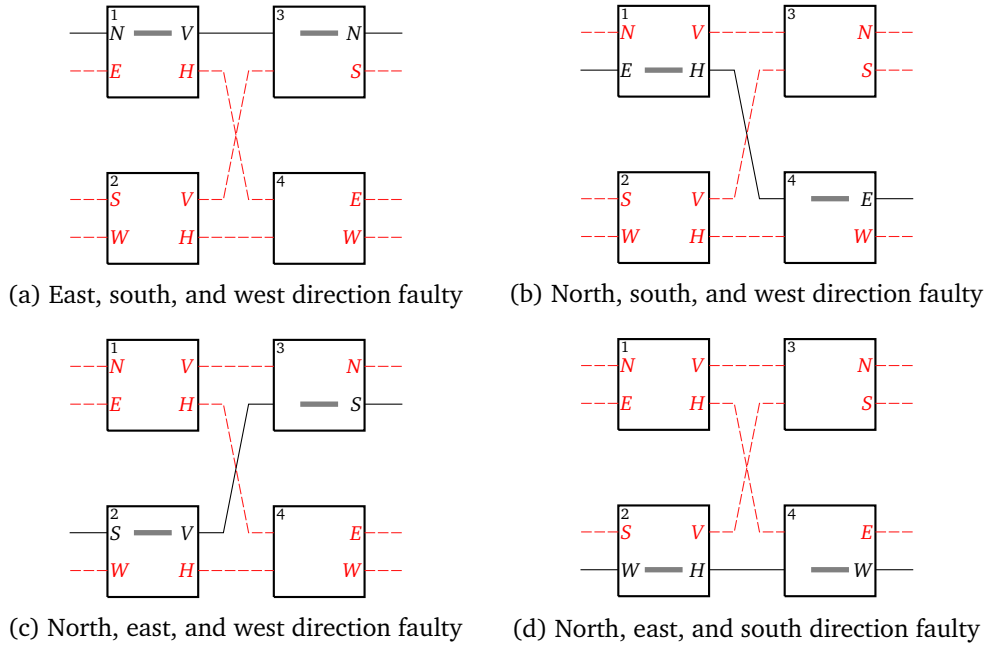


Figure 4.4: All three-directions fault situations of a Banyan network.

an arbitrary state as no flits can arrive at these elements.

In terms of fault tolerance, the two-directions fault situations are more complicated. At these situations, two routing decisions remain. In total there are $\binom{4}{2} = 6$ two-directions fault situations. As a Banyan network consists of four switching elements, it is obvious that at least two out of the six possible two-directions fault situations can not be handled by a single switching element. This means, some kind of coordination between the switching elements is required at these two situations. Therefore, all two-directions fault situations can be divided into unambiguous and ambiguous fault situations. Figure 4.5 shows the four unambiguous fault situations, which all consist of one fully functional switching element, two partially functional switching elements, and one faulty switching element. The two partially functional elements have to be in the state for an identity permutation, due to the same reasons already explained at one-direction fault situations. Thus, the only fully functional switching element, which can be in an arbitrary state, decides between the two remaining routing decisions.

The two ambiguous fault situations are depicted in Figure 4.6. At these two situations, all four switching elements are partially functional, i.e. they are connected to one faulty input or output. Here, also two routing decisions remain. For instance, if the north and the west direction are faulty (cf. Figure 4.6a), the two remaining routing decisions

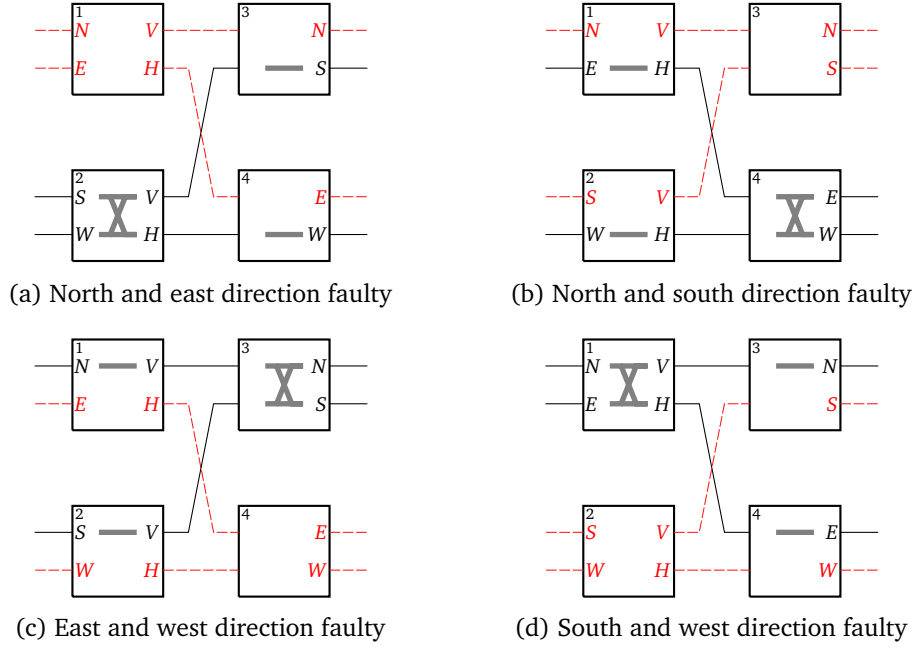


Figure 4.5: All unambiguous two-directions fault situations of a Banyan network.

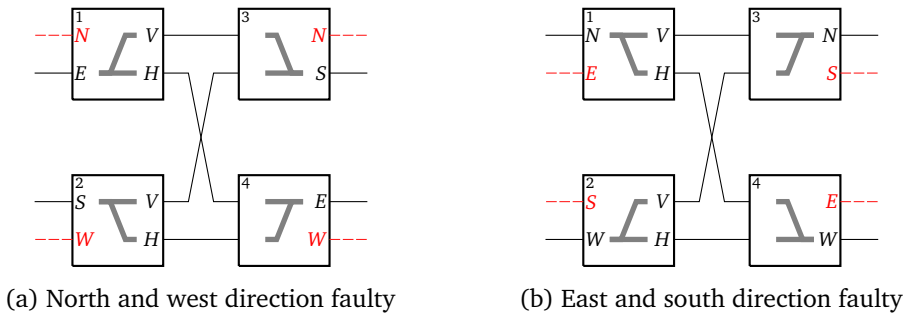


Figure 4.6: All ambiguous two-directions fault situations of a Banyan network. Two different configurations exist for these two situations.

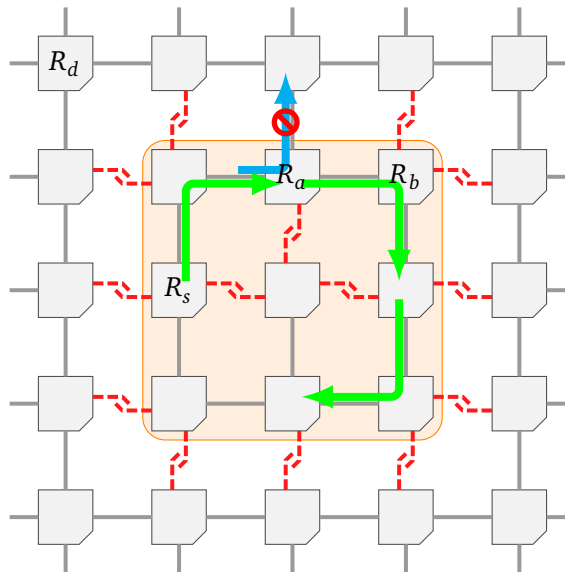
are¹⁵ $E \rightarrow S$, $S \rightarrow E$ and $E \rightarrow E$, $S \rightarrow S$. Please note that a reflection of flits (e.g. $E \rightarrow E$, $S \rightarrow S$) can be required to overcome complex fault situations, even if this permutation seem useless at first glance. Bad routing decisions, which cause that flits are routed into a wrong direction, have to be reversible. Such routing decisions can not be avoided without global fault information, and they might require a reflection to undo them.

For both remaining routing decisions, all four switching elements have to be in the same state, either $z(s_i) = 0$ for $E \rightarrow E$, $S \rightarrow S$, or $z(s_i) = 1$ for $E \rightarrow S$, $S \rightarrow E$, $i \in \{1, 2, 3, 4\}$. If one switching element is in a different state than the other elements, a faulty link would be used. Thus, these two situations complicate the fault handling, due to the following reasons: At these two situations, every switching element has to evaluate the complete fault information f_i , whereas local fault information for the two input ports is sufficient for all other fault situations. Furthermore, every switching element has to coordinate its routing decision with other switching elements. In particular, s_1 has to consider the state of s_2 , and vice versa, which increases the critical path of the Banyan network.

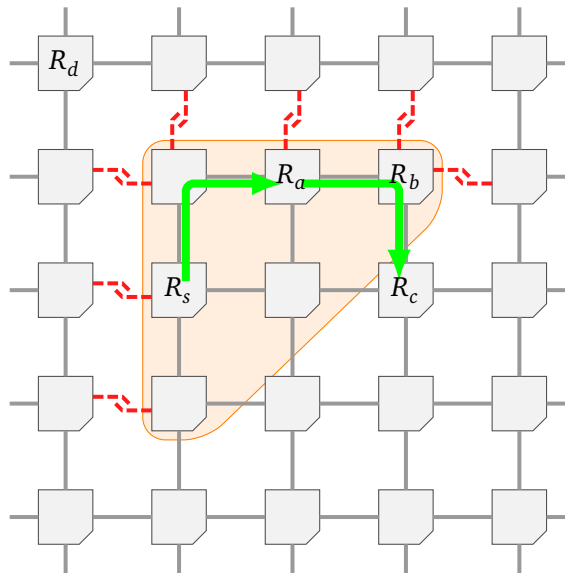
In summary, at all one-direction fault situations, at all three-directions fault situations, and even at four out of the six possible two-directions fault situations, it is sufficient to set the state of partially functional switching elements to 0, i.e. to the identity permutation, to avoid faulty links. This can be implemented in a decentralized manner and every switching element has to consider just local information. This does not apply to the two two-directions fault situations of Figure 4.6, which require a coordination between switching elements.

Besides the requirement for coordination, Banyan networks are internal blocking, i.e. not every permutation between all input and output ports is possible. In the fault-free case, the internal blocking characteristic might be acceptable, as a Banyan network has low hardware requirements and, even more importantly, it is guaranteed that the highest prioritized flit can be transferred to a productive direction. However, if links fail, this can not be guaranteed anymore. This aspect is illustrated by means of a short example. Figure 4.7a shows a 2D mesh of routers and the dashed red lines represent faulty links. In the depicted fault situation, every path between source router R_s and destination router R_d includes the north link of router R_a . In particular, every shortest path between R_s and R_d contains R_a , whereby a flit arrives at R_a from the west and should be forwarded to the north. Unfortunately, this turn from the west to the north is not possible, as the link in the south of R_a is faulty. Even if the link in the south is not required for this transmission, the states of both switching elements s_2 and s_3 of R_a are predefined to $z(s_2) = 0$ and $z(s_3) = 0$ (cf. Figure 4.3c), i.e. to the identity permutation, due to this fault situation. Consequently, flits arriving at R_a from the west can only be transmitted to the east or be reflected to the west. Nevertheless, every path from R_s to R_d has to include R_a 's north link. As this link is only reachable for flits which arrived at

¹⁵ $A \rightarrow B$ means, a flit from input port A is forwarded to output port B .



(a) Livelock occurs if routers never reflect flits



(b) Reflections might cause livelocks

Figure 4.7: Two fault situations, the first situation requires a reflection of flits, whereas reflections create a risk of livelocks at the second situation.

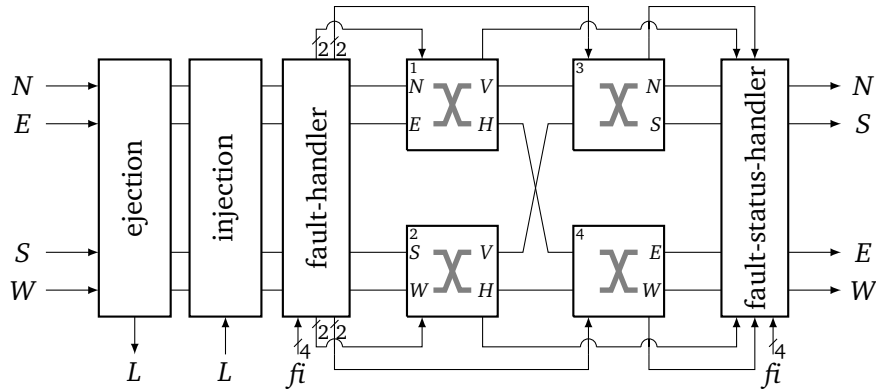


Figure 4.8: Overview of the FaFNoC router architecture based on a Banyan network.

R_a from the east, every path has to include router R_b . This means, if R_s transmits a flit with destination R_d to the north, which is reasonable, as this corresponds to a shortest path, the flit has to be reflected eventually. Otherwise, the flit would be routed on a circular path in clockwise direction inside the orange highlighted area, as indicated by the green arrows in Figure 4.7a. Figure 4.7b shows a second, different fault situation. However, from R_b 's perspective, both fault situations depicted in Figure 4.7 do not differ from each other. At Figure 4.7b, a reflection of flits from R_b back to R_a creates the risk of livelocks. In this fault situation, solely greedy, destination-oriented routing decisions are not sufficient to leave the orange highlighted area. Instead, a flit with destination R_d has to be routed intentionally away from R_d , to reach R_d finally. In conclusion, Banyan networks require the reflection of flits to overcome some fault situations and avoid livelocks, since some routing decisions can become impossible due to link failures. On the other hand, this complicates the design, as reflections in turn can lead to livelocks, or even the oscillating of flits between adjacent routers.

FaFNoC based on a Banyan network

As mentioned before, a Benes network, instead of a Banyan network, is deployed as interconnection architecture at FaFNoC. Nevertheless, a Banyan network based architecture was developed as part of this thesis, to justify the design decision and to show the resulting benefits and drawbacks. Hereinafter, the Banyan network and Benes network based versions of FaFNoC are referred to as FaFNoC-Banyan and FaFNoC-Benes, respectively, where necessary. An overview of FaFNoC-Banyan is depicted in Figure 4.8. Besides the deployed permutation network, the most significant differences to FaFNoC-Benes are (1) the *fault-handler* and (2) the *return flag*.

Instead of adding complex logic to all four switching elements, a central fault-handler is deployed, which adjusts the switching elements' states in case of faulty links. Towards

this end, the fault-handler has a 4 bit wide fault information input fi , which provides the functional capability of the links for the four compass directions. Furthermore, the fault-handler has four 2 bit wide fault control outputs fc_i , $i \in \{1, 2, 3, 4\}$, whereas the output fc_i controls the behavior of the switching element s_i . At $fc_i = 00$, switching element s_i operates according to its routing algorithm (normal operation). At $fc_i = 01$, the state is set to $z(s_i) = 0$ and at $fc_i = 10$ to $z(s_i) = 1$. If at least one faulty link exists, the fault-handler takes over the routing decision by setting its fault control outputs fc_i according to Figures 4.3 to 4.6. This means that the switching elements connected to a faulty link have to perform an identity permutation, except for the two situations depicted in Figure 4.6. In those two situations, the fault-handler prevents a reflection of flits, unless a flit's return flag is set, as described in the following.

As shown before, reflecting flits can cause livelocks. Nevertheless, FaFNoC-Banyan requires reflections to overcome some complex fault situations, since the deployed 4×4 Banyan network is internal blocking. To avoid livelocks, the used routing algorithm avoids reflections, which means that flits are not routed back to their arriving direction. Nevertheless, if a flit's productive output direction is occupied by a higher prioritized flit, the flit can be reflected. Furthermore, the flit structure of FaFNoC-Banyan is extended by one single bit, referred to as return flag, to allow productive reflections. If a router R_y can not transmit a flit to its healthy productive direction due to a faulty link, as the case at router R_a in Figure 4.7a, the current router R_y sets the flit's return flag to 1. If this bit is set, a receiving router R_z tries to reflect the flit back to the sending router R_y . Hence, the flit arrives at router R_y again one network clock cycle later, but from a different direction, which allows to transmit the flit to its productive direction. An example of the return flag handling is depicted in Figure 4.9. The return flag handling is implemented in the fault-handler and fault-status-handler. The fault-handler adjusts the switching elements' states in case the highest prioritized flit's return flag is set. The fault-status-handler, which is explained in Section 4.3.3, sets and clears this bit, dependent on the switching elements' states.

4.3.2 Substitute Benes Networks for Banyan Networks

FaFNoC utilizes a permutation network as interconnection architecture, because of the benefits mentioned in Section 3.4. However, as shown in the last section, Banyan networks suffer from three problems when it comes to fault tolerance:

1. They are internal blocking, i.e. not every permutation between all input ports and output ports is supported.
2. Coordination between switching elements is required, e.g. by a central component like the fault-handler.
3. Link failures can cause that fully functional links are unusable.

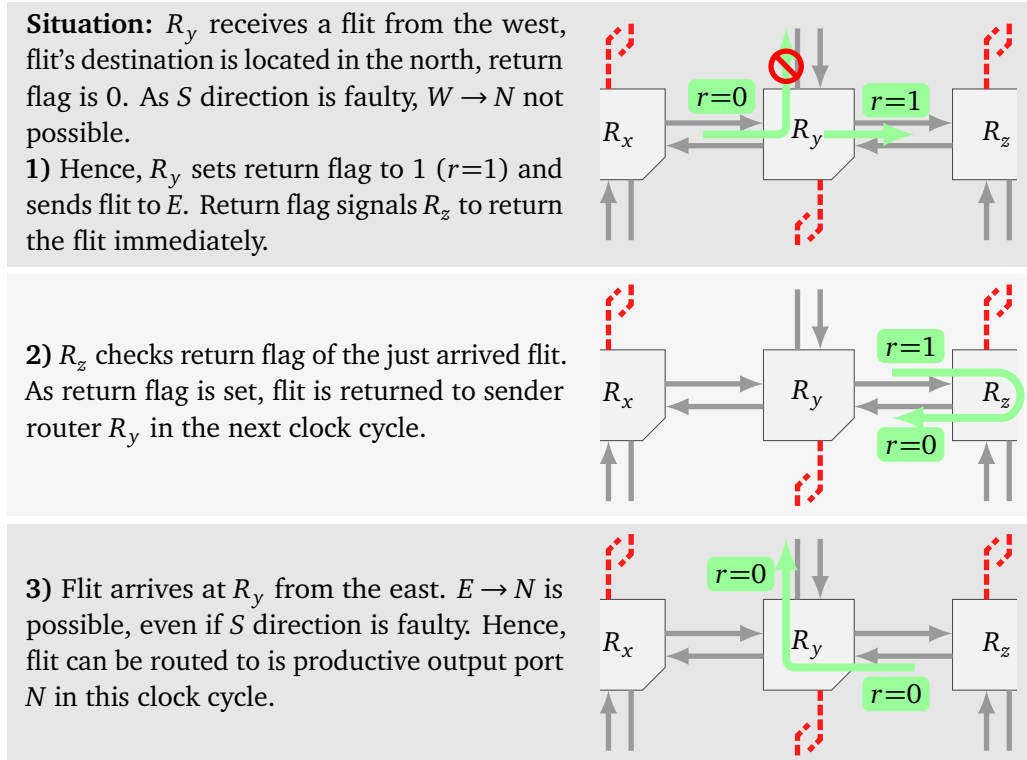


Figure 4.9: Return flag handling of FaFNoC-Banyan illustrated.

The second problem prevents a decentralized implementation of the permutation network, whereas the other two problems might impair the performance of a faulty NoC. Non-fault-tolerant router architectures based on a Banyan network, like CHIPPER, are also internal blocking, which means that some routing decisions are not supported. However, the influence on the achievable performance of a partially faulty NoC can be significantly higher than that of a fully functional NoC, as the number of possible routing decisions, and the number of paths between the routers, is already reduced at a faulty network. Moreover, the third problem of the list above is a consequence of the internal blocking characteristic as well as the required adjustments to the states of the switching elements.

To avoid these three problems of Banyan networks, a Benes network is deployed at the fault-tolerant FaFNoC router architecture. In this section, it is shown that Benes networks provide a solution for all three problems of Banyan networks. To this end, firstly, several important properties of permutation networks are presented, and it is shown that some properties of CHIPPER's Banyan network should not be adopted. Secondly, Banyan networks and Benes networks are compared theoretically. Thirdly, it

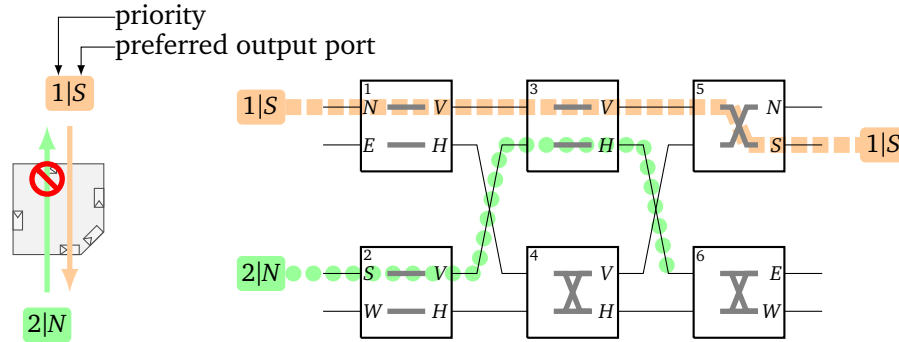


Figure 4.10: A 4×4 Benes network with configuration adopted from CHIPPER's Banyan network. The used configuration is the reason for the depicted deflection.

is demonstrated that the deployed Benes network allows an efficient implementation of the fault tolerance concept.

Configuration of the deployed Benes network An important property of a permutation network is the arrangement of the switching elements' I/O directions, hereinafter referred to as the *configuration* of a permutation network. Figure 4.10 shows a Benes network with the same configuration as used at CHIPPER's Banyan network. There, the input ports of the permutation network, which are the input ports of s_1 and s_2 , are arranged according to the compass directions. The output ports of the permutation network, which are the output ports of s_5 and s_6 , are arranged according to the vertical axis (s_5) and the horizontal axis (s_6). According to the compass directions, the vertical axis consists of N and S , and the horizontal axis consists of E and W . The switching elements s_1, \dots, s_4 route flits with a productive direction in the north or in the south to their first output port V , which is the vertical axis, and flits with a productive direction in the east or in the west to their second output port H , which is the horizontal axis. As explained in Section 3.3.2, this particular configuration allows an efficient implementation of the routing algorithm. Furthermore, this configuration enables that the most frequent permutation is supported by the Banyan network, even though it is internal blocking. However, as Benes networks are rearrangeable non-blocking, this particular configuration might not be necessary. Indeed, at a Benes network, CHIPPER's configuration leads to undesirable deflections of flits, as depicted in Figure 4.10. There, both arriving flits, the orange one and the green one, have a destination along the vertical axis. Hence, both flits are routed to s_3 , where they collide. Switching element s_3 can transfer only one flit to the vertical axis, and thus the lower prioritized flit is deflected to the horizontal axis. Figure 4.11 shows the Benes network which is deployed at FaFNoC-Benes. An explanation why this configuration was chosen follows after the next paragraph.

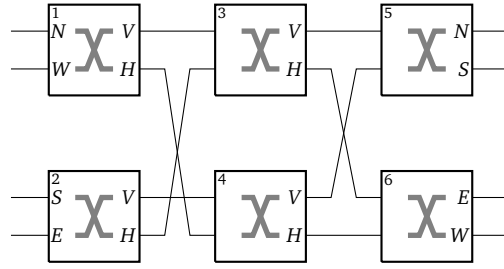


Figure 4.11: FaFNoC's Benes network, with a changed arrangement of input ports as well as a changed wiring between the first and the second stage of switching elements.

Classification and limitations of Benes networks Both Banyan networks and Benes networks are multistage logarithmic networks. In contrast to Banyan networks, which consist of $\text{ld}(N)$ stages of $\frac{N}{2}$ switching elements¹⁶, Benes networks consist of $2 \cdot \text{ld}(N) - 1$ stages of $\frac{N}{2}$ switching elements, whereby N denotes the number of I/O ports [LL04, p. 12]. Specifically, this means that the deployed 4×4 Benes network consists of six switching elements, which are arranged in three stages (cf. Figure 4.11). On account of the additional stages of switching elements, Benes networks are *rearrangeable non-blocking, multi path* networks, whereas Banyan networks are internal blocking, single path networks. At a multi path network, multiple paths between the network's I/O ports exist, whereas only one predefined path exists at a single path network. At a 4×4 Benes network, there are exactly two different paths between every input port and every output port. Rearrangeable non-blocking means that the network is capable of connecting any input port to every free output port, but existing connections may have to be rearranged to this end. However, at FaFNoC, only single flits are routed and connections last only one network clock cycle. Hence, there are no connections which have to be rearranged, as deflection routing is used. On the other hand, the configuration of a Benes network influences the possible permutations, e.g. $S \rightarrow N$, $N \rightarrow S$ is not possible with the configuration depicted in Figure 4.10. Therefore, it has to be shown that the configuration used at FaFNoC-Benes, which is depicted in Figure 4.11, is non-blocking, i.e. all $4! = 24$ permutations are supported. This can be shown by a case distinction. At every permutation, two flits destined for the vertical axis, referred to as V -flits, and two flits destined for the horizontal axis, referred to as H -flits, exist. In the first case, it is assumed that one V -flit and one H -flit arrive at both switching elements s_1 and s_2 . In this case, no deflection occurs at s_1 and s_2 , independently of the flits' priorities, as both switching elements can route both flits to their V and H output port. Consequently, at s_3 and s_4 exactly one H -flit and one V -flit arrive. Thus, also at these two switching elements no deflection occurs. Hence, both V -flits arrive at s_5 , and

¹⁶Herein, it is assumed that they are constructed using 2×2 switching elements.

Routing decision	Banyan	Benes	crossbar
$N \rightarrow S, S \rightarrow N$	✓	✓	✓
$N \rightarrow S, E \rightarrow N$	✗	✓	✓
$N \rightarrow S, E \rightarrow S \vee W$	✗	✗	✓
$N \rightarrow S, E \rightarrow S$	✗	✗	✗

Table 4.2: Possible and impossible routing decisions of different interconnection architectures.

both H -flits arrive at s_6 . As one of the V -flits is destined for N and the other one for S , also at s_5 no deflection occurs. The same holds for both H -flits which arrive at s_6 . Thus, the deployed Benes network supports every permutation according to the assumption that one V -flit and one H -flit arrive at s_1 and s_2 . In the second case, it is assumed that the flits arrive not as assumed in the first case, i.e. both V -flits or both H -flits arrive at s_1 . Hence, at s_2 , also two H -flits or two V -flits arrive. As s_1 and s_2 have one H and one V output port, only the higher prioritized flit can be routed to its productive direction, and the lower prioritized flit has to be deflected. However, as s_1 transmits flits destined for the V axis to s_3 , and s_2 transmits such flits to s_4 , exactly one V -flit and one H -flit arrive at s_3 and s_4 . Hence, no deflection occurs at these second stage switching elements and therefore also not at the third stage switching elements. In conclusion, the deployed Benes network is non-blocking, i.e. it supports every permutation.

Since the deployed Benes network is non-blocking, an increased number of routing decisions is supported, compared to a Banyan network, which is internal blocking. To demonstrate the limitations of Benes networks, in particular, compared to crossbars, Table 4.2 shows several routing decisions and if they are possible with a Banyan network, a Benes network, and a crossbar. At all four routing decisions, two flits arrive at a router, and it is assumed that the flit arriving from the north has a higher priority. The first routing decision is possible with all interconnection architectures, which means that both incoming flits can be routed to their preferred output port. The second routing decision is not possible with a Banyan network, as s_1 can transfer only one flit to the vertical axis, and hence, it has to deflect the lower prioritized flit to the horizontal axis. At the third routing decision, it is assumed that the second flit, which arrives at S , has two productive output ports, E and W . Switching elements, however, have to select one of the two productive directions, as they can transmit a flit either to the vertical axis or to the horizontal axis. Here, it is assumed that the vertical axis is prioritized, i.e. Y -first routing is used. Unfortunately, this leads to a collision at s_5 , as a Benes network has no central controller. The last routing decision is not possible with any interconnection architecture, as both flits have only one productive output port, which is the same port for both flits.

Fault tolerance and Benes networks As Benes networks belong to the group of multi path networks, several configurations of a 4×4 Benes network exist which solve the before mentioned problems of Banyan networks. To justify the used configuration, some desirable characteristics of a permutation network are shown first. Afterwards, it is shown how the permutation network is constructed, and that the desirable characteristics are achieved.

First of all, the permutation network should support as many routing decisions as possible, in particular, under fault conditions. Furthermore, an efficient implementation of the routing algorithm should be supported. This includes that no central component, or coordination between switching elements, is required. The switching elements should operate completely independently from each other, using only local information, i.e. using only the two received flits and fault information for the two connected input ports.

The herein used Benes network is constructed as follows:

1. The switching elements' output port arrangement is adopted from CHIPPER, as this allows an efficient implementation of the routing algorithm, due to the reasons explained in Section 3.3.2.
2. The wiring between the first and the second stage of switching elements is changed to avoid collisions at the second stage (cf. Figure 4.10). This corresponds to a permutation of the two output ports of s_2 . Hence, s_1 and s_2 send flits, with a destination along the same axis, to different switching elements.
3. The input port direction are arranged in a way that every input port is connected to its corresponding output port if all switching elements perform an identity permutation: $d \rightarrow d, \forall d \in \{N, E, S, W\} \Leftrightarrow z(s_i) = 0, \forall i \in \{1, 2, 3, 4\}$.

The third point simplifies the required adjustments to avoid faulty links. At a Banyan network, every switching element which is connected to a faulty link has to perform an identity permutation, except in the case of the two fault situations depicted in Figure 4.6. This ensures that a faulty input port, i.e. an input port connected to a faulty link, is connected to its corresponding output port, and hence, the faulty link can not be used by flits from other input ports. Since there is only a single path between two arbitrary input and output ports of a Banyan network, these adjustments are unambiguous. In contrast, multiple paths between an input port and output ports exist at a Benes network, as those networks belong to the group of multi path networks. Nevertheless, every switching element has to determine its state by using only local information to enable a decentralized implementation of the fault tolerance concept. The third point ensures that all input ports are connected to their corresponding output ports if all switching elements perform an identity permutation. Thereby, it is also ensured that an arbitrary

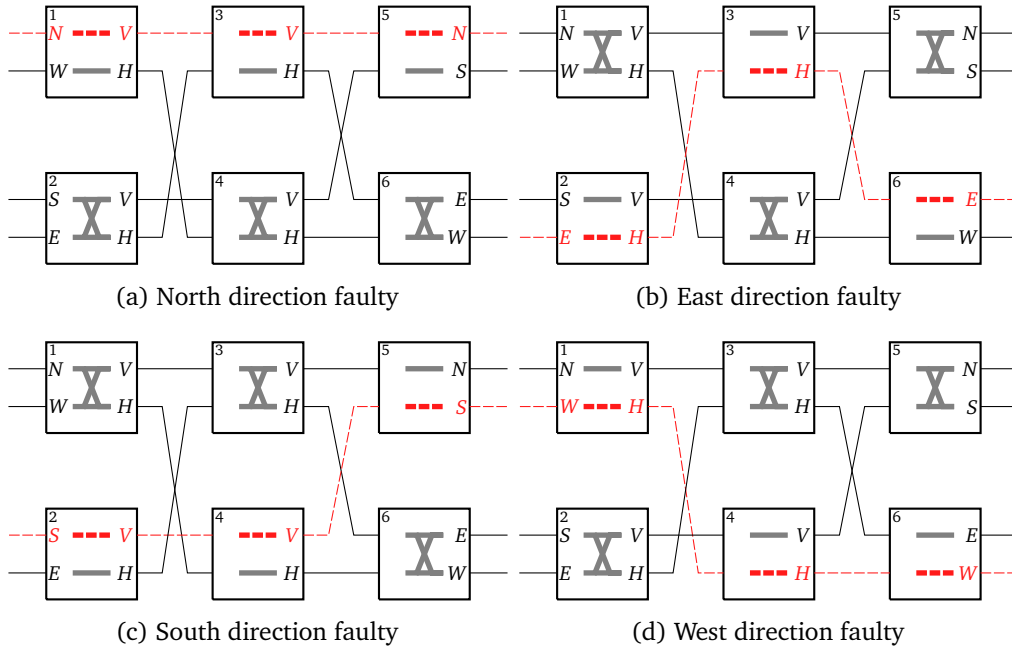


Figure 4.12: All one-direction fault situations of a Benes network.

input port is connected to its output port if all switching elements associated¹⁷ to the corresponding direction perform an identity permutation. Thus, switching elements can adjust their states based on solely local fault information, i.e. the fault status of the two directions associated to the switching element.

To demonstrate that no coordination between the switching elements is required, Figures 4.12 to 4.14 show all one-direction, two-directions, and three-directions fault situations of the deployed 4×4 Benes network. As the case at Banyan networks, one-direction and three-directions fault situations are easy to handle, as either a single faulty path or a single functional path exists. However, in contrast to Banyan networks, the same applies to two-directions fault situations at Benes networks. The deployed Benes network consists of six switching elements, whereas one fully functional switching element exists at each of the six possible two-directions fault situations (cf. Figure 4.13). This means, the fully functional switching element can decide between both remaining routing decisions, and no coordination between the switching elements is required.

Figures 4.12 to 4.14 show that no faulty link is used if every switching element

¹⁷At a Benes network, a switching element s_i is referred to as *associated to a direction d* if either s_i is a first stage switching element, i.e. it is directly connected to d , or s_i is connected to d in the case that $z(s_i) = 0$, whereas s_j are switching elements of the preceding stages. For instance, i_1 and i_2 of s_4 are referred to as associated to S and W , respectively, even if they might be physically connected to other directions.

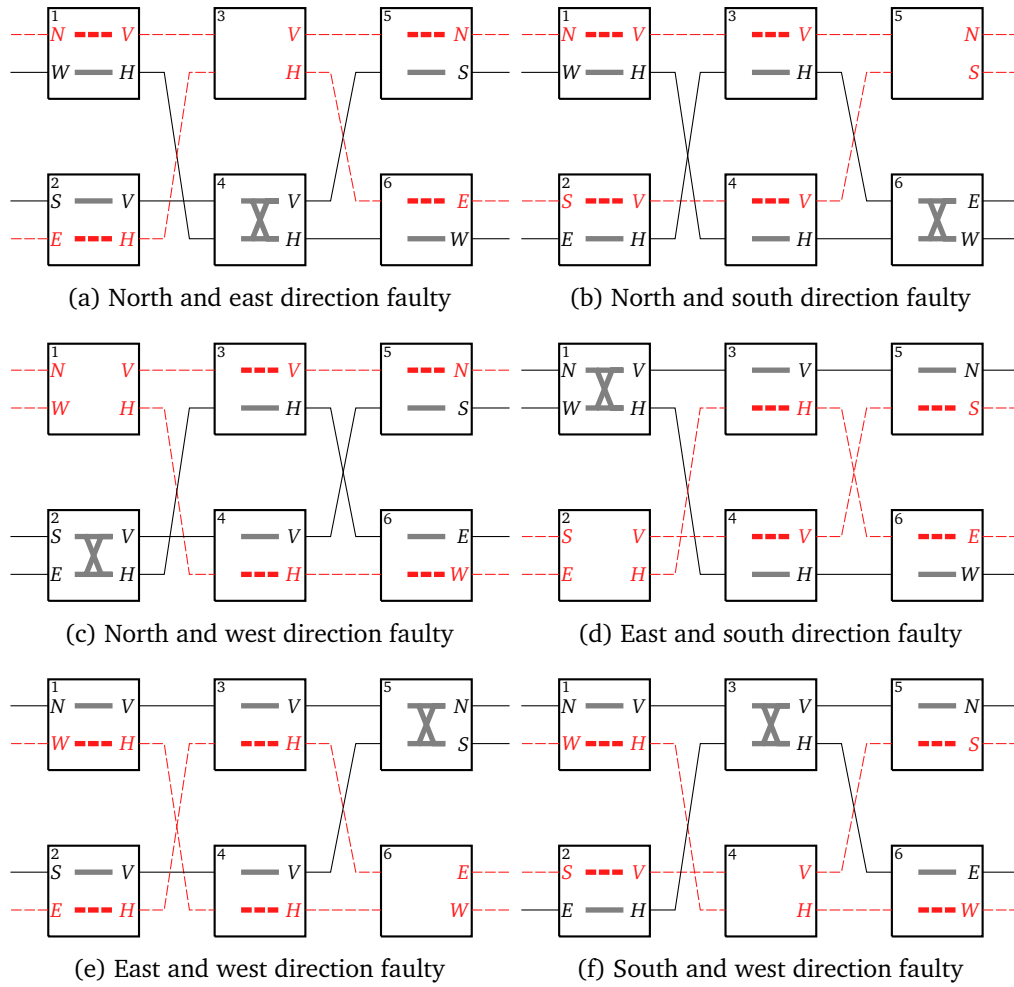


Figure 4.13: All two-directions fault situations of a Benes network.

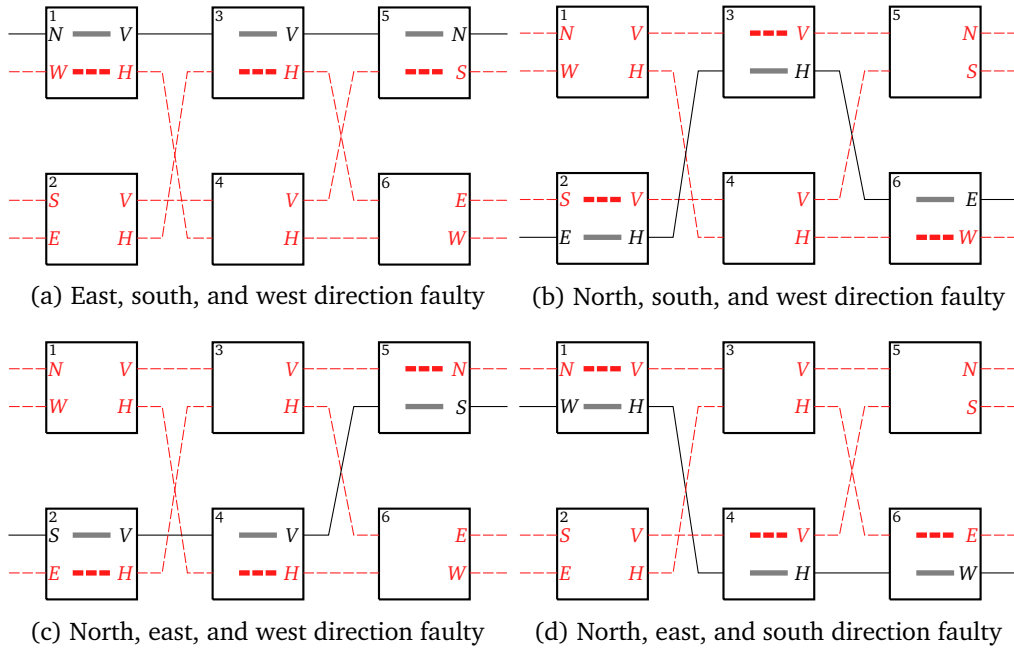


Figure 4.14: All three-directions fault situations of a Benes network.

connected to a faulty direction performs an identity permutation, i.e. their state is 0. However, even if these predefined states are sufficient to avoid faulty links, they are not required, as multiple paths exist between any tuple of input port and output port. For instance, if the north direction is faulty, the north input port could be connected to the north output port by using s_4 instead of s_3 , which would lead to different states. Nevertheless, the chosen and depicted states ensure that the fault situations are additive. This means, the required state changes to avoid two one-direction fault situations correspond to the required state changes of the resulting two-directions fault situation, and this also holds for all other combinations of fault situations. Hence, additivity simplifies a decentralized implementation of the fault tolerance concept, as required adjustments depend only on local fault information.

4.3.3 Concept of Fault-aware Flits

In Sections 4.3.1 and 4.3.2, adjustments to the permutation network to prevent the utilization of faulty links have been shown. Unfortunately, these adjustments are required, but they are not sufficient to guarantee the delivery of flits to their destination. For instance, if fully functional routers reflect flits back to faulty routers, flits might oscillate between these routers, which leads to livelock situations. In general, link failures can cause that the regular structure of a 2D mesh topology disintegrates into

an irregular one. Hence, routing algorithms which require this regular structure can not cope with complex fault situations, which can arise if several links fail. Some kind of global fault knowledge or fault awareness is required to overcome complex fault situations and avoid livelocks of flits. Existing fault-tolerant and additionally deflection routing based router architectures use either routing tables (e.g. FTDR, FTDR-H) or fault information of neighbored routers (e.g. FON). Routing tables in combination with Q-learning provide good performance as well as high fault tolerance, but the hardware requirements increase with the network's size, as the routing tables' size grows. Thus, routing algorithms based on tables do not scale with high NoC dimensions. If the fault information of neighbored routers is used, the hardware requirements per router are constant, but the fault tolerance is limited as the exchange of fault information is restricted to a small number of hops. Fault situations which exceed this number of hops can not be tolerated and therefore this method is only practical for small fault situations.

The non-fault-tolerant CHIPPER architecture excels by its energy efficiency, low hardware requirements, simplicity, and speed. Thus, these characteristics are preserved as far as possible at FaFNoC. Deflection routing dispenses with buffers and, instead of being buffered, flits are deflected in case of collisions. This means, flits are not stored in routers, but kept on the links instead. The concept of FaF, developed for the FaFNoC architecture, transfers this approach to the fault-information keeping. Instead of adding fault awareness to every router, this information is added to the flits. The basic idea of FaF is adopted from the maze solving algorithm *wall follower*, at which one follows either the left wall or the right wall to find a way out of a maze. This means, if a flit is deflected away from its destination because of a faulty link, the flit is turned to *fault region evasion mode*. In this mode, it is attempted to route this flit around the faulty region by performing either only left turns or only right turns until the region is overcome.

To this end, the flit structure is extended by two fields for fault tolerance, $tDir$ and $tDst$. Both fields together represent a flit's *fault-status*, which is denoted by fs . A flit's turn direction is stored in $tDir \in \{R, L, -\}$, whereas R , L , and $-$ denote right turns, left turns, and no turns (normal mode), respectively. If a flit's turn direction field is set, i.e. $tDir \neq -$, the switching elements route this flit according to the turn direction, and the routing algorithm is ignored. The second field $tDst$, stores the flit's turn distance, which is the Manhattan distance from the router, at which the flit switched to region evasion modus, to the flit's destination. Thus, at an $N \times N$ NoC, $tDir$ and $tDst$ require 2 bit and $\text{ld}(2N)$ bit, respectively. These fields are set and cleared by the *fault-status-handler*, which is the last component in the data path of the FaFNoC architecture (cf. Figure 4.2). For the sake of clarity, the fault-status-handler is depicted as a single component, which has I/O ports for all four compass directions. However, the fault-status-handler in fact consists of four separate sub-components, and all four directions are handled individually. The pseudo code of the fault-status-handler is shown in Algorithm 8, and a detailed illustration of switching element s_5 and the fault-status-handler for the north port is depicted in

Algorithm 8 Fault-status-handler

```

1: for each flit  $f$  arriving from  $\{N, E, S, W\}$  do
    //  $dflctd$  is true if at least one switching element can not route  $f$  to its preferred direction
    // This does not mean that  $f$  is deflected away from its destination!
2:    $dflctd \leftarrow f$  has been deflected at  $s_i$ ,  $i \in \{1, 2, 3, 4\}$ 
3:    $nDst \leftarrow$  Manhattan distance from next router to  $f(dst)$  //  $f(dst)$  denotes  $f$ 's destination
4:    $cDst \leftarrow$  Manhattan distance from current router to  $f(dst)$ 
5:   if  $f(tDir) \neq -$  then // flit is in region evasion modus
6:     if  $dflctd \vee nDst < f(tDst)$  then // if flit is deflected or new shortest distance will be reached
7:        $f(tDir) = -$  // stop region evasion modus
8:        $f(tDst) = 0$ 
9:     end if
    // start region evasion modus if:
    // 1) flit has not been deflected by higher prioritized flit,
    // 2) flit is routed away from its destination,
    // 3) this is due to a faulty link ( $pDir$  denotes the flit's productive direction).
10:  else if  $dflctd \wedge cDst < nDst \wedge f(pDir)$  is faulty then
11:     $f(tDst) = cDst$ 
    // For instance, a flit routed to  $N$  is turned to the left if it has arrived from  $W$ . This corresponds to:
    //  $z(s_1) = 1 \wedge z(s_3) = 0 \wedge z(s_5) = 0 \vee z(s_1) = 0 \wedge z(s_4) = 1 \wedge z(s_5) = 1$ 
12:    if  $f$  is turned to the left then
13:       $f(tDir) = R$ 
14:    else if  $f$  is turned to the right then
15:       $f(tDir) = L$ 
16:    else if  $f(dst)$  on the right then //  $f$  is reflected or follows a straight line
17:       $f(tDir) = R$ 
18:    else
19:       $f(tDir) = L$ 
20:    end if
21:  end if
22: end for

```

Figure 4.15.

In general, a flit is turned to fault region evasion modus if the flit is deflected away from its destination due to a faulty link (cf. Algorithm 8, line 10). In such a case, the flit's turn distance field $tDst$ is set to the Manhattan distance from the current router's address to the flit's destination address (cf. line 11), and additionally, the flit's turn direction field $tDir$ is set to the appropriate direction to surround the faulty region (cf. lines 12 to 20). This means, if the flit is deflected away from its destination and a 90° turn is performed additionally, only one turn direction is suitable, as the other one can lead to a livelock. For instance, if a flit f is deflected to the right, $f(tDir)$ has to be set to L , and $f(tDir) = R$ might lead to a livelock. In case of a reflection, i.e. no turn is performed, $f(tDir)$ is set to R or L depending on f 's destination relative to the current

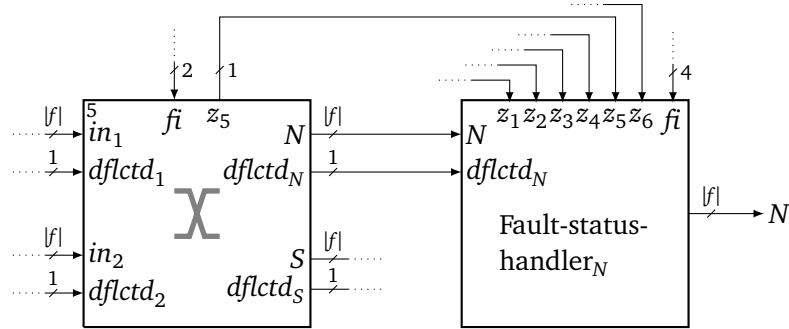


Figure 4.15: Detailed illustration of switching element s_5 and fault-status-handler for north port, whereas $z_i = z(s_i)$, $i \in \{1, \dots, 6\}$.

router. However, this is only a heuristic, which emerged as efficient, and both turn directions are suitable in case of a reflection.

The flit is turned again to normal mode if the faulty region has been overcome, i.e. a router which is closer to the destination as $tDst$ has been reached (cf. lines 5 to 9). Special attention has to be paid if a flit, which is in fault region evasion modus, gets deflected. The permutation network turns the direction of a flit which is in this mode whenever possible, in order to route the flit along the faulty region's boundary. However, if a flit is deflected, it can be transmitted away from the faulty region. In such a case, turning the flit whenever possible might lead to a circular path and the flit might never leave the fault region evasion modus again, i.e. a livelock occurs. Therefore, a flit is only turned to fault region evasion modus if it is not deflected, and it leaves this mode immediately if it is deflected. The fault-status-handler, however, can not detect deflections efficiently. In order to determine if a flit has been deflected by another flit, the permutation network is extended by one $dflctd$ signal per I/O port, which is set if the corresponding flit is deflected (cf. Figure 4.15). For instance, $dflctd_N$ is set if the flit, which will be routed to the north, has been deflected by a higher prioritized flit. Please note that this does not mean that this flit is routed away from its destination. If the flit is in fault region evasion modus, the flit can be deflected closer to its destination, however, the flit should have been routed away from its destination according to the evasion modus.

Figure 4.16 shows an example, at which a flit is routed from its source router R_s to its destination router R_d . At router R_s , both the north and the west are productive directions for the flit. Here, it is assumed that the north is chosen, and hence, the flit is routed to R_a . Unfortunately, R_a can only deflect the flit to the east, as R_a 's north link and west link are faulty. As the flit is not deflected by another flit and additionally, it is deflected away from its destination due to a faulty link, R_a 's fault-status-handler sets the flit's fault-status field fs . The turn direction $tDir$ is set to L , as R_a has turned the flit

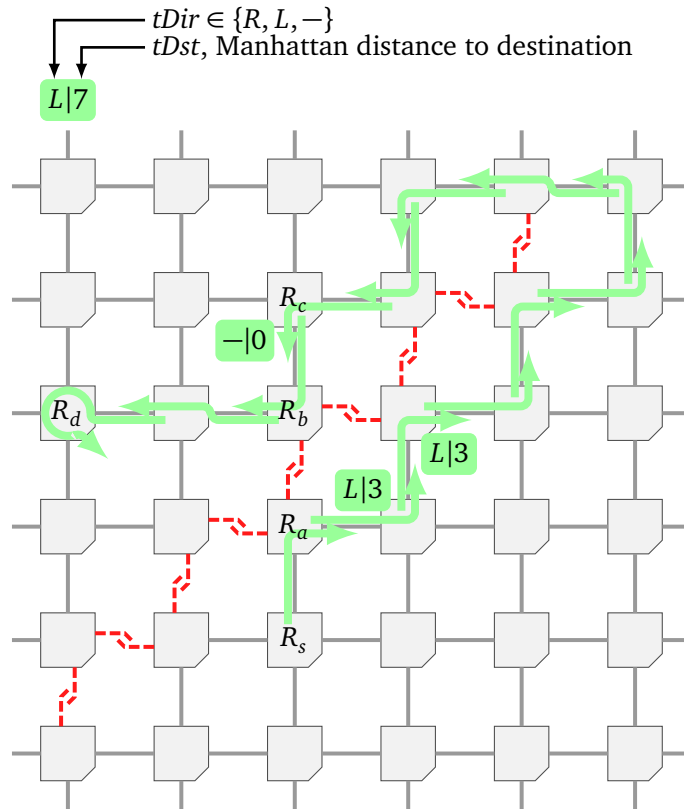


Figure 4.16: Path of a flit which is routed from R_s to R_d . At R_a , the flit is turned to region evasion modus, and at R_c , back to normal mode.

to the right, and the turn distance $tDst$ is set to three, which is the Manhattan distance $d_1(R_a, R_d)$. Thus, the next routers along the depicted path try to route this flit in 90° to the left. Furthermore, every router checks if the Manhattan distance from the next router along the flit's path to R_d is smaller than $tDir = 3$. By this means, the flit is routed to R_c , which clears the flit's fault status, as the next router R_b is only two hops away from R_d . Finally, the flit is routed in normal mode from R_b to R_d .

4.3.4 Summary and Complete Overview of FaFNoC Router Architecture

FaFNoC is a fault-tolerant router architecture, which is based on deflection routing and additionally utilizes a permutation network. To the best of my knowledge, no other fault-tolerant NoC router architecture exists which is based on a permutation network. Permutation networks distinguish from crossbars by the fact that flit prioritization, path arbitration, and routing are performed in a decentralized manner by the switching elements. Combined with deflection routing, this yields a small, energy efficient, and fast router architecture. Existing non-fault-tolerant, permutation network based router architectures, as CHIPPER or MinBD, utilize a 4×4 Banyan network. In Section 4.3.1, the drawbacks of Banyan networks in terms of fault tolerance, as well as the necessary adjustments to avoid faulty links, have been shown. These drawbacks include i.a. the requirement for coordination between switching elements, as well as unusable, but healthy links due to faulty links. In Section 4.3.2, it has been shown that a 4×4 Benes network requires no coordination between switching elements for fault tolerance, and it supports more permutations than a 4×4 Banyan network. On the other hand, a 4×4 Benes network consists of one additional stage of switching elements, compared to a 4×4 Banyan network. To demonstrate the benefits of a Benes network based architecture, both FaFNoC-Banyan and FaFNoC-Benes are compared in Section 4.4.

The adjustments to avoid faulty links, which have been presented in Sections 4.3.1 and 4.3.2, are necessary, but not sufficient to overcome more complex fault situations, which may arise at higher fault rates. Some kind of fault awareness is required to overcome those situations, as some flits have to be routed intentionally away from their destination to reach this destination eventually. Both FaFNoC-Banyan and FaFNoC-Benes utilize the concept of FaF, which is inspired by the maze solving algorithm *wall follower*. There, the flits, instead of the routers, are aware of the encountered fault situations. To this end, the flit structure is extended by a fault-status field, which in turn consists of two fields, $tDir$ and $tDst$. At an 8×8 NoC, the fault-status field's size is 6 bit. Additionally, FaFNoC-Banyan requires one bit, denoted as the return flag, which signals an adjacent router to reflect a flit immediately. The resulting flit structure is depicted in Figure 4.17. Please note that the return flag is not required at FaFNoC-Benes. The fault-status field, as well as the return flag, is set and cleared by the fault-status-handler. At FaFNoC-Banyan, the required routing decisions, like specific turns or a reflection, are

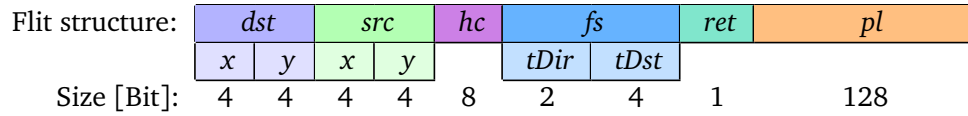


Figure 4.17: FaFNoC’s flit structure for an 8×8 NoC, consisting of destination and source address field, hop count field *hc*, fault-status field *fs*, which in turns consists of *tDir* and *tDst*, return flag field *ret* (only required at FaFNoC-Banyan), and payload field *pl*.

realized by the central fault-handler, which adjusts the switching elements’ state. At FaFNoC-Benes, the routing decisions are performed by the switching elements, based on solely local information, i.e. based on the received flits and the fault information of the two connected links.

In general, an important aspect of fault-tolerant routing algorithms is guaranteed delivery, i.e. the abstinence of livelocks. Nevertheless, a formal proof of guaranteed delivery is omitted herein as the general idea is highly influenced by the well-known wall-follower algorithm. Instead, the evaluation in the subsequent section is primarily simulation based. The presented results show, FaFNoC-Benes was able to deliver all flits, even at high fault rates and different traffic scenarios. Nevertheless, a livelock detection and resolution approach is required to detect disconnected nodes. Therefore, a FaFNoC router discards every flit which has reached its maximum hop count of 255.

4.4 Evaluation of FaFNoC Router Architecture

In this section, performance and hardware costs of the FaFNoC router architecture are evaluated. Further, the results are compared to the existing router architectures FON [Fen+10b], FTDR [Fen+10a], and FTDR-H [Fen+10a], which are also fault-tolerant and additionally based on deflection routing. Towards this end, these three router architectures were reimplemented. All router architectures are simulated for an 8×8 NoC using an in-house cycle accurate simulator implemented in VHDL. As described in Section 2.6.2, every router is connected to a traffic generator which is able to generate different synthetic traffic, or receives the network traffic from Netrace. If a flit could not be injected due to congestion, the flit is stored in an injection queue and injected as soon as possible. The flit sizes are chosen to fit 128 bit of payload. Simulations are executed 5000 clock cycles and $\approx 5 \times 10^7$ clock cycles for synthetic traffic and application traffic, respectively. For comparability, every reported value is an average value of three simulations. Hence, every plot in Figure 4.18 consists of 54 simulations, and every plot in Figure 4.21 consists of 60 simulations. The location of link failures, and also traffic in case of random traffic, are generated by a uniform distributed random

process with the same seed for every router architecture. Only connectivity was ensured at the link failure placement. The three generated fault situations are depicted in Appendix A.1.

To evaluate hardware costs, the router architectures are synthesized using Xilinx's XST. Please note that XST synthesizes a design for FPGAs and does not use a standard cell library like ASIC synthesis software. Nevertheless, this enables a comparison of hardware requirements and achievable speed of the different router architectures.

4.4.1 Non-fault-tolerant Architecture

The fault-tolerant router architectures compared in this section utilize different interconnection architectures. A crossbar is deployed at FON, FTDR, and FTDR-H, whereas a Banyan network and a Benes network are utilized at FaFNoC-Banyan and FaFNoC-Benes, respectively. Obviously, the interconnection architecture of a router has an impact on hardware costs as well as on performance. Theoretically, a Banyan network is the most restrictive interconnection architecture and the crossbar the least restrictive architecture (cf. Table 4.2). In order to assess the impact of the different interconnection architectures of a router, the Banyan network of the non-fault-tolerant baseline router architecture is replaced by a crossbar as well as by a Benes network and various traffic scenarios are simulated. The simulation results for uniform random traffic with a variable injection probability are depicted in Figure 4.18. Figure 4.18a shows the average hop count of all ejected flits, which is limited by 255 hops, as an 8 bit hop count field is used. The average latency (cf. Figure 4.18b) further considers the queue time, which is theoretically unlimited. The throughput θ , which is the number of ejected flits per clock cycle per node, is depicted in Figure 4.18c. These results show, the crossbar based router architectures achieve the best performance values, i.e. the lowest average hop count and the lowest average latency, as well as the highest throughput. The worst values are achieved with the internal blocking Banyan network. As not all permutations are possible for such a network, this restricts the number of supported routing decisions. The deployed Benes network is non-blocking, and hence, every permutation between all I/O ports is theoretically possible. However, in contrast to a crossbar based architecture, a central component which adjusts the permutation network is avoided and the switching elements are self-routing. This means that all switching elements only use local information, which is a drawback in terms of achievable routing decision quality on the one hand, but also a beneficial property that contributes to a fast and energy efficient router architecture on the other hand.

Figure 4.19 shows the synthesis results, i.e. the required number of LUTs, REGisters (REGs), and the achievable frequency, for the non-fault-tolerant architectures. The Benes network based router architecture has slightly higher hardware requirements, as well as a lower maximum frequency, compared to the Banyan network based architecture, due to the third stage of switching elements. The crossbar based router architecture

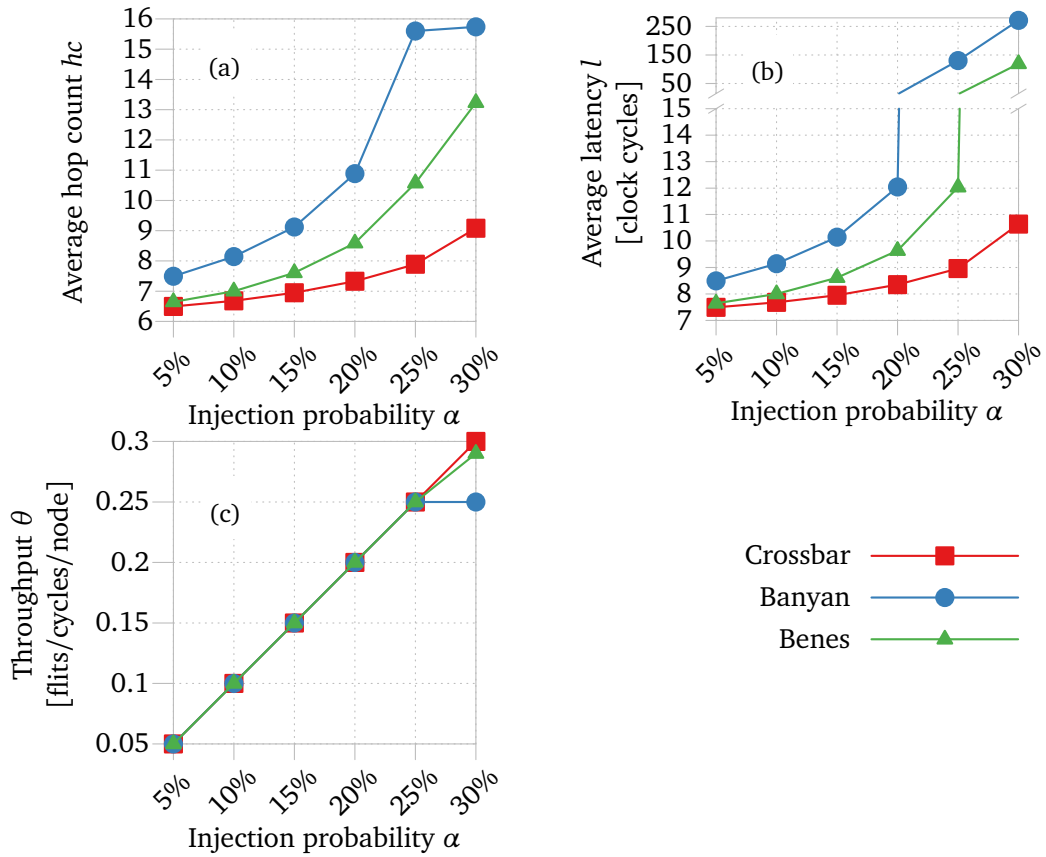


Figure 4.18: Simulation results for non-fault-tolerant router architectures with different interconnection architectures, uniform random traffic, and a variable injection probability.

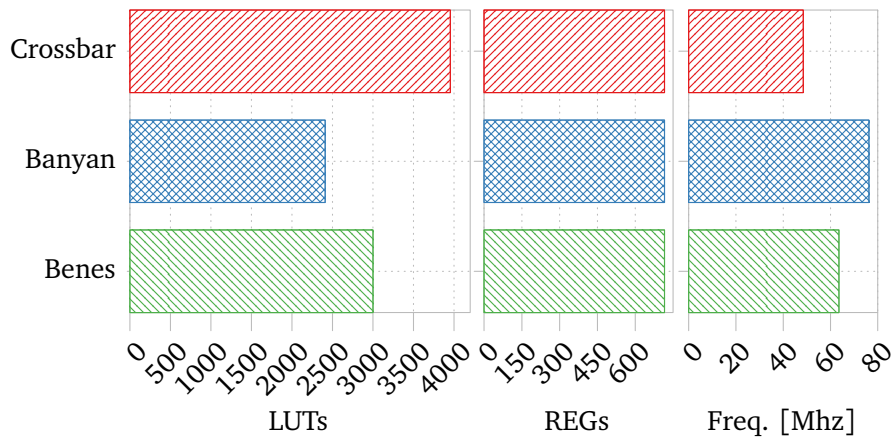


Figure 4.19: Synthesis results for one non-fault-tolerant router, with different deployed interconnection architectures.

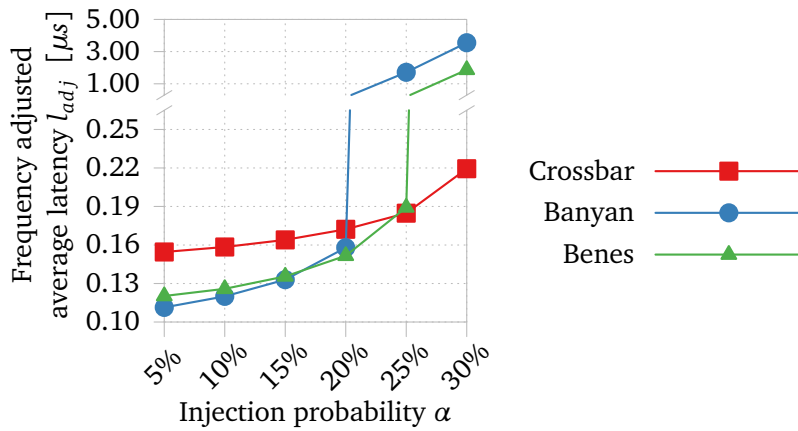


Figure 4.20: Frequency adjusted average latency for non-fault-tolerant router architectures. The depicted values are based on the average latencies of Figure 4.18 and the maximum clock frequencies of Figure 4.19.

has the highest hardware requirements, mainly because of the sorting and arbitration logic, which is not required at permutation network based architectures. In summary, the synthesis results of the Benes network based router architecture lie in between the values of the crossbar based architecture and the Banyan network based architecture, just as the performance values.

So far, the average latency has been considered without assuming a specific technology and is given in units of clock cycles. However, the actually occurring latency is also affected by the implementation, which is technology dependent. The herein presented simulation results and synthesis results have shown that the lowest average latency and the highest maximum frequency are achieved in reverse order. For instance, the lowest average latency is achieved with a crossbar based router architecture, but this architecture has the lowest maximum frequency, i.e. the longest critical path. Consequently, the frequency adjusted average latency is another valuable performance metric, as it combines simulation and synthesis results. Figure 4.20 shows the average latency of Figure 4.18 adjusted for clock frequencies of Figure 4.19. In contrast to the solely simulation based average latency, the frequency adjusted average latency of both permutation network based router architectures is lower than the average latency of the crossbar based router architecture up to an injection probability of 20%. If the network is highly congested, i.e. $\alpha > 25\%$, the higher saturation point of the crossbar based router architecture pays off.

4.4.2 Fault-tolerant Architecture

In this section, the three existing fault-tolerant router architectures FTDR, FTDR-H, and FON, are compared to the herein introduced architectures FaFNoC-Banyan and FaFNoC-Benes. They are evaluated in terms of their fault tolerance, i.e. the impact of different link failure rates, denoted by λ , and the achievable performance is investigated. The assumed fault model considers permanent link failures, as caused, among others, by aging effects (cf. Section 4.1). To this end, the same three evaluation criteria as for the non-fault-tolerant router architectures are used, i.e. average hop count, average latency, and throughput. However, the fault tolerance of some of the herein evaluated router architectures is limited, and thus, some flits might be undeliverable due to complex fault situations. To prevent that those undeliverable flits stay in the network forever, a livelock resolution approach is required. Therefore, all evaluated router architectures discard flits if the flits' hop count field overflows. Discarded flits are logged as lost flits, which is the fourth evaluation criteria. In general, a method to deal with lost flits is required, e.g. those flits have to be detected and retransmitted. However, as only the networks themselves are simulated and compared herein, discarded flits are never injected again. Consequently, lost flits do not appear in the plots for hop count, latency, and throughput.

Synthetic Traffic Performance

Figure 4.21 shows the simulation results for random traffic as well as for transpose traffic. Even more synthetic benchmarks were simulated, however, as the results are similar to the depicted results, they are omitted for reasons of clarity. Flits are injected at every router with an injection probability of $\alpha = 10\%$, which corresponds to ≈ 32000 flits in total per simulation. Four different link failure rates between a healthy NoC, i.e. $\lambda = 0\%$, and $\lambda = 30\%$ are simulated. At $\lambda = 30\%$, the network is saturated and the achieved throughput decreases with all evaluated router architectures (cf. Figures 4.21g and 4.21h). This means, the routers inject new flits into their corresponding injection queue, but not all flits can be injected into the network.

The depicted results show that all router architectures perform similarly at a fault-free network and at very low injection probabilities (e.g. $\lambda = 10\%$). However, at high fault rates, the achieved performances vary significantly, whereby FTDR performs best, followed by FaFNoC-Benes. FTDR learns the interconnection topology, by using Q-learning. Thus, FTDR can tolerate even very irregular topologies, as they occur at high fault rates. FaFNoC-Benes and FaFNoC-Banyan perform second best and third best, respectively, as these architectures can also tolerate complex fault situations, but non-shortest paths are used to surround faulty regions. FON as well as FTDR-H perform significantly worse than the other router architectures, due to their limited fault tolerance. As FON just uses two-hop fault information, only fault situations which

4.4 Evaluation of FaFNoC Router Architecture

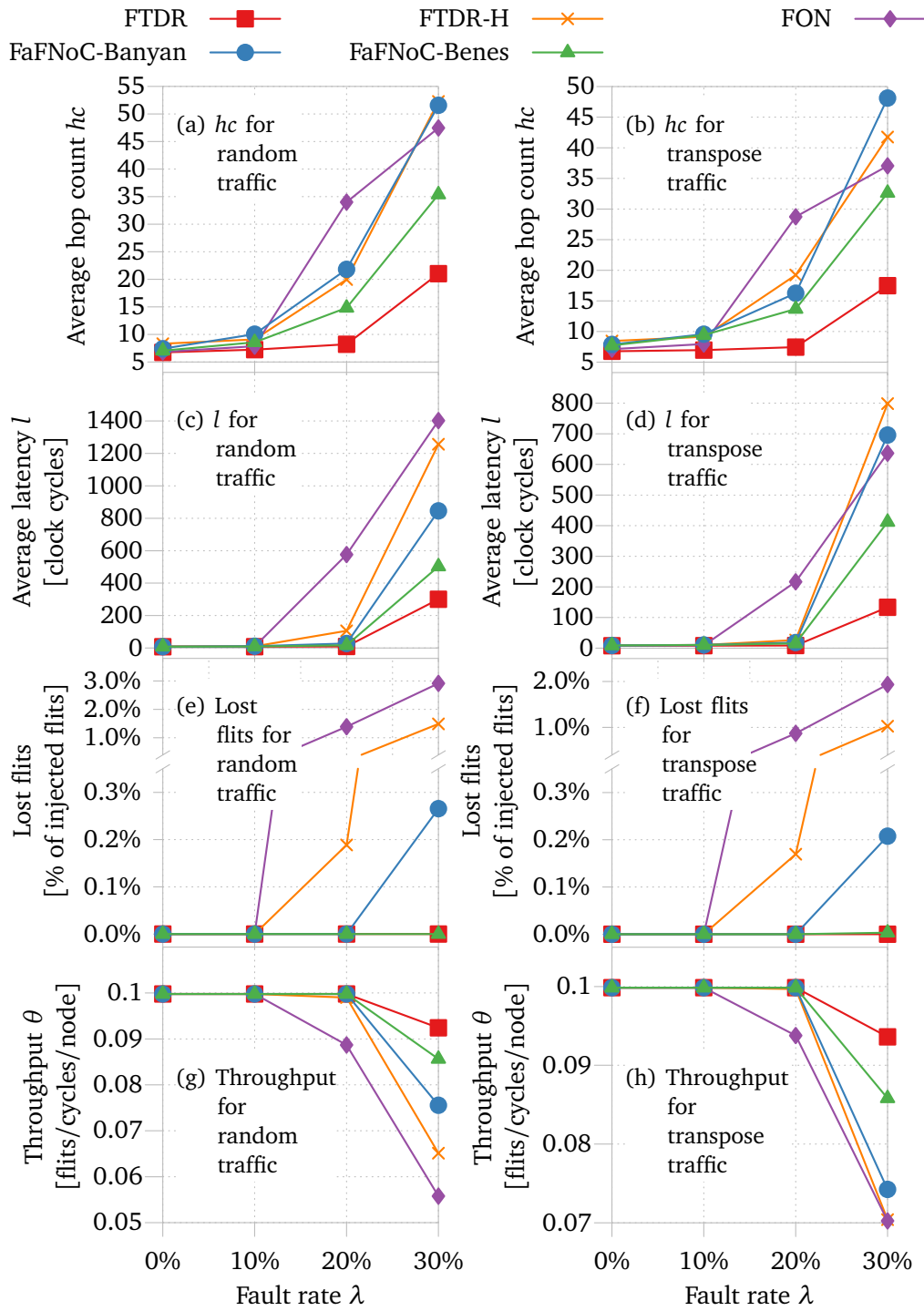


Figure 4.21: Simulation results for **random traffic** and **transpose traffic**, an injection probability of $\alpha = 10\%$, and a variable number of random link failures.

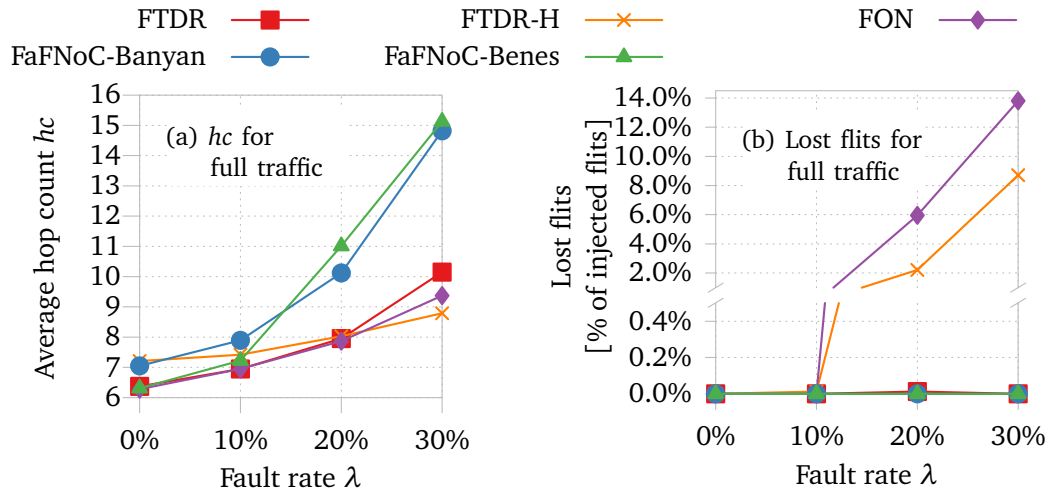


Figure 4.22: Average hop count and percentage of lost flits for **full traffic**, an injection probability of $\alpha = 10\%$, and a variable number of random link failures. At full traffic, every router sends a flit to every other router in the network.

do not exceed this number are tolerable. FTDR-H utilizes a local-routing-table as well as a region-routing-table at every router. Thus, complex fault situations, which cause that the only path between a source and a destination traverses at least two regions, can not be tolerated with FTDR-H. As a result, a crucial number of flits are lost with FON as well as with FTDR-H (cf. Figures 4.21e and 4.21f). Lost flits pose a problem, as those flits have to be retransmitted, whereby a different path should be used at retransmission, but moreover, they interfere with other flits. As flits stay in the network until they are ejected or their hop count field overflows, lost flits have a high priority before they are discarded, which leads to a large number of unproductive deflections.

At both depicted synthetic traffic benchmarks, a relatively small number of flits is lost with FaFNoC-Banyan at $\lambda = 30\%$. However, in contrast to FON and FTDR-H, this is not a systematic problem. There, flits can get lost as they have to be in fault region evasion mode for a long period of time to reach their destination. However, at this high fault rate, the network is congested and the flits are deflected repeatedly during their first time in the network. As a flit's mode is turned from fault region evasion mode to normal mode if it gets deflected, flits frequently have to leave this mode before a faulty region is overcome. Only after a flit is higher prioritized than the remaining flits, i.e. its hop count is higher, it is not deflected and stays in fault region evasion mode for a sufficiently long period. At least when a flit's hop count reaches $hc = 255$, it is discarded to avoid a livelock. Thus, flits can get lost if the remaining time, after their priority is high enough not to be deflected, is not sufficient to reach their destination.

To demonstrate that flit loss at FaFNoC-Banyan is caused by high network load and

the consequential deflections, whereas flit loss at FON and FTDR-H is caused by their limited fault tolerance, Figure 4.22 shows the simulation results for *full traffic*. There, every router sends a flit to every other router. Additionally, it is ensured that at most one flit is in the network at every time. This is achieved by waiting 255 clock cycles between two consecutive injections. Thus, flits can not be deflected by other flits and an injection queue is not required for this kind of traffic, as a new flit is only injected after the old flit has been ejected. Therefore, the average latency corresponds to the average hop count plus one, and the throughput is almost constant and equal for all router architectures and all fault rates. As an 8×8 NoC is simulated, exactly $64 \cdot 63 = 4032$ flits are injected into the network, and 4032 flits minus the number of lost flits are ejected. Thus, only the average hop count and lost flits are depicted in this figure. The results show that no flits are lost with FaFNoC-Banyan, FaFNoC-Benes, and FTDR. At both FON and FTDR-H, the percentages of lost flits even increased, compared to random and transpose traffic. In contrast, the average hop count of FON and FTDR-H seems surprisingly low at first glance, in particular, at a fault rate of $\lambda = 30\%$. However, the depicted average hop counts do not include the lost flits. If lost flits are taken into account, the average hop count of these two architectures increases significantly. For instance, if an average hop count of $hc = 9$ and 10% lost flits are assumed, the average hop count including the lost flits increases to $9 \cdot 0.9 + 255 \cdot 0.1 = 33.6$, and the lost flits are still not delivered.

Application Performance

Application performance, in addition to synthetic traffic performance, is valuable to compare and assess different router architectures. An important metric for such a comparison is the achieved execution time of the simulated benchmarks. However, some herein evaluated router architectures are not capable of delivering all messages, and even retransmission of these undeliverable messages is not sufficient to guarantee their delivery. Consequently, the simulated execution time is not meaningful. Nevertheless, evaluation metrics as average hop count, average latency, and lost flits are valuable, as application traffic has a different temporal and spatial distribution than synthetic traffic. Thus, two PARSEC benchmarks, blackscholes and canneal, are simulated, as described in Section 2.6.2. The handling of undeliverable flits does not differ to synthetic traffic. Those flits are discarded when their hop count field overflows, and they are never injected again. Furthermore, message dependencies are ignored, as new flits, which are dependent on undelivered flits, could not be injected otherwise.

Figure 4.23 shows the simulation results for four different fault rates. Every depicted value is an average of three simulations, which are based on the fault situations of Appendix A.1. Throughput plots are not shown here, as the total execution times are determined by the last injection of the benchmark. Thus, the execution times are almost independent of the simulated router architecture. Furthermore, an equal number of flits is injected by every simulation, and the same number of flits is ejected, minus the number

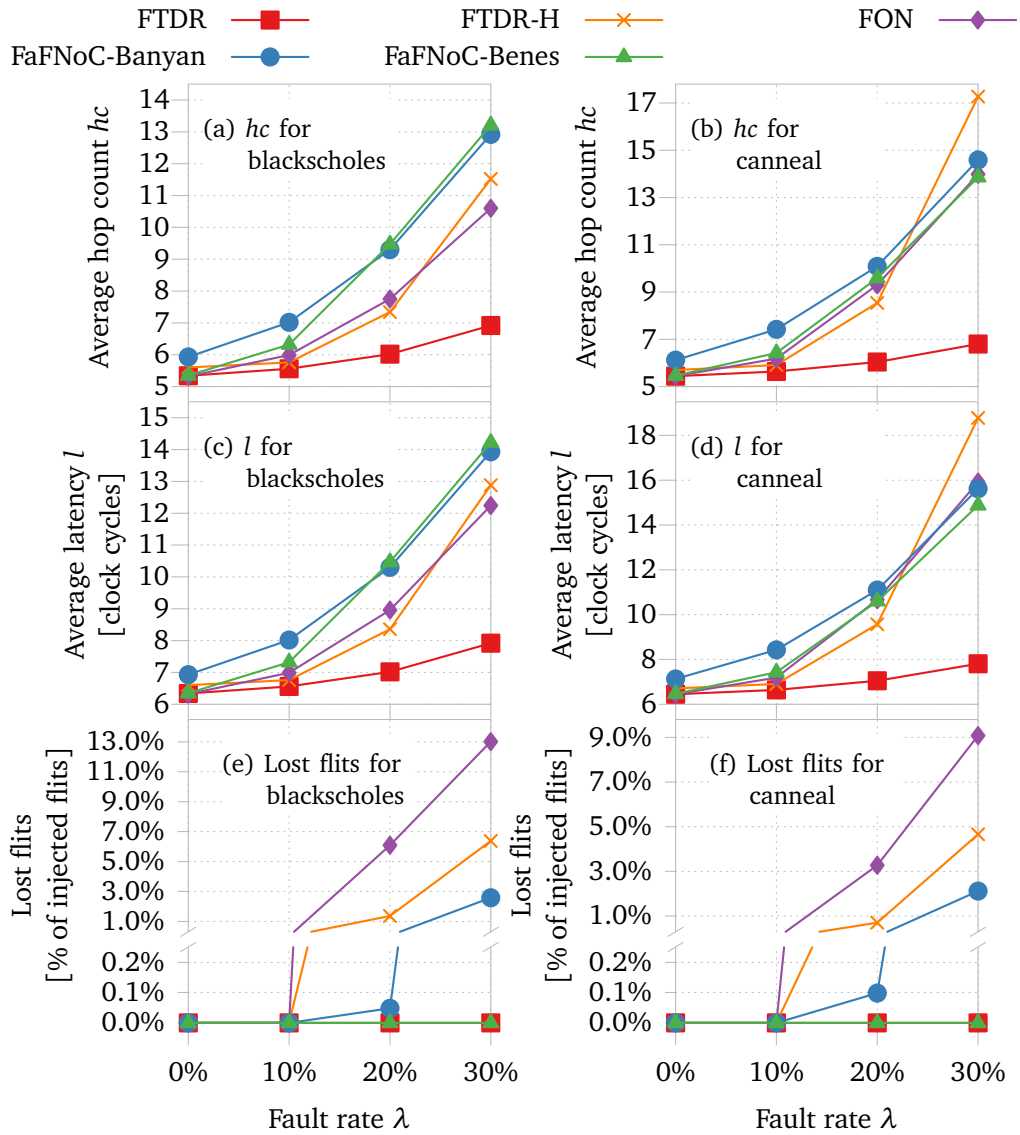


Figure 4.23: Simulation results for two PARSEC benchmarks, **blacksholes** and **canneal**, and a variable number of random link failures.

of lost flits. This means, the used injection queues are sufficiently dimensioned to buffer bursts of injections. At both simulated PARSEC benchmarks, the average injection probability is relatively low (cf. Figure 2.15), even though temporary bursts of injections exist. Most of the time, the network is not congested, and therefore, the simulation results are more similar to those of full traffic than to those of both presented synthetic traffic benchmarks. In particular, the achieved average hop counts and average latencies are lower compared to random traffic and transpose traffic, whereas the percentages of lost flits are higher.

Hardware Costs

Figure 4.24 shows the hardware requirements, i.e. the number of required LUTs and REGs for a single router, of all herein compared router architectures and four NoC dimensions. Here, a flit size which corresponds to a payload size of 128 bit is assumed. It can be seen that the required number of LUTs and REGs for FTDR and FTDR-H increases with the NoC dimension, due to the deployed routing tables. The three remaining router architectures require a comparable amount of LUTs and REGs per router, which is constant for all NoC dimensions. Hence, the required numbers of LUTs and REGs are only quoted above the rightmost bars of Figure 4.24. In particular, the synthesis results show, the hardware requirements of the third stage of switching elements at FaFNoC-Benes are almost offset by the nonexistent fault-handler and the more efficient fault tolerance logic.

All router architectures require the exchange of fault information between adjacent routers. At FON, 4 bit per output port are required to transmit the 2-hop fault information. At FTDR and FTDR-H, additional 6 bit per output port are needed to transmit Q-values, which are estimated distances between the current router and the flits' destinations. At both FaFNoC router architectures, the flit structure is extended by 6 bit for the fault status field. At FaFNoC-Banyan, one additional bit for the return flag is necessary. A constant size of the fault status field is used herein, even if a smaller fault status field would have been sufficient at smaller NoC dimensions, as a constant Q-value size is used at FTDR and FTDR-H.

Figure 4.25 shows the achievable frequencies, which are similar for the three crossbar based router architectures (FTDR, FTDR-H, and FON), and slightly higher for FaFNoC-Banyan and FaFNoC-Benes, which are based on a permutation network.

Frequency Adjusted Average Latency

As it has already been shown for the non-fault-tolerant basis router architecture, the actually occurring average latency is affected by the implementation of the router architecture and, in particular, by the achievable maximum clock frequency. Figure 4.26 shows the frequency adjusted average latencies for both synthetic traffic patterns of

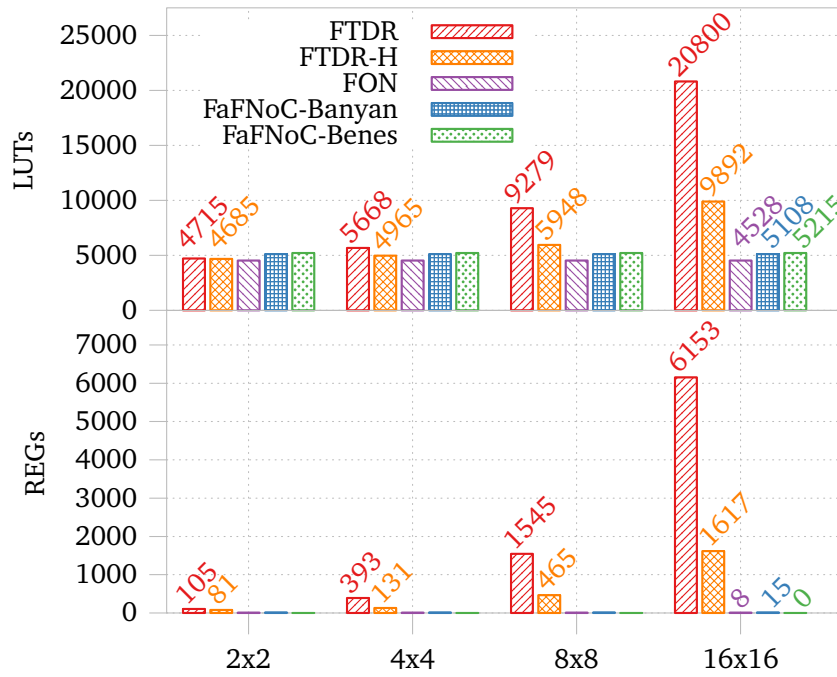


Figure 4.24: Hardware requirements for the herein compared fault-tolerant router architectures and four different NoC dimensions.

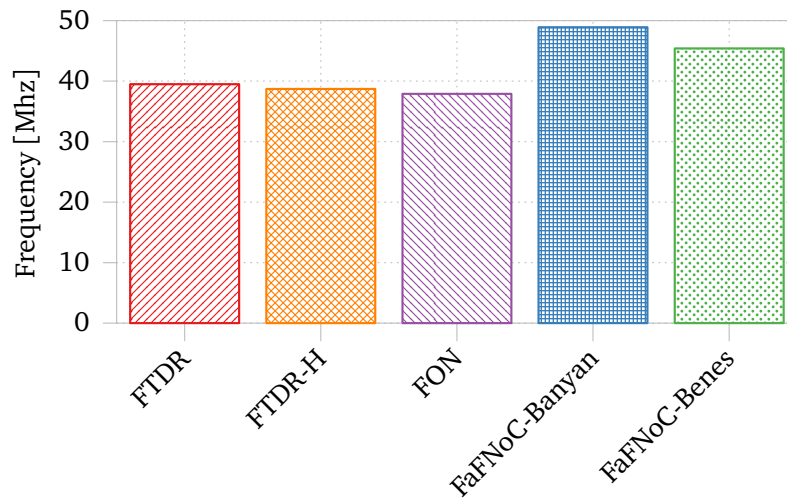


Figure 4.25: Achievable frequencies for the herein compared fault-tolerant router architectures.

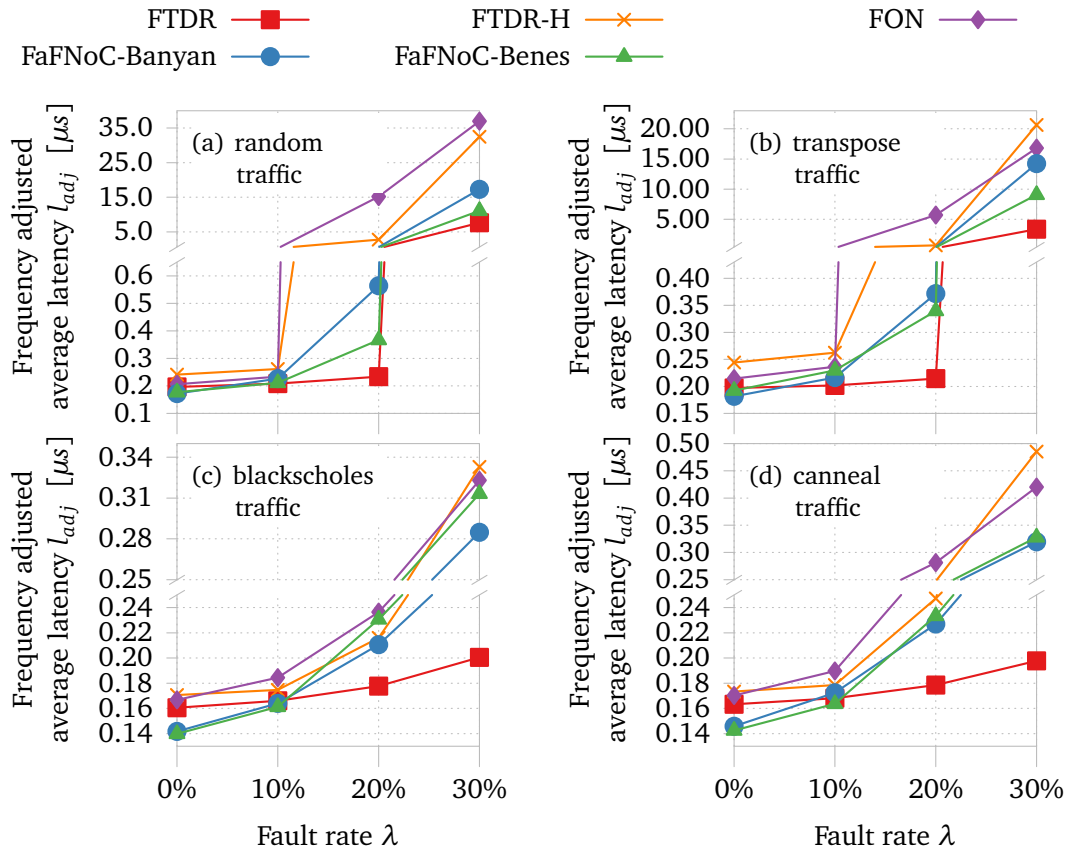


Figure 4.26: Frequency adjusted average latency for fault-tolerant router architectures and different traffic types. The depicted values are based on the average latencies of Figures 4.21 and 4.23 and the maximum clock frequencies of Figure 4.25.

Figure 4.21 as well as the for two evaluated PARSEC benchmarks of Figure 4.23. Compared to the solely simulation based average latencies, the results shift in favor of the two permutation network based router architectures. At low fault rates, FaFNoC-Banyan and FaFNoC-Benes achieve a lower average latency than the crossbar based router architectures. At higher fault rates, the lowest average latency is achieved with FTDR, despite of its lower maximum clock frequency.

4.5 Summary and Conclusion of Chapter 4

Shrinking manufacturing processes cause that NoCs are frequently deployed, as they provide a scalable communication infrastructure, but they also necessitate fault tolerance

concepts. A short introduction to fault tolerance has been given in Section 4.1, to enable a thematic classification of fault tolerance concepts at NoCs. The main focus of this chapter is on fault-tolerant routing algorithms. Such algorithms exploit the inherent path redundancy, which is present in most topologies, to avoid faulty components. Several fault-tolerant and additionally deflection routing based algorithms have been presented in Section 4.2. In Section 4.3, the FaFNoC router architecture has been introduced, which was developed as part of this thesis. In this context, the following contributions have been made:

- The drawbacks of Banyan networks, which are frequently deployed at non-fault-tolerant router architectures, regarding fault tolerance have been identified.
- It has been proposed to substitute Benes networks for Banyan networks, as Benes networks provide a solution for all identified problems of Banyan networks. Within this scope, the required changes of Benes networks to avoid faulty links have been presented.
- The concept of fault-aware flits has been developed, which is a viable solution to overcome complex fault situations without requiring costly routing tables.

Finally, the FaFNoC architecture has been evaluated and compared to existing fault-tolerant and deflection routing based architectures in Section 4.4. The results have shown that FaFNoC-Benes requires only slightly more hardware resources compared to FaFNoC-Banyan, whereas a significantly better performance is achieved. Furthermore, the results have shown that a significant number of flits is lost if solely local fault information is used, as the case at FON and FTDR-H. Consequently, FON and FTDR-H perform far worse than FaFNoC-Benes, at comparable hardware requirements. Only FTDR performs significantly better, but FTDR's hardware requirements increase with the used NoC dimension, which makes it impractical for large networks.

Design of Deflection Routing based Network on Chips

Contents

5.1 Introduction and Motivation	113
5.1.1 Effect of the Link Width on Buffered, Packet Switched NoCs	114
5.1.2 Effect of the Link Width on Deflection Routing based NoCs	115
5.2 Related Work	117
5.3 The optimal Link Width	119
5.3.1 Effect of the Flit Size on Hardware Costs	119
5.3.2 Effect of the Flit Size on Performance	123
5.4 TwoPhases - An Alternating Transmission Scheme	132
5.4.1 Methodology of <i>TwoPhases</i>	133
5.4.2 Transmission Methods	139
5.4.3 Evaluation	141
5.5 Summary and Conclusion of Chapter 5	149

5.1 Introduction and Motivation

The design space of current NoCs is huge, due to their complexity and the multiplicity of design parameters.

“NoCs design space is considerably larger when compared to a bus-based solution, as different routing and arbitration strategies can be implemented as well as different organizations of the communication infrastructure.”
[CML12, p. 11]

The design parameters affect the performance, the area requirements, and the energy efficiency of a NoC. However, the determination of optimal values for all design parameters can be a challenging task. Models (e.g. ORION [Kah+12; KLN12b]) as well as simulators (e.g. NNSE [Lu+05], Noxim [Cat+15], NIRGAM [Jai]) can support system designers in early design stages to determine these values. At deflection routing, some of the parameters are already predefined or nonexistent, as they are related to buffer handling and deflection routing is usually bufferless. The number of virtual channels and the deployed flow control scheme are two of those predefined parameters. Nevertheless, there are still many design parameters which have to be specified. One very important parameter of deflection routing based architectures is the link width. To motivate and demonstrate the significance of this design parameter, the impact of the link width on buffered and bufferless NoCs is pointed out in the following.

5.1.1 Effect of the Link Width on Buffered, Packet Switched NoCs

As described in Chapter 2, buffered NoCs frequently utilize packet switching or, in particular, wormhole flow control. Here, messages are decomposed into packets, which represent the data units for routing and sequencing [DT04]. Packets are further divided into flits, whereby a flit is the data unit for bandwidth and buffer allocation. Flits can be in turn divided into phits. A phit is the data unit that can be transferred over a link within one clock cycle. Thus, the link width equals the phit size ($|LI| = |ph|$). Frequently, the flit size $|f|$ equals the phit size $|ph|$ for on-chip networks [SKH08]. This means, flits are not (de)serialized and a flit is transmitted over a link as a whole. Hence, the link width equals the flit size in this case ($|LI| = |f| = |ph|$). An equal link width and flit size is assumed hereinafter, unless otherwise stated.

For packet switched networks, there are three types of flits to be distinguished: head flits, body or data flits, and tail flits. Only head flits contain routing information. Body and tail flits just follow the path of their head flit towards the destination. This principle allows the transfer of large messages over much smaller links. Furthermore, the overhead, i.e. the data which is not payload, is kept at a reasonable level, as one packet consists of one head flit h , an arbitrary number of body flits b_i , and at most one tail flit t . Whereby, most of the overhead is contained in the head flit h .

Therefore, a reduced link width, or flit size, at a packet switched network leads to just more flits per packet. As the number of flits increases, the network load also increases, which in turn leads to an increasing average latency as well as a decreasing throughput, due to the higher network utilization. On the other hand, hardware costs decrease. Thus, there is a trade-off between performance and hardware costs. However, there is still only one head flit and at most one tail flit per packet, assuming that the flit structure is large enough to fit the complete head information. As a consequence, only the number of body flits increases, in the case of smaller links.

5.1.2 Effect of the Link Width on Deflection Routing based NoCs

Unfortunately, the decomposition of a message, as described in Section 5.1.1, is not possible for deflection routing based NoCs. Such NoCs do not store or discard flits in case of collisions. Instead, flits are deflected to potentially non-shortest paths. A basic prerequisite to avoid livelocks at deflection routing is, to transmit the highest prioritized flit always to a productive port at every router. As there are no flit buffers, a router has to calculate a new routing decision or permutation if a new head flit is received, as this head flit could be the highest prioritized flit. This in turn means, if packet switching is combined with deflection routing, two flits of one packet could be separated from each other. In particular, it could not be ensured that body and tail flits can follow their head flit. Hence, at deflection routing every flit has to carry routing information, or in other words, has to be a head flit. For deflection routing based NoCs, it is not possible to packetize large messages into packets with many flits. Instead, messages are decomposed into several independently routed head flits, or into several packets, whereby each packet consists of exactly one head flit.

Figure 5.1 illustrates this before mentioned problem. Here, it is assumed that packet switching is combined with deflection routing and packets, which consist of one head flit h and at least one body flit b , are routed. In this example, the first flit which arrives at router R_y is the orange head flit. As the destination of this orange head flit is in the north, and the north port is idle, the flit is routed to the north. In the next clock cycle, the green head flit arrives from the right neighbored router, and additionally the orange body flit arrives from the left neighbored router. It is assumed that the green packet's destination lies also in the north, and the green packet is higher prioritized than the orange one. Thus, there are two options. The first option is, the orange body flit is routed to the north, as body flits do not contain any routing information, and hence, this flit could not follow its head flit otherwise. However, this implies that the green head flit has to be deflected and the packets' priority is disregarded. This contradicts to the prerequisite of deflection routing, that the highest prioritized flit is always routed to a productive direction. Therefore, this approach would lead to livelocks eventually. The second option takes the packets' priorities into account, and the green head flit is routed to the north. This in turn implies that the orange body flit is deflected instead. In this case, the body flit is separated from its head flit. Thus, the body flit needs routing information, or in other words, every flit has to be a head flit, as it is the case for standard deflection routing.

One simple and obvious solution to transmit data, which exceeds the payload size of one flit, is the utilization of several independently routed head flits. However, this means that a reduced link width leads to more flits per message, not per packet, as the case for a packet switched network. As every flit is a head flit, i.e. every flit contains routing information, the routing information overhead per message increases significantly. Further, the flits can arrive out of order, which requires reassembling of the received

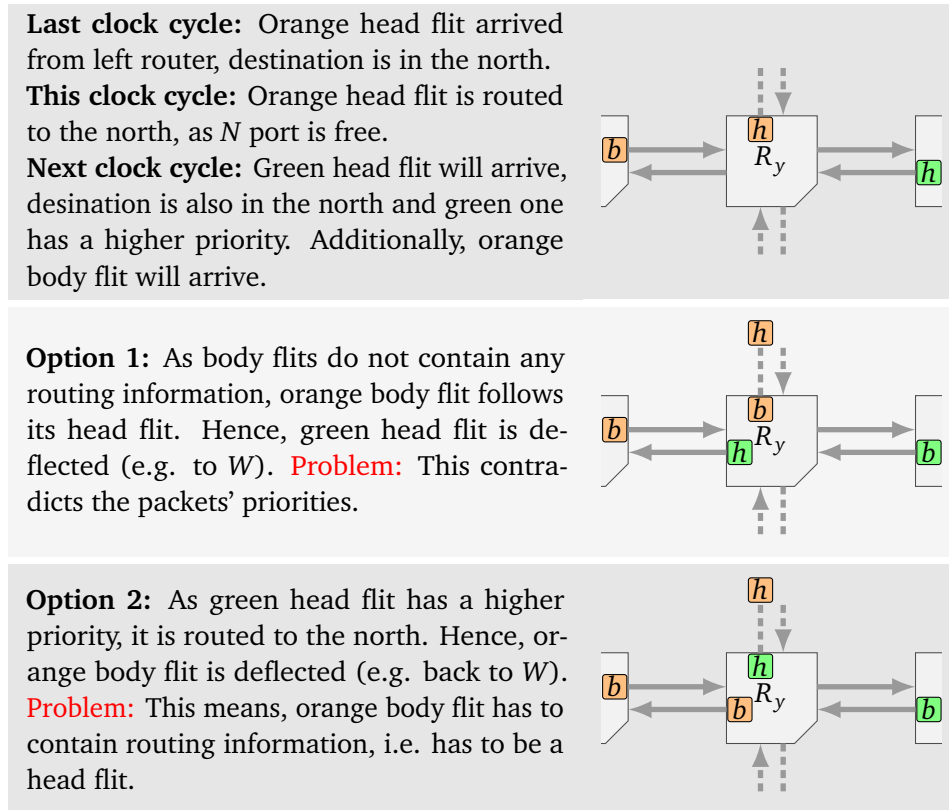


Figure 5.1: Livelock problem of deflection routing based NoCs illustrated.

flits at the receiver side. To this end, sequencing information, like a flit-ID, is required for every received flit. This increased overhead for routing information and sequencing leads to an immense performance degradation (cf. Section 5.3.2). On the other hand, the deployment of unnecessary wide links and large flits results in high hardware costs and can even be impossible due to hardware restrictions. Therefore, it is particularly important for deflection routing based architectures to determine an appropriate flit size and link width. Please note that an appropriate link width for deflection routing based NoCs can be significantly higher than for buffered, packet switched NoCs. Due to the impact of the link width on deflection routing based NoCs, it is important to:

1. identify the effect of this design parameter on the hardware costs as well as on the performance of a NoC. The link width for permutation network based router architectures is considered in Section 5.3.
2. investigate if new methods can be developed to route messages that exceed the link width and additionally mitigate the effects caused by very wide links. A new

transmission method that reduces routing overhead is presented in Section 5.4.

5.2 Related Work

Interconnection network simulators can help to evaluate the influence of design and configuration parameters on the performance of NoCs. BookSim [Jia+13], Noxim [Cat+15], and NIRGAM [Jai] are among the most frequently used NoC simulators. However, they are all designed for buffered, packet switched NoCs, and hence, have to be adapted for bufferless, deflection routing based NoCs. One of the few simulators which is designed for deflection routing is NNSE [Lu+05].

Power-performance simulators, like ORION [Wan+02], ORION2.0 [Kah+12; Kah+09], and McPAT [Li+09], enable early design space power-performance tradeoffs. ORION simulates an interconnection network, whereas McPAT is a modeling framework for multicore and manycore architectures. McPAT as well as ORION (up to version 2.0) model the circuit structure of each basic building block. In case of a router, these blocks include i.a. flit buffers, arbiters, and crossbars. An analytical model for each building block is used to estimate the power and area of a system. Mismatches between the actual RTL code of the building blocks and the assumed logic structure can exist, which may lead to large estimation errors. Hence, at ORION3.0 [KLN12a; KLN12b; KLN15], the models are derived from post place and route data. As all these power simulators are developed for packet switched NoCs, the presented models and formulas are not directly applicable for deflection routing based NoCs.

Most of the more analytical work related to flit size and link width also considers buffered, packet switched NoCs. Ye, Benini, and Micheli introduced an on-chip communication power model for MPSoCs in [YBM03]. They quantified the effect of the packet size on performance as well as on energy consumption. A large packet size leads to higher miss penalties, but also to a lower cache miss rate. Performance wise, they found that the optimal packet size is 64 B for all evaluated SPLASH benchmarks. In terms of energy, a larger packet size reduces the cache and memory energy, but in turn also leads to a higher network energy. However, a packet size which always achieves the least energy consumption, independent from the investigated application, was not found. The authors only considered the packet payload size in this work, and a constant link width of 64 bit was assumed. Therefore, their approach is orthogonal to the problem of the optimal link width, which also effects performance and energy consumption.

In [Lee+13a], Lee et al. gave a guideline how the appropriate flit size or link width for wormhole flow control based router architectures can be determined. The authors developed models for the cost as well as for the latency and analyzed workload characteristics. They concluded that the links should be as wide as the smallest packet size plus the header overhead. Their work is similar to our approach, presented in Section 5.3. However, as they assumed a buffered, packet switch router architecture, their outcome

is not applicable to deflection routing based architectures.

Ogras, Hu, and Marculescu presented equations for bandwidth and latency as functions of the channel width for packet switched NoCs in [OHM05]. They also identified the optimal channel width as one of eight open research problems of NoC design. Network channel design was also identified as an outstanding research problem in [Mar+09].

Investigating the effect of the link width on performance and hardware costs is important, in particular for deflection routing based NoCs. However, as stated before, methods or transmission schemes which allow the transfer of messages that exceed the flits' payload size are also indispensable. One such transmission scheme is serialization. There, the flits are serially transferred from router to router. Serialization has been examined with different emphasis in the past, mainly for buffered NoCs. Serialization of the costly TSVs, which can be used for interlayer interconnection of 3D chips, is investigated for packet switched routers in [Ghi+13] and deflection routing in [Lee+13b]. The authors in [CLC12] proposed serialization to preserve partially defect links of packet switched NoCs.

Moscibroda and Mutlu presented in [MM09] a router architecture which uses the wormhole principle and is still based on deflection routing. The livelock problem mentioned in Section 5.1.2 and illustrated in Figure 5.1 is solved in this approach by worm truncation. In the case of a collision, the lower prioritized worm is truncated, and thus, the higher prioritized worm can be routed to its productive direction. As body flits usually do not contain routing information, the first body flit of the truncated worm's second part has to become a new head flit. Thus, every router stores the header information of every packet and additionally a mapping of packets to output ports. With this information, every router is able to create new head flits out of body flits. The authors assumed that the header information is transmitted by dedicated wires, which are only used if a head flit is transferred. A comparison to packet switched, buffered NoCs is made, by comparing the costs of these additional wires versus flit buffers. In contrast, herein it is assumed that all available wires are used for payload. This implies that head flits contain less payload information than body flits. Furthermore, depending on the frequency of truncations, the performance of their approach degrades to standard deflection routing, where many flits are routed independently.

Lin, Lin, and Tang also introduced a wormhole-switched NoC, entitled making-a-stop [LLT12]. Their architecture consists of one additional register array per router, which has to have the size of the largest packet. If all productive ports of the currently highest prioritized packet are occupied, the packet is stored in this register array. Hence, the deflection rate is reduced. If a lower prioritized packet is already stored in the register array, the lower prioritized packet is evicted and deflected to an idle output port. Unfortunately, depending on the priorities of the incoming flits, the deflection rate can be high, despite of the register array. Further, the buffer requirements can be substantial, as the register array's size depends on the maximum packet size.

Another bufferless router architecture which utilizes wormhole routing is presented in [TB11]. They utilized pipeline registers at the channels as storage elements, which hold the flits until the next output port is ready. Further, they developed the express flow control scheme to avoid unnecessary stall cycles. There, an additional signal is added to each input port, which enables that body and tail flits of a packet can follow their corresponding head flit immediately. Without express flow control, one stall cycle is required due to the latency of the signal, which indicates an idle register in the succeeding router. Even if this approach is bufferless, it is not based on deflection routing. Instead, it resembles to a buffered, wormhole routing based router architecture with a buffer depth of a single flit.

5.3 The optimal Link Width

As packetization is not possible at deflection routing based NoCs, it is particularly important to determine the appropriate link width for such NoCs [RK16b]. The link width has been investigated already for packet switching based NoCs in [Lee+13a], but not yet for deflection routing. Deflection routing is not linked to any specific router architecture. However, permutation network based router architectures, like CHIPPER [FCM10; FCM11] or MinBD [Fal+12; Fal+11], have several advantages compared to crossbar based architectures in relation to deflection routing (cf. Section 3.3). In Section 5.3.1, the asymptotic hardware requirements of such a permutation network based router architecture are analyzed as a function of the flit size $|f|$ and the number of ports P , which corresponds to the radix of the routers. The results show that the requirements increase linearly with the flit size and quadratically with the number of ports. Synthesis results confirm these findings. To demonstrate the effect of the link width on the performance, results for different synthetic traffic patterns and several injection probabilities, as well as application performance for three PARSEC benchmarks are presented in Section 5.3.2.

5.3.1 Effect of the Flit Size on Hardware Costs

An overview of the herein assumed router architecture is depicted in Figure 5.2. It consists of an ejection stage, an injection stage, and a permutation network. The ejection stage enables the extraction of flits, which have reached their destination, from the network to the local port L . Accordingly, the injection stage enables the transmission of flits from the L port to the network if at least one port is idle. The permutation network consists of four switching elements s_1, \dots, s_4 and replaces the commonly used crossbar. Each switching element s_i can either swap both inputs or pass the inputs to the corresponding outputs.

The ejection arbiters, which are the first three depicted components in Figure 5.2, are

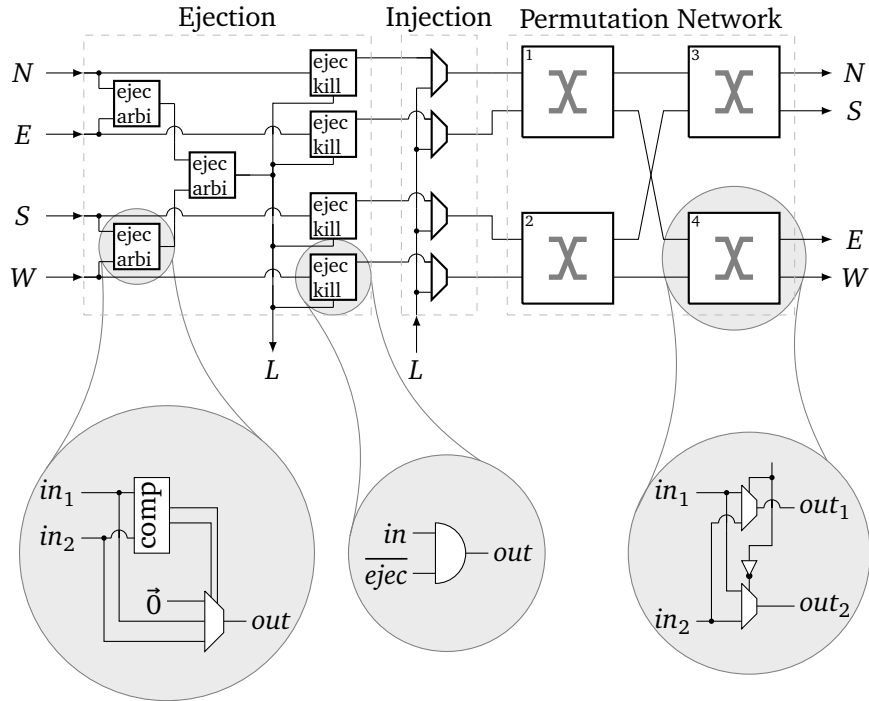


Figure 5.2: Permutation network based router architecture

responsible for selecting the flit which is allowed to be ejected to the local port L . The number of ejection arbiters equals the number of nodes of a perfect binary tree with $\frac{P}{2}$ leaves. As such a tree consists of $P - 1$ nodes, $P - 1$ ejection arbiters are required. Every ejection arbiter compares the destination address as well as the hop count field hc of the two input flits to determine at most one flit that is allowed to eject the router. The hc is compared, as an oldest flit first prioritization scheme is used. If none of the two input flits has reached its destination, an empty flit ($\vec{0}$) is passed to the output. Each arbiter consists of one $3|f|$ -to- $|f|$ MUX and control logic. This logic sets the select inputs of the MUX in the following manner:

$$\text{out} = \begin{cases} \vec{0} & \text{if } in_1(dst) \neq pos \text{ and } in_2(dst) \neq pos \\ in_1 & \text{if } in_1(dst) = pos \text{ and } in_2(dst) \neq pos \\ in_2 & \text{if } in_1(dst) \neq pos \text{ and } in_2(dst) = pos \\ in_1 & \text{if } in_1(dst) = pos \text{ and } in_2(dst) = pos \text{ and } in_1(hc) \geq in_2(hc) \\ in_2 & \text{else} \end{cases}$$

Hence, the logic primarily consists of three comparators. The first two compare the router's position pos with the destination dst of flit in_1 and in_2 . The third one compares

both flits' hop count field hc . For the sake of clarity, these three comparators are depicted as one joint comparator in the lower left of Figure 5.2. Thus, the control logic's hardware requirements depend on the size of the hop count field as well as on the size of the destination field. However, the hardware costs of the control logic are independent of the link width $|LI|$ or flit size $|f|$. Hence, for an increasing flit size, the ejection arbiter's hardware requirements are dominated by the *MUX*.

The second part of the ejection stage are the ejector kill elements, which clear the wires or bits of an ejected flit. The amount of ejector kill elements equals the number of ports P . Depending on the used hardware, they consist of $|f|$ logic gates, e.g. *AND* gates as depicted in Figure 5.2, or one $MUX_{2|f|:|f|}$. For the sake of simplicity, one $MUX_{2|f|:|f|}$ is assumed in the following analysis. A flit will be deleted, or rather the corresponding bits will be set to 0, if this flit has been selected by the ejection arbiters (signal $ejec = 1$). This signal is generated by the ejection arbiters' control logic, and hence, the hardware requirements of one ejector kill element just depend on the flit size $|f|$.

The injection stage allows that one flit can be injected from the local port into the network. It consists of P injection elements, whereas each element allows the injection only if the corresponding router port is idle. Hence, a comparator as well as an arbiter, which avoids multiple injections of the same flit, is required per port. The comparator checks the idleness of the port. However, only the hc field has to be compared, as the hop count of every valid flit is greater than zero. To summarize, the hardware requirements of an injection element are one $MUX_{2|f|:|f|}$, a comparator, and an arbiter, whereas the costs of the latter two components are independent of $|f|$.

In general, the deployed permutation network is a Banyan network, which consists of 2×2 permutation blocks or switching elements. Therefore, a router with P ports requires $\frac{P}{2} \cdot \log_2(P)$ switching elements [Pat01]. Every switching element consists of two $MUX_{2|f|:|f|}$, one inverter, and control logic to set the selection inputs of the two $MUX_{2|f|:|f|}$ (cf. lower right of Figure 5.2). In our implementation, the control logic calculates the productive direction of the higher prioritized flit. As the hop count is used for prioritization, the hc fields of both input flits have to be compared. The productive direction is based on the current router's position pos and the higher prioritized flit's destination address. If the comparators in the ejection stage provide *greater* and *lower* signals, no additional hardware is required for the comparison of the destination. In any case, the control logic's hardware requirements are constant for one switching element and independent of the flit size $|f|$.

Thus, the HardWare Requirements (*HWR*) of one router are given by:

$$\begin{aligned}
 HWR(\text{router}) &\propto (P-1) \cdot (MUX_{3|f|:|f|} + CL) && // \text{ ejection arbiters} \\
 &+ P \cdot MUX_{2|f|:|f|} && // \text{ ejection kill elements} \\
 &+ P \cdot (MUX_{2|f|:|f|} + CL) && // \text{ injection stage} \\
 &+ \frac{P}{2} \cdot \log_2(P) \cdot (2 \cdot MUX_{2|f|:|f|} + INV + CL) && // \text{ permutation network} \quad (5.1)
 \end{aligned}$$

Here, *CL* denotes the hardware requirements of the before mentioned parts of the control logic. The aim of this examination is to figure out an asymptotic course of the hardware requirements of a permutation network based router architecture as a function of the flit size $|f|$. Hence, all components which are independent of the flit size $|f|$ are summed up as *CL*. Please note that the hardware requirements of the parts which are independent of the flit size are negligibly small compared to the costs of the flit size dependent parts. For a router with four ports ($P = 4$), as it is the case for the assumed 2D mesh topology, this leads to

$$HWR(\text{router}) \propto 3 \cdot MUX_{3|f|:|f|} + 16 \cdot MUX_{2|f|:|f|} + CL \quad (5.2)$$

Furthermore, if $MUX_{x|f|:|f|} = |f| \cdot MUX_{x:1}$ and $MUX_{3:1} = 2 \cdot MUX_{2:1}$ is assumed, the final hardware costs of a router are given by:

$$\begin{aligned}
 HWR(\text{router}) &\propto 3 \cdot |f| \cdot MUX_{3:1} + 16 \cdot |f| \cdot MUX_{2:1} + CL \\
 &\propto 22 \cdot |f| \cdot MUX_{2:1} + CL \quad (5.3)
 \end{aligned}$$

In other words, in contrast to [Lee+13a], these results indicate that the hardware requirements of one router increase only linearly, and not quadratically, with the flit size $|f|$.

Synthesis Results

To get an impression of the real hardware costs, and confirm the theoretical results, the router architecture depicted in Figure 5.2 was synthesized using Xilinx's XST [Xil15]. Please note that XST synthesizes a design for FPGAs and does not use a standard cell library like ASIC synthesis software. Nevertheless, this enables us to compare the hardware requirements for different flit sizes and link widths. Figure 5.3 shows the number of required LUTs for one router and different flit sizes / link widths. The two largest router parts are the permutation network and the ejection arbiter blocks. Both increase linearly with the flit size, as expected from the theoretical results. The rest (mainly the ejector kill and injection stage) also increases linearly. The number of required LUTs for one entire router increases by about 80% if flit size and link width are doubled.

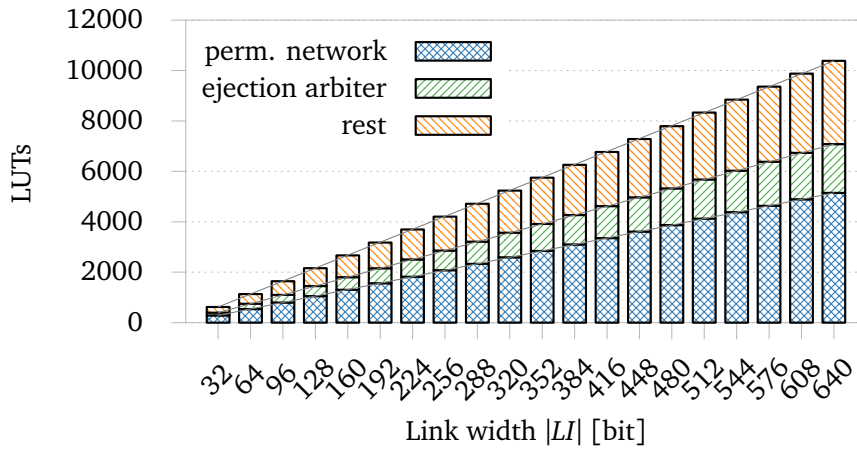


Figure 5.3: Synthesis results for one router and different flit sizes.

5.3.2 Effect of the Flit Size on Performance

In Section 5.3.1, the hardware costs for different flit sizes have been determined. The effect of the flit size on performance metrics like throughput and latency is less obvious, as those values depend on the present network traffic. On the other hand, the calculation of most statistic measures, as diameter and bisection bandwidth, does not differ fundamentally between bufferless, permutation network based router architectures and buffered, crossbar based architectures. Therefore, the performance evaluation in this section is primarily simulation based.

Link width If the impact of the link width on the network performance is investigated, all reasonable link widths have to be determined first. Let us assume that the link width equals the flit size, as it is the case in most NoC implementations. Every flit has to contain a $|hd|$ wide header, which is required for routing. Furthermore, it contains $|pl|$ payload bits. If the size of the message to be transferred exceeds the size of the flits' payload field, the message has to be divided into several sub-messages. All flits corresponding to the sub-messages are independently routed from sender to receiver. Hence, these flits can arrive out of order at their destination, as some may make a detour because of deflections. As the original message can be distributed over several flits, the flit structure has to be extended by a flit-ID field id , which determines the sequence of the flits of one message. The size of the flit-ID field is denoted as $|id|$. Thus, the link width and flit size is given by

$$|LI| = |f| = |pl| + |hd| + |id| \quad (5.4)$$

With Equation (5.4), the lower and the upper limit of the flit size or the link width is

straightforward. Flits have to be at least as large as the header width $|hd|$, plus one bit of payload, as every transferred flit should contain any information content, and the size of the flit-ID, which is required for reassembly. Furthermore, a flit size that exceeds the maximum message size $|msg|_{\max}$ plus the header information does not offer any advantage. If the links are as wide as the largest message plus the size of the header information, no flit-ID is required and $|id| = 0$. Hence, the link width which has to be considered is limited by:

$$\begin{aligned} |LI| \in [|LI|_{\min}, |LI|_{\max}] &= [1 + |hd| + |id|, |msg|_{\max} + |hd|] \\ &= [1 + |hd| + \lceil \text{ld}(|msg|) \rceil, |msg|_{\max} + |hd|] \end{aligned} \quad (5.5)$$

It can be assumed that the message size, as well as the header width, is fixed. Hence, the flit-ID field, which just stores a sequence number, is the only unknown variable in Equation (5.4). For $|LI|_{\min}$, exactly one bit of payload is transferred, and hence, $\#f = |msg|$ flits are required to transmit a message msg . This leads to $|LI|_{\min} = 1 + |hd| + \text{ld}(|msg|)$. In general, the size of the flit-ID field depends only on the number of flits per message $\#f$. This number in turn depends on the size of a message, which has to be distributed among several flits, and the capacity of the flits' payload field: $\#f \cdot |pl| \geq |msg|$. With Equation (5.4), the number of flits per message, for a fixed $|msg|$ and $|hd|$, and a concrete $|LI|$, is given by:

$$\#f = \left\lceil \frac{|msg|}{|pl|} \right\rceil \stackrel{(5.4)}{=} \left\lceil \frac{|msg|}{|LI| - |hd| - |id|} \right\rceil = \left\lceil \frac{|msg|}{|LI| - |hd| - \lceil \text{ld}(\#f) \rceil} \right\rceil \quad (5.6)$$

The number of flits per message $\#f$ appears on the left side, as well as in the denominator, of Equation (5.6). This equation can be transformed (cf. Appendix A.2) with the Lambert W function to:

$$\#f \geq \frac{-|msg| \cdot \ln(2)}{W(-|msg| \cdot \ln(2) \cdot 2^{|hd| - |LI|})} \quad (5.7)$$

Therefore, calculating the number of flits per message for a given link width can be complex. However, as the computing effort of an iterative program which solves Equation (5.6) is quite low, such a program was used instead of Equation (5.7). Depicted in Figure 5.4 is the number of flits per message for a concrete example ($|msg| = 32$ bit, $|hd| = 16$ bit) and a variable link width. The link width varied between $|LI|_{\min} = 22$ bit and $|LI|_{\max} = 48$ bit. It can be seen, not every link width in this range is reasonable, i.e. Pareto optimal, as some link widths lead to the same number of flits per message, and additionally to more unused link wires. The Pareto optimal link widths are highlighted in orange. If e.g. the Pareto optimal link width of $|LI| = 26$ bit is used, this leads to $\#f = 4$ flits per message, $|id| = 2$ bit, and a payload size of $|pl| = 8$ bit. The non-optimal link width of $|LI| = 27$ bit would also lead to the same numbers, but additionally to more unused link wires.

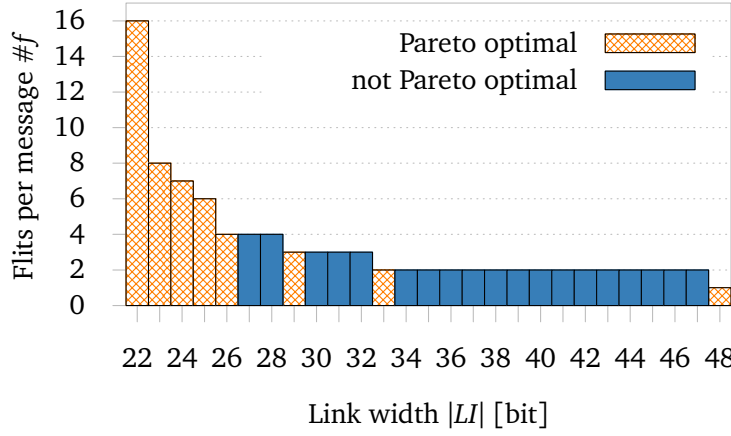


Figure 5.4: Number of transferred flits per message $\#f$ calculated according to Equation (5.6), for $|msg| = 32$ bit, $|hd| = 16$ bit, and a variable link width $|LI|$.

If the number of flits per message has to be calculated, Equation (5.6) can be used. However, if the main goal is to just determine the Pareto optimal link widths, a less computationally intensive method exists. Rather than calculating the required number of flits per message for a given link width, the link width for a given number of flits per message can be calculated:

$$\begin{aligned} \left\lceil \frac{|msg|}{\#f} \right\rceil &= |pl|_{\#f} = |LI|_{\#f} - |hd| - |id| \\ \Rightarrow |LI|_{\#f} &= \left\lceil \frac{|msg|}{\#f} \right\rceil + |hd| + |id| \end{aligned} \quad (5.8)$$

Here, $|pl|_{\#f}$ and $|LI|_{\#f}$ denote the minimum payload size and the minimum link width, respectively, which are required for a specific number of flits per message. Figure 5.5 shows the required link width for a variable $\#f \in [1, |msg|]$. It can be seen, that the same Pareto set as for Figure 5.4 is achieved. The Pareto optimal values are highlighted in yellow.

Latency An important performance metric of a NoC is the average latency of all transferred messages. If deflection routing and several independently routed flits are used, the latency l of one message is given by the time difference of the last ejected flit and the injection time of the first flit of this message:

$$l = \max_{1 \leq i \leq \#f} (t_{\text{ejec}}(f_i)) - t_{\text{injec}}(f_1) \quad (5.9)$$

Here, the ejection time and injection time of a flit f_i are denoted as $t_{\text{ejec}}(f_i)$ and $t_{\text{injec}}(f_i)$, respectively. Please note that the last received flit does not necessarily have to be the

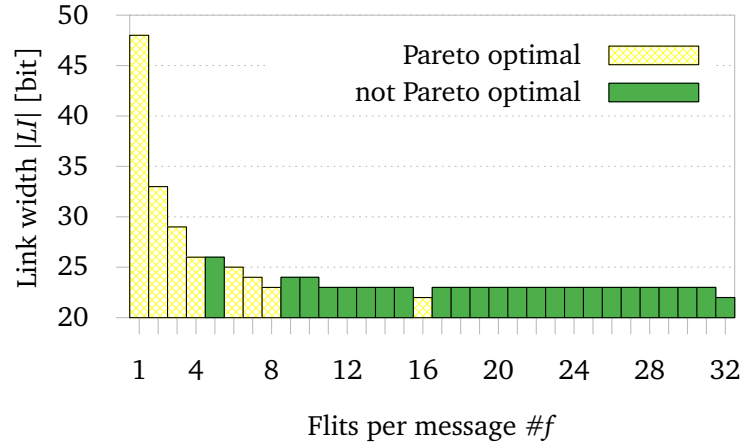


Figure 5.5: Link width $|L|$ calculated according to Equation (5.8), for $|msg| = 32$ bit, $|hd| = 16$ bit, and a variable number of flits per message $\#f$.

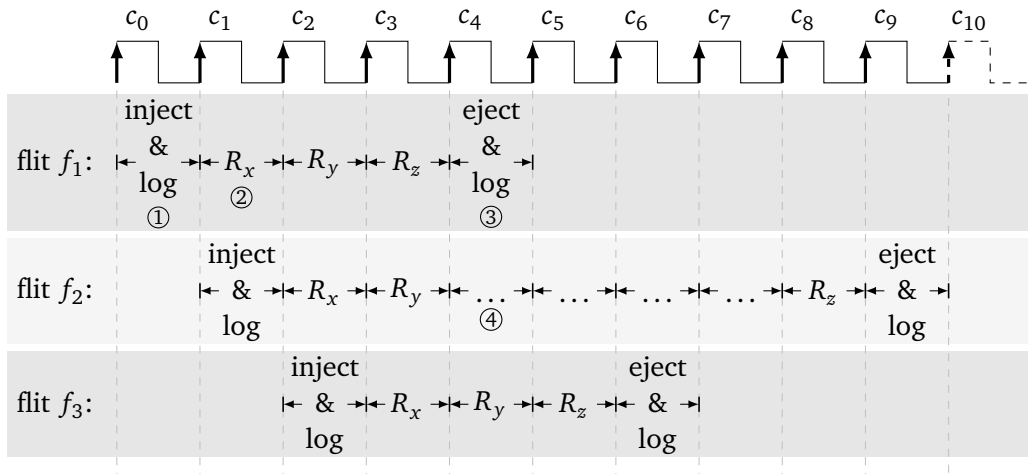


Figure 5.6: Transmission of three flits between two two-hop neighbors.

last injected one. Figure 5.6 shows an example of one message that consists of three flits being transmitted from switch R_x to its two-hop neighbor R_z . In clock cycle c_0 , the first flit is injected (cf. ① in Figure 5.6) from the core and the log message is printed. In the next cycle c_1 , the first flit f_1 traverses router R_x and the second flit f_2 is injected ②. In clock cycle c_2 , f_1 traverses R_y , whereas f_2 traverses R_x and f_3 is injected. In clock cycle c_4 , flit f_1 is ejected at R_z ③. It is assumed that flit f_2 , which has to traverse R_z in this cycle to follow the shortest path between R_x and R_z , is deflected (cf. ④ in Figure 5.6) because of contention and arrives only in clock cycle c_8 . Hence, flit f_3 has overtaken the second flit, as f_3 followed the direct path to R_z .

If a sufficiently dimensioned injection queue is deployed, as it is the case in the simulations presented in the following sections, it can be assumed that flit injections are never blocked. In other words, flits are deemed as injected as soon as they are stored in an injection queue. This means, the i -th flit f_i of a message msg_i is injected $i - 1$ clock cycles after the first flit f_1 has been injected. Hence, the latency l for a message which consists of $\#f$ flits is given by:

$$l = \max_{1 \leq i \leq \#f} (l_i + i - 1) \quad (5.10)$$

whereas l_i denotes the latency of the i -th flit.

If there is always an idle input port at every router in every network clock cycle, the cores can inject new flits directly into the network. In this case, an injection queue is not required and the latency of a flit is determined by the time the flit spends in the network. Hence, the latency of a flit f is given by: $l_f = hc_f + 1$. With Equation (5.10), this leads to:

$$l \geq l_{\text{no queue}} = \max_{1 \leq i \leq \#f} (hc_i + i) \quad (5.11)$$

If additionally no congestion occurs, and hence, no deflections at all occur, the hop count of a flit equals the Manhattan distance between the source node and the destination node of this flit. With Equation (5.11), this leads to:

$$l_{\text{no queue}} \geq l_{\text{no deflections}} = \max_{1 \leq i \leq \#f} (d_1(f_i) + i) = d_1(msg) + \#f = l_{\text{min}} \quad (5.12)$$

Here, the Manhattan distance of a flit f_i and a message msg is denoted by $d_1(f_i)$ and $d_1(msg)$, respectively. Equation (5.12) is also a lower bound of a message's latency, as no congestion in the network as well as in the injection queues is assumed.

Simulated Results with Uniform Random Traffic

To evaluate and compare the performance characteristics of different link widths and flit sizes, an 8×8 NoC is simulated using our in-house cycle accurate simulator implemented in VHDL. Every router is connected to a traffic generator, which is able to generate

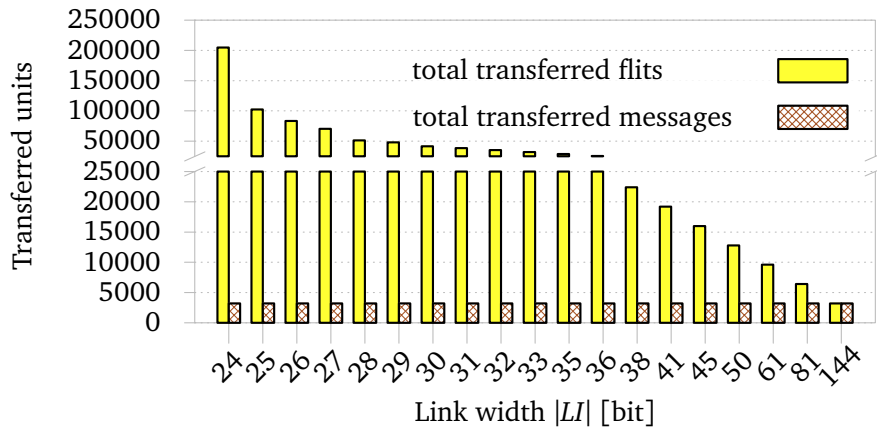


Figure 5.7: Number of transferred flits and transferred messages for a variable flit size / link width.

different types of network traffic. Messages are injected at every router with a given injection probability, which was set to $\alpha = 1\%$ or $\alpha = 0.1\%$. Please note that the real network load is significantly higher than these low injection probability might suggest. For the herein used simulation parameters, messages are divided in up to 64 flits. Thus, the injection probability of a flit is also up to 64 times higher than the message injection probability. If a flit could not be injected at a certain router due to congestion, the flit is stored in the injection queue and is injected as soon as possible. A header information of $|hd| = 16$ bit and a message size of $|msg| = 128$ bit is assumed. All simulations are executed for 5000 clock cycles. Hence, the total number of transferred messages varied between 320 and 3200, for $\alpha = 0.1\%$ and $\alpha = 1\%$, respectively.

According to Equation (5.5), the minimum link width is $|LI|_{\min} = 24$ bit, and the maximum link width is $|LI|_{\max} = 144$ bit. If 144 bit wide links are used, every message can be transferred in one single flit. One could assume that exactly 128 flits are required if 24 bit wide links are utilized. Indeed, Equation (5.8) holds: $|LI|_{\#f=128} = \lceil \frac{128}{128} \rceil + 16 + \text{ld}[128] = 24$ bit. However, if 2 bit of payload per flit are transferred, 64 flits are required and this leads also to $|LI|_{\#f=64} = \lceil \frac{128}{64} \rceil + 16 + \text{ld}[64] = 24$ bit. Hence, $|LI| = 24$ bit, $\#f = 128$ is not a Pareto optimal point. Figure 5.7 shows the number of transferred flits, as well as the number of transferred messages, for every Pareto optimal link width between $|LI|_{\min}$ and $|LI|_{\max}$, and an injection probability of $\alpha = 1\%$. As depicted, the number of flits per message increases exponentially for smaller link widths (e.g. twice as much flits if the link width is reduced from 25 bit to 24 bit) and a constant number of 3200 transferred packets.

According to Equation (5.12), the minimum latency of a message is restricted by the Manhattan distance of the message's source and destination node, and the number

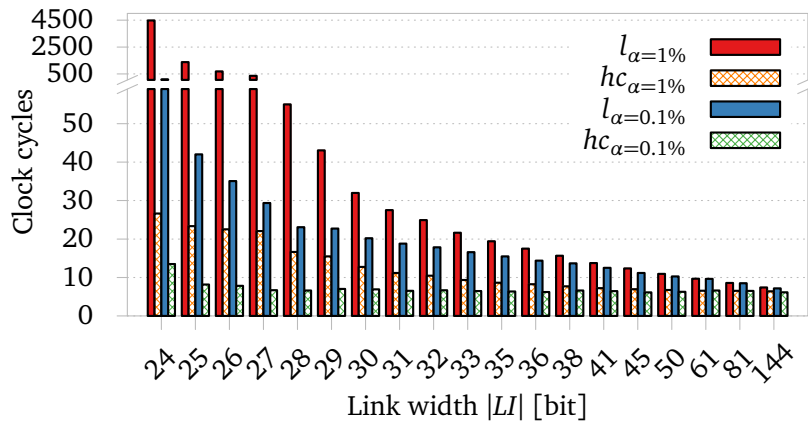


Figure 5.8: Average latency l and average hop count hc for traffic with **uniform** length, injection probabilities of $\alpha = 1\%$ and $\alpha = 0.1\%$, and a variable link width / flit size.

of flits per message. Therefore, as $\#f$ increases exponentially, also the latency has to increase in this order of magnitude for smaller link widths. Simulation results for a variable flit size / link width and uniform random traffic with 128 bit of payload per message are depicted in Figure 5.8. This figure shows the average latency l , as well as the average hc , for two different injection probabilities of $\alpha = 1\%$ and $\alpha = 0.1\%$. It can be seen that the average latency increases exponentially for both, $\alpha = 1\%$ and $\alpha = 0.1\%$. The latency is influenced by $\#f$ directly, due to the exponentially growing number of flits per message, as indicated in Equation (5.12). However, the latency is also influenced by $\#f$ indirectly, as a higher number of flits per message also leads to a higher network load. An increased network load in turn leads to a rising number of deflections and more frequent detours of flits. In our simulations, the network load even rose up to the saturation point of the NoC for small flit sizes. If the network is fully saturated, deflections occur very frequently, but even more importantly, new flits can only be injected if flits which are already in the network are ejected before. The new flits have to stay in the injection queue for this period of time. As every flit has a hop count field of 8 bit, an upper bound for the time a flit stays in the network can be given. In contrast, no upper bound for the queue time exists. According to Equation (5.11), the latency of the complete message is $l_{\text{no queue}} = \max_{1 \leq i \leq \#f} (hc_i + i) \leq 255 + \#f$ if an 8 bit hc -field and a nonexistent queue time is assumed. However, the difference between the increase of the latency and the hop count is much higher, as depicted in Figure 5.8. Hence, the indirect influence of an increased $\#f$ on the latency is considerably greater than the direct influence, due to the unbounded queue time.

In summary, despite the exponential increase of the latency, a reduction of the link

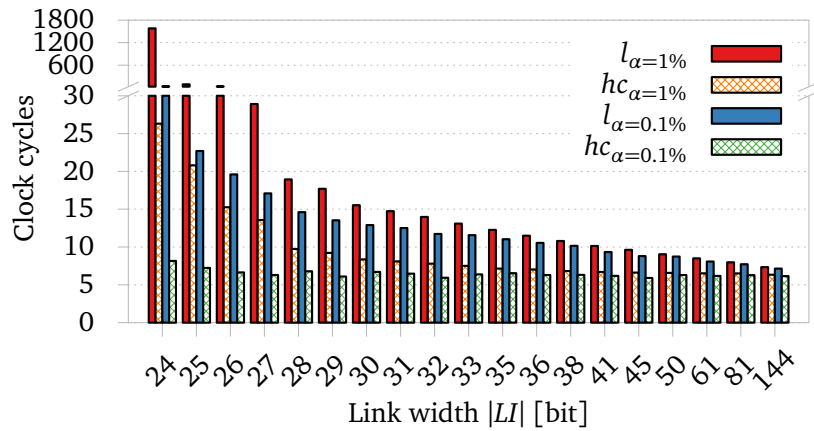


Figure 5.9: Average latency l and average hop count hc for traffic with **non-uniform** length, an injection probability α of 1% and 0.1%, and a variable link width / flit size.

width can be an option for low injection probabilities and short messages. If the link width is reduced from 144 bit to 61 bit, the injection probability would be three times higher, but the average latency for an injection probability of 1% would deteriorate only by 2.3 clock cycles.

Simulated Results with Non-Uniform Random Traffic

A uniform message size of $|msg| = 128$ bit was used for the simulations of Figure 5.8. However, real traffic for MPSoCs usually does not have a uniform length [YBM03]. Instead, it consists of larger messages (e.g. data fetch or data update packets) as well as shorter messages (e.g. memory access request packets). As the message size affects the number of flits $\#f$, which in turn influences the network load, the simulations are repeated with traffic which is more similar to real MPSoC traffic. For Figure 5.9 half of the transferred messages have a size of 128 bit and the other packets consist of just one flit per message. Hence, the number of transferred messages remained equal to the simulations with a uniform message length, but the total number of transferred flits is reduced. The factor of reduction depends on the utilized $|LI|$, or rather $\#f$. For a link width of $|LI| = 24$ bit, 64 flits per message are required if a uniform message length of $|msg| = 128$ bit is assumed. For a non-uniform message length of $|msg| \in \{1 \text{ bit}, 128 \text{ bit}\}$, only 32.5 flits per message are required on average. As the network load is reduced, the average latency is also improved compared to the simulations with uniform message length. The maximum average latency, which occurred at a link width of 24 bit, decreased from 4473 clock cycles to 1580 clock cycles. If, for instance, the injection probability is 1% and an average latency of 15 clock cycles is tolerable, a link

$ Ll $ [bit]	576	288	144	96	64	32	16
#f large messages	1	2	4	6	9	18	36
#f small messages	1	1	1	1	1	2	4

Table 5.1: Simulated link widths, and the consequential number of flits per large message as well as per small message.

	link width	576	288	144	96	64	32	16
Exec. time [10^6 cycles]	blackscholes	50.95	50.95	50.95	50.95	50.95	50.95	50.96
	canneal	55.52	55.52	55.52	55.52	55.52	55.52	55.52
	x264	52.98	52.99	52.99	53.00	53.00	53.03	53.06
Inj. prob. α [10^{-3}]*	blackscholes	1.38	2.04	3.36	4.68	6.67	13.33	26.66
	canneal	1.69	2.47	4.05	5.62	7.97	15.95	31.89
	x264	0.29	0.44	0.73	1.02	1.45	2.90	5.80
Avg. hc	blackscholes	6.38	6.38	6.39	6.41	6.46	6.70	7.71
	canneal	6.78	6.78	6.79	6.82	6.89	7.28	8.81
	x264	6.53	6.52	6.55	6.61	6.73	7.31	8.93
Avg. l [cycles]	blackscholes	7.38	7.87	8.84	9.83	11.32	16.48	28.12
	canneal	7.78	8.25	9.21	10.18	11.66	16.98	30.34
	x264	7.53	8.03	9.05	10.12	11.76	17.98	42.05

*Injected flits per clock cycle per node

Table 5.2: PARSEC simulation results, for seven different link widths / flit sizes.

width of 41 bit will be required for uniform traffic length ($|msg| = 128$ bit) and a link width of 31 bit will be required for non-uniform traffic ($|msg| \in \{1 \text{ bit}, 128 \text{ bit}\}$).

Application Performance

In order to assess the impact of a reduced flit size / link width on application performance, the three PARSEC benchmarks, blackscholes, canneal, and x264, are also simulated. For further details concerning the benchmarks and the simulation parameters, please refer to Section 2.6.2. The benchmark traces, which are provided by Netrace [HGK10; HK10], consist of several message types with two different message sizes. Large messages have a size of $72B = 576$ bit and small messages have a size of $8B = 64$ bit. Table 5.1 shows the simulated link widths, and the consequential number of flits per messages for both message types. The considered link widths ranged from 576 bit, which corresponds to a single flit per message, to 16 bit, which corresponds to 36 flits per large message and 4 flits per small message.

Simulation results are depicted in Table 5.2. The first considered value is the execution time. As described in Section 2.6.2, the simulated clock cycles are restricted to approximately 50 million clock cycles, due to simulation times of up to several weeks. Only with x264, the execution time increased slightly for lower link widths. With blackscholes, as well as with canneal, the execution time is almost constant for all link widths. This can be traced back to two main reasons. First, there is only a short chain of message dependencies, at least at the end of the simulation time of these two benchmarks. Second, the average injection probability is quite low, even for $|LI| = 16$, and hence, the NoC is not saturated most of the time. Due to the low traffic load, the average hop count increased also hardly for all simulated link widths. However, the average latency of the whole message increased significantly for lower link widths. According to Equation (5.11), the latency of a message which consists of $\#f$ flits has to be greater or equal to $l_{\text{no queue}} = \max_{1 \leq i \leq \#f} (hc_i + i) \geq hc_{\text{avg}} + \#f$. For instance, the blackscholes traffic consists of 48% larger messages and 52% smaller messages (cf. Table 2.4). For a link width of 16 bit, which corresponds to an average hop count of 7.71, this leads to $l \geq 0.48 \cdot (36 + 7.71) + 0.52 \cdot (4 + 7.71) = 27.07$. Hence, the increase of the latency is dominated by the message fragmentation.

To summarize, the PARSEC benchmark results show, a reduced link width does not have to affect the execution time if the network is neither congested nor the performance bottleneck. For higher injection probabilities, e.g. due to an increased ratio of core frequency to NoC frequency, a performance decrease is to be expected.

5.4 TwoPhases - An Alternating Transmission Scheme

In Section 5.3, the link width and its effect on the performance as well as on the hardware costs for a permutation network based router architecture have been considered. As deflection routing based architectures do not allow packet switching, the link width is of particular importance for such NoCs. Wide links enable high throughputs and prevent the fragmentation of large messages into too many sub-messages. These advantages are gained by higher hardware requirements. According to Section 5.3.1, the hardware costs for one router only increase linearly with the link width or flit size. Nevertheless, limited hardware resources can prevent a link width equal to the maximum message size. Besides of hardware limits, there are further reasons for smaller link widths and router sizes. First, the saved area and energy can be used for other components. Second, router design and wiring is much simpler, as a high number of wires is difficult to place and route. For off-chip routing, the limited number of pins plays a further crucial role. On the other hand, small link widths lead to a significant routing overhead, as every flit has to be a head flit. Furthermore, costly reassembly of all received flits is required, as the flits can arrive out of order.

Hence, transmission schemes for deflection routing based NoCs are desirable, which

allow the transmission of messages that exceed the link width and which additionally mitigate the negative effects caused by very wide links. In this section, an alternating transmission scheme for deflection routing is presented, called *TwoPhases* [RK16a; RK16c], which enables packets that consist of several flits. Thereby, smaller links and a significant reduction of the routing overhead is achievable. The general methodology of this transmission scheme, including the prerequisites and an analytical performance evaluation, is presented in Section 5.4.1. In Section 5.4.2, existing and more obvious transmission methods are introduced. Finally, these existing approaches are compared to *TwoPhases* in Section 5.4.3.

5.4.1 Methodology of *TwoPhases*

At deflection routing, there is generally no distinction between packets and flits, as every flit has to carry routing information. This is different at the new transmission scheme *TwoPhases*. Here, every packet consists of exactly two flits, at least if a single-cycle router architecture is assumed. These two flits are one head flit h , which carries the necessary routing information, and one body flit b , which follows its head flit. Thereby, the ratio of routing overhead to payload can be improved, as the body flit contains no routing overhead.

Prerequisites and General Operating Mode

In order to apply *TwoPhases*, some prerequisites have to be fulfilled. First, as *TwoPhases* is based on deflection routing, every router must have at least as many output ports as input ports. Second, if the routers represent vertices and the links represent edges, the corresponding graph has to be bipartite. In other words, all routers have to be divisible into two disjoint sets S_0 and S_1 , so that every router of S_0 is only neighbored to routers of S_1 and vice versa. Third, the link width and router width has to be at least as wide as the header information. Furthermore, a single-cycle router architecture is assumed. Results for multi-cycle router architectures are presented on Page 138. All these requirements can be fulfilled for a 2D mesh topology, which is assumed as the chosen topology. Here, the division into two disjoint sets corresponds to a checkerboard pattern (cf. Figure 5.10).

As mentioned before, *TwoPhases* allows the transmission of packets instead of solely flits, as it is the case for standard deflection routing. These packets consist of exactly one head flit h and one body flit b . The body flits only contain payload, and in particular, no routing information. Hence, they have to follow their head flit immediately, in order to reach their destination. Every network clock cycle, every router alternates between two different modes. These two modes are the head mode (routers are colored red) and the body mode (routers are colored blue). In head mode, routers expect only head flits, and determine new routing decisions based on head flits that have just arrived. In

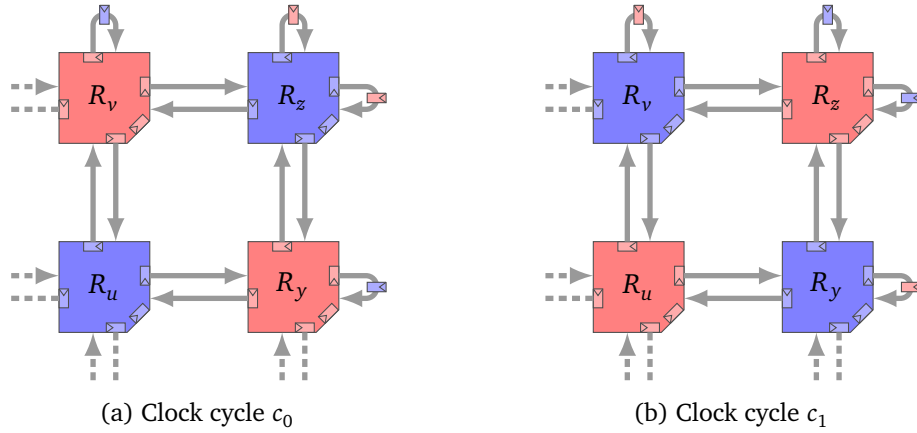


Figure 5.10: Mesh topology with *TwoPhases* for two consecutive clock cycles. In clock cycle c_0 , routers of S_0 (S_1) are in head mode (body mode), and thus, colored in red (blue). In the subsequent clock cycle c_1 , all routers are in their alternated mode.

body mode, routers expect only body flits, and merely hold their configuration from the last clock cycle (e.g. the routing decision as well as the injection and ejection state). As the routers alternate between these two modes, a router's mode changes to head mode (body mode) in the next clock cycle if and only if this router is in body mode (head mode) in the current clock cycle. An example with two consecutive clock cycles c_0 and c_1 is depicted in Figures 5.10a and 5.10b. In general, the mode of an arbitrary router R_i at clock cycle c_j can be expressed as follows¹⁸:

$$\begin{aligned}
 R_i \text{ in head mode (colored red)} &\Leftrightarrow R_i \in S_0 \wedge c_{j \bmod 2} \equiv c_0 \vee \\
 &\quad R_i \in S_1 \wedge c_{j \bmod 2} \equiv c_1 \\
 R_i \text{ in body mode (colored blue)} &\Leftrightarrow R_i \in S_0 \wedge c_{j \bmod 2} \equiv c_1 \vee \\
 &\quad R_i \in S_1 \wedge c_{j \bmod 2} \equiv c_0
 \end{aligned} \tag{5.13}$$

Even if deflection routing is inherently free from deadlocks, livelocks can occur. At standard deflection routing, livelocks can be avoided if the highest prioritized flit is always routed one hop closer to its destination, and never deflected away from it. The livelock problem usually prevents the use of packet switching at deflection routing based NoC, as shown in Section 5.1.2. However, packets which consist of two flits are used at *TwoPhases*. Thus, the abstinence of livelocks has to be shown for this new transmission scheme. At *TwoPhases*, data flits just follow their head flits. In particular, routing computation and the path arbitration takes place only in head mode. Thus,

¹⁸ c_j denotes the j -th clock cycle and, for reasons of clarity, $c_j, j \bmod x \equiv y$ is abbreviated as $c_{j \bmod x} \equiv c_y$

it is sufficient to show that head flits do not livelock, i.e. every head flit reaches its destination eventually, to guarantee the abstinence of livelocks. The general operating mode of *TwoPhases* in head mode equals the process at standard deflection routing. Standard deflection routing is livelock free if the highest prioritized flit is routed to a productive direction, which is also the case at *TwoPhases*. Thus, it is sufficient to show that head flits do not leave head mode, i.e. only arrive at routers which are in head mode in this network clock cycle. The network interfaces inject new flits only if the corresponding router is in head mode. Therefore, a head flit is always in the correct mode in its first network clock cycle. Additionally, the bipartite network graph as well as the alternating router modes ensure that this is maintained until the flit reaches its destination. Therefore, head flits do not livelock and as a consequence, *TwoPhases* is livelock free.

There is one last prerequisite, which does not affect all deflection routing based NoCs. At some implementations, the output ports of routers, which are placed at the border of a 2D mesh topology, are fed back to the corresponding inputs ports. This ensures that the same number of input and output ports are used and the same router architecture can be deployed, independent of a router's position. Further, links act as a kind of buffer space at deflection routing. However, as routers in *TwoPhases* alternate between head mode and body mode, a flit which is sent to such a loop link would arrive one clock cycle later at the same router. A head flit would now arrive in body mode and vice versa. To prevent this, a simple register has to be inserted in every loop link (cf. Figure 5.10). These registers delay every flit by one clock cycle. Thus, it is ensured that head flits (body flits) arrive only if the router is in head mode (body mode).

Performance

In this section, the theoretical performance and the costs of *TwoPhases* are compared to standard deflection routing. The network latency for one specific flit with standard deflection routing is $l_{DR} = hc + 1$, whereas hc denotes a flit's hop count. At *TwoPhases*, flits can only be injected if the corresponding router is in head mode, which is the case in half of the clock cycles. Further, after a head flit has arrived at its destination, one additional clock cycle is required to transmit the corresponding body flit to its destination. Thus, the theoretical minimum network latency for one specific flit, which arrives with a hop count of hc , is given by:

$$l_{TP} = l_{DR} + 0.5 + 1 = hc + 2.5 \quad (5.14)$$

Equation (5.14) gives only a lower bound for the latency, as a network without congestion is assumed. However, if the NoC is congested, flits may not be injectable into the network. In this case, packets might stay in the injection queue for several clock cycles. Hence, the latency of a packet consists of the network latency and the queue latency, whereas the

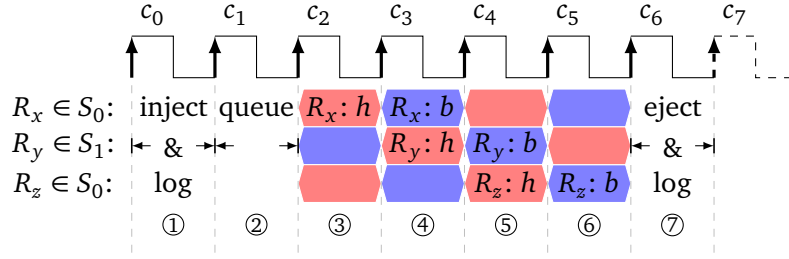


Figure 5.11: Transmission of one packet, which is routed with *TwoPhases* from R_x to its two-hop neighbor R_z via R_y .

latter can dominate the packet latency if the network is highly congested. Simulation results are presented in Section 5.4.3.

An example for one packet transmission is depicted in Figure 5.11. Here, a packet is transmitted from R_x to its two-hop neighbor R_z . In clock cycle c_0 , the head flit h is injected and the core prints the log message (cf. ① in Figure 5.11). In the subsequent cycle c_1 , the flit has to stay in the queue ②, as $R_x \in S_0$ and routers of S_0 can only accept head flits in clock cycles with $c_{j \bmod 2} \equiv c_0$. In cycle c_2 , the head flit h traverses the router R_x ③ and h will arrive at R_y at the subsequent clock pulse edge. Please note, for clock cycles c_2, \dots, c_5 , the flits as well as the modes of all three routers are depicted. Router R_x , R_y , and R_z are depicted in the first, second, and third row, respectively (cf. left of Figure 5.11). In cycle c_3 , the body flit b traverses R_x and h traverses R_y ④. In cycle c_4 , b traverses R_y , whereas h traverses R_z ⑤. In cycle c_5 , the body flit traverses R_z , whereby the complete packet has arrived at R_z ⑥. Finally, the packet is ejected ⑦ and the receiving core prints the log message in cycle c_6 .

Even if the average latency may increase compared to standard deflection routing, the header overhead decreases by more than half. For a link width of $|LI|$ bits and $|hd|$ header bits, only $|LI| - |hd|$ payload bits can be transmitted per flit with standard deflection routing. In contrast, $2 \cdot |LI| - |hd|$ payload bits per packet are possible with *TwoPhases*. Therefore, the link width and also the width of all internal router architecture parts can be reduced, which allows resource and energy savings. If the same link width is used, the throughput can be increased compared to standard deflection routing. In general, if a message of $|msg|$ bits and additionally $|hd|$ bits of header information should be transmitted with *TwoPhases*, the required link width is given by:

$$|LI| = \max \left\{ \frac{|msg| + |hd|}{2}, |hd| \right\} \quad (5.15)$$

The benefits of *TwoPhases* compared to standard deflection routing gain in importance if small flits, and therefore small link widths and router widths, are deployed. For such small flit sizes, which can be necessary due to resource constraints, the header overhead is particularly significant.

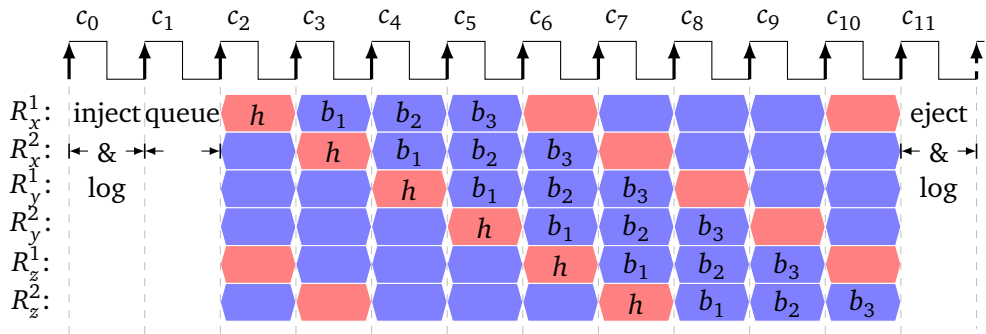


Figure 5.12: Transmission of one packet, which is routed with *TwoPhases* and a pipelined router architecture with two stages from R_x to its two-hop neighbor R_z via R_y .

Optimality of Decomposition into Flits

TwoPhases enables the use of packets which consist of exactly two flits, one head flit and one data flit. The head flits arbitrate the paths, whereas the body flits follow their head flits. Legitimately, one may ask why not more than two flits are used. Unfortunately, it is not possible to split packets into more than two flits at deflection routing based NoCs, which are based on single-cycle router architectures. This can be proved by a counter-example.

We consider two neighbored routers R_x and R_y . It is assumed that these routers have a single-cycle router architecture and packets consist of at least three flits, a head flit h and at least two flits b_1, b_2 . To avoid livelocks, head flits may only arrive at clock cycles at which the receiving router is in head mode. This assumes that only head flits and no body flits are allowed to arrive at the input ports of router R_x in clock cycle c_0 . As R_x could transmit one of those head flits to router R_y , R_y expects only head flits in clock cycle c_1 . At the same time, router R_x processes the first body flit b_1 , which follows its head flit h . Now, it is possible that R_y decides to transmit one of the just received head flits to R_x . It is even possible that the head flit h , which was sent from R_x to R_y , is reflected back to R_x . This is the case if all other output ports of R_y are occupied by higher prioritized flits. In cycle c_2 (when R_x should process the second body flit b_2 , as all packets consist of at least three flits, following the adoption), a head flit arrives at router R_x again. However, this is not allowed, as in every clock cycle at every router solely head flits or solely body flits are permitted. Therefore, a decomposition of packets into more than two flits is not possible here.

Pipelined Router Architectures

For single-cycle router architectures, packets can not consist of more than two flits. This is not the case if the router architecture is pipelined. At pipelined routers with n stages, it takes exactly n cycles to transmit a flit from an input port to an output port. Therefore, a head flit will arrive at a router of the same set exactly after $2n$ clock cycles. Thus, packets consist of $2n$ flits, one head flit h and $2n - 1$ body flits b_1, \dots, b_{2n-1} . In contrast to single-cycle router architectures, a head flit is processed just by one pipeline stage of a router within one clock cycle, not by the complete router. Accordingly, the concept of a router's mode has to be adapted. If a router is stated to be in head mode, the first pipeline stage is in head mode and the router can receive new head flits from its neighbored routers.

The division into S_0 and S_1 remains unchanged. Routers of S_0 are in head mode at clock cycles $c_{j \bmod 2n} \equiv c_0$ and routers of S_1 at clock cycles $c_{j \bmod 2n} \equiv c_n$. Accordingly, the routers of S_0 and S_1 are in body mode at clock cycles $c_{j \bmod 2n} \not\equiv c_0$ and $c_{j \bmod 2n} \not\equiv c_n$, respectively. In contrast to *TwoPhases* for single-cycle router architectures, two neighbored routers can be in body mode simultaneously, but never can be in head mode at the same time. At a single-cycle router architecture, two adjacent routers are always in different states. The abstinence of livelocks is still guaranteed, as in every network clock cycle at every router either head flits or body flits arrive.

An example for a packet transmission from router R_x to its two-hop neighbor R_z via R_y is depicted in Figure 5.12. A router architecture with $n = 2$ pipeline stages is assumed for this example. Hence, it takes two clock cycles to transmit a flit from one router to another router. Packets consist of one head flit h , and three body flits b_1, b_2 , and b_3 . It is assumed that $R_x, R_z \in S_1$, and $R_y \in S_0$. Thus, R_x is in head mode at clock cycles c_2, c_6, c_{10} , and in body mode at the remaining depicted clock cycles. As mentioned above, a router is in head mode if the first pipeline stage of this router is in head mode. The first and second pipeline stage of router R_x are denoted as R_x^1 and R_x^2 , respectively. Please note, the six pipeline stages are depicted on the left side of Figure 5.12. These labels correspond to the processed flits and router modes depicted for clock cycle c_2, \dots, c_{10} .

In summary, for a pipelined router architecture, the header overhead can be reduced even more. At n pipeline stages, $2n$ flits per packet and $2n \cdot |LI| - |hd|$ payload bits per packet are possible. However, the average latency of a packet may increase as well. In general, the latency of a packet which has been routed with a n -cycle pipelined router

architecture is given by:

$$\begin{aligned}
 l_{TP_{\text{pipelined}}} &= \frac{\sum_{i=1}^{2n-1} i}{2n} + hc \cdot n + (2n - 1) + 1 \\
 &= \frac{(2n-1) \cdot 2n}{2 \cdot 2n} + (hc + 2) \cdot n = (hc + 3) \cdot n - 0.5 \quad (5.16)
 \end{aligned}$$

As the case for single-cycle router architectures, the injection of new head flits is only possible if the router is in head mode, which is the case in $\frac{1}{2n}$ of all clock cycles. Hence, the average queue time is given by the first term of Equation (5.16). Every router requires n clock cycles to transmit a flit to the next hop. Thus, $hc \cdot n$ clock cycles are required to transmit the head flit to its destination. As $2n$ flits per packet exist, $2n - 1$ additional clock cycles are required to eject the rest of the packet. Finally, one more clock cycle is required to process the packet at the destination and print the log message.

5.4.2 Transmission Methods

In this section, an overview of existing methods to transmit messages which exceed the link width is given, and their performance characteristic is shown.

MultiFlit

The most obvious method to transmit data which exceeds the size of one flit is the transmission in multiple portions, hereinafter referred to as *MultiFlit*. For *MultiFlit*, the message to be transferred is divided into as many flits as necessary, whereas all flits are routed from sender to receiver independently. This transmission method has been assumed in Section 5.3 and the performance of *MultiFlit* has already been investigated in 5.3.2. Hence, only the performance characteristics are briefly summarized here.

The main advantage of this method is that the number of flits is variable. However, these flits can arrive out of order at their destination, as they are routed independently from each other and some may make a detour because of deflections. Thus, all flits which belong to one message have to be received to be able to reassemble the complete message. Further, the flit structure has to be extended by a flit-ID field, which determines the sequence of the flits of one message, as the original message is distributed over several flits. As every flit has to carry routing information, and additionally the flit-ID field has to be appended, the overhead (transmitted data, which is not payload) can be significant, depending on the flit size. The transferred overhead is given by: $\#f \cdot (|hd| + |id|)$. Further, the links have to be as wide as the flits ($|LI| = |f|$).

Serialization

Another method to transfer large amounts of data over small links and additionally avoid the separation into several flits, is the serialization and deserialization of flits. For buffered NoCs, this method has been investigated extensively [Ghi+13; CLC12; HN12; YKH10]. Unfortunately, the results for buffered NoCs can not be transferred to bufferless, deflection routing based NoCs. For buffered, wormhole flow control based NoCs, the routing decision can be calculated as soon as the routing information and, in particular, the destination address of the packet has arrived at a router. In case of a successful buffer arbitration, even the data transfer to the next router can be performed immediately. This is not possible for deflection routing, at least not without major adaptations of the router architecture [MM09]. Bufferless, deflection routing based router architectures merely permute all inputs and outputs. There is no arbitration of buffer space. Instead, it is exploited that flits stay in a router for just one clock cycle, and hence, routers can receive new flits every clock cycle. Therefore, at *Serialization*, every flit is indeed transmitted in several phits, but the routing decisions will nevertheless be calculated synchronously by all routers of the NoC in the same clock cycle. Thereby the abstinence of livelocks is guaranteed, as the highest prioritized flit always gets assigned to its preferred direction.

At *Serialization*, the link width $|LI|$ equals the phit size $|ph|$, whereas the router width corresponds to the flit size $|f|$. Every flit is divided into $\#ph = \lceil \frac{|f|}{|LI|} \rceil$ phits of $|LI|$ bits, which are serially transferred over the links. The router architecture has to be extended by additional buffers at every input and control logic, which controls the phit transmission and the sorting into the correct buffer space. The buffer width has to be $|f| - |LI|$ as all phits except the last one must be stored. This method allows a reduction of the link width up to 1 bit, but the router architecture is still based on the complete flit size. Hence, the link width is limited by:

$$|LI| \in [|LI|_{\min}, |LI|_{\max}] = [1, |msg|_{\max} + |hd|] \quad (5.17)$$

However, the router architecture size can even increase for smaller link widths, due to additional buffers and control logic (cf. Section 5.4.3). In general, if a message size of $|msg|$, a header width of $|hd|$, and $\#ph$ phits per message are assumed, the required link width is given by:

$$|LI|_{\#ph} = \left\lceil \frac{|msg| + |hd|}{\#ph} \right\rceil \quad (5.18)$$

Figure 5.13 shows the transmission of one packet sent from router R_x to R_z via R_y which consists of four phits ph_4, \dots, ph_1 . In clock cycle c_0 , the whole packet is injected and the core prints the log message ①. As all routers operate synchronously and the last phit ph_1 , which completes the whole packet, is expected at clock cycles $c_{j \bmod 4} \equiv c_0$, the packet can not be transmitted to router R_x immediately. This is why the packet

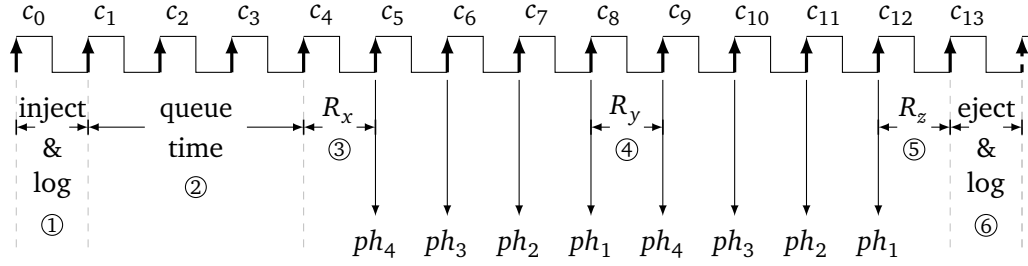


Figure 5.13: Transmission of one message with Serialization between two-hop neighbors.

stays in the queue ② until the fourth clock pulse edge. In the fourth clock cycle, the whole packet traverses R_x ③, but only one phit can be transmitted to R_y at the next clock pulse edge. All four phits are transmitted to R_y after the eighth clock pulse edge. During clock cycle c_8 , the packet traverses R_y ④ and the phit transmission to R_z begins. In clock cycle c_{12} , the whole packet has arrived at R_z ⑤ and the packet can be ejected ⑥ in clock cycle c_{13} .

In general, the theoretical minimum latency of one packet, which arrived with a hop count of hc and consists of $\#ph$ phits, is given by:

$$\begin{aligned}
 l_{SER} &= \#ph \cdot (hc - 1) + 2 + \frac{\#ph - 1}{2} \\
 &= \#ph \cdot (hc - 0.5) + 1.5
 \end{aligned} \tag{5.19}$$

The last term in Equation (5.19) gives the average queue time, as new flits can be injected only every $\#ph$ clock cycles. Please note that this queue time must not be confused with the queue time due to network congestion. The constant 2 is added as it takes one clock cycle to traverse the last router and one clock cycle to transmit the flit to the core and print the log message. The most significant term for the majority of the flits is the first term, which gives the serialization and deserialization time. As the packet is serialized at every router, except the router which receives the packet, this takes $\#ph \cdot (hc - 1)$ clock cycles.

5.4.3 Evaluation

In this section, *TwoPhases* is evaluated and compared to the transmission schemes introduced in Section 5.4.2. One may ask, how the three transmission methods cope with small links. Table 5.3 shows all Pareto optimal link widths for all three methods, a message size of $|msg| = 32$ bit, and a header width of $|hd| = 16$ bit. The same values have been used for the example depicted in Figures 5.4 and 5.5 in Section 5.3.2. Hence,

Method	Pareto optimal link widths												
<i>MultiFlit</i>	48	33	29	26	25	24	23	22					
<i>TwoPhases</i>	24												
<i>Serialization</i>	48	24	16	12	10	8	7	6	5	4	3	2	1

Table 5.3: All Pareto optimal link widths for $|msg| = 32$ and $|hd| = 16$.

the values of *MultiFlit* are taken from there. At *Serialization*, the header information is transmitted only once and can be distributed over several phits. Therefore, the minimum link width is a single bit and the Pareto optimal link widths are determined by means of Equation (5.18). The link width for $\#ph = 1$ is always Pareto optimal and all other Pareto optimal link widths are found by iterating from 2 to $|LI|_{\max}$. A Pareto optimal point is found if $|LI|_{\#ph=i-1} > |LI|_{\#ph=i}$. For *TwoPhases*, there is only one Pareto optimal link width, which is 24 bit, according to Equation (5.15). Of course, *TwoPhases* could be combined with *MultiFlit* or *Serialization* to enable a further reduction of the link width. This means, for instance, several independently routed flits can be used, which would correspond to *TwoPhases* combined with *MultiFlit*. However, as this would significantly increase the design space, pure *TwoPhases* is compared to *MultiFlit* as well as to *Serialization*, whereas similar conditions are used for all of them. Thus, exactly two independently routed flits per message are used for *MultiFlit* and two phits per message for *Serialization*.

One advantage of *MultiFlit*, compared to both other transmission schemes, is that the number of flits per message $\#f$ does not have to be constant. With *Serialization* as well as with *TwoPhases*, all messages have to have a uniform length, as it is the case for standard deflection routing. However, real network traffic can have a non-uniform length [YBM03], i.e. it can consist of larger messages (e.g. data fetch and data update packets) as well as shorter messages (e.g. memory access request packets). Therefore, the three transmission methods are compared to a fourth one, *MultiFlit*_{50:50}. For *MultiFlit*_{50:50}, half of the messages consist of two flits and the other messages consist of only one flit.

Synthetic Traffic Performance

All transmission schemes are simulated for an 8×8 NoC using our in-house cycle accurate simulator implemented in VHDL. Every router is connected to a traffic generator, which is able to generate different synthetic workloads. At every router, messages are injected with a given injection probability. If a message could not be injected at a certain router due to congestion, the corresponding flit is stored in the injection queue and is injected as soon as possible. The injection probability α is varied from 0.1% up to the saturation point of the NoC. Simulations are executed for 5000 clock cycles and every message consists of 128 bit payload and 16 bit header information. As mentioned before, two

flits per message are used for *MultiFlit* and two phits per message for *Serialization*. Thus, for *TwoPhases* as well as for *Serialization*, every flit, or rather packet in case of *TwoPhases*, has a length of 144 bit and 72 bit wide links are used. As *MultiFlit* uses several independently routed flits to transmit larger messages, every flit has to carry routing information and a flit-ID field for sequencing. At these simulations, every flit has a length of $64 + 16 + 1$ bit, whereas the last bit is required to store the flit-ID. Therefore, the links are 81 bit wide for *MultiFlit* and exactly two flits are transmitted per message.

Figure 5.14 shows the average hop count hc as well as the average latency l for all four transmission methods and four different synthetic traffic benchmarks, as described in Section 2.6.1. Please note that all vertically stacked plots share a common x -axis, which denotes the injection probability. The displayed injection probabilities specify the likelihood of a message injection. As a message consists of two flits (in case of *MultiFlit* and *TwoPhases*) or two phits (in case of *Serialization*), the real injection probabilities are up to two times higher.

MultiFlit has the highest average hop count in most simulations (cf. left plots of Figure 5.14) as the deflection rate is higher due to the increased number of independently routed flits. This effect increases with the injection probability and occurs for all four evaluated workloads. The lowest average hop count is achieved with *MultiFlit*_{50:50}, because of the lower network load. For instance, *MultiFlit*_{50:50} is the only transmission scheme that allows an injection probability higher than 10% with bit-complement traffic (cf. Figure 5.14 (e)).

Even more important than the average hop count of the flits is the message latency. The latency of a message can differ significantly from the hop count of the messages' flits, due to the queue time. Furthermore, at *MultiFlit*, all independently routed flits of a message have to be received before the data can be processed. The average latencies which were determined by our simulations are depicted in Figure 5.14 (b), (d), (f), and (h). Please note the broken y -axis for plots (b), (d), and (f). The theoretical minimum average latencies for all four transmission methods, which are based on Equations (5.11), (5.14) and (5.19) are also depicted (smaller point sizes and a brighter color). A zero queue time, and hence, a NoC which is not congested, is assumed for these theoretical lower bounds. At *TwoPhases* and *Serialization*, the difference between the measured latency and the theoretical latency is the queue time. This does not apply for *MultiFlit*, as the herein depicted theoretical latency is based on Equation (5.11). At *MultiFlit*, two flits per message are routed independently and the last injected flit does not have to be the last ejected flit, due to deflections. Hence, the difference between the theoretical latency and the measured latency includes the queue time (due to network congestion) as well as the time caused by varying arrival times of the two flits. The results show, for low injection probabilities, *Serialization* has a higher latency than the other approaches, due to the serialization and deserialization time, and *MultiFlit* and *TwoPhases* perform equally well. However, at higher injection probabilities, the latency with *MultiFlit* increases significantly due to the increased network congestion.

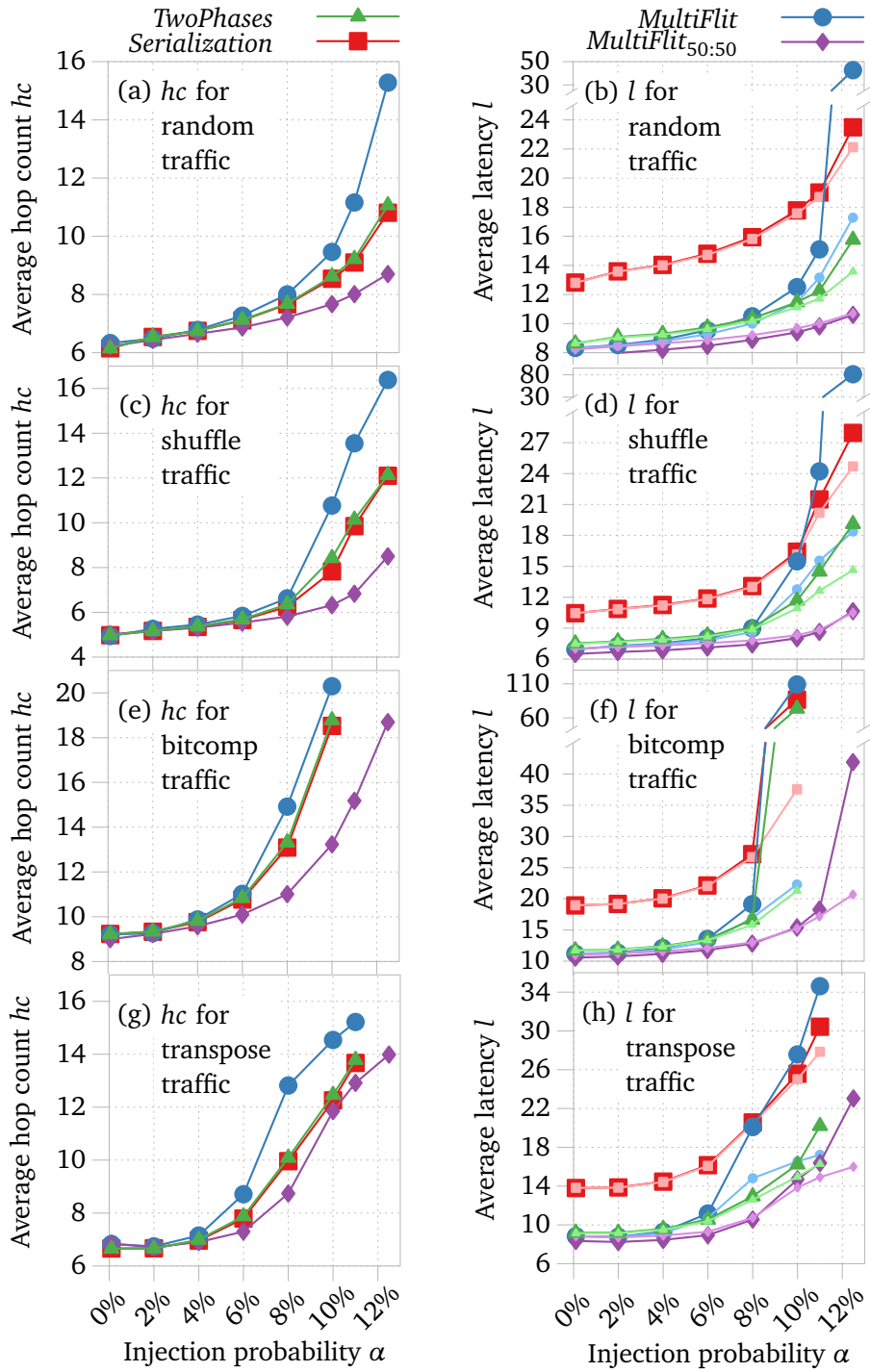


Figure 5.14: Avg. hop count hc and latency l for a variable injection probability α and different synthetic workloads. Please note the common x-axis for all plots.

In particular, this applies for the measured latency of *MultiFlit*, which is considerably higher than the corresponding theoretical latency, due to the above mentioned reasons.

Application Performance

In order to evaluate application performance, three PARSEC benchmarks are simulated additionally, as described in Section 2.6.2. These benchmarks are simulated with *TwoPhases*, *Serialization*, *MultiFlit*, as well as with standard deflection routing. For *TwoPhases* and *Serialization*, all messages are transferred by two flits or two phits, respectively. At *MultiFlit*, the larger messages are transmitted by two independently routed flits, and the smaller messages require only a single flit. At standard deflection routing, which is the performance baseline, the link width is doubled compared to *TwoPhases* and *Serialization*. Hence, large messages as well as small messages are transferred by a single flit.

Table 5.4 shows the simulation results for three different clock frequency ratios. More precisely, it shows the execution time, which is the number of clock cycles that are required to transmit the predetermined number of messages. Further, the injection probability is depicted, which is the number of injected messages per clock cycle per node. Finally, the average hop count of all flits as well as the average latency of the complete messages is shown.

As the network has not been congested in almost all clock cycles (cf. the low injection probabilities in Table 5.4), the messages' latencies are dominated by the time the messages spend in the network. In contrast to the simulations with synthetic traffic, the average queue time, which can arise due to network congestion, is close to zero for these simulations. Hence, the time difference between the average latency and the average hop count is caused by message fragmentation. At *MultiFlit*, for instance, both independently routed flits have to be ejected to be able to reassemble the original message. For standard deflection routing, the average latency l equals the average hop count hc plus one, which is the lower bound for the latency, as every message is transmitted by a single flit.

Figure 5.15 illustrates the speedup of *TwoPhases*, *Serialization*, and *MultiFlit* compared to standard deflection routing for all three evaluated benchmarks. In the first plot of Figure 5.15, an equal core and network frequency (clock ratio $CR = 1$) is assumed. Due to the halved link width, also a reduction of the performance is to be expected. However, as depicted in Table 5.4, the execution times are very close to each other (less than 100 clock cycles at a runtime of $\approx 5 \times 10^7$ clock cycles), even if the average latencies are slightly different. Hence, the NoC is not the performance bottleneck in these configurations.

To increase the network load, the simulations were repeated with two decreased NoC clock frequencies, but an unchanged core frequency. The two reduction factors are 64 and 128, denoted as clock ratio $CR = 64$ and $CR = 128$, respectively. Please note,

			<i>TwoPhases</i>	<i>Serial-ization</i>	<i>MultiFlit</i>	Deflection routing
clock ratio CR = 1	Exec. time [10^6 cycles]	blackscholes	50.95	50.95	50.95	50.95
		canneal	55.52	55.52	55.52	55.52
		x264	52.99	52.99	52.99	52.98
	Inj. prob. α [10^{-3}]	blackscholes	1.38	1.38	1.38	1.38
		canneal	1.69	1.69	1.69	1.69
		x264	0.29	0.29	0.29	0.29
	Avg. <i>hc</i>	blackscholes	6.40	6.39	6.38	6.38
		canneal	6.82	6.79	6.78	6.78
		x264	6.69	6.55	6.52	6.53
	Avg. <i>l</i> [cycles]	blackscholes	8.94	13.74	7.87	7.38
		canneal	9.83	14.87	8.25	7.78
		x264	11.66	16.18	8.03	7.53
clock ratio CR = 64	Exec. time [10^6 cycles]	blackscholes	1.54	1.75	1.48	1.46
		canneal	2.16	2.42	2.01	1.93
		x264	1.75	2.05	1.58	1.54
	Inj. prob. α [10^{-3}]	blackscholes	45.57	40.19	47.52	48.29
		canneal	43.37	38.38	46.13	48.60
		x264	8.92	7.72	9.90	10.16
	Avg. <i>hc</i>	blackscholes	7.67	6.98	7.43	7.00
		canneal	8.57	7.76	7.82	7.39
		x264	7.34	6.93	6.90	6.76
	Avg. <i>l</i> [cycles]	blackscholes	10.85	14.04	9.14	8.00
		canneal	12.38	15.32	9.56	8.39
		x264	16.18	19.91	8.52	7.76
clock ratio CR = 128	Exec. time [10^6 cycles]	blackscholes	1.05	1.18	0.98	0.91
		canneal	1.44	1.60	1.35	1.25
		x264	1.16	1.26	1.09	1.09
	Inj. prob. α [10^{-3}]	blackscholes	66.65	59.82	71.67	76.91
		canneal	63.42	57.34	69.67	75.20
		x264	13.43	12.45	14.34	14.40
	Avg. <i>hc</i>	blackscholes	8.78	7.84	8.14	7.15
		canneal	10.67	10.53	9.69	8.39
		x264	7.69	7.06	7.14	6.87
	Avg. <i>l</i> [cycles]	blackscholes	12.33	15.82	10.04	8.16
		canneal	14.65	18.18	12.22	9.44
		x264	16.78	19.56	8.80	7.87

Table 5.4: PARSEC simulation results, for three different clock ratios.

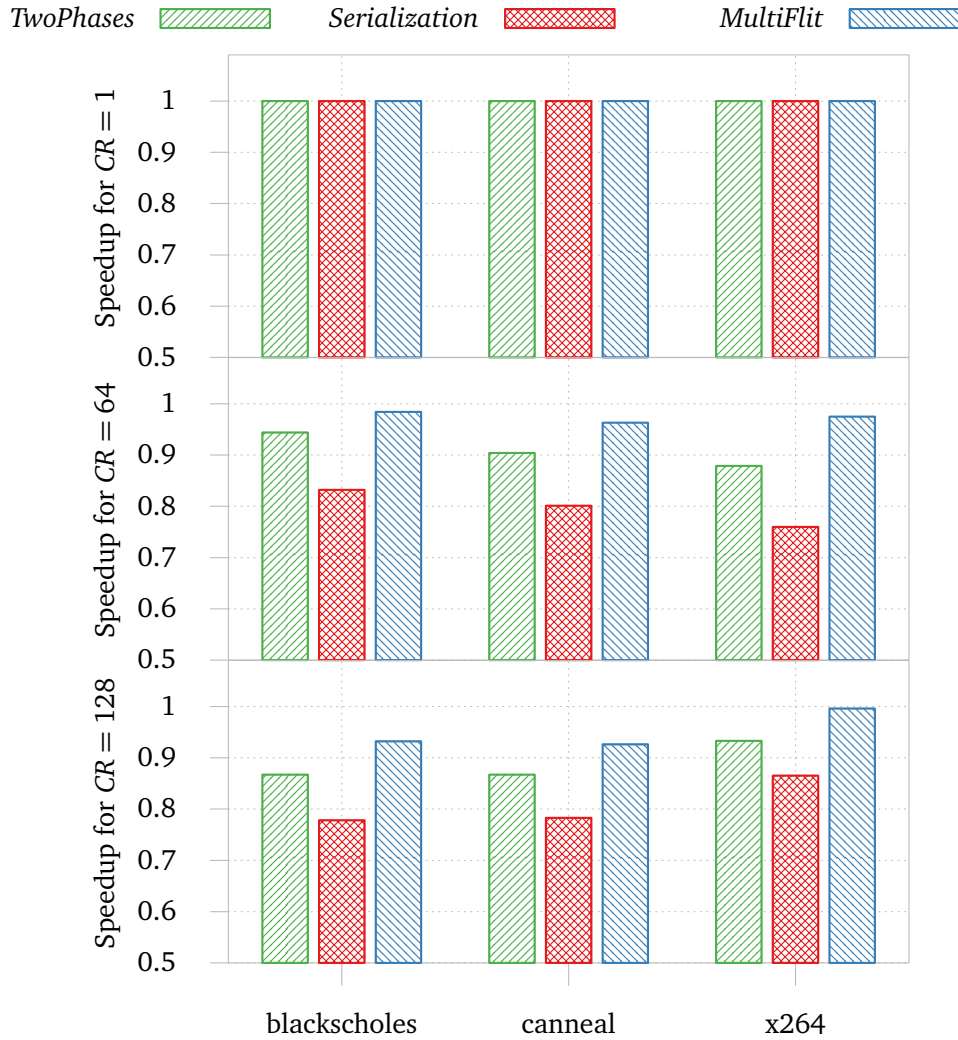


Figure 5.15: Speedup for all evaluated PARSEC simulations, compared to standard deflection routing with links that are as wide as the largest packet. The upper, middle, and lower plot shows the results for a clock ratio of $CR = 1$, $CR = 64$, and $CR = 128$, respectively.

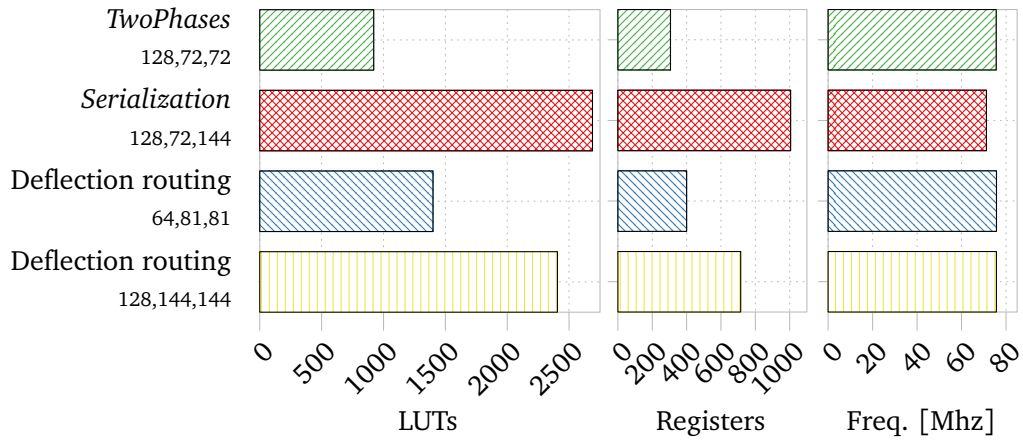


Figure 5.16: Synthesis results for different router architectures. The numbers underneath the labels on the left indicate the used payload size, link width, and router width, respectively.

Netrace ensures that inter-message dependencies are still met. Hence, the resulting injection probabilities increased by a factor lower than 64 or 128 (cf. Table 5.4). Further, the increase of injection probabilities varied between benchmarks and transmission methods. The highest injection probability with *TwoPhases* and $CR = 1$, for instance, is achieved for canneal. However, if the network clock frequency is reduced, the injection probability for blackscholes is higher than for canneal. Due to the higher network load, also the execution times for the four evaluated transmission times varied. The middle plot of Figure 5.15 shows the speedup compared to standard deflection routing and a clock ratio of $CR = 64$. All transmission methods suffer from a performance decrease compared to standard deflection routing. This result was expected, as also the link width is reduced by half for *TwoPhases* and *Serialization*, and by almost half for *MultiFlit*. A more detailed evaluation of the hardware costs is presented in the next section. The best performance is achieved with *MultiFlit*, as larger packets and smaller packets are approximately equally probable, which corresponds to *MultiFlit*_{50:50}. However, even if the resulting average injection probability with $CR = 64$ is much higher compared to $CR = 1$, it is below 5% for all simulations. From the results for synthetic traffic patterns, it is to be expected that the performance gap increases for even higher injection probabilities.

Hardware Costs

To evaluate the hardware cost, all router architectures were synthesized using Xilinx's XST [Xil15]. As described in Section 2.6.2, a Virtex-6 XC6VLX75T FPGA was selected

as target device and the synthesis parameters were changed to optimize for area with normal effort. Figure 5.16 shows the number of required LUTs and the number of registers for a single router. The achievable frequencies are based on the synthesis results for a NoC of dimension 8×8 . The comma separated numbers underneath the labels on the left indicate the required payload size, link width, and router width for the corresponding transmission schemes. Thus, these values are set to transmit 128 bit of payload, or rather 64 bit in case of *MultiFlit*. With *TwoPhases* for instance, each packet consists of 128 bit payload, the link width is 72 bit and the internal router structures also have a width of 72 bit. It can be seen that *TwoPhases* has the least hardware requirements of all presented transmission methods, as both, the link width and router width is smaller compared to the other architectures. Further, only small hardware modifications are required for *TwoPhases*, compared to standard deflection routing. Every router is just extended by a simple one bit state machine, which controls the router's mode, and several registers, which store the router's state when the router switches from head mode to body mode. *Serialization* has the highest demands, because of the additional hardware for serialization and deserialization. In particular, the number of required slice registers is substantially higher than for the other transmission schemes, as additional buffers are required for deserialization. Hence, input buffers as well as output buffers are needed at *Serialization*. Nevertheless, the achievable frequency is the lowest, due to the before mentioned serialization and deserialization hardware. The requirements for a NoC with standard deflection routing, as the case for *MultiFlit*, are depicted in blue. It can be seen that *MultiFlit* has higher hardware requirements than *TwoPhases*, as wider links (81 bit instead of 72 bit) have to be deployed to transmit the same amount of payload. The last bar in this plot shows the hardware requirements for a NoC with routers which are able to transmit 128 bit payload in solely one flit. Hence, this can be considered as an upper bound for the reasonable hardware requirements.

5.5 Summary and Conclusion of Chapter 5

The utilized link width is an important design parameter of NoCs, as it influences the hardware costs directly and may affect the performance of the network. Usually¹⁹, the link width equals the flit size. However, at standard deflection routing, there is no distinction of packets and flits, and every flit contains routing information. Thus, the link width is of particular importance for deflection routing based NoCs.

The effect of the link width on hardware costs and performance has already been investigated for crossbar based, packet switched networks, but not yet for permutation network, deflection routing based NoCs. In Section 5.3, the hardware requirements of permutation network based architectures have been considered, and the theoretical results have been confirmed by synthesis results. Further, equations for performance

¹⁹If flits are not serialized and deserialized

influencing values, as the expedient link widths, have been developed, and performance results for synthetic traffic as well as PARSEC applications have been presented. The outcomes can help system designers to identify the optimal link width and have shown, that a link width determined by the maximum message size does not have to be the most appropriate choice.

Finally, *TwoPhases*, a method to reduce the routing overhead of transferred messages by more than half has been introduced in Section 5.4. Performance and hardware requirement comparisons for *TwoPhases* and existing approaches have shown that *TwoPhases* is efficiently implementable and outperforms the existing approaches in case of a uniform message length. At non-uniform message lengths, *MultiFlit* benefits from a varying number of flits per message.

Concluding Remarks

Contents

6.1 Contributions of this Thesis	151
6.2 Future Work	153

Reliable and power efficient NoCs are of vital importance for the development of future many-core processors. Deflection routing based NoCs constitute a promising approach, as the amount of power consuming flit buffers can be significantly reduced. Further advantages of deflection routing are a low router latency and an inherent abstinence of routing dependent deadlocks. However, bufferless deflection routing also poses several challenges and issues related to the design and application of such NoCs.

6.1 Contributions of this Thesis

In this thesis, the following two key problems related to the design of reliable, power efficient, and deflection routing based NoCs are addressed:

Fault tolerance Future systems have to be able to cope with different types of failures, as it is predicted that they will be ubiquitous in these systems. In Chapter 4, the fault-tolerant FaFNoC router architecture has been introduced. This architecture stands out for the deployed interconnection architecture, which is a Benes network, and the used method to tolerate complex fault situations, which is the concept of FaF.

Permutation networks enable a short router latency, as flit prioritization, route computation, and path allocation are implemented in a decentralized manner. At all existing router architectures based on deflection routing, the utilized permutation networks correspond to Banyan networks. However, these architectures are non-fault-tolerant. In this thesis, several major problems of Banyan networks in terms of fault tolerance have been identified. Furthermore, it has been proven that Benes networks provide a

solution for all identified problems of Banyan networks. Compared to Banyan networks, Benes networks require more hardware resources but support more routing decisions. A complete router architecture for both permutation networks has been implemented and evaluated. Synthesis results have shown that the hardware requirements of both router architecture versions are almost equal, as the Banyan network based architecture requires an additional component which controls the switching elements' behavior in case of link failures. In terms of performance, the Benes network based architecture outperforms the Banyan network based architecture significantly.

If several links fail, complex fault situations can arise which are hard to overcome. Existing fault-tolerant and additionally deflection routing based NoC architectures utilize either routing tables or fault information of adjacent routers. However, routing tables do not scale with a high number of nodes, and fault information of adjacent routers provides only limited fault tolerance. In this thesis, a scalable alternative to these two approaches has been presented, namely the concept of FaF. There, flits which are deflected due to faulty components try to surround the faulty region. Towards this end, a *fault status* field is added to the flit structure and the router architecture is extended by the *fault-status-handler*. This component sets a flit's fault status field to start the fault region evasion modus and clears the field if the faulty region is overcome.

To assess performance and hardware requirements, simulation results for different types of traffic as well as synthesis results have been shown. FaFNoC-Benes outperforms most existing router architectures and, in particular, all flits are delivered to their destination, even at extremely high fault rates. The only router architecture which performs better than FaFNoC-Benes utilizes routing tables, which causes that hardware requirements depend on the number of nodes in the network. The hardware requirements of the FaFNoC router architecture are independent of the number of nodes. Thus, the herein presented FaFNoC architecture provides a scalable solution which is appropriate for NoCs with a huge number of nodes.

Design of Deflection Routing based NoCs The application of deflection routing is linked to a number of challenges. One of these challenges is a proper dimensioning of the link width, which usually corresponds to the flit size. Compared to packet-switched networks, the appropriate link width can be significantly larger, as packet switching and deflection routing can not be combined and every flit has to contain routing information. Furthermore, oversizing and undersizing the link width are not an option, due to the constrained resources on a chip and the importance of a short message latency.

To identify an optimal link width, the effect of the link width on performance and on hardware costs of a permutation network based router architecture has been investigated in Chapter 5. In addition to theoretical performance metrics and formulas for the expected hardware requirements, simulation and synthesis results have been presented. The results have shown that a link width determined by the maximum message size

does not have to be the most appropriate choice.

Furthermore, *TwoPhases*, an alternating transmission scheme for deflection routing based architectures has been developed. *TwoPhases* allows a reduction of the routing overhead to payload ratio by transmitting packets which, in its simplest form, consist of one head flit and one data flit. Comparisons to existing transmission schemes, which enable a reduction of the link width, have shown that *TwoPhases* performs better in case of a uniform message length. At non-uniform message lengths, a transmission scheme which is based on several independently routed flits performs best.

6.2 Future Work

The design of fault-tolerant and deflection routing based NoCs is an important research area, which is even gaining in importance due to ongoing advances in technology and manufacturing processes. Several encouraging results in this area have been achieved and presented in this thesis. Nevertheless, there are pending issues and interesting research opportunities in both areas, the design of fault-tolerant NoCs and the design of deflection routing based NoCs.

The focus of the herein presented FaFNoC router architecture is on tolerating permanent faults at the link and router level. However, as manufacturing processes are shrinking, non-permanent faults, i.e. transient and intermittent faults, also will become more frequent. An holistic approach, which includes several fault tolerance methods with various kinds of redundancy and which operates at different layers, is required to develop fault-tolerant NoCs. Moreover, fault detection is not considered herein. Both tolerating non-permanent faults as well as fault detection are topics of current research. However, an integration of these methods and techniques into a permutation network based router architecture, like FaFNoC, has not been investigated so far.

The appropriate link width is an important parameter of deflection routing based NoCs, yet there are some further pending issues. Several drawbacks of deflection routing have been mentioned in Chapter 3. One of these drawbacks is the reassembly problem, which is also related to the design of network interfaces. The only existing approach which addresses this issue is restricted to certain types of many-core systems, since MSHRs are used as reassembly buffers. Consequently, an approach which is applicable if MSHRs are not available is still missing.

Finally, the herein presented evaluations and findings offer potential for future expansions. In this thesis, the frequently deployed 2D mesh topology was assumed. Various 3D integration schemes have been developed recently, which necessitate 3D interconnections. Depending on the used technology and interconnection method, a 2D mesh might soon no longer be the appropriate topology and routers with a higher radix are required. At permutation networks, this would increase the hardware requirements and, in particular, decrease the maximum frequency of the router architecture.

Furthermore, there is still room for improvement in the herein used evaluation methodology. Power consumption is a key factor of future many-core systems. Thus, the evaluation would benefit from static and dynamic power analysis as well as ASIC synthesis results.

Appendix **A**

Appendix

A.1 Fault Situations of Chapter 4

In Chapter 4, the fault tolerance of five different router architectures has been evaluated. All reported simulation results are averages of three different fault situations. These three fault situations are depicted in Figures A.1 to A.3. For every fault situation, the following four fault probabilities have been evaluated: $\lambda = 0\%$, $\lambda = 10\%$, $\lambda = 20\%$, and $\lambda = 30\%$, whereby the faults of λ_1 are a subset of the faults of λ_2 if $\lambda_1 \leq \lambda_2$. For instance, at $\lambda = 30\%$, the link failures depicted in yellow, orange, red are faulty. At $\lambda = 20\%$, only the link failures depicted in yellow and orange are faulty.

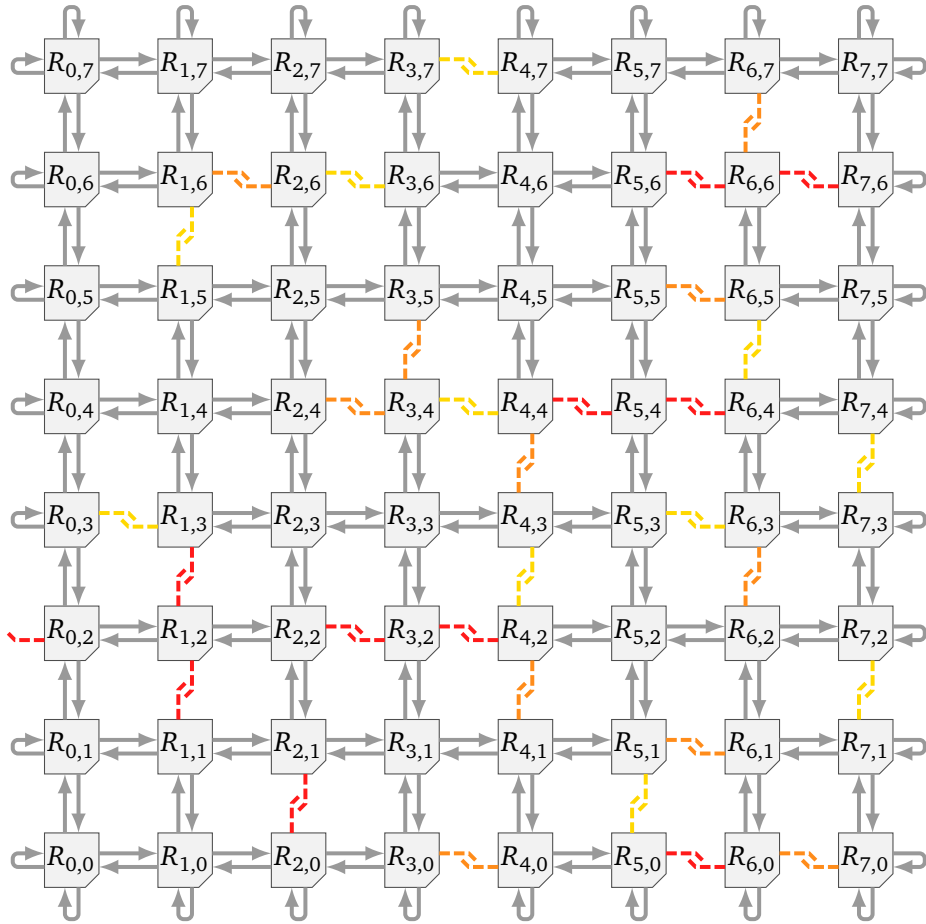


Figure A.1: First simulated fault situation. The link failures depicted in yellow, orange, and red correspond to $\lambda = 10\%$, $\lambda = 20\%$, and $\lambda = 30\%$, respectively.

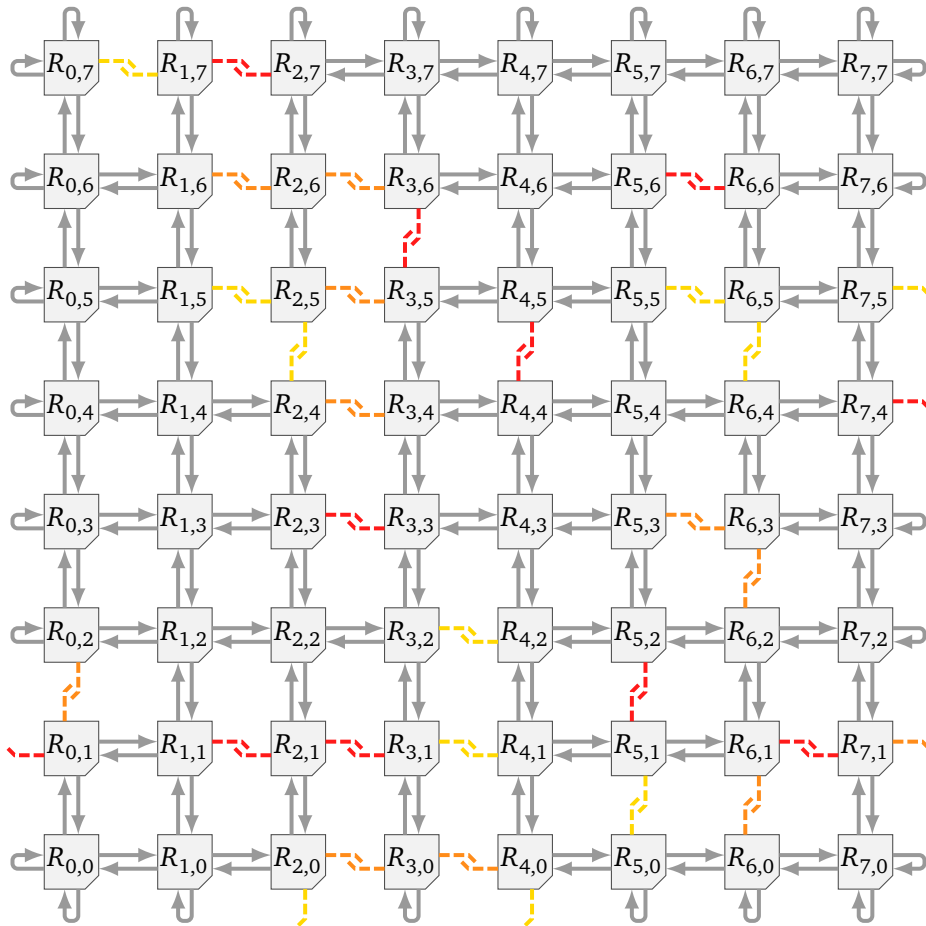


Figure A.2: Second simulated fault situation. The link failures depicted in yellow, orange, and red correspond to $\lambda = 10\%$, $\lambda = 20\%$, and $\lambda = 30\%$, respectively.

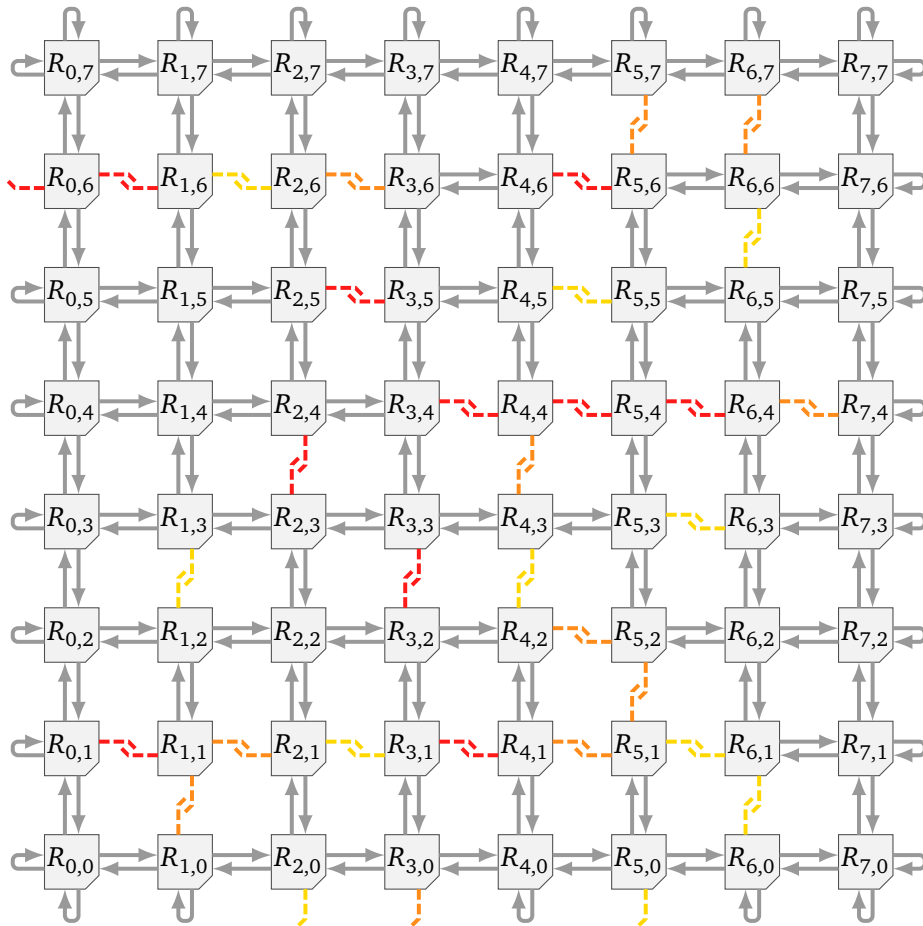


Figure A.3: Third simulated fault situation. The link failures depicted in yellow, orange, and red correspond to $\lambda = 10\%$, $\lambda = 20\%$, and $\lambda = 30\%$, respectively.

A.2 Lambert W function

If the two ceiling functions of Equation (5.6) are omitted, i.e. with the inequations depicted in Equation (A.1), Equation (5.6) can be solved with the Lambert W function:

$$\frac{|msg|}{|LI| - |hd| - \text{ld}(\#f) - 1} + 1 > \#f \geq \frac{|msg|}{|LI| - |hd| - \text{ld}(\#f)} \quad (\text{A.1})$$

$$\begin{aligned} & \#f = \frac{|msg|}{|LI| - |hd| - \text{ld}(\#f)} \\ \Leftrightarrow & \frac{|msg|}{\#f} = |LI| - |hd| - \text{ld}(\#f) \\ \Leftrightarrow & \frac{-|msg|}{\#f} - \text{ld}(\#f) = |hd| - |LI| \quad | + \text{ld}(\#f) | \cdot (-1) \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} - \text{ld}(\#f)} = e^{|hd| - |LI|} \\ \Leftrightarrow & (e^{\frac{-|msg|}{\#f} - \text{ld}(\#f)})^{\ln(2)} = (e^{|hd| - |LI|})^{\ln(2)} \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2) - \frac{\ln(\#f)}{\ln(2)} \cdot \ln(2)} = e^{(|hd| - |LI|) \cdot \ln(2)} \quad | \text{ with } (a^x)^y = a^{x \cdot y} \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2) - \ln(\#f)} = 2^{|hd| - |LI|} \quad | \text{ with } a^x = b^{x \cdot \log_b(a)} \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2)} \cdot e^{-\ln(\#f)} = 2^{|hd| - |LI|} \quad | \text{ with } a^{x+y} = a^x \cdot a^y \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2)} \cdot e^{\ln(\frac{1}{\#f})} = 2^{|hd| - |LI|} \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2)} \cdot \frac{1}{\#f} = 2^{|hd| - |LI|} \quad | \text{ with } e^{\ln(x)} = x \\ \Leftrightarrow & e^{\frac{-|msg|}{\#f} \cdot \ln(2)} \cdot \frac{-|msg| \cdot \ln(2)}{\#f} = -|msg| \cdot \ln(2) \cdot 2^{|hd| - |LI|} \quad | \cdot (-|msg| \cdot \ln(2)) \\ \Leftrightarrow & \frac{-|msg| \cdot \ln(2)}{\#f} = W(-|msg| \cdot \ln(2) \cdot 2^{|hd| - |LI|}) \quad | \text{ Lambert W function} \\ \Leftrightarrow & \#f = \frac{-|msg| \cdot \ln(2)}{W(-|msg| \cdot \ln(2) \cdot 2^{|hd| - |LI|})} \quad (\text{A.2}) \end{aligned}$$

Publications of the Author

- [RK16a] Armin Runge and Reiner Kolla. **An Alternating Transmission Scheme for Deflection Routing Based Network-on-Chips**. In: *Architecture of Computing Systems – ARCS 2016*. Springer International Publishing. Springer Nature, 2016, pp. 48–59. DOI: 10.1007/978-3-319-30695-7_4 (cit. on pp. 5, 133).
- [RK16b] Armin Runge and Reiner Kolla. **Consideration of the Flit Size for Deflection Routing based Network-on-Chips**. In: *1st International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems (AISTECS)*. Association for Computing Machinery (ACM), 2016, 5:1–5:6. DOI: 10.1145/2857058.2857060 (cit. on pp. 5, 119).
- [RK16c] Armin Runge and Reiner Kolla. **TwoPhases: A Transmission Scheme to Reduce the Link Width at Deflection Routing based Network-on-Chips**. In: *Journal of Systems Architecture* (Dec. 2016). DOI: 10.1016/j.sysarc.2016.12.001 (cit. on pp. 5, 133).
- [RK16d] Armin Runge and Reiner Kolla. **Using Benes Networks at Fault Tolerant and Deflection Routing based NoCs**. In: *Proceedings of the 10th International Symposium on Networks-on-Chip*. Institute of Electrical and Electronics Engineers (IEEE), Sept. 2016. DOI: 10.1109/NOCS.2016.7579325 (cit. on pp. 6, 68).
- [Run12a] Armin Runge. **Determination of the Optimum Degree of Redundancy for Fault-prone Many-Core Systems**. In: *Zuverlässigkeit und Entwurf - 6. GMM/GI/ITG-Fachtagung*. VDE VERLAG GmbH, 2012 (cit. on pp. 5, 69).
- [Run12b] Armin Runge. **Reliability Enhancement of Fault-prone Many-core Systems Combining Spatial and Temporal Redundancy**. In: *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems*. Institute of Electrical & Electronics Engineers (IEEE), June 2012. DOI: 10.1109/hpcc.2012.233 (cit. on pp. 5, 69).
- [Run15a] Armin Runge. **FaFNoC: A Fault-tolerant and Bufferless Network-on-chip**. In: *Procedia Computer Science* 56 (2015), pp. 397–402. DOI: 10.1016/j.procs.2015.07.226 (cit. on pp. 5, 68).

- [Run15b] Armin Runge. **Fault-tolerant Network-on-Chip based on Fault-aware Flits and Deflection Routing**. In: *Proceedings of the 9th International Symposium on Networks-on-Chip*. ACM. Association for Computing Machinery (ACM), 2015, p. 9. DOI: 10.1145/2786572.2786585 (cit. on pp. 5, 68).

Bibliography

- [Acc16] Accellera. **Open Core Protocol Specification**. Ed. by Accellera. 2016. URL: <http://accellera.org/downloads/standards/ocp> (cit. on p. 8).
- [ARM16] ARM. **AMBA Specifications**. Ed. by ARM. 2016. URL: <https://www.arm.com/products/system-ip/amba-specifications> (cit. on p. 8).
- [Avi+04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. **Basic Concepts and Taxonomy of Dependable and Secure Computing**. In: *IEEE Trans. Dependable Secur. Comput.* 1.1 (Jan. 2004), pp. 11–33. DOI: 10.1109/TDSC.2004.2 (cit. on p. 68).
- [Bar64] Paul Baran. **On distributed communications networks**. In: *IEEE transactions on Communications Systems* 12.1 (Mar. 1964), pp. 1–9. DOI: 10.1109/tcom.1964.1088883 (cit. on pp. 3, 19, 38).
- [Bat68] K. E. Batcher. **Sorting Networks and Their Applications**. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS '68 (Spring). Atlantic City, New Jersey: ACM, 1968, pp. 307–314. DOI: 10.1145/1468075.1468121 (cit. on pp. 43, 44).
- [BD06] James Balfour and William J. Dally. **Design Tradeoffs for Tiled CMP On-chip Networks**. In: *Proceedings of the 20th Annual International Conference on Supercomputing*. ICS '06. Cairns, Queensland, Australia: ACM, 2006, pp. 187–198. DOI: 10.1145/1183401.1183430 (cit. on p. 13).
- [BDM07] Paul Bogdan, Tudor Dumitraş, and Radu Marculescu. **Stochastic Communication: A New Paradigm for Fault-Tolerant Networks-on-Chip**. In: *VLSI design 2007* (2007), pp. 1–17. DOI: 10.1155/2007/95348 (cit. on p. 71).
- [Bie11] Christian Bienia. **Benchmarking Modern Multiprocessors**. PhD thesis. Princeton University, Jan. 2011 (cit. on p. 31).
- [BM06] Tobias Bjerregaard and Shankar Mahadevan. **A Survey of Research and Practices of Network-on-chip**. In: *ACM Computing Surveys* 38.1 (June 2006), 1–es. DOI: 10.1145/1132952.1132953 (cit. on p. 8).
- [Bor05] Shekhar Borkar. **Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation**. In: *IEEE Micro* 25 (6 Nov. 2005), pp. 10–16. DOI: 10.1109/MM.2005.110 (cit. on p. 2).

- [Bor07] Shekhar Borkar. **Thousand Core Chips - A Technology Perspective**. In: *DAC*. Association for Computing Machinery (ACM), 2007. DOI: 10.1145/1278480.1278667 (cit. on pp. 2, 3, 38).
- [Bor10] Shekhar Borkar. **Future of Interconnect Fabric: A Contrarian View**. In: *Proceedings of the 12th ACM/IEEE International Workshop on System Level Interconnect Prediction*. SLIP '10. ACM, 2010, pp. 1–2. DOI: 10.1145/1811100.1811101 (cit. on pp. 3, 38).
- [Cat+15] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. **Noxim: An open, extensible and cycle-accurate network on chip simulator**. In: *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. July 2015, pp. 162–163. DOI: 10.1109/ASAP.2015.7245728 (cit. on pp. 29, 114, 117).
- [Chi00] Ge-Ming Chiu. **The Odd-Even Turn Model for Adaptive Routing**. In: *IEEE Transactions on Parallel and Distributed Systems* 11.7 (July 2000), pp. 729–738. DOI: 10.1109/71.877831 (cit. on p. 18).
- [CLC12] Changlin Chen, Ye Lu, and Sorin D Cotofana. **A Novel Flit Serialization Strategy to Utilize Partially Faulty Links in Networks-on-Chip**. In: *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), May 2012, pp. 124–131. DOI: 10.1109/nocs.2012.22 (cit. on pp. 118, 140).
- [CM11] Chris Craik and Onur Mutlu. **Investigating the Viability of Bufferless NoCs in Modern Chip Multi-processor Systems**. Tech. rep. SAFARI Technical Report, TR-SAFARI-2011-004, Carnegie Mellon University, 2011 (cit. on p. 42).
- [CML12] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. **Reliability, Availability and Serviceability of Networks-on-chip**. Springer Science & Business Media, 2012. DOI: 10.1007/978-1-4614-0791-1 (cit. on pp. 8, 67, 113).
- [CMM15] Yu Cai, Ken Mai, and Onur Mutlu. **Comparative evaluation of FPGA and ASIC implementations of bufferless and buffered routing algorithms for on-chip networks**. In: *Quality Electronic Design (ISQED), 2015 16th International Symposium on*. IEEE. 2015, pp. 475–484 (cit. on pp. 3, 42).
- [Cop+08] Marcello Coppola, Miltos D. Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia, and Lorenzo Pieralisi. **Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC**. 1st. Boca Raton, FL, USA: CRC Press, Inc., Sept. 2008. DOI: 10.1201/9781420044720 (cit. on pp. 13, 14).

- [Cou15] Rachel Courtland. **Gordon Moore: The Man Whose Name Means Progress**. In: *IEEE Spectrum SPECIAL REPORT: 50 Years of Moore's Law* 30 (2015) (cit. on p. 1).
- [Den+74] Robert H Dennard, Fritz H Gaensslen, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. **Design of ion-implanted MOSFET's with very small physical dimensions**. In: *IEEE Journal of Solid-State Circuits* 9.5 (Oct. 1974), pp. 256–268. DOI: 10.1109/JSSC.1974.1050511 (cit. on p. 1).
- [DKM03] Tudor Dumitraş, Sam Kerner, and Radu Mărculescu. **Towards on-chip fault-tolerant communication**. In: *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. ACM. Association for Computing Machinery (ACM), 2003, pp. 225–232. DOI: 10.1145/1119772.1119817 (cit. on p. 71).
- [DT04] William James Dally and Brian Patrick Towles. **Principles and practices of interconnection networks**. Elsevier, 2004 (cit. on pp. 10, 18, 41, 114).
- [DYL02] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. **Interconnection Networks: An Engineering Approach**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002 (cit. on p. 11).
- [Esm+11] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. **Dark Silicon and the End of Multicore Scaling**. In: *ACM SIGARCH Computer Architecture News* 39.3 (July 2011), pp. 365–376. DOI: 10.1145/2024723.2000108 (cit. on p. 3).
- [Fal+11] Chris Fallin, Gregory Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu. **MinBD: A Minimally-Buffered Deflection Router Approaching Conventional Buffered-Router Performance**. Tech. rep. 2011-008. Carnegie Mellon University, Sept. 2011 (cit. on pp. 38, 40, 41, 51, 52, 119).
- [Fal+12] Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu. **MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect**. In: *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), May 2012, pp. 1–10. DOI: 10.1109/NOCS.2012.8 (cit. on pp. 38, 40, 41, 51, 52, 119).
- [FCM10] Chris Fallin, Chris Craik, and Onur Mutlu. **CHIPPER: A Low-complexity Bufferless Deflection Router**. Tech. rep. 2010-001. Carnegie Mellon University, Dec. 2010 (cit. on pp. 3, 25, 38, 40–42, 47–51, 53, 55, 119).

- [FCM11] Chris Fallin, Chris Craik, and Onur Mutlu. **CHIPPER: A low-complexity bufferless deflection router**. In: *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. Institute of Electrical & Electronics Engineers (IEEE), Feb. 2011, pp. 144–155. DOI: 10.1109/HPCA.2011.5749724 (cit. on pp. 25, 38, 40, 41, 47–49, 53, 55, 119).
- [Fen+10a] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang. **A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip**. In: *Proceedings of the Third International Workshop on Network on Chip Architectures*. NoCArc '10. Atlanta, Georgia: ACM, 2010, pp. 11–16 (cit. on pp. 45, 74, 75, 100).
- [Fen+10b] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang. **FoN: Fault-on-Neighbor aware routing algorithm for Networks-on-Chip**. In: *SOC Conference (SOCC), 2010 IEEE International*. IEEE. 2010, pp. 441–446 (cit. on pp. 45, 74, 100).
- [Fen+11a] Chaochao Feng, Jinwen Li, Zhonghai Lu, Axel Jantsch, and Minxuan Zhang. **Evaluation of deflection routing on various NoC topologies**. In: *2011 9th IEEE International Conference on ASIC*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), Oct. 2011, pp. 163–166. DOI: 10.1109/asicon.2011.6157147 (cit. on p. 53).
- [Fen+11b] Chaochao Feng, Minxuan Zhang, Jinwen Li, Jiang Jiang, Zhonghai Lu, and Axel Jantsch. **A Low-overhead Fault-aware Deflection Routing Algorithm for 3D Network-on-Chip**. In: *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), July 2011, pp. 19–24. DOI: 10.1109/isvlsi.2011.42 (cit. on p. 76).
- [Fen+13] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Minxuan Zhang, and Zuocheng Xing. **Addressing Transient and Permanent Faults in NoC With Efficient Fault-Tolerant Deflection Router**. In: *IEEE Trans. VLSI Syst.* 21.6 (June 2013), pp. 1053–1066. DOI: 10.1109/tvlsi.2012.2204909 (cit. on pp. 76, 77).
- [Fic+09a] David Fick, Andrew DeOrio, Gregory Chen, Valeria Bertacco, Dennis Sylvester, and David Blaauw. **A highly resilient routing algorithm for fault-tolerant NoCs**. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. DATE '09. Nice, France: European Design and Automation Association, Apr. 2009, pp. 21–26. DOI: 10.1109/date.2009.5090627 (cit. on p. 72).

- [Fic+09b] David Fick, Andrew DeOrio, Jin Hu, Valeria Bertacco, David Blaauw, and Dennis Sylvester. **Vicis: a reliable network for unreliable silicon**. In: *Proceedings of the 46th Annual Design Automation Conference*. DAC '09. San Francisco, California: ACM, 2009, pp. 812–817. DOI: 10.1145/1629911.1630119 (cit. on p. 72).
- [GHD15] GHDL. **GHDL**. Ed. by Tristan Gingold. 2015. URL: <http://ghdl.free.fr/> (cit. on p. 30).
- [Ghi+13] Yan Ghidini, Matheus Moreira, Lucas Brahm, Thais Webber, Ney Calazans, and Cesar Marcon. **Lasio 3D NoC vertical links serialization: Evaluation of latency and buffer occupancy**. In: *Integrated Circuits and Systems Design (SBCCI), 2013 26th Symposium on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), Sept. 2013, pp. 1–6. DOI: 10.1109/sbcc.2013.6644891 (cit. on pp. 118, 140).
- [GN92] Christopher J. Glass and Lionel M. Ni. **The Turn Model for Adaptive Routing**. In: *ACM SIGARCH Computer Architecture News* 20.2 (June 1992), pp. 278–287. DOI: 10.1145/146628.140384 (cit. on pp. 15, 16, 18).
- [Góm+11] Crispín Gómez, María E. Gómez, Pedro López, and José Duato. **How to Reduce Packet Dropping in a Bufferless NoC**. In: *Concurr. Comput. : Pract. Exper.* 23.1 (Jan. 2011), pp. 86–99. DOI: 10.1002/cpe.1606 (cit. on p. 38).
- [Gra+06] Paul Gratz, Changkyu Kim, Robert McDonald, Stephen W Keckler, and Doug Burger. **Implementation and evaluation of on-chip network architectures**. In: *2006 International Conference on Computer Design*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), Oct. 2006, pp. 477–484. DOI: 10.1109/iccd.2006.4380859 (cit. on pp. 3, 38).
- [HGK10] Joel Hestness, Boris Grot, and Stephen W. Keckler. **Netrace: Dependency-driven Trace-based Network-on-chip Simulation**. In: *Proceedings of the Third International Workshop on Network on Chip Architectures*. NoCArc '10. Atlanta, Georgia, USA: ACM, 2010, pp. 31–36. DOI: 10.1145/1921249.1921258 (cit. on pp. 31, 131).
- [HJL09] Mitchell Hayenga, Natalie Enright Jerger, and Mikko Lipasti. **SCARAB: a single cycle adaptive routing and bufferless network**. In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: Association for Computing Machinery (ACM), 2009, pp. 244–254. DOI: 10.1145/1669112.1669144 (cit. on p. 38).

- [HK10] Joel Hestness and Stephen W. Keckler. **Netrace: Dependency-Tracking Traces for Efficient Network-on-Chip Experimentation**. **Technical Report TR-10-11**. Tech. rep. The University of Texas at Austin, May 2010 (cit. on pp. 31, 131).
- [HLH02] Ching-Fang Hsu, Te-Lung Liu, and Nen-Fu Huang. **Performance analysis of deflection routing in optical burst-switched networks**. In: *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. Vol. 1. IEEE. Institute of Electrical & Electronics Engineers (IEEE), 2002, pp. 66–73. DOI: 10.1109/infcom.2002.1019247 (cit. on p. 39).
- [HNJ12] Robert Hesse, Jeff Nicholls, and Natalie Enright Jerger. **Fine-Grained Bandwidth Adaptivity in Networks-on-Chip Using Bidirectional Channels**. In: *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), May 2012, pp. 132–141. DOI: 10.1109/nocs.2012.23 (cit. on p. 140).
- [Hos+07] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. **A 5-GHz Mesh Interconnect for a Teraflops Processor**. In: *IEEE Micro* 27.5 (Sept. 2007), pp. 51–61. DOI: 10.1109/MM.2007.4378783 (cit. on pp. 3, 38).
- [How+10] Jason Howard, Saurabh Dighe, Yatin Hoskote, Sriram Vangal, David Finan, Gregory Ruhl, David Jenkins, Howard Wilson, Nitin Borkar, Gerhard Schrom, et al. **A 48-core IA-32 message-passing processor with DVFS in 45nm CMOS**. In: *2010 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE. Institute of Electrical and Electronics Engineers (IEEE), Feb. 2010, pp. 108–109. DOI: 10.1109/isscc.2010.5434077 (cit. on p. 35).
- [HSG09] Andreas Hansson, Mahesh Subburaman, and Kees Goossens. **aelite: A flit-synchronous network on chip with composable and predictable services**. In: *Proceedings of the conference on design, automation and test in Europe*. European Design and Automation Association. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2009, pp. 250–255. DOI: 10.1109/date.2009.5090666 (cit. on p. 38).
- [ITR11] ITRS. **The International Technology Roadmap for Semiconductors, Design**. 2011 (cit. on p. 69).
- [ITR15] ITRS. **International Technology Roadmap for Semiconductors 2.0 2015 Edition Executive Report**. 2015 (cit. on p. 1).
- [Jai] Lavina Jain. **NIRGAM** (cit. on pp. 29, 114, 117).

- [Jia+13] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim. **A detailed and flexible cycle-accurate Network-on-Chip simulator**. In: *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*. official ref for booksim. Apr. 2013, pp. 86–96. DOI: 10.1109/ISPASS.2013.6557149 (cit. on pp. 29, 117).
- [Jon+14] Gnaneswara Rao Jonna, John Jose, Rachana Radhakrishnan, and Madhu Mutyam. **Minimally buffered single-cycle deflection router**. In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. EDAA, 2014, p. 310. DOI: 10.7873/date.2014.323 (cit. on pp. 41, 47).
- [Jos+13] John Jose, Bhawna Nayak, Kranthi Kumar, and Madhu Mutyam. **DeBAR: Deflection based adaptive router with minimal buffering**. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. EDAA, Mar. 2013, pp. 1583–1588. DOI: 10.7873/DATE.2013.322 (cit. on pp. 41, 47).
- [Kah+09] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. **ORION 2.0: a fast and accurate NoC power and area model for early-stage design space exploration**. In: *Proceedings of the conference on Design, Automation and Test in Europe*. European Design and Automation Association. Institute of Electrical & Electronics Engineers (IEEE), Apr. 2009, pp. 423–428. DOI: 10.1109/DATE.2009.5090700 (cit. on p. 117).
- [Kah+12] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. **ORION 2.0: A Power-Area Simulator for Interconnection Networks**. In: *IEEE Trans. Very Large Scale Integr. Syst.* 20.1 (Jan. 2012), pp. 191–196. DOI: 10.1109/TVLSI.2010.2091686 (cit. on pp. 114, 117).
- [Kas+16] E. Kasapaki, M. Schoeberl, R. B. Sørensen, C. Müller, K. Goossens, and J. Sparsø. **Argo: A Real-Time Network-on-Chip Architecture With an Efficient GALS Implementation**. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.2 (Feb. 2016), pp. 479–492. DOI: 10.1109/TVLSI.2015.2405614 (cit. on p. 38).
- [KG15] Nachiket Kapre and Jan Gray. **Hoplite: Building austere overlay NoCs for FPGAs**. In: *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. Institute of Electrical and Electronics Engineers (IEEE), Sept. 2015, pp. 1–8. DOI: 10.1109/fpl.2015.7293956 (cit. on p. 29).
- [Kim09] J. Kim. **Low-cost router microarchitecture for on-chip networks**. In: *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*. Association for Computing Machinery (ACM), Dec. 2009, pp. 255–266. DOI: 10.1145/1669112.1669145 (cit. on p. 49).

- [KK07] Y. B. Kim and Y. B. Kim. **Fault Tolerant Source Routing for Network-on-chip**. In: *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*. Institute of Electrical & Electronics Engineers (IEEE), Sept. 2007, pp. 12–20. DOI: 10.1109/DFT.2007.14 (cit. on p. 71).
- [KLN12a] Andrew B Kahng, Bill Lin, and Siddhartha Nath. **Comprehensive modeling methodologies for NoC router estimation**. Tech. rep. CS2012-0989. UCSD, Sept. 2012 (cit. on p. 117).
- [KLN12b] Andrew B Kahng, Bill Lin, and Siddhartha Nath. **Explicit Modeling of Control and Data for Improved NoC Router Estimation**. In: *Proceedings of the 49th Annual Design Automation Conference*. ACM. Association for Computing Machinery (ACM), 2012, pp. 392–397. DOI: 10.1145/2228360.2228430 (cit. on pp. 114, 117).
- [KLN15] Andrew B. Kahng, Bill Lin, and Siddhartha Nath. **ORION3.0: A Comprehensive NoC Router Estimation Tool**. In: *Embedded Systems Letters, IEEE* 7.2 (June 2015), pp. 41–45. DOI: 10.1109/LES.2015.2402197 (cit. on p. 117).
- [KPP06] Michael Kistler, Michael Perrone, and Fabrizio Petrini. **Cell Multiprocessor Communication Network: Built for Speed**. In: *IEEE Micro* 26.3 (May 2006), pp. 10–23. DOI: 10.1109/mm.2006.49 (cit. on p. 35).
- [KR09] A. Kohler and M. Radetzki. **Fault-tolerant architecture and deflection routing for degradable NoC switches**. In: *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*. Institute of Electrical & Electronics Engineers (IEEE), May 2009, pp. 22–31. DOI: 10.1109/NOCS.2009.5071441 (cit. on p. 73).
- [Kro81] David Kroft. **Lockup-free Instruction Fetch/Prefetch Cache Organization**. In: *Proceedings of the 8th Annual Symposium on Computer Architecture*. ISCA '81. Minneapolis, Minnesota, USA: IEEE Computer Society Press, 1981, pp. 81–87. DOI: 10.1145/285930.285939 (cit. on p. 51).
- [KSR10] Adán Kohler, Gert Schley, and Martin Radetzki. **Fault Tolerant Network on Chip Switching With Graceful Performance Degradation**. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.6 (June 2010), pp. 883–896. DOI: 10.1109/tcad.2010.2048399 (cit. on pp. 73, 77).
- [Lee+13a] J. Lee, C. Nicopoulos, S. J. Park, M. Swaminathan, and J. Kim. **Do we need wide flits in Networks-on-Chip?** In: *2013 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Institute of Electrical & Electronics Engineers

- (IEEE), Aug. 2013, pp. 2–7. DOI: 10.1109/ISVLSI.2013.6654614 (cit. on pp. 117, 119, 122).
- [Lee+13b] Jinho Lee, Dongwoo Lee, Sunwook Kim, and Kiyoun Choi. **Deflection routing in 3D network-on-chip with limited vertical bandwidth**. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 18.4 (Oct. 2013), p. 50. DOI: 10.1145/2505011 (cit. on pp. 47, 118).
- [Li+09] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. **MCPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures**. In: *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 42. New York, New York: ACM, 2009, pp. 469–480. DOI: 10.1145/1669112.1669172 (cit. on p. 117).
- [LL04] Guy Lemieux and David Lewis. **Design of Interconnection Networks for Programmable Logic**. Norwell, MA, USA: Kluwer Academic Publishers, 2004, pp. 1–8. DOI: 10.1007/978-1-4757-4941-0_1 (cit. on p. 89).
- [LLT12] Jing Lin, Xiaola Lin, and Liang Tang. **Making-a-stop: A new bufferless routing algorithm for on-chip network**. In: *Journal of Parallel and Distributed Computing* 72.4 (Apr. 2012), pp. 515–524. DOI: 10.1016/j.jpdc.2012.01.001 (cit. on p. 118).
- [Lu+05] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. **NNSE: Nostrum Network-on-Chip Simulation Environment**. In: *Proceedings of Swedish System-on-Chip Conference, Stockholm, Sweden, April 2005*. 2005 (cit. on pp. 45, 114, 117).
- [LZJ06a] Ming Li, Qing-An Zeng, and Wen-Ben Jone. **DyXY: a proximity congestion-aware deadlock-free dynamic routing method for network on chip**. In: *Proceedings of the 43rd annual Design Automation Conference*. DAC '06. San Francisco, CA, USA: ACM, 2006, pp. 849–852. DOI: 10.1145/1146909.1147125 (cit. on p. 72).
- [LZJ06b] Zhonghai Lu, Mingchen Zhong, and Axel Jantsch. **Evaluation of on-chip networks using deflection routing**. In: *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. GLSVLSI '06. Philadelphia, PA, USA: ACM, 2006, pp. 296–301. DOI: 10.1145/1127908.1127977 (cit. on pp. 53, 55).
- [Mar+09] Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. **Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives**. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28.1 (Jan. 2009), pp. 3–21. DOI: 10.1109/tcad.2008.2010691 (cit. on p. 118).

- [Mic+10] George Michelogiannakis, Daniel Sanchez, William J Dally, and Christos Kozyrakis. **Evaluating bufferless flow control for on-chip networks**. In: *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*. IEEE Computer Society, 2010, pp. 9–16 (cit. on p. 42).
- [Mil+04a] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. **The Nostrum backbone—a communication protocol stack for Networks on Chip**. In: *VLSI Design, 2004. Proceedings. 17th International Conference on*. 2004, pp. 693–696. DOI: 10.1109/ICVD.2004.1261005 (cit. on p. 45).
- [Mil+04b] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch. **Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip**. In: *Proceedings of the conference on Design, automation and test in Europe - Volume 2. DATE '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 20890–. DOI: 10.1109/date.2004.1269001 (cit. on p. 45).
- [Mil02] Mikael Millberg. **The Nostrum Protocol Stack and Suggested Services Provided by the Nostrum Backbone**. Tech. rep. TRITA-IMIT-LECSR02:01. Draft v 0.1.48. Stockholm, Sweden: Institute of Microelectronics and Information Technology, Royal Institute of Technology (KTH), Nov. 2002 (cit. on pp. 38, 40, 45).
- [Mil11] Mikael Millberg. **Architectural Techniques for Improving Performance in Networks on Chip**. PhD thesis. KTH Royal Institute of Technology, 2011 (cit. on p. 45).
- [MJ07] Mikael Millberg and Axel Jantsch. **Increasing NoC performance and utilisation using a Dual Packet Exit strategy**. In: *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), Aug. 2007, pp. 511–518. DOI: 10.1109/dsd.2007.4341516 (cit. on p. 45).
- [MM09] Thomas Moscibroda and Onur Mutlu. **A case for bufferless routing in on-chip networks**. In: *Proceedings of the 36th annual international symposium on Computer architecture*. ISCA '09. Austin, TX, USA: ACM, 2009, pp. 196–207. DOI: 10.1145/1555754.1555781 (cit. on pp. 38, 40–42, 45, 46, 118, 140).
- [Moo98] G. E. Moore. **Cramming More Components Onto Integrated Circuits**. In: *Proceedings of the IEEE* 86.1 (Jan. 1998), pp. 82–85. DOI: 10.1109/JPROC.1998.658762 (cit. on p. 1).

- [MWM04] Robert Mullins, Andrew West, and Simon Moore. **Low-Latency Virtual-Channel Routers for On-Chip Networks**. In: *ACM SIGARCH Computer Architecture News* 32.2 (Mar. 2004), p. 188. DOI: 10.1145/1028176.1006717 (cit. on p. 25).
- [Nyc+12] George P Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan. **On-chip networks from a networking perspective: congestion and scalability in many-core interconnects**. In: *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. Vol. 42. SIGCOMM '12 4. Helsinki, Finland: ACM, Sept. 2012, pp. 407–418. DOI: 10.1145/2377677.2377757 (cit. on p. 42).
- [OHM05] Umit Y Ogras, Jingcao Hu, and Radu Marculescu. **Key research problems in NoC design: a holistic perspective**. In: *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. CODES+ISSS '05. ACM, 2005, pp. 69–74. DOI: 10.1145/1084834.1084856 (cit. on p. 118).
- [OWB12a] Gadi Oxman, Shlomo Weiss, and Yitzhak (Tsahi) Birk. **Buffered Deflection Routing for Networks-on-chip**. In: *Proceedings of the 2012 Interconnection Network Architecture: On-Chip, Multi-Chip Workshop*. INA-OCMC '12. ACM, 2012, pp. 9–12. DOI: 10.1145/2107763.2107766 (cit. on p. 41).
- [OWB12b] Gadi Oxman, Shlomo Weiss, and Yitzhak (Tsahi) Birk. **Streamlined Network-on-chip for Multicore Embedded Architectures**. In: *Proceedings of the 25th International Conference on Architecture of Computing Systems*. ARCS'12. Springer-Verlag, 2012, pp. 238–249. DOI: 10.1007/978-3-642-28293-5_20 (cit. on p. 55).
- [Pat01] Achille Pattavina. **Interconnection Networks**. In: *Switching Theory, Architectures and Performance in Broadband ATM Networks*. John Wiley & Sons, Ltd, 2001, pp. 53–90. DOI: 10.1002/0470841915.ch2 (cit. on p. 121).
- [Pir+04] Matthew Pirretti, Greg M Link, Richard R Brooks, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. **Fault tolerant algorithms for network-on-chip interconnect**. In: *IEEE Computer Society Annual Symposium on VLSI*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), 2004, pp. 46–51. DOI: 10.1109/isvlsi.2004.1339507 (cit. on p. 71).
- [PJ06] Sandro Penolazzi and Axel Jantsch. **A High Level Power Model for the Nostrum NoC**. In: *Proceedings of the 9th EUROMICRO Conference on Digital System Design*. IEEE Computer Society, 2006, pp. 673–676. DOI: 10.1109/DSD.2006.9 (cit. on p. 45).

- [PZ11] Sudeep Pasricha and Yong Zou. **NS-FTR: a fault tolerant routing scheme for networks on chip with permanent and runtime intermittent faults**. In: *Proceedings of the 16th Asia and South Pacific Design Automation Conference*. ASPDAC '11. Yokohama, Japan: IEEE Press, Jan. 2011, pp. 443–448. DOI: 10.1109/aspdac.2011.5722231 (cit. on p. 72).
- [QS13] Muhammad Yasir Qadri and Stephen J Sangwine. **Multicore Technology: Architecture, Reconfiguration, and Modeling**. Ed. by Stephen Sangwine. CRC Press, July 2013. DOI: 10.1201/b15268 (cit. on p. 11).
- [Rad+13] Martin Radetzki, Chaochao Feng, Xueqian Zhao, and Axel Jantsch. **Methods for Fault Tolerance in Networks on Chip**. In: *ACM Computing Surveys* 1 (2013), p. 35 (cit. on pp. 69, 70).
- [Rad11a] M. Radetzki. **Fault-Tolerant Differential Q Routing in Arbitrary NoC Topologies**. In: *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*. Institute of Electrical & Electronics Engineers (IEEE), Oct. 2011, pp. 33–40. DOI: 10.1109/EUC.2011.36 (cit. on p. 76).
- [Rad11b] Martin Radetzki. **Fehlertolerantes differentielles Q-Routing für On-Chip-Verbindungsnetzwerke mit beliebiger Topologie**. In: *Zuverlässigkeit und Entwurf - 5. GI/GMM/ITG-Fachtagung*. 2011 (cit. on p. 76).
- [Rah13] Rezaur Rahman. **Intel® Xeon Phi™ Coprocessor Architecture and Tools: The Guide for Application Developers**. Springer Nature, 2013. DOI: 10.1007/978-1-4302-5927-5 (cit. on p. 35).
- [RK11] Martin Radetzki and Adán Kohler. **Cost-Based Deflection Routing for Intelligent NoC Switches**. In: *Solutions on Embedded Systems*. Ed. by Massimo Conti, Simone Orcioni, Natividad Martínez Madrid, and E.D. Ralf Seepold. Dordrecht: Springer Netherlands, 2011, pp. 77–90. DOI: 10.1007/978-94-007-0638-5_6 (cit. on p. 73).
- [Saw+11] Shankar Sawant, Utpal Desai, Gururaj Shamanna, Lokesh Sharma, Mandar Ranade, Anil Agarwal, Sampath Dakshinamurthy, and Rajagopal Narayanan. **A 32nm Westmere-EX Xeon® enterprise processor**. In: *2011 IEEE International Solid-State Circuits Conference*. Institute of Electrical and Electronics Engineers (IEEE), Feb. 2011, pp. 74–75. DOI: 10.1109/ISSCC.2011.5746225 (cit. on p. 1).
- [Sei85] Charles L Seitz. **The cosmic cube**. In: *Communications of the ACM* 28.1 (Jan. 1985), pp. 22–33. DOI: 10.1145/2465.2467 (cit. on p. 13).

- [Shp+15] Alexander Shpiner, Erez Kantor, Pu Li, Israel Cidon, and Isaac Keslassy. **On the Capacity of Bufferless Networks-on-Chip**. In: *Parallel and Distributed Systems, IEEE Transactions on* 26.2 (Oct. 2015), pp. 492–506. DOI: 10.1109/allerton.2012.6483296 (cit. on p. 38).
- [SKH08] Erno Salminen, Ari Kulmala, and Timo D Hamalainen. **Survey of network-on-chip proposals**. In: *white paper, OCP-IP* (2008), pp. 1–13 (cit. on p. 114).
- [SMG14] Radu Andrei Stefan, Anca Molnos, and Kees Goossens. **dAElite: A TDM NoC Supporting QoS, Multicast, and Fast Connection Set-Up**. In: *IEEE Transactions on Computers* 63.3 (Mar. 2014), pp. 583–594. DOI: 10.1109/tc.2012.117 (cit. on p. 38).
- [Sod+16] Avinash Sodani, Roger Gramunt, Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, and Yen-Chen Liu. **Knights Landing: Second-Generation Intel Xeon Phi Product**. In: *IEEE Micro* 36.2 (Mar. 2016), pp. 34–46. DOI: 10.1109/mm.2016.25 (cit. on p. 35).
- [Sod16] Avinash Sodani. **Knights Landing Intel Xeon Phi CPU: Path to Parallelism with General Purpose Programming**. keynote speech at 29th International Conference on Architecture of Computing Systems (ARCS) 2016. Nuremberg, Apr. 4, 2016 (cit. on p. 4).
- [SRK10] Gert Schley, Martin Radetzki, and Adán Kohler. **Degradability Enabled Routing for Network-on-Chip Switches**. In: *it-Information Technology* 52.4 (Jan. 2010), pp. 201–208. DOI: 10.1524/itit.2010.0592 (cit. on p. 73).
- [TB11] A. T. Tran and B. M. Baas. **Design of Bufferless On-Chip Routers Providing In-Order Packet Delivery**. In: *SRC Technology and Talent for the 21st Century (TECHCON)*. Sept. 2011, S14.3 (cit. on p. 119).
- [Val+10] Mojtaba Valinataj, Siamak Mohammadi, Juha Plosila, and Pasi Liljeberg. **A fault-tolerant and congestion-aware routing algorithm for Networks-on-Chip**. In: *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2010 IEEE 13th International Symposium on*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), 2010, pp. 139–144. DOI: 10.1109/ddecs.2010.5491798 (cit. on p. 72).
- [Val82] Leslie G. Valiant. **A Scheme for Fast Parallel Communication**. In: *SIAM Journal on Computing* 11.2 (May 1982), pp. 350–361. DOI: 10.1137/0211027 (cit. on p. 15).

- [Van+08] Sriram R Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, et al. **An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS**. In: *IEEE Journal of Solid-State Circuits* 43.1 (Jan. 2008), pp. 29–41. DOI: 10.1109/jssc.2007.910957 (cit. on pp. 3, 35).
- [Wan+02] Hang-Sheng Wang, Xinpeng Zhu, Li-Shiuan Peh, and S. Malik. **Orion: a power-performance simulator for interconnection networks**. In: *Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*. 2002, pp. 294–305. DOI: 10.1109/MICRO.2002.1176258 (cit. on p. 117).
- [Wen+07] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. **On-Chip Interconnection Architecture of the Tile Processor**. In: *IEEE Micro* 27.5 (2007), pp. 15–31. DOI: 10.1109/MM.2007.89 (cit. on pp. 3, 13, 35, 38).
- [Wu03] Jie Wu. **A Fault-Tolerant and Deadlock-Free Routing Protocol in 2D Meshes Based on Odd-Even Turn Model**. In: *IEEE Trans. Comput.* 52.9 (Sept. 2003), pp. 1154–1169. DOI: 10.1109/tc.2003.1228511 (cit. on p. 72).
- [Xil12] Xilinx. **Virtex-6 FPGA CLB User Guide**. Ed. by Xilinx. 2012. URL: https://www.xilinx.com/support/documentation/user_guides/ug364.pdf (cit. on p. 30).
- [Xil15] Xilinx. **ISE WebPACK Design Software**. [Online; accessed 22-June-2016]. 2015 (cit. on pp. 29, 122, 148).
- [YA12] Qiaoyan Yu and Paul Ampadu. **Transient and Permanent Error Control for Networks-on-Chip**. Springer Science & Business Media, 2012. DOI: 10.1007/978-1-4614-0962-5 (cit. on pp. 67, 68).
- [YBM03] Terry Tao Ye, Luca Benini, and Giovanni De Micheli. **Packetized on-chip interconnect communication analysis for MPSoC**. In: *Design, Automation and Test in Europe Conference and Exhibition, 2003*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), 2003, pp. 344–349. DOI: 10.1109/DATE.2003.1253632 (cit. on pp. 117, 130, 142).
- [YKH10] Zhiyao Joseph Yang, Akash Kumar, and Yajun Ha. **An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee**. In: *2010 International Conference on Field-Programmable Technology*. IEEE. Institute of Electrical & Electronics Engineers (IEEE), Dec. 2010, pp. 389–392. DOI: 10.1109/fpt.2010.5681443 (cit. on p. 140).

Affidavit

I hereby confirm that my thesis entitled *Advances in Deflection Routing based Network on Chips* is the result of my own work. I did not receive any help or support from commercial consultants. All sources and / or materials applied are listed and specified in the thesis.

Furthermore, I confirm that this thesis has not yet been submitted as part of another examination process neither in identical nor in similar form.

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, die Dissertation *Advances in Deflection Routing based Network on Chips* eigenständig, d.h. insbesondere selbständig und ohne Hilfe eines kommerziellen Promotionsberaters, angefertigt und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet zu haben.

Ich erkläre außerdem, dass die Dissertation weder in gleicher noch in ähnlicher Form bereits in einem anderen Prüfungsverfahren vorgelegen hat.

Würzburg, 2017

(Armin Runge)