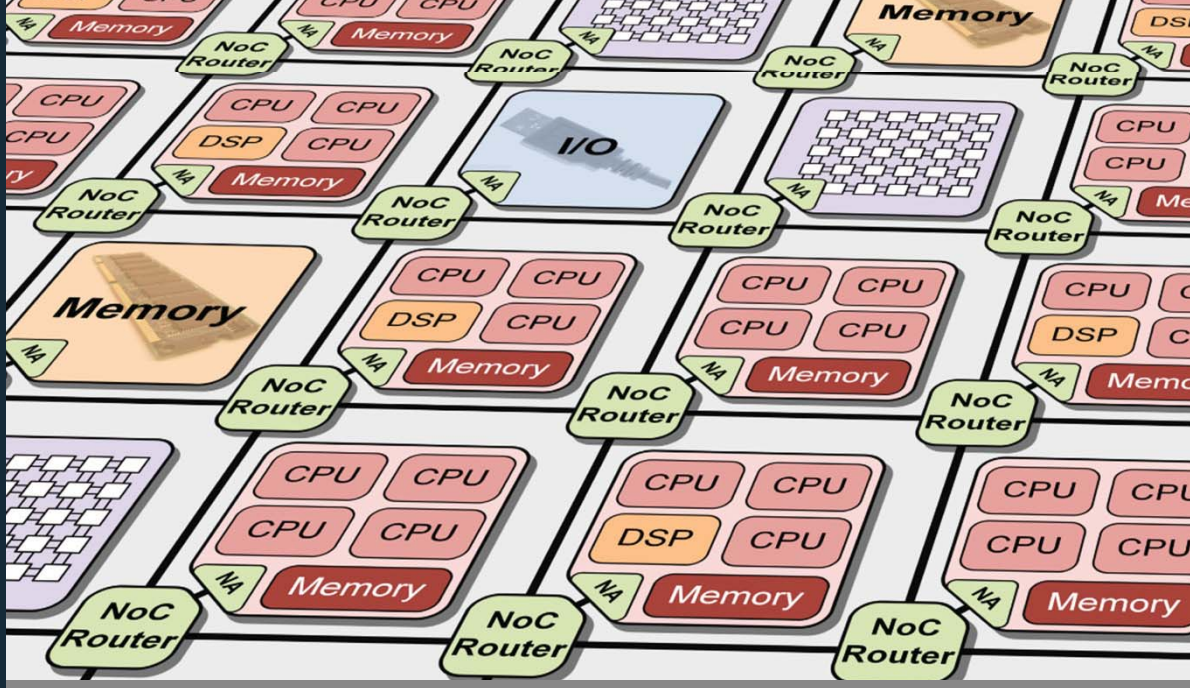


Jan Heißwolf

A Scalable and Adaptive Network on Chip for Many-Core Architectures

A Scalable and Adaptive Network on Chip for Many-Core Architectures

Zur Erlangung des akademischen Grades eines

DOKTOR-INGENIEURS (Dr.-Ing.)

von der Fakultät für
Elektrotechnik und Informationstechnik
am Karlsruher Institut für Technologie (KIT)
genehmigte

DISSERTATION

von

Dipl.-Ing. Jan Heißwolf

geboren in Backnang

Tag der mündlichen Prüfung:
11.11.2014

Hauptreferent: Prof. Dr.-Ing. Dr. h. c. Jürgen Becker
Korreferent: Prof. Dr. sc.techn. Andreas Herkersdorf

Abstract

The continuous observance of Moore's law has enabled to continuously implement increasingly powerful single-core processors than in the past. The increase of clock frequency and complexity of the microarchitecture were previously the main means enabling this performance enhancement. However, physical and architectural limitations necessitated a rethinking in recent times. Instead of increasing the performance of a single core, the number of cores is elevated today in order to enhance the computing power of a system. This concept of multi-core and many-core architectures enables to increase the performance almost linearly with the number of cores. However, existing bus-based communication infrastructures emerged as a limiting factor for this method of performance increase. Consequently, networks on chip have been proposed to create scalable many-core processor systems with respect to communication.

Inspired by these developments, a network on chip based communication infrastructure is presented in this work. In order to simplify and accelerate the design of future many-core systems, a modular simulator-based evaluation and design methodology for networks on chip is introduced. As a basis for the proposed modular communication system, a scalable state of the art network on chip is developed in this work. This network is extended by novel mechanisms for quality of service, self-optimization and fault tolerance.

In the context of quality of service, a novel approach for run-time adaptive bandwidth reservation, for end-to-end connections is proposed. It enables adjustable, hard guarantees for throughput and latency. An approach for region-based allocation of communication resources, enables run-time establishment of virtual networks in order to isolate the communication of individual applications. Novel self-optimization strategies are intended to increase the performance of the network and optimize its power consumption without necessitating complex software management. An innovative approach for detection, localization and treatment of permanent errors prepares the presented network on chip architecture for future technologies; with such errors being expected to occur frequently. With these aspects and concepts, this work provides a thorough and flexible approach for scalable communication in future many-core processor systems.

Zusammenfassung

Die kontinuierliche Einhaltung des mooreschen Gesetzes hat in der Vergangenheit die Realisierung immer leistungsfähigerer Einkernprozessoren ermöglicht. Die Steigerung der Taktfrequenz und komplexer werdende Mikroarchitekturen waren die maßgeblichen Mittel um diese Leistungssteigerung zu ermöglichen. Physikalische Limitierungen und Grenzen bei der Komplexität der Mikroarchitekturen haben in jüngerer Zeit ein Umdenken nötig gemacht. Statt die Performanz eines einzelnen Kerns zu steigern, wurde nun die Anzahl der Prozessorkerne erhöht, um die Rechenleistung eines Systems zu erhöhen. Dieses Prinzip der Mehr- und Multikernprozessoren bietet nahezu eine beliebige Leistungssteigerung. Als ein limitierender Faktor bei diesem Prinzip der Performanzsteigerung stellten sich bestehende, busbasierte Kommunikationsinfrastrukturen heraus. Networks on Chip stellen den benötigten Paradigmenwechsel dar, der es ermöglichen soll, skalierbare Mehrkernprozessorsysteme zu realisieren.

Inspiriert durch diese Entwicklungen wird in dieser Arbeit eine Network-on-Chip-basierte Kommunikationsinfrastruktur vorgestellt. Um den komplexen Entwurf zukünftiger Mehrkernsysteme zu vereinfachen und zu beschleunigen, wird ein simulator- und baukastenbasierter Ansatz zur Evaluierung und Generierung von Networks on Chip vorgestellt. Als Basis des modularen Baukastens wurde ein skalierbares Network on Chip entwickelt, das dem Stand der Technik entspricht. Dieses Netzwerk wird im Rahmen dieser Arbeit um neuartige Mechanismen für Quality of Service, Selbstoptimierung und Fehlertoleranz erweitert.

Im Kontext von Quality of Service wird ein neuartiges Konzept zur laufzeitadaptiven Bandbreitenreservierung für Ende-zu-Ende-Verbindungen mit anpassbaren harten Garantien für Durchsatz und Latenz realisiert. Ein Konzept zur regionsbasierten Allokation von Kommunikationsressourcen ermöglicht die Laufzeitrealisierung virtueller Netzwerke und damit die Isolation einzelner Anwendungen. Neuartige Selbstoptimierungsstrategien sollen dazu dienen, die Performanz des Netzwerkes zu steigern und dessen Energieverbrauch zu optimieren, ohne das ein komplexes Softwaremanagement nötig ist. Ein innovatives Konzept für die Erkennung, Lokalisierung und Behandlung von permanenten Fehlern bereitet die vorgestellte Architektur auf zukünftige Technologien vor, bei denen vermehrt mit solchen Fehlern zu rechnen ist. Mit der Summe all dieser Aspekte und Konzepte bietet diese Arbeit einen ganzheitlichen und flexiblen Ansatz für die skalierbare Kommunikation in zukünftigen Mehrkernprozessorsystemen.

Vorwort

Die vorliegende Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am Institut für Technik der Informationsverarbeitung (ITIV) innerhalb des Karlsruher Instituts für Technologie (KIT). In dieser Zeit habe ich nicht nur einen sehr vielseitigen Arbeitsalltag mit Lehre, Studentenbetreuung, Implementierungsarbeiten und Forschung erleben dürfen, sondern ich habe vor allem großartige Menschen getroffen und mit ihnen zusammengearbeitet. Ohne nun eine lange Liste von Namen zu nennen, möchte ich mich bei allen bedanken, die das ITIV zu dem gemacht haben was es ist und dafür gesorgt haben, dass ich immer gerne zur Arbeit gekommen bin.

Meinem Doktorvater, Herrn Prof. Jürgen Becker, danke ich für die Anstellung am ITIV und sein Vertrauen, für seine Unterstützung, die freie Arbeitsweise und die Betreuung dieser Arbeit. Herrn Prof. Andreas Herkersdorf danke ich für die gute und erfolgreiche Zusammenarbeit im InvasIC Projekt, für das Korreferat dieser Arbeit und die damit verbundene Zeit und Mühe. Auch bei Herrn Prof. Teich möchte ich mich an dieser Stelle für die gute Zusammenarbeit im Teilprojekt B5 und für die Initiierung des InvasIC Projektes bedanken. Des Weiteren gilt mein Dank Herrn Prof. Hohmann, Herrn Prof. Dostert und Herrn Prof. Jelonnek für ihre Rolle als Prüfer und die damit verbundene Zeit.

Im Umfeld des InvasIC Projektes, in dem die vorliegende Arbeit entstanden ist, hatte ich die Möglichkeit viele nette Kollegen kennenzulernen und mit ihnen zusammenzuarbeiten. Die interdisziplinäre und interkulturelle Teamarbeit habe ich sehr genossen und dabei viel gelernt. Besonders möchte ich mich bei Aurang Zaib und Andreas Weichslgartner für die gute und erfolgreiche Zusammenarbeit bedanken. Die vielen Telefongespräche, das gemeinsame Publizieren und Hardware-Debugging hat immer bestens funktioniert. Darüber hinaus möchte ich mich bei Ralf König und Timo Stripf für ihre Unterstützung im InvasIC Projekt und die zahlreichen fachlichen Diskussionen bedanken.

Als ich vor etwas mehr als fünf Jahren am ITIV angefangen habe, wurde Michael Rückauer zu meinem Büronachbarn. Michael stand mir als gelernter Informatiker immer mit Rat und Tat zur Seite. Für seine Hilfe, die vielen spannenden und bereichernden Gespräche und Diskussionen bin ich ihm sehr dankbar. Auch bei Stephanie Friederich bedanke ich mich für die tolle Büronachbarschaft im letzten Jahr und die sehr gute Zusammenarbeit im InvasIC Projekt.

Während meiner Zeit am ITIV habe ich zahlreiche studentische Arbeiten betreut. Ohne diese Arbeiten wäre die vorliegende Dissertation sicher nicht in dieser Art und in diesem Umfang möglich gewesen. Für die umfangreichen Implementierungsarbeiten im Rahmen von Bachelor- und Masterarbeiten möchte ich mich hier ganz herzlich bei allen betreuten Studenten bedanken. Besonders hervorheben möchte ich die Arbeit meiner wissenschaftlichen Hilfskräfte Martin Kupper, Maximilian Singh und Simon Bischof. Die Zusammenarbeit mit ihnen war für mich immer sehr abwechslungsreich und hat mir viel Freude bereitet, auch dafür bedanke ich mich.

Bei meinen Eltern bedanke ich mich dafür, dass sie immer an mich geglaubt und mich jederzeit bedingungslos unterstützt haben. Sie erkannten meine Leidenschaft für Technik schon in jungen Jahren und förderten diese. Die Unterstützung meiner Eltern ging so weit, dass sie es mir bereits in jungen Jahren ermöglicht haben ein eigenes IT-Gewerbe zu führen. Für all diese Unterstützung bin ich ihnen sehr dankbar. Auch bei meinem Bruder Steffen, sowie meinem gesamten Freundeskreis möchte ich mich an dieser Stelle für ihre Unterstützung und die Ablenkung von der Arbeit bedanken.

Mein größter Dank gilt Marina. Du hast immer Verständnis für die langen Arbeitszeiten und die wenige gemeinsame Freizeit gehabt. Du hast mich immer unterstützt und mir den Rücken freigehalten, damit ich mich der vorliegenden Arbeit widmen konnte. Für deine Unterstützung, dein Verständnis und deine unendliche Liebe danke ich dir von ganzem Herzen.

Karlsruhe, im Dezember 2014

Jan Heißwolf

Contents

1. Introduction

The steady improvement in the production of digital circuits allows continuously higher integration densities. Thus, systems with increasing complexity can be implemented on a single chip. Gordon Moore predicted this evolution already in 1965 [179]. According to *Moore's law*, the number of transistors in a dense integrated circuit doubles every 18 to 24 months. Figure 1.1 shows Moore's prediction and different semiconductor devices as well as their transistor counts and release dates. The diagram shows how well Moore's law has been adhered to, until today.

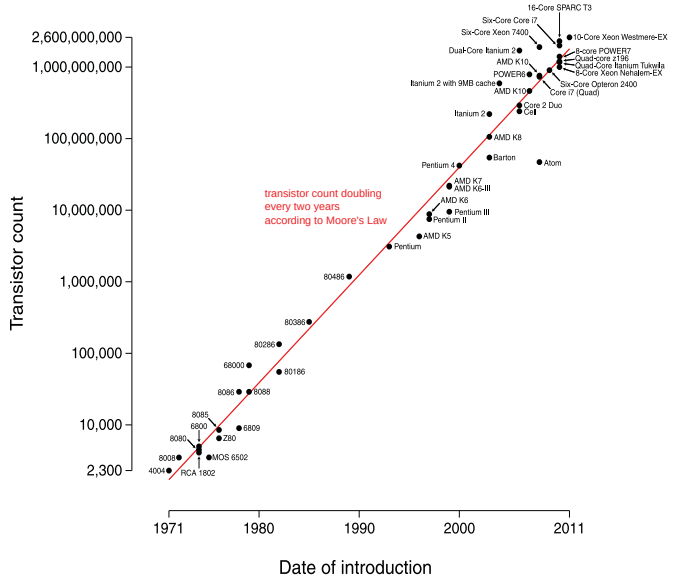


Figure 1.1.: Transistor count evolution for different microprocessors between 1971-2011 (black) and the predictions according to Moore's law (red) [264].

According to the *International Technology Roadmap for Semiconductors (ITRS)* [120] it is expected that Moore's law can be observed for at least the next decade[1].

[1] At the years 2020-2025 many physical dimensions are expected to be crossing the 10 nm threshold. It is expected that as dimensions approach the 5-7 nm range it will be difficult to operate any

1

The ever-increasing number of transistors has been utilized in the past to improve the performance of integrated circuits by increasing their complexity. In addition, the increasing switching speed resulting from new technology nodes enabled to enhance the clock frequency of integrated circuits continuously. In the past, both factors played an important role in improving the performance of microprocessors and *central processing units (CPUs)*.

In recent years, the technology improvement could hardly be used to further increase the performance of single cores according to *Pollack's rule*, which reads *"performance increases roughly proportional to the square root of the increase in complexity"* [32]. Figure 1.2 shows integer computation performance enhancement of new microarchitectures against area increase from the previous generation microarchitecture; assuming the same process technology.

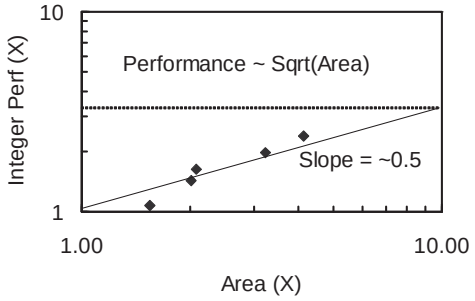


Figure 1.2.: Microarchitecture performance enhancement and area requirements according to *Pollack's rule* [32].

In addition to Pollack's rule, various physical limitations prevent designers from further increasing performance of an individual processor; nowadays. The wire latencies increase relative to the gate delays for shrinking technologies [215, 111]; notably in large monolithic circuits, such signal propagation delays limit the clock frequency.

An even more essential problem from the physical perspective is the power dissipation P_{total}, which is composed of a static and a dynamic part:

$$P_{total} = P_{dyn} + P_{stat} \tag{1.1}$$

The static component P_{stat} of the power dissipation consists mainly of leakage power [42]. The leakage power in turn is proportional to the leakage current and

transistor structure that is utilizing the MOS physics as the basic principle of operation. ITRS expects that new devices, such as the tunnel transistors, will allow a smooth transition from traditional *complementary metal oxide semiconductor (CMOS)* to this new class of devices to reach these new levels of miniaturization [120].

the supply voltage. The largest share of the total power of a CMOS circuit is the dynamic power P_{dyn}, which can be estimated as follows according to [275, 42]:

$$P_{dyn} = C_{load} \cdot V_{dd}^2 \cdot f + I_{sc} \cdot V_{dd} \tag{1.2}$$

C_{load} is the capacity that must be charged or discharged, V_{dd} is the supply voltage, f the clock frequency and I_{sc} the short circuit current through p-MOS and n-MOS transistor during switching. The clock frequency f depends on the supply voltage. A higher V_{dd} increases the switching speed. Thus, the supply voltage must be increased to enhance the clock frequency above a certain threshold, which is circuit dependent. The correlation between f and V_{dd} results in a drastic increase of the dynamic power consumption when increasing the clock frequency of a circuit. Although V_{dd} could be decreased in the past for new technologies, the increased transistor density and the higher clock frequencies led to a dramatic raise in power density [30]. Figure 1.3 shows the progress in power density between 1990 and 2010. The peak in power density was reached in 2005 with the release of *Intel's Pentium 4* processor' whose power density is nearly comparable with that of a nuclear reactor.

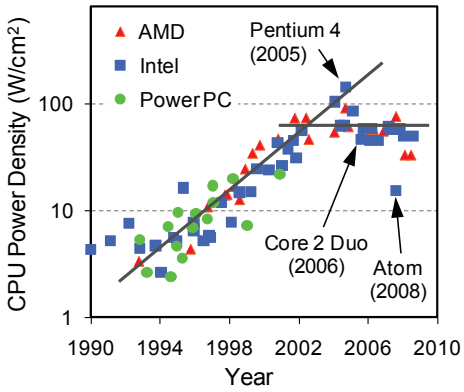


Figure 1.3.: Near-exponential increase of CPU power density [205].

According to *Pollack's rule* and especially due to the power density limitations resulting from a continuous increase of clock frequency and integration density, the performance of single cores could hardly be further increased. This was the main reason for integrating multiple cores on a single integrated circuit die around the year 2005. Figure 1.3 shows that such *chip multiprocessors (CMPs)* helped to limit the power density. CMPs, also known as multi-core or many-core architectures, overcome the limitations of Pollack's rule and can be used to increase the theoretical performance of a system with the number of cores

relative to the die area. Since the introduction of the first multi-core architectures, such as Intel's Core 2 Duo [91], IBM's Cell architecture [201] and SUN's Niagara multi-core [142], the number of cores was continuously increased. Following that trend, the first many-core architectures that consist of dozens of cores, have been developed for research purpose. The most well-known representatives of existing many-core architectures are the *Tile64* processor [18] and *Intel's Larrabee* [224] and *Single-chip Cloud Computer (SCC)* [113] architectures. However, with the growing number of cores, the bandwidth and complexity of the on-chip communication infrastructure must be increased as well to cope with the growing requirements.

1.1. Prologue

Today, many-core architectures are widely accepted as processor architectures of the future [33]. They solve the physical limitations that prevent designers from further increasing the performance of single-cores, as previously described. The improvement in performance gained by the use of multiple processor cores depends very much on the degree of parallelism of the software. In particular, parallel execution is limited by the fraction of the software that can be executed in parallel and simultaneously on multiple cores. This effect is described by *Amdahl's law* [10]:

$$S(p,n) = \frac{1}{(1-p) + \frac{p}{n}} \tag{1.3}$$

$S(p,n)$ is the speedup experienced from parallel execution on n cores compared to a sequential execution of a program. The amount p can be executed in parallel and the amount $(1-p)$ can only be executed sequential. Amdahl's law shows that many-core architectures require applications that offer a degree of parallelism in order to utilize the cores of the architecture. This is a huge challenge from the software perspective, involving programming languages, compilers, operating systems and especially applications and algorithms [169]. All of these aspects are subject to the current research, as described in detail in chapter 3.

The original law of Amdahl does not consider any architectural aspects and neglects other aspects, such as synchronization overhead. Thus, different extensions have been proposed in the past [6] for refinement. One of the most famous extensions of Amdahl's law [109] assumes a hardware architecture where the cores can either be used separately for parallel execution, or coupled to accelerate the sequential part $(1-p)$ of the program, as enabled by the *KAHRISMA* architecture [141][KSHB11]. The communication requirements of parallel applications and the architectural communication capabilities have not been addressed.

1.1.1. Amdahl's Law from the Communication Perspective

In the following paragraph, an attempt will be made to extend Amdahl's law, taking into account communication requirements of parallel applications as well as the communication capabilities of a many-core architecture. This is done to show how the underlying communication infrastructure impacts the execution of a parallel application. In order to take the communication requirements of a parallel program p into account, it is split up in two parts:

$$p = p_l + p_g \leq 1 \tag{1.4}$$

p_l is the part of the parallel program that is independent from global communication. This means that the communication is only core local[2]. The part of the program that requires global communication[3] is p_g.

Taking communication into account, the total amount of global communication of a parallel program using n cores must be known. In general, the total amount of communication c is the sum of the communication or bandwidth of all nodes:

$$c(n) = \sum_{i=0}^{n-1} c_{core}(n) \approx n \cdot c_{core} \tag{1.5}$$

Without loss of generality, it is assumed that all nodes of a parallel application have the same amount of communication c_{core}, as provided by the approximation in equation 1.5. A similar assumption was made by Amdahl for the computation that is divided equally between the cores in equation 1.3.

From the communication perspective, an additional aspect must be taken into account: the capabilities of the hardware architecture. Without loss of generality it is assumed for the following that all cores use the same communication infrastructure with the capacity or bandwidth c_{arch}. The bandwidth c_{arch} is used to define an impact factor of the communication infrastructure of a many-core architecture with respect to the communication $c(n)$ of a parallel application:

$$f_c(n) = max(1, \frac{c(n)}{c_{arch}}) = max(1, \frac{n \cdot c_{core}}{c_{arch}}) \geq 1 \tag{1.6}$$

The impact factor $f_c(n)$ estimates the influence on the part of a parallel program p_g, which requires global communication. It is equal to 1 as long as the bandwidth

[2]Local communication refers to communication that does not cross the boundaries of a single core of the architecture. A core can for sure contain local memory or an exclusive cache hierarchy, which can be accessed by local communication.

[3]Global communication crosses the boundaries of a core and can interfere with global communication of other cores.

of the architecture is sufficient for the application. p_g is smaller than one, if the application is affected by bandwidth limitations of the architecture.

Using equation 1.4, 1.5 and 1.6, Amdahl's law can be extended as follows:

$$S(p_l, p_g, f_c(n)) = \frac{1}{(1 - (p_l + p_g)) + \frac{p_l}{n} + \frac{p_g}{n} \cdot f_c(n)} \tag{1.7}$$

For the extended speedup function, it is assumed that the sequential part of the program $(1 - (p_l + p_g))$ is not affected by architectural bandwidth limitations due to the relatively small communication requirements of the sequential execution.

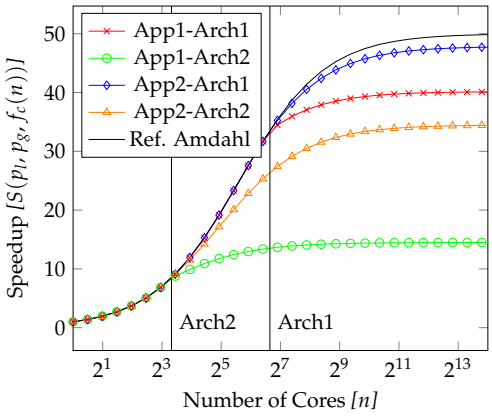


Figure 1.4.: Amdahl's law from the communication perspective for two different applications (both with $p = 0.98$) on two different architectures: *App1*: Much communication ($p_l = p_g = 0.49$), *App1*: Little communication ($p_l = 0.89, p_g = 0.09$) - *Arch1*: High communication bandwidth ($c_{arch} = 100 * c_{core}$), *Arch2*: Low communication bandwidth ($c_{arch} = 10 * c_{core}$).

Figure 1.4 shows the speedup S for two different applications and two architectures, with parallelism n between 1 and 16384. The extension of Amdahl's law, given in equation 1.7, is applied to calculate S. *App1* is an application with relatively high communication requirements, whereas *App2* has very moderate communication requirements. Both application characteristics are applied on two different communication architectures. *Arch1* is able to process a high amount of parallel communication and *Arch2* has relatively low communication capabilities. The original law of Amdahl is used as a reference (*Ref. Amdahl*) for the theoretical speedup of both applications[4]. The results provided in figure 1.4 show that

[4]Both applications have the same degree of parallelism $p = 0.98$ used in equation 1.3

communication infrastructure can have a huge impact on the performance of a parallel application. *App1* with its high communication requirements can profit significantly from the high communication bandwidth provided by *Arch1*.

1.2. Motivation

The previous investigations showed that communication requirements of applications and their satisfaction by the architecture can have a huge impact on the speedup gained from parallel execution. Conventional bus-based interconnections, such as the widely used AMBA bus [79] developed by ARM, do not fulfill the requirements of many-core architectures. Their essential limitation, with respect to such architectures, is the bandwidth. It must be shared between all cores [101]. Another drawback is the physical realization that becomes more complex with a growing number of cores. In general, it can be said that buses provide a limited scalability with respect to large many-core architectures. However, such architectures are feasible within one decade from the technology perspective [33].

Because of these disadvantages, *networks on chip (NoCs)* have been proposed by L. Benini and G. De Micheli [19], W. J. Dally [61], A. Jantsch [122] and others [145] as communication infrastructure for future many-core systems. In contrast to on-chip bus systems, networks on chip enable parallel communication. The aggregated bandwidth of a NoC scales with the network size. This enables scalability with respect to large architectures. As discussed in section 1.1.1, sufficient bandwidth of the communication infrastructure is necessary to gain the potential speedup of a parallel application.

In addition to these advantages, in terms of performance, NoCs are also beneficial from the technology perspective [101, 175]. For NoCs, directed point-to-point connections are utilized, independent of the network size. This simplifies the integration. Network wires can be pipelined because the network protocol is globally asynchronous. This allows to limit the wire length and can enable to achieve high clock frequencies.

Networks on chip can be seen as the key innovation on the architectural level in order to implement future many-core architectures with hundreds of cores. The research and development of networks on chip comprises many different challenges [192, 170] which must be solved. One of the most important aspects is the NoC architecture itself, which comprises the topology of the network, the structure and the functionality of its components. Depending to the use case of the NoC, the components must fulfill different functional and nonfunctional requirements:

Scalability: Depending on the size of the architecture, different strategies for routing and resource allocation have to be selected.

Quality of Service: Delay and throughput guarantees may be required for safety critical or real-time applications, but can also improve the performance of other parallel programs.

Software management: Depending on the programming model[5], different software interfaces and transmission schemes must be supported. Additionally, a management interface for the NoC features must be provided. Self-optimization can be employed to reduce the burden of management.

Energy efficiency: Energy saving techniques play a key role in future systems with billions of transistors. Therefore, the power budget for the communication infrastructure must be limited and the energy efficiency improved.

Reliability/Dependability: Depending on the employed technology and the size of an architecture, fault tolerance and reliability of the on-chip components is an important aspect. For future technology nodes, addressed by NoC-based architectures, reliability and fault tolerance are expected to play a key role [105] to improve the yield during production and the availability of a system during operation.

The variety of different requirements for on-chip communication of future architectures result in a huge challenge for the realization of an integral concept. This challenge must be mastered to reach the goal of a many-core architecture with hundreds of cores [33].

1.3. Contribution

In the past, extensive research has been done, addressing individual aspects that are required for the realization of large NoC-based architectures [23, 210]. However, in most cases, only single aspects are addressed by existing concepts and architectures. The complexity of many-core architectures and the diversity of the research challenges, such as; energy efficiency, dependability and scalability make it very challenging to address the problem in its entirety.

However, the development and evaluation of an integral, but flexible architecture concept for scalable on-chip communication is a focus of this work. In order to achieve this goal, approved existing concepts for scalable and flexible on-chip communication shall be identified as a basis. The identified concepts shall be applied to create a scalable communication infrastructure; suitable for large

[5] Well known parallel programming models are: *message passing interface (MPI)* [234], OpenMP [56] or *partitioned global address space (PGAS)*.

multiprocessor systems on a chip (MPSoCs). This base architecture can be considered as a **state of the art on-chip communication infrastructure** for future many-core architectures.

Apart from scalability, the base architecture shall be developed with the focus on design-time configurability and expandability; in order to embed it in a **semi-automated design flow**. This design flow shall enable a fast, but accurate evaluation of different configuration options and comfortable extension of the base architecture. A cycle-accurate simulation model and a *hardware description language (HDL)* model of the architecture are necessary to enable rapid and accurate evaluation of implementation alternatives with respect to a short time to market. The simulation model enables fast and accurate performance measurements. An HDL model shall be employed to evaluate technology dependent aspects[6] of a specific configuration meeting the performance requirements. The goal is to establish a design flow that allows fast evaluation of implementation alternatives and new features, as well as subsequent generation of a synthesizable design. This methodology shall simplify and accelerate the design of future NoC-based MPSoC architectures.

Moreover, this design flow is employed for further research towards the goal of a holistic communication concept. As further requirements for such a communication infrastructure, *quality of service (QoS)* support, self-optimization and dependability are identified. These aspects shall be addressed using the established design flow and base architecture. The base architecture shall be extended gradually towards a holistic NoC design.

The rising number of components increases the demand for **quality of service** support in order to isolate critical communication flows. Consequently, a scalable communication infrastructure must provide QoS mechanisms. The design, implementation and evaluation of scalable QoS mechanisms is an essential research objective of this work. In this context, mechanisms for a run-time adaptive allocation of communication resources, according to the application's requirements, shall be investigated. The provision of run-time adaptive guarantees for latency and throughput of point-to-point connections are goals to be achieved. Run-time management of connections and communication resources must be addressed, enabling QoS support from the software perspective. In addition to connection-oriented QoS support, communication resource allocation on the granularity of applications and regions of the architecture must be supported to overcome current limitations and to enable a region-based management. Virtualization of the communication infrastructure shall be investigated to enable isolation of the communication of entire applications.

[6] Technology dependent aspects are power consumption, area or achievable clock frequency of the design. These aspects are closely related to the technology node.

The size and complexity of large multiprocessor system on a chip architectures leads to high power consumption and aggravates the management. In order to reduce the burden of management for the software and *operating system (OS)* and to limit the energy footprint of a NoC, **self-optimization strategies** for the communication infrastructure are necessary. Self-optimization strategies for power minimization, performance improvement and load balancing in the NoC shall be investigated in this work to achieve the goal of a holistic communication approach. In particular, transparent NoC-internal use of QoS communication is evaluated to optimize the performance and energy footprint for communication. A strategy for balancing the distribution of QoS end-to-end connection is proposed. Fine-grained power gating of NoC components shall further reduce the **power consumption** of the communication infrastructure.

Dependability of the communication infrastructure is another aspect that is addressed in order to achieve the goal of a holistic NoC design. Dependability and fault tolerance will play an ever increasing role when using future technology nodes for *very-large-scale integration (VLSI)*. The focus will be on **permanent errors**, resulting from process fluctuations, or aging effects. In existing work, very simple networks have been assumed when fault tolerance schemes have been investigated. In contrast, this work attempts to create a fault tolerance concept for a very complex scalable network with QoS and self-optimization support. A scalable scheme for detection and localization of defects and faults shall be developed and evaluated by the use of accurate gate-level simulations. Based on the localization scheme, faulty routers shall be disabled. Therefore, a transparent bypass scheme for faulty routers and regions is desired. When no faults are present, the alternative use of the bypass scheme is considered in order to minimize the overhead for fault tolerance.

The realization of all these aspects shall lead to an integral concept for communication in future many-core architectures.

1.4. Outline

The work is organized as follows. In chapter 2, the principles and fundamentals for the present work are defined and explained. The integration of various components on a single chip, which necessitate the development of advanced on-chip communication concepts, is described in section 2.1. Multi-core and many-core architectures can be considered as a subcategory of such *systems on a chip (SoCs)*, described in section 2.2. At the end of the section, an overview of existing NoC-based multi-core and many-core architectures is provided. General principles and concepts for on-chip communication are presented in section 2.3. Bus systems and point-to-point connections are discussed as a predecessor of on-

chip networks. Finally, networks on chip are introduced in section 2.4. The basic components, different topologies, switching schemes, flow control mechanisms and routing schemes are introduced and discussed. In this context, state of the art for each of these aspects is summarized. At the end of the chapter, the fundamentals for fault tolerance and dependability are presented.

The context of this work is discussed in chapter 3. In particular, the concept of invasive computing and the InvasIC hardware architecture are described in section 3.1 and 3.2 respectively. The individual components of the heterogeneous InvasIC architecture are presented and their communication requirements are discussed. The software perspective and a scalable hardware management concept for invasive computing is described in section 3.3. It is reflected by the NoC management scheme presented in chapter 6. The concept for multi-FPGA prototyping, which has a close relation to the NoC implementation and its topology, is summarized in section 3.4.

The general concept of this work is described in chapter 4. Therefore, general communication requirements and constraints of future many-core architectures are analyzed and identified in section 4.1. Additionally, the requirements of the InvasIC architecture are discussed. The fulfillment of these requirements by existing network on chip architectures is discussed in section 4.2. The limitations of existing architectures are used as motivation for the general concept presented next. This concept and the basic design decision for the communication infrastructure, presented in this work, are described in section 4.3. In particular, aspects of scalability, quality of service, distributed management and reliability are addressed. A template-based design methodology for the proposed NoC architecture is introduced in section 4.3.5. It is the basis for the semi-automated design flow that is described in section 4.4.

The basic network on chip design is described in detail in chapter 5. The functional principle and implementation of the NoC router is explained in section 5.1. A simulation framework for the proposed NoC template is introduced in section 5.2 and is used for evaluation as well as design space exploration. The simulation framework and the HDL implementation, introduced in section 5.3, constitute the proposed semi-automated design flow. Its usability is demonstrated by a case study, which is presented in section 5.4. At the end of chapter 5, various extensions of the base architecture are introduced. These extensions are employed for the concepts and mechanisms presented subsequently.

Chapters 6-8 represent the actual research and contribution of this work. They rely on the general concept from chapter 4 and use the base architecture from chapter 5, as well as the simulation framework and the HDL implementation for evaluation.

Quality of service aspects are addressed in chapter 6. An overview of the state of the art in this field of research is provided in section 6.1. A concept and

implementation for end-to-end connections, with run-time adjustable bandwidth and latency guarantees, is introduced in section 6.2. A run-time management scheme for QoS resources is presented and evaluated in section 6.3. This scheme is extended in section 6.4 to enable communication resource allocation for individual regions and applications. The exclusive allocation of communication resources enables so-called virtual networks.

Different strategies for self-optimization are addressed in chapter 7. A separate discussion of related work is provided for each of the proposed strategies. First, a concept named rerouting is introduced and evaluated in section 7.1. Rerouting balances the distribution of QoS end-to-end connections automatically. Two strategies for optimizing the overall communication by automatic setup of end-to-end connections are presented in section 7.2. Adaptive data collection schemes for software management and optimization are evaluated in section 7.3. At the end of the chapter, a self-optimizing power management concept using fine-grained power gating is described and evaluated.

A dependability concept for the proposed NoC is introduced in chapter 8. At the beginning of the chapter, related work is discussed in section 8.1. Subsequently, a fault detection and localization scheme for the complex NoC, developed in this work, is introduced and evaluated in section 8.2. The localization scheme is used to disable and bypass faulty routers. This is enabled by the architecture extension, introduced and investigated in section 8.3. More specifically, a second network layer is introduced to bypass faulty routers. However, it can also be employed for power saving.

Subsequently, this work is concluded in chapter 9, with an outlook on future work presented in section 9.2.

2. Fundamentals

The network on chip, presented in the following, addresses future multiprocessor systems on a chip and many-core architectures. This chapter provides the fundamentals for many-core architectures and networks on chip.

First, the general principle of systems on a chip is introduced in section 2.1. Subsequently, the fundamentals of MPSoCs and many-core architectures are described in section 2.2. The basic principles for on-chip communication are provided in section 2.3. The latest evolution of these on-chip communication infrastructures are networks on chip. NoCs and their basic mechanisms are introduced in section 2.4. At the end of the chapter, the basic principles for reliable and fault tolerant systems are introduced as a basis for the concept presented in chapter 8.

2.1. System on a Chip

Due to the continuous observance of Moore's law (see chapter 1) the integration density increases steadily. This trend enabled to implement not only single components but complete systems on a single piece of silicon. Such systems, which have been implemented on a *printed circuit board (PCB)* in the past, are named *system on a chip (SoC)* if they are realized on a single die [82]. They combine multiple digital components (e.g. processing cores, accelerators and peripherals) with mixed-signal (e.g. digital/analog converter or analog/digital converter) and analog components (e.g. sensors or filters). Systems on a chip enable to build more complex and compact embedded systems by integrating most of the required functionality on a single chip. The single chip integration also reduces the power consumption of a system [133] compared to a PCB realization. This enables the construction of high-performance, compact mobile devices that can cope with the limited power budget of batteries [134, 40].

Figure 2.1 shows the evolutionary process of SoC development. With the increasing number of components of a SoC, the amount of on-chip communication grows. This leads to more complex communication infrastructures, as described in detail in section 2.3 and 2.4. Moreover, the computation demands of SoCs grow persistently. In order to meet these demands, the number of processor cores

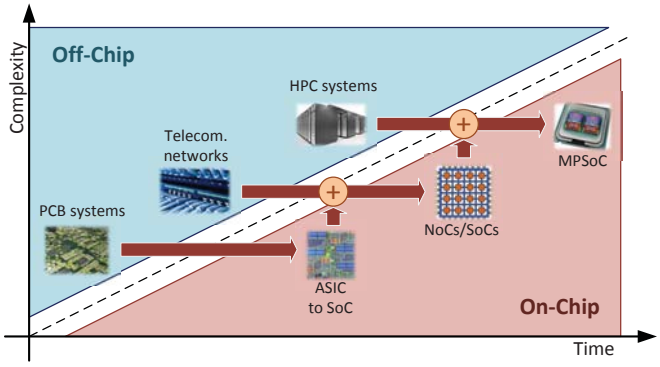


Figure 2.1.: Evolution from PCB-based systems to fully integrated MPSoCs [253].

integrated on a chip was increased. Such SoCs with multiple processing cores are named *multiprocessor system on a chip (MPSoC)* [268]. If the MPSoC mainly consists of processing cores, it is often referred to as multi-core architectures or chip multiprocessor.

2.2. Multi-Core and Many-Core Architectures

"A multi-core processor is an integrated circuit (IC) to which two or more processors have been attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks." [216]. The processing units or *processing elements (PEs)* can be carried out as *reduced instruction set computer (RISC)* [199] or *complex instruction set computer (CISC)* [200]. More exotic processing schemes are *very long instruction word (VLIW)* [77] or *explicitly parallel instruction computing (EPIC)* [221].

Typically, each processing unit (referred to as core) has an exclusive cache-hierarchy that is used to hide memory latencies for data and instruction access. Typically, this hierarchy consists of two separate L1 caches for data and instructions and optionally a larger but slower L2 cache that is shared between data and instructions. A global communication infrastructure is used for communication between the processing cores, main memory access and *input/output (I/O)* access; as described in section 2.3 and 2.4.

If the architecture consists of a large number of cores it is mainly referred to as many-core architecture[1]. According to [108] multi-core or many-core architectures

[1] In the following architectures with dozens of cores will be referred to as many-core architecture.

can be built as shared memory architectures or distributed memory architectures as explained below in more detail.

2.2.1. Shared Memory Architecture

Through their caches all cores can access the main memory of the systems and other components; such as peripherals and I/O-buses. For the communication between the single cores and the memory subsystem, shared buses are commonly used. The bandwidth of this global communication infrastructure, as well as the bandwidth to the main memory, are shared by all cores. This bandwidth is an important factor for the performance of such an architecture, but limits scalability. Figure 2.2 shows the basic structure of a symmetric shared memory multiprocessor.

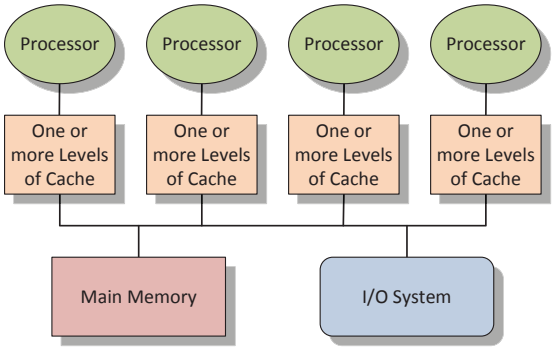


Figure 2.2.: Basic structure of a shared memory multiprocessor.

This type is currently by far the most popular organization for a multi-core architecture [108]. One of the main reasons is that such shared memory systems simplify programming by offering a consistent view of the memory. This consistent view is achieved by the use of cache coherency protocols [13, 103]. The cache coherency protocol ensures that values are updated in different caches, if they are modified. This ensures that all caches consistently return the same value for a given address. The communication overhead of the cache coherency protocol increases with the number of cores. Thus, it is expected that cache coherency can hardly be achieved in architectures containing dozens of cores. However, Intel managed to implement cache coherency on its Xeon Phi Many Integrated Core architecture with 32-72 cores [53] to keep programming as easy as possible.

2.2.2. Distributed Memory Architecture

Distributed memory architectures overcome the bandwidth limitations of shared memory multiprocessor systems. This is achieved by equipping each core with a separate memory and optional dedicated I/O components. Figure 2.3 shows the basic architecture of such a distributed memory multiprocessor. Due to the separate memory for each core or node, the memory bandwidth must not be shared. Since the memory bandwidth in such a system grows linearly with the number of cores, distributed memory architectures provide a good scalability. The larger number of processors also increases the need for a high-bandwidth interconnect. Networks on chip can be used to fulfill these increased bandwidth requirements, as detailed in section 2.4.

"*Distributing the memory among the nodes has two major benefits. First, it is a cost-effective way to scale the memory bandwidth if most of the accesses are to the local memory in the node. Second, it reduces the latency for accesses to the local memory.*" [108]

MPI is widely used in distributed memory architectures as communication concept [235]. By means of `send` and `receive` commands, the instances of a parallel program can communicate with each other. MPI is considered to be more efficient if the receiving buffers are implemented in the target node. This requirement can easily be fulfilled with *non-uniform memory access (NUMA)* architectures; introduced in the next section.

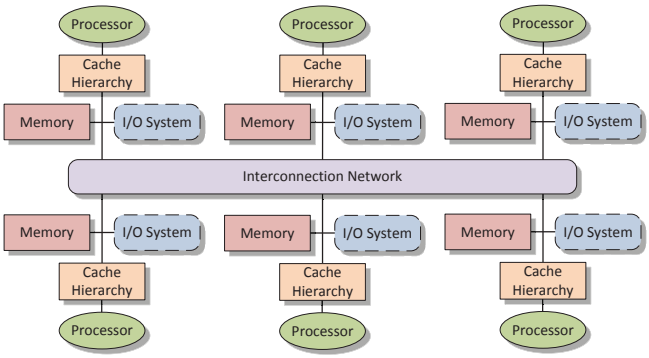


Figure 2.3.: Basic architecture of a distributed memory multiprocessor with an interconnection network that connects all the nodes.

2.2.2.1. Non-Uniform Memory Access

A NUMA architecture is a distributed memory architecture that enables direct access to all memories of the system by each node, but with varying latencies

and bandwidths. In order to enable this, the architecture must have a common address space. Such architectures are referred to as distributed shared memory architectures. In contrast to distributed non-shared memory architectures, NUMA can enable *single program, multiple data (SPMD)* programming style for distributed memory systems. The fact that NUMA allows direct access to each memory simplifies the programming of these architectures. The PGAS parallel programming model uses these advantages. PGAS is used in modern programming languages, including *Co-array Fortran* [187] or *X10* [44].

When a network on chip is used as a communication infrastructure it must implement access to the memory of all the nodes in order to enable NUMA. The hardware implementation within a NoC mainly affects the *network interface (NI)* as described in section 2.4.1.1.

2.2.3. Hybrid Distributed Shared Memory Architecture

In the *high-performance computing (HPC)* domain, mixtures of distributed- and shared memory systems have a long and successful history [17]. Today, multi-core shared memory architectures with a common main memory are used for individual nodes. These nodes are connected with a multidimensional-network, creating a distributed memory architecture on a large scale. Thus, the advantages of easy programmability and cache coherency can be applied within individual nodes, while ensuring scalability of the entire system.

2.2.4. Distributed Shared Memory Architecture

Hybrid *distributed shared memory (DSM)* systems, discussed in the previous paragraph, do not enable direct memory access between different nodes of the architecture. In contrast, there exist (non-hybrid) distributed shared memory systems that combine the advantages of distributed memory and shared memory systems. Such architectures provide a virtual address space shared among the nodes or cores of the architecture. The memory is distributed over the architecture and shared locally between neighboring cores. Depending on the global communication infrastructure, a distributed shared memory system can have a NUMA characteristic. DSM enables ease of programming, reduced hardware complexity and portability. (1) Well-established shared memory programming paradigms can be used. (2) The physically distributed memory can be implemented with low implementation cost. (3) Scalability is ensured due to the absence of hardware

bottlenecks[2] resulting from globally shared resources [206]. Because of these advantages, it is a desirable concept for future many-core architectures.

2

2.2.5. Existing Many-Core Architectures

Selected many-core architectures are introduced in the following sections. The selection is limited to architectures that are complete and have been integrated on silicon. All of them use a network on chip as on-chip interconnect. The terminology used for the characterization of the NoCs of these architectures is described in detail in section 2.4.

The presence of NoC-based many-core architectures should serve as a motivation for NoC research; addressed within this work.

2.2.5.1. Tilera

The first commercial NoC-based many-core architecture that was available on silicon is the TILE64 architecture [18] from *Tilera*; first introduced in 2007. The company Tilera can be seen as a spin-out of the Massachusetts Institute of Technology, as a result of their earlier multi-core research on the Raw architecture [242] and compiler [2].

TILE64 The TILE64 MPSoC consists of 64 tiles[3] arranged as an 8x8 mesh, as shown in figure 2.4. Each tile contains a three-way issue, scalar VLIW processor with three functional units per core. Additionally, each node contains a cache hierarchy consisting of an L1 and an L2 cache, as well as a router (named switch). A 2D *direct memory access (DMA)* engine is located in each tile. This engine supports block-copy functions between the main memory and the caches or between two caches. Due to the mechanisms for direct data exchange between caches and memories, the architecture can be regarded as distributed memory system. However, the four *double data rate (DDR)* controllers located at two boundaries of the 8x8 array provide the impression of a shared memory architecture. In addition to the *memory controllers (MCs)* different I/O interfaces, such as PCI-e, XAUI and RGMII supply over 40 Gb/s of I/O bandwidth

A special feature of the TILE64 architecture is its interconnection network. It consists of five independent 2D mesh networks [263], each of them supporting a distinct function: The *static network (STN)*, the *tile dynamic network (TDN)*, the *user*

[2] This statement is only valid when assuming that the access to the different memories is fairly distributed. This must be ensured by the software memory management.

[3] The name "tiled architecture" is derived from the regular arrangement of the processor nodes or cores, which have a tile-like structure.

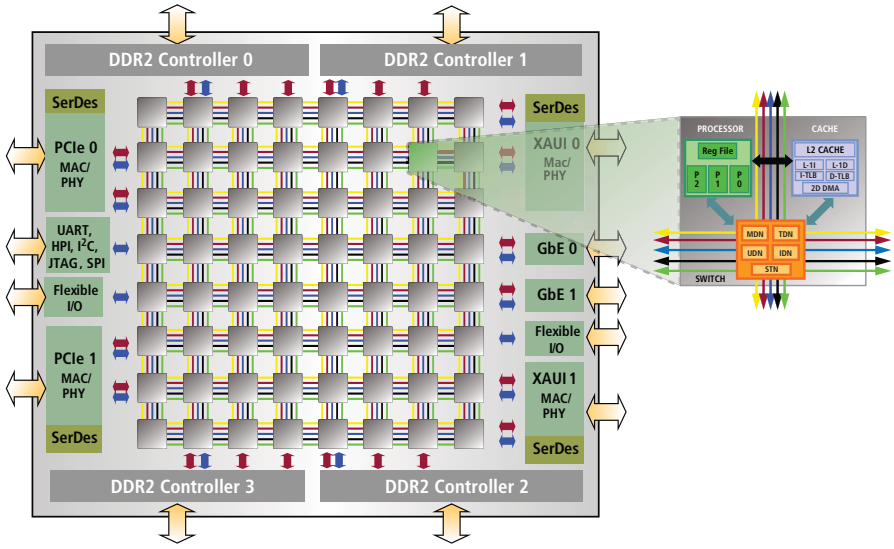


Figure 2.4.: Block diagram and tile architecture of the TILE64 processor [250].

dynamic network (UDN), the *memory dynamic network (MDN)* and the *input/output dynamic network (IDN)*. The STN is a software-driven and software-routed network for low-latency scalar communication between the tiles. The MDN and TDN are dynamically routed networks that are part of the memory subsystem. The UDN and IDN are software-driven, dynamically routed networks for message-oriented communication. In [18] it is stated that *"All networks have single-cycle hop latency from tile to tile"*, which leads to the assumption that they must be circuit switched. The aspect that several networks are used for different communication types is a unique characteristic of the TILE64 architecture. The total network supports a bisection bandwidth (see section 2.4.2) of 240 GB/s.

TILE-Gx100 TILE-Gx is the successor of the TILE64 architecture shipped from the year 2010. TILE-Gx100 [213] is an incarnation of this architecture with 100 tiles arranged in a 10x10 array. It uses a 64 bit processor and DDR3 memory controllers. The bandwidth of the network on chip, which is now named iMesh, was increased compared to TILE64. It has an aggregated bandwidth of 200 Tb/s.

2.2.5.2. Single-chip Cloud Computer

The *Single-chip Cloud Computer (SCC)* is an experimental processor design that originates from Intel's multi-core research. As part of the Tera-scale Computing Research Program of Intel, hardware and software approaches are being studied that consist of a large number of computing units. The first generation of processors of this initiative was an 80-core design [257]. The second generation is the Single-chip Cloud Computer [113].

The SCC is also a tiled architecture, consisting of 24 homogenous nodes or tiles. A block diagram of the architecture is shown in figure 2.5. Each tile consists of two IA-32 cores, where each core is connected to its own L1 and L2 cache. Cache coherency is not implemented in the entire architecture. Furthermore, each tile has a *message passing buffer (MPB)* that is used for direct communication between the tiles. The L2 caches of both cores and the MPB are connected to a router, which is part of the 2D mesh NoC of the SCC.

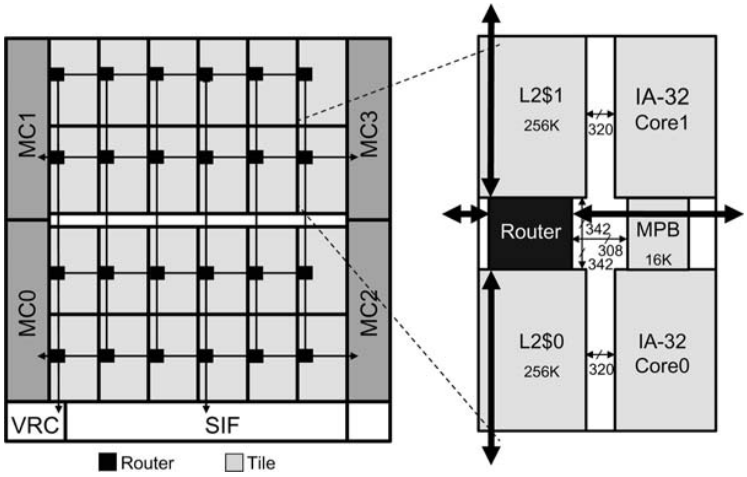


Figure 2.5.: Block diagram and tile architecture of the 48 core SCC [113].

The SCC is no classical distributed memory architecture, but also no pure shared memory architecture. With its MPBs, which are shared between two cores, it can be considered as a distributed shared memory system. However, taking into account the four memory controllers at the borders of the architecture, it looks like a shared memory organization. The memory that is connected through the four DDR3 MCs is shared between all tiles of the architecture. Thus, the SCC can also be considered as a hybrid distributed shared memory architecture with noticeable differences to the HPC architectures introduced in section 2.2.3.

The SCC dispose of a fine-grained power management. It consists of 8 *voltage islands (VIs)* and 28 *frequency islands (FIs)*, where two VIs supply the NoC and periphery with the remaining 6 voltage islands being divided among the cores. Consequently, four tiles form a voltage island. Each tile has its own FI. Software-based power management protocols use the voltage and frequency islands through *dynamic voltage and frequency scaling (DVFS)*.

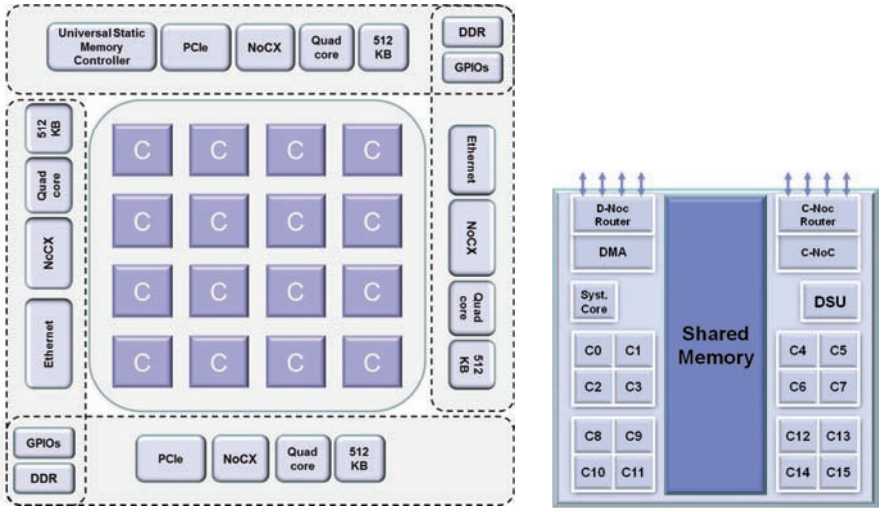
The network on chip of the SCC is a packet switching network consisting of 6x4 routers arranged in a 2D mesh topology. The routers are connected by two uni-directional 144 bit links[4] operating at a clock frequency of 2 GHz. Two message classes and eight *virtual channels (VCs)* ensure deadlock-free routing and maximize bandwidth utilization. Dimension-ordered XY routing is used to eliminate network deadlocks. Individual links offer a bandwidth of 64 GB/s, enabling the total network to support 256 GB/s of bisection bandwidth [113].

2.2.5.3. MPPA Architecture

The Kalray *multi-purpose processor architecture (MPPA)* architecture is another available single-chip, many-core processor. The current version of this architecture is the MPPA-256 which integrates 256 user cores and 32 system cores in 28 nm CMOS technology. These cores are distributed across 16 compute tiles. Figure 2.6(a) gives an overview on the entire MPPA-256 architecture. Four I/O subsystems, one on each side of the tile array, enable communication with the peripherals and with the DDR3 memory. The detailed structure of a compute tile is provided in figure 2.6(b). All cores of the tile have a VLIW architecture with 16 cores (*C0-C15*) being dedicated to application code. The 17th core (*Syst. Core*) is reserved for the system. It is distinguished by its privileged connections to the NoC interfaces through event lines and interrupts [127]. Each core has a private 2-way associative instruction and data cache. In addition to the cores, each compute tile includes a DMA unit and a *debug support unit (DSU)*. A shared memory is located in each tile and can be accessed directly by all cores. Consequently, the MPPA architecture can be classified as a distributed shared memory architecture, whereas the external DDR memories represent the shared memory.

The *D-NoC* router and the *C-NoC* router, shown in figure 2.6(b), connect each tile to the two on-chip networks of the MPPA architecture. Both networks have a 2D-wrapped-around torus structure and use wormhole packet switching. The D-NoC is dedicated to high bandwidth data transfers and supports quality of service. It is used to transfer data between tiles or between a tile and the external DDR memory. The C-NoC is dedicated to control purposes. It is used for flow control,

[4] The two uni-directional links of each SCC router port form a full-duplex connection between each router pair.



(a) Global architecture (b) Compute cluster tile

Figure 2.6.: (a) Block diagram of the MPPA-256 architecture [127] and (b) the internal structure of a compute tile (C) [63].

power management and application software messages. According to [63] both networks support unicast and multicast communication.

2.2.5.4. Morpheus

In contrast to the previous discussed architectures, MORPHEUS is a heterogeneous system [247]. It was fabricated during 2010 using an ST 90 nm technology [259].

The platform combines three different array-based reconfigurable architectures, referred to as *heterogeneous reconfigurable engines (HREs)*, and an ARM processor to form a SoC. The different HRE-types target different flavors of signal processing: *Pact XPP-III* is a coarse-grained reconfigurable stream processor. It consist of an array computational elements communicating through configurable data channels. *DREAM* is a reconfigurable processor composed of a RISC core and a mid-grain reconfigurable data path. The third HRE-type is named *FlexEOS*. It is an embedded *field programmable gate array (FPGA)*.

Figure 2.7 gives an overview on the MORPHEUS SoC. The *advanced RISC machines (ARM)* processor serves as a host processor for the entire platform. The implemented control scheme is based on the MOLEN polymorphic processor

paradigm; introduced in [258]. The host processor controls the work flow of the HREs through register mapped interfaces.

The communication infrastructure of MORPHEUS consists of two systems: An *advanced microcontroller bus architecture (AMBA)* AHB bus is used to access all HRE exchange registers and system peripherals. It is also used to configure the second communication system, the so-called Spidergon network on chip [50]. Spidergon is a packet-based communication infrastructure, adopting wormhole switching. This NoC is dimensioned for the bandwidth-intensive communication between the HREs and has a link size of 64 bit. The Spidergon NoC also connects the computational resources of the SoC with the external global shared memory.

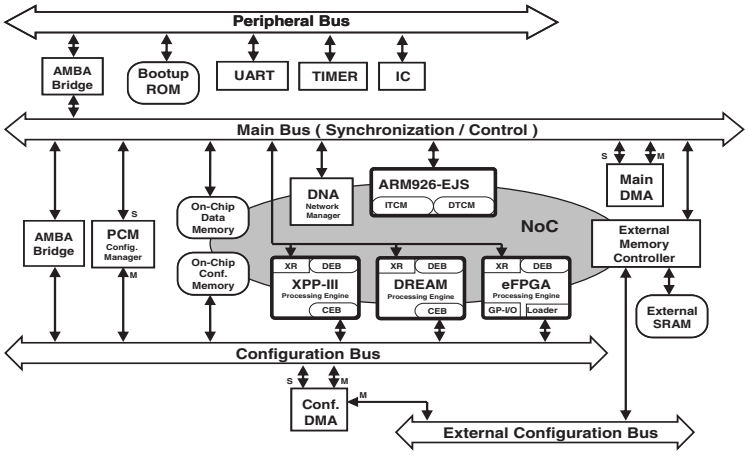


Figure 2.7.: Block diagram of the heterogeneous reconfigurable MORPHEUS architecture [259].

2.3. Rudimentary On-Chip Communication

Systems on a chip [133] and especially MPSoCs [268] have been available for more than one decade, as described earlier. The number of components that are integrated on a piece of silicon is increasing steadily. This leads to a rise in communication on the chip. Depending on the number of components and their communication requirements, different communication schemes are preferred. Typically various communication schemes are combined to deploy a many-core architecture, such as the architectures presented previously. In the following sections, the most important strategies for on-chip communication will be discussed.

2.3.1. Direct Connections

The simplest and probably most common type of connection is a direct connection or point-to-point connection. As the name suggests, this connection type is limited to two participants. They are linked using a direct parallel or serial connection. Point-to-point connections have the advantage of easy implementation with low overhead. They can be pipelined easily depending on the frequency requirements of the connection and the distance between the communication partners. The bandwidth can be matched to the requirements of the communication partner by adjusting the number of wires or the clock frequency. Due to their numerous advantages, point-to-point connections are used whenever possible. They are also used as sub-systems to implement more complex communication schemes, such as bus systems or networks on chip; presented later.

2.3.2. Bus Systems

Compared to direct connections, bus systems enable a flexible communication between an arbitrary number of communication partners. Therefore, individual components are typically connected via a standard interface. The bus interface defines the physical interface as well as the protocol for communication. The advantage of standardization is that the individual components of a SoC can be easily connected [154]; accelerating the design and implementation of such systems.

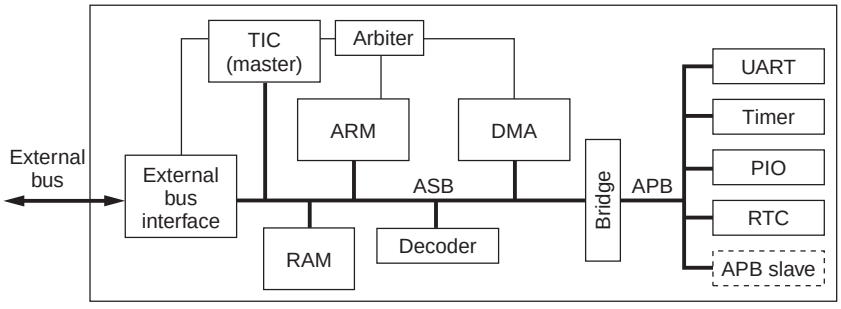


Figure 2.8.: AMBA bus-based system on a chip example with an ARM microcontroller [79].

Widely used representatives for on-chip bus systems are AMBA [79], developed by ARM, and the CoreConnect [154] bus system used by IBM. Figure 2.8 shows a typical configuration of an AMBA-based bus system. It consists of a fast *advanced*

system bus (ASB)[5] with high bandwidth and a slower *advanced peripheral bus (APB)*; with both buses being connected via a bridge. Frequently used components with high bandwidth requirements, such as the processor (ARM), memory (RAM), DMA unit or special accelerators (Decoder), are attached to the faster ASB. Components with low bandwidth requirements, such as a timer, a PIO or *universal asynchronous receiver transmitter (UART)* interfaces, are attached to the APB. The use of different buses and their decoupling enables to save resources and thus energy [211]; relevant in battery-powered systems.

2.3.2.1. Components

The following section describes the components of a bus system in more detail. Given the relatively large diversity between different bus systems, the following description is based on the AMBA bus system.

Arbiter The arbiter manages the access to the shared bus. It can either be implemented in a distributed way by the bus participants or as a separate module with a connection to all devices that are able to request bus access. The AMBA bus system uses a centralized arbitration scheme to implement the bus without tri-state logic[6]. A bus arbiter is responsible for solving conflicts or collisions[7] on the bus. For this purpose, the arbiter prioritizes the access to the bus. The arbiter grants the bus access to the component that momentarily has the highest priority. The granted participant is then allowed to put data on the bus. Once the access is finished, a new arbitration decision takes place.

Bus Interface The bus interface connects the bus devices with the actual bus system. It must carry out the physical interface of the bus and the bus protocol. In many cases, the bus interface of a component only implements a subset of the bus features. Special features, such as split transactions or burst transfers, might be omitted.

The AMBA *advanced high-performance bus (AHB)* distinguishes between two fundamental types of bus interfaces: The *master interface* is used for components that have the capability to request bus access. These components are usually named bus-masters. Once the access is granted, the bus can be used by the master to read or write data to another bus participant. The *slave interface* is used for components that receive read requests and write requests from bus-masters and

[5] In recent systems, the AHB or AXI replaces the ASB.

[6] Tri-state logic allows an output port to have a high impedance state in addition to the 0 and 1 logic levels.

[7] Collisions can also be prevented by an appropriate arbitration at design-time.

process them accordingly. Such components are commonly named slaves. In case of read requests, a slave interface is responsible for blocking the bus or issuing a split transaction until the requested data can be returned to the requesting bus-master. A component can either have only a master or a slave interface or it is even equipped with both types of interfaces, such as the network adapter presented in section 3.2.2.1.

Bridge A bridge is used to connect different types of bus systems or to connect two bus systems of the same type. If different bus systems are connected by a bridge, it is responsible for protocol as well as signal conversion. Figure 2.8 shows a bridge that is used to connect the AMBA ASB and the slower APB, which has a different protocol. Since only slave-devices are connected through the APB, it is only equipped with a slave-interface at the AHB side. In contrast, bidirectional bridges are used to connect two identical bus systems. Bidirectional bridges have a master and a slave interface at both ends. As discussed in section 2.3.2.3, such bridges can be used for bus splitting.

2.3.2.2. Shared Bus

A system where all participants use the same communication medium is named a shared bus; which is the most common type of bus. A shared bus has a very low delay once the bus access is granted. However, the following formula is used to calculate the worst case access delay of a component t_{access} for an access to a bus with n communication partners:

$$t_{access} = t_{comp,max} \cdot n \tag{2.1}$$

In equation 2.1 it is assumed that each component has a maximum bus access time $t_{comp,max}$. Furthermore, it is assumed that all components have fair access to the bus[8]. Using a shared bus, all communication partners n share the communication bandwidth b_{bus}. Taking the same assumption as for equation 2.1, the bandwidth b_{comp} of a component can be calculated as follows:

$$b_{comp} = \frac{b_{bus}}{n} \tag{2.2}$$

Equation 2.1 and 2.2 show that the performance of the communication infrastructure depends on the number n of the components attached to the bus. With an increasing number of participants, the efficiency of the bus decreases.

[8] A fair access to the bus can be enabled by realizing a round-robin or TDM arbitration within the bus arbiter).

2.3.2.3. Bus Splitting and Segmentation

Bus splitting or segmented buses can be used to cope with larger numbers of components, but also for other reasons, such as deadlock avoidance. In a segmented bus system different shared buses are connected by bridges to form larger, more complex systems. The system in figure 2.8 uses two buses to decouple faster components with high bandwidth requirements from peripherals.

The bridge is used to decouple the different address ranges of the single buses. As such, it is possible to communicate in parallel within different bus segments when the communicating participants are attached to the same segment. This increases the effective bandwidth and the access latency of a segmented bus system compared to a single bus, because the number of components n attached to each bus is reduced compared to a non-segmented bus system[9]. However, an increased latency can arise when communicating over several segments because of the separate access latency for each segment; exhibited in equation 2.1. The concept of segmentation is also applied in NoCs, introduced in section 2.4. Using a NoC, the bandwidth grows linearly with the number of component, because each new component is adding one or multiple new segments to the communication system.

2.3.2.4. Crossbars and Bus Matrices

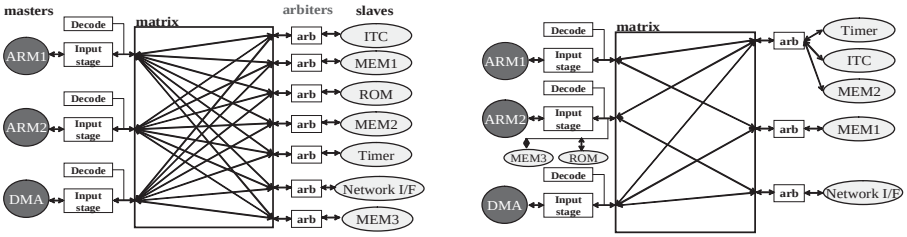
Bus matrices can be considered as an evolution of segmented buses. In a bus matrix or crossbar, the communicating components are directly connected via a multiplexer network. Several disjoint pairs can communicate in parallel with each other. Arbitration is only needed if two components (masters) have to communicate simultaneously with the same endpoint (slave). Therefore, each endpoint of a bus matrix has a separate arbiter. If no arbitration is required, all masters are able to communicate with the full bandwidth of the bus system. The latencies for bus access can typically be kept very low, if a bus matrix is used.

Figure 2.9 shows two different types of bus matrices. In figure 2.9(a) a fully connected crossbar is shown. All masters can communicate directly with all the slaves. The implementation cost c of a fully connected crossbar with i inputs (masters) and o outputs (slaves) can be approximated as follows:

$$c \sim i \cdot o \tag{2.3}$$

Equation 2.3 shows that the implementation cost increase quadratic with the number of masters and slaves. Therefore, the number of connected components

[9] This statement is only valid if the bus is segmented properly. A good segmentation reduces the number of cycles where the bridge must be crossed.



(a) Full bus matrix

(b) Partial bus matrix

Figure 2.9.: Two different bus matrix types – (a) fully connected bus matrix connecting all components and (b) partially connected matrix connecting only a subset of components [195].

should be limited in order to keep implementation costs moderate. Due to this limitation, the scalability of bus matrices is limited.

One way to reduce the implementation cost is the realization of a partial bus matrix. Figure 2.9(b) shows a partial bus matrix that reduces the number of connections between masters and slaves by excluding some paths between masters and slaves. This optimization is based on the assumption that each master only communicates with a subset of slaves.

The concept of a crossbar or bus matrix is reused in NoCs. Each router of a NoC can be considered to be a crossbar. The number of ports, and therefore the complexity, is limited by the topology of the NoC.

2.4. Networks on Chip

As already indicated in the previous sections, networks on chip are inspired by the evolution of bus systems. Further sources of inspiration are the internet [20] and HPC systems. Both, the internet and an HPC system consist of a very large number of end nodes, which are connected by routers and switches to form a scalable network. NoCs originated from the trend that the number of components to be connected to a SoC increased continuously. Especially for large multi-core and many-core architectures, as introduced in section 2.2, bus-based communication is not sufficient due to the limited scalability with respect to the large number of components of such systems.

Many different aspects must be considered when realizing a network on chip. The choice of the switching scheme, topology, routing method and many other parameters have an influence on the following: power consumption, implementa-

tion complexity, performance, quality of service, management and scalability of the network. The most important implementation options and aspects of NoCs will be discussed in the following sections.

2.4.1. Components

A NoC generally consists of routers (referred to as switches), that are connected by links to form a network according to the used topology. The individual nodes, which are often referred to as tiles, are attached via network adapters.

2.4.1.1. Network Adapter

The *network adapter (NA)* or *network interface (NI)* is used to attach system components to the NoC communication infrastructure. The NA can be seen as a wrapper providing a view of the network, consistent with the I/O protocol of the component or subsystem attached to it. Thus, the NA is a protocol converter that maps the I/O protocol of the processing node or tile into the protocol used within the NoC. Essentially, each NA has two ports or interfaces: one *NoC interface*, which is used to attach the NoC, and one *tile interface*, which is used to attach the components of the tile to the NoC. Figure 2.10 shows the basic structure of an NA with its two interfaces.

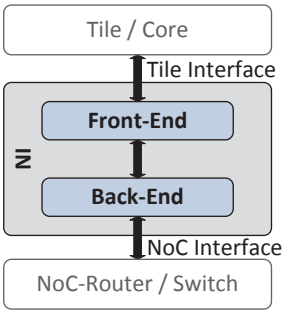


Figure 2.10.: Simplified block diagram of a network adapter with an interface to the tile and the NoC.

According to [175], a network adapter consists of a *front-end* and a *back-end*. The front-end implements a standardized point-to-point or bus protocol allowing core reuse across several platforms. A point-to-point interface is often used to attach a single core. In other systems where more complex tiles are attached to the NoC, bus systems are used tile-internally. The use of existing bus protocols, such as

AMBA [79], CoreConnect [154] or VCI [55], enables backward compatibility. This simplifies the reuse of the NoC and other SoC components [132]. Depending on the supported features of the NoC, the NA might provide different software interfaces:

- **Memory communication**: Memories of other tiles or a main memory might be mapped into the address range of the NA. The NA enables transparent load/store of data and instruction fetching, when accessing tile-external memory.
- **Message passing communication**: Message passing support can be implemented within the network adapter. Intel's Single-chip Cloud Computer supports message passing by dedicated message buffers that can be used for direct communication between the tiles. Message generation and buffer management is performed by the network interface at the transmit side and at the receive side respectively.
- **Network management**: NoCs might provide special features, such as quality of service support by exclusive resource allocation, a monitoring infrastructure or programmable adaptivity. If these features can be managed by the software, the NA must provide a management interface that is typically constituted by memory mapped registers located within the NA.

Using the ISO/OSI reference model [256], the front-end can be considered as the implementation of the session layer. The front-end uses the services offered by the NoC. These services are part of the implementation of the transport, network, data-link and physical layer within the back-end of the NA. Thus, the back-end of the NA must perform the packetization and routing related functions. If source routing is used (see section 2.4.5.1), routing tables within the NA might be used to determine the route of a packet or connection. When distributed routing is used (see section 2.4.5.2), the NA only has to encode information about the destination node in the packet. The network and data-link layer inside the network adapter ensures reliability of the communication flows. Error-detection and recovery techniques [184] can be used to enable fault tolerance. Flow control mechanisms are used to regulate upstream data transmission and to prevent buffer overflows and back pressure. The physical layer of the NA might do frequency conversion in case of multiple clock domains [189] or support special coding schemes for power saving [194].

2.4.1.2. Router

A router (also named switch) is responsible for forwarding the data from an input port to an output port. This procedure is repeated in different routers until the data reach the destination node. The functioning of a router is similar to that of a

bus matrix described in section 2.3.2.4. One of the main components, known as crossbar or multiplexer network, connects the inputs and outputs of the router.

Depending on the complexity of the NoC, a router implements up to four layers of the ISO/OSI reference model [256]: The transport and network layer, that enable transparent transfer of data between end nodes, the data-link layer, that provides the services for router-to-router communication (e.g. flow control) and the physical layer representing the realization of inter-router communication.

Figure 2.11 shows the structure of an elementary packet switching NoC router. Packet switching necessitates buffering, as described in detail in section 2.4.3.2. Consequently, buffers are typically located at each input port of the packet switching router to temporarily store incoming data until they can be forwarded. The router shown in figure 2.11 implements a distributed routing scheme, where each router takes an independent routing decision. Based on this routing decision the output port can be allocated to forward the data. The data transmission is subsequently managed by the *transmission control unit*. The arbitration between multiple inputs that want to forward data through the same output, is performed by the *reservation table*. The *crossbar* comprises the multiplexer network that connects the inputs to the output ports.

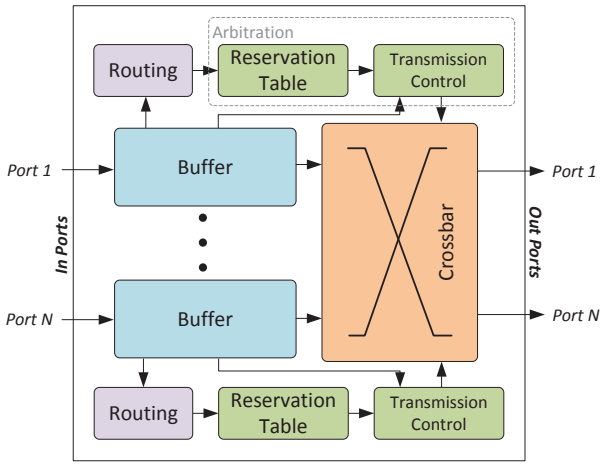


Figure 2.11.: Block diagram of a packet switching router.

The implementation costs of a router depend on the number of input and output ports. Similar to the bus crossbar discussed in section 2.3.2.4, the complexity grows approximately quadratic with the number of router ports.

Additional details about the functional principles of a packet switching router are provided in section 5.1.

2.4.1.3. Links

The link is the physical connection between two routers or between a router and a network adapter. It is part of the physical layer of the NoC. In case of a 2D silicon implementation, the links are typically created by parallel wires. "Due to the relatively small wire pitch in silicon technologies, and to the availability of several layers for wiring, physical wires are a cheap commodity for networks." [175] Consequently, data and control signals are typically separated to have a cleaner and better performing design. Because of the low overhead for wiring, the links are usually implemented bidirectional with separate wires. However, for particular topologies, global wires may be required; as described in detail in section 2.4.2. For such links, repeaters in the form of buffers or registers are needed to ensure scalability.

In case of a 3D integration, link design becomes more complex due to the restrictions of *through-silicon vias (TSVs)* [204]. The additional requirements resulting from the characteristics of TSVs lead to more complex link designs which must cope with faulty TSVs [165] or limitations with respect to their number [164]. In addition, there are other approaches for on-chip networks that have a large impact on the link design. An example are wireless[10] [86] and optical/photonic[11] [226] networks on chip.

2.4.2. Topology

The topology of a network has a noticeable impact on implementation cost, performance and floorplanning of a NoC-based architecture.

Implementation cost of each router is affected by the topology because it dictates the number of router ports. This in turn affects the implementation cost of the crossbar according to equation 2.3.

The performance of a network is affected by the topology because it dictates the bandwidth. As a measure for the bandwidth of a network, the *bisection bandwidth* or bisection width is widely accepted. The bisection bandwidth is defined as the bandwidth between two parts of a network, which is segmented into two (roughly) equal parts [108]. Clark D. Thomborson could show that the bisection width of a network has a noticeable impact on the performance of distributed applications, such as parallel sorting [249] or a *discrete Fourier transform (DFT)* [248].

[10] Wireless networks use miniaturized on-chip antennas for communication between routers or nodes [86].

[11] Optical networks might use photonic switching elements controlled by electronic circuit switching routers [226].

According to [248], the bisection width is defined relative to the number of cores or nodes N connected by the network[12].

Compared to off-chip networks, floorplanning is another important aspect that must be taken into account when selecting a topology for an on-chip network. In general, floorplanning of tiled architectures with regular topologies is easier [107] and can improve the circuit performance due to the short links. However, irregular topologies might be beneficial for application-specific designs. A floorplaning methodology for irregular topologies was presented in [183].

In the following the most relevant network topologies will be introduced and discussed. For each topology, the implementation costs, the performance and the suitability of an *application-specific integrated circuit (ASIC)* implementation will be evaluated. An overview of different network topologies is provided in figure 2.12.

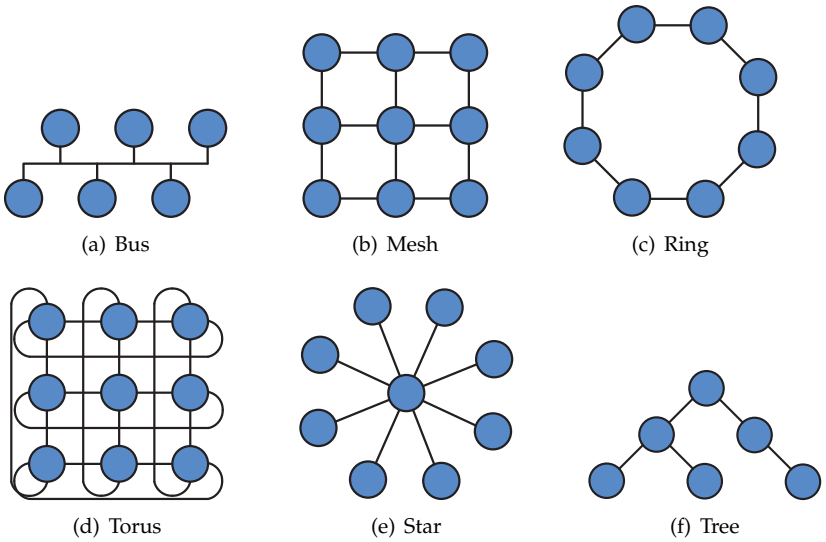


Figure 2.12.: Six different common network topologies.

Bus The topology shown in figure 2.12(a) represents a shared bus. It has low implementation cost; as explained in section 2.3.2.2. An efficient floorplan of a bus-based system is limited to a small number of components due to the long wires required in larger systems. The bisection width for N nodes is 1. Thus, it is independent of the number of nodes, limiting the scalability of the bus.

[12] In the following paragraph it is assumed that each link has a bandwidth or capacity of 1.

Mesh The two-dimensional mesh, shown in figure 2.12(b), is a very popular NoC topology [175]. It has good floorplanning capabilities due to its regularity. All links have the same length; which is limited. The number of ports per router is independent of the total number of nodes; enabling scalability. The bisection width of a 2D mesh for an architecture with N nodes is $\sqrt{N}$. The diameter[13] of a 2D mesh is $2(\sqrt{N}-1)$. The 2D mesh is used in many existing NoCs, such as the QNoC [29] and Nostrum [177, 178].

Ring The ring is a topology that is widely used in off-chip networks. A ring topology is shown in figure 2.12(c) and can be seen as a 1D torus (multi-dimensional tori are explained later). The number of ports per node is 2, independent of the number of nodes connected by the ring. The ring can be implemented in a way that all links have the same (short) length, enabling efficient floorplanning. The bisection width of a ring topology is 2; therefore providing limited scalability with respect to the bisection bandwidth. The Proteo [230] architecture uses a ring topology for on-chip communication. Intel's commercial Xeon Phi architecture [53] also uses a ring topology for its main interconnect.

Torus A torus can be seen as the extension of a ring to multiple dimensions. It adds wrap-around links to the 2D mesh; as shown in figure 2.12(d). In order to reduce the length of the wrap-around links, the torus can be folded. When folding, the distance between two connected nodes can be limited to two, independent of the number N of nodes. The regularity of a folded torus simplifies floorplaning. Compared to the 2D mesh, a 2D torus has a double bisection width of $2\sqrt{N}$ but only slightly higher implementation cost. Its diameter, which is relevant for the latency, is $\sqrt{N}$. The NoC presented in [61] uses a torus topology.

Star Using a star topology, as shown in figure 2.12(e), all nodes are connected to a common node. This common node is often referred to as a super-node. It can be implemented as a crossbar (cf. section 2.3.2.4). The crossbar enables low latency between all nodes but limits the scalability of the star topology due to the implementation cost that grow quadratic with the number of nodes. The floorplaning of the star topology can result in long links for larger numbers of nodes. However, the bisection width is $N/2$ according to [248] and scales well due to the use of a centralized crossbar in the super-node.

Tree A tree topology is typically used in *local area networks (LANs)*. According to [175], in a k-ary tree, each router has k children. The network adapters are

[13] The diameter is a good measure for the latency of the network.

attached only to the leaves of the tree. A schematic diagram of a 2-ary tree is provided in figure 2.12(f). The floorplaning capabilities of tree topologies depend on the number and type of nodes. A good placement might be found for small heterogeneous architectures where the nodes have different sizes. When scaling the size of an architecture, floorplanning of trees becomes difficult. The bisection width of a tree is 1, due to concentration of all traffic at the root node. In order to increase the bisection bandwidth the bandwidth towards the root node can be increased as proposed in [156]. Such a fat tree or folded butterfly has a large bisection width. The drawback is the high cost of implementation. A tree-based on-chip interconnect is presented in [101].

2.4.2.1. 3D Topologies

The previous discussion only focused on topologies, suitable for 2D integration. However, 3D integration is a promising technology for large future systems [139]. Therefore, 3D topologies for networks on chip will be briefly discussed now. In [72] it was shown that tree topologies are suitable for 3D integration although they have a limited scalability. A tree topology can profit from a decreased average distance between the nodes in case of 3D integration, resulting in better performance and significant gain in power dissipation and area overhead. However, 3D integration targets large SoCs. That is why scalable regular topologies are most suitable for 3D integration. For a 3D mesh-based topology significant gains in terms of throughput, latency and power dissipation have been demonstrated in [72]. However, this is achieved on the cost of additional area, which results from the higher number of ports of a 3D mesh or torus router. Taking equation 2.3, the implementation cost c of a 3D and a 2D mesh or torus router can be estimated as follows[14]:

$$c_{3D} \sim 7 \cdot 6 = 42 > c_{2D} \sim 5 \cdot 4 = 20 \tag{2.4}$$

Since the implementation costs of the other router components (e.g. buffer, routing units, etc.) grows only linearly with the number of ports, the total relative implementation overhead of a 3D router should be lower than the relative overhead of the crossbar. A NoC router for 3D architectures is presented in [135]. It uses a partially-connected 3D crossbar structure to reduce the implementation cost of the crossbar.

The throughput gain of a 3D torus or mesh compared to the 2D variant results from the higher bisection width. It is $(\sqrt[3]{N})^2$ for a 3D mesh and $2(\sqrt[3]{N})^2$ for a 3D torus. The improvement in terms of latency compared to a 2D integration results

[14] For the overhead estimation it is assumed that each input is only connected to six of the outputs, since the data do not enter and leave the router through the same port.

from the decreased diameter, which is $3(\sqrt[3]{N}-1)$ for the mesh and $3\sqrt[3]{N}/2$ for the torus.

In [124] a 3D mesh and torus NoC have been compared in terms of latency and throughput. The presented results approve the theoretical advantages of the torus compared to the mesh; resulting from the higher bisection width and decreased diameter. In addition the authors proposed a deadlock-free routing mechanism for 3D tori.

Due to the fact that a 3D mesh or torus is just an extension in the dimension, their scalability and floorplaning capabilities are comparable to the 2D variant.

2.4.3. Switching Schemes

The previous sections showed that the choice of the topology can have a noticeable impact on scalability, performance and implementation cost. The switching technique is another important aspect with respect to implementation cost, performance, flexibility and QoS requirements of the network. It defines the data flow through the routers and the granularity of data transfers. The minimum granularity of a data transfer is typically the link size, measured in bits. However, an exception is a network with *spatial division multiplexing (SDM)* techniques [157] where the physical link is divided into sub-links. The unit of data transferred in a single cycle on a link without SDM techniques is known as *phit*. One or multiple phits are combined to one *flow control digit (flit)*. These flits represent the granularity that is used for switching or flow control. Several flits form a packet and multiple packets can be used for a complete message.

There are two basic modes for data transfer within a network: *circuit switching* and *packet switching*. In circuit switching networks, an end-to-end connection is established between sender and receiver before the actual data transmission takes place. In contrast, packet switching networks enable to inject packets without previous connection set-up. Packets constituting a message make their way independently of each other in a packet switching network. Each packet could e.g. use a different route and experience a different delay.

The choice of the switching method has significant impact on resource requirements, power consumption, flexibility, performance and QoS support of the network on chip. The impact of different switching methods on all these parameters will be described now.

2.4.3.1. Circuit Switching

Circuit switching aims to send rather large messages in their entirety from one module to another. Therefore, an end-to-end connection must be established before data transmission. Such an end-to-end connection is a physical path consisting of a series of links and routers. To establish a circuit switching connection in a distributed way, a setup or head flit can be used. It is transmitted from the sender to the receiver, reserving a series of links along the way. The multiplexers and crossbars within the routers are configured during setup to connect the allocated links. If the setup flit arrives at the receiving node without conflicts (occupied links), an acknowledge flit is sent back to the transmitter[15]. Once a connection is established successfully, the data transfer can be performed without any needs for interruption. The connection behaves like a circuit between the two communicating nodes. Figure 2.13(a)(left) shows the setup and data transmission phase of a circuit switching connection over time. Three routers *R1*, *R2* and *R3* are used by the connection. The per-router delay is one cycle in the example provided, due to pipeline registers within the routers[16]. Figure 2.13(a)(right) shows the pipelined circuit switching data transmission. It can be seen that the delay of the header or setup flit is larger than the latency for the actual data transmission. The reason is the additional routing and switching delay required for the transmission of the header and acknowledge flits. After transmission of a message, a tail flit is used to release the circuit switching connection.

Circuit switching networks have a high initial latency for circuit setup, resulting from routing and acknowledgment. This setup phase must be completed before data transmission starts. However, after setup, data transmission is very efficient due to the low latency and the high bandwidth. The low latency results from the absence of buffers and arbitration. Due to the initial circuit setup, buffering and arbitration is not required because it is ensured that all data can be forwarded directly after reception. Avoiding buffers not only reduces the latency, it also lowers the implementation cost and thereby the energy footprint of the router. The drawback of circuit switching networks are their limited flexibility. The number of parallel communication flows is limited by the number of available links. Frequently changing communication partners would lead to a huge overhead and inefficiency due to connection setup. On the other hand, the amount of resources per connection increases with the distance of the communication partners. Considering the increase in setup time with the distance of the communication partners in large architectures, the scalability of circuit switching is limited [175].

[15] If a link is occupied by another circuit switching connection, setup fails and the data transmission fails for now.

[16] Registers in a circuit switching NoC enable scalability. The reason is the limited length of a paths in the network independent of the number of nodes.

Circuit switching is suitable for frequent communication or static transmission scenarios; where the communication partners are fixed for a long period. Consequently, the efficiency of circuit switching depends on the payload size to be transmitted after setup. The initial time for connection setup becomes negligible if the circuit is used for an extended period of time.

SoCBUS was one of the first representatives of a circuit switching network on chip. With respect to the power efficiency, circuit switching NoCs have been shown to be very efficient. A circuit switching CMOS integration of a 2D mesh NoC with 64 nodes and an power efficiency of 363 $Gb/s/W$ was presented in [11]. Packet switched request circuits are used to set up new connections. Wolkotte et al. proposed a reconfigurable circuit switching router design and showed its low power consumption and chip area compared to packet switching [269]. The Æthereal NoC investigated by Goossens et al. [92, 93] combines circuit switching with the virtual channel concept (see section 2.4.3.3) using a TDM scheme for VC scheduling. In Æthereal, circuit switching is combined with a packet switching network that offers best-effort communication.

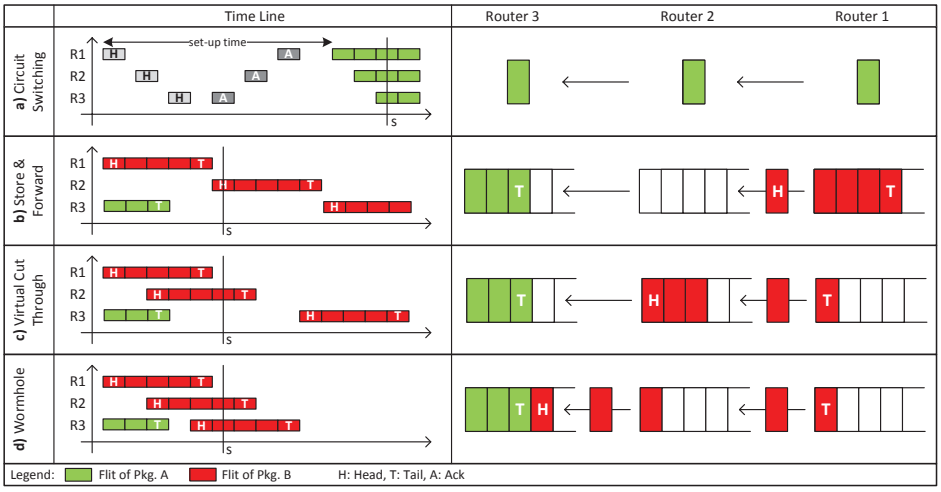


Figure 2.13.: Basic circuit (a) and packet switching techniques (b,c,d). *Right:* Time line of the data transmission of the three routers (*R1-R3*). *Left:* Snapshot of the router buffers (*R1-R3*) at time *s*.

2.4.3.2. Packet Switching

In contrast to circuit switching, packet switching networks do not require a reservation before data transmission takes place. Different packages of a large

message may take different routes through the network; perhaps with a large variance in latency. The absence of a reservation phase enables direct injection of a packet into the network. Thus, a packet switching network can be considered to be more flexible in terms of changing communication partners. Omitting the setup procedure, which could possibly fail, simplifies the usability and reduces the management overhead of a NoC. Best-effort communication, enabled by packet switching, allows to communicate to different partners in parallel without taking resource constraints into account. Nevertheless, the available resources should not be completely ignored when using packet switching. If different transmissions want to use the same link at the same time, a so-called *contention* appears. It requires an arbitration between the packets that are competing for the link usage. The arbiter decides which packet is allowed to use the link directly. All other competing packets must wait and have to be sorted until the link is available again. Therefore, buffers must be built within the routers to store the packets in case of contention. These buffers lead to increased implementation cost and power consumption compared to a circuit switching network.

The comparison of the performance of circuit and packet switching networks on chip is not obvious. Referring to the latency, a circuit switching network is invincible if the connection is already established. However, if data must be transmitted spontaneously, packet switching can beat circuit switching in terms of delay. Using a packet switching NoC the packet can be sent directly, whereas a circuit switching variant would have to wait for a successful connection setup. Referring to the throughput, a packet switching NoC is generally more efficient. The reason is the more flexible link usage of packet switching. Each link is only utilized for the time a packet is transmitted. Subsequently, it can directly be used by other packets waiting in the router buffers. This leads to a very flexible and efficient link utilization resulting in a high general throughput. In contrast, the throughput of a circuit switching network strongly depends on the load of the established connection. Reserved links remain unused if the corresponding connection is idle. To increase the link utilization of both packet and circuit switching networks, virtual channels, introduced in section 2.4.3.3 can be used.

In general, packet switching networks do not support QoS in terms of latency or throughput. However, *virtual cut through (VCT)* switching and wormhole switching can be used to enable end-to-end connections and *guaranteed service (GS)* communication in packet switching networks; as explained in chapter 6.

Store and Forward The *store and forward (SAF)* flow control or switching method enables a simple packet switching implementation. A packet is only forwarded from one router to the next, if there is enough space available in its buffers to store the whole packet. Store and forward has the advantage that a packet transmission between two routers never stalls after it was initiated, because it was

ensured previously that all data are accepted by the receiver. The disadvantage compared to more advanced switching schemes is the increased delay resulting from the fact that a packet is received in its entirety before it is forwarded again. Figure 2.13(b)(left) shows the data transmission of a SAF switching network for three routers: *R1*, *R2* and *R3*. In this scenario, the buffer of router *R3* is occupied by a packet (red). A second packet (green) will be transmitted on the same path from *R1* to *R2*. Figure 2.13(b)(right) illustrates that point in time *s* where the packet is completely stored in the buffer of *R1* and is now forwarded to the empty buffer of *R2*. The buffer of R2 is empty and can store the entire packet (red).

A huge disadvantage of store and forward is the large buffer, which is necessary to store the largest supported packets as a whole. Since the buffer size of NoCs is critical with respect to implementation cost and power consumption [214], SAF is not very common for NoCs. However, the Nostrum NoC [177] implicitly carries out SAF by limiting the packet size to one flit.

Virtual Cut Through In order to reduce the latency compared to store and forward, virtual cut through switching can be used. It forwards the first flit of a packet as soon as space for the entire packet is available in the receiving router. In contrast to SAF, the packet does not have to be received completely before the available flits are forwarded. If there is not enough space available, the whole packet is buffered. Figure 2.13(c)(left) shows the progression of a VCT data transmission of two packets. In contrast to SAF, shown in figure 2.13(b), the red packet can be forwarded earlier from *R1* to *R2*, since it is not required to be received completely by *R1*. A snapshot of the buffers is provided in figure 2.13(c)(right) for time *s*. Since the required buffer size for VCT is identical to SAF, its realization would result in large buffers and implementation cost. One of the few NoCs that apply virtual cut through switching is implemented in the SCC architecture [113].

Wormhole The wormhole switching scheme is an evolution of VCT. It reduces the buffer requirements to one flit. This is achieved by forwarding each flit of a packet directly after reception. For flit transmission in wormhole switching, a packet must not be completely received and the following router does not need to have buffer space available for the entire packet. This enables to reduce the delay compared to store and forward and virtual cut through and leads to an area efficient implementation of a NoC router; due to the low buffer requirements. The disadvantage of wormhole switching is the potential blocking of links by packets that are left strung out over multiple routers. All the links between the header and the tail of the packet are then blocked for other transmissions; potentially leading to contention. Virtual channels, introduced in section 2.4.3.3, assist in reducing the chance of contention by enabling processing and forwarding of multiple packets in parallel. The risk of deadlocks in wormhole switching requires a careful selection

of the routing algorithm; as described in detail in section 2.4.5.5 and section 2.4.5 respectively.

Figure 2.13(d)(left) shows the time line of a data transmission in a wormhole switched network. Compared to VCT, presented in figure 2.13(c)(left), the red packet can be forwarded sooner from router *R2* to *R3* because the buffer of router *R3* (see figure 2.13(d)(right)) must not be empty; like with virtual cut through switching.

According to [175], almost all packet switching NoCs use wormhole switching due to its low buffer requirements and low delay. The Hermes [180] NoC uses wormhole switching in combination with dimension-ordered routing to avoid deadlocks (see section 2.4.5). Intel implemented an 80-Tile architecture in 65 nm using a wormhole packet switching NoC [257]. In Æthereal [214] wormhole packet switching is used for the *best-effort (BE)* router. QNoC [29] uses wormhole switching in combination with different service classes; each with separate buffers in the router. This enables interleaved data transmission of packets that belong to different service classes. The MANGO network on chip [25, 26] and the router design presented in [130] both combine wormhole packet switching with VCs (see section 2.4.3.3) to avoid contentions. All the previous discussed networks use wormhole switching in a regular mesh topology. In contrast, the SPIN micro-network [1] applies wormhole packet switching to a fat tree topology.

2.4.3.3. Virtual Channels

The concept of *virtual channels (VCs)* was introduced by Dally [58] to decouple buffer resources from transmission resources. This decoupling allows active messages to pass blocked messages using network bandwidth that would otherwise be left idling. In general, a predefined number of parallel channels (VCs) is built at each port of the router. These virtual channels share the physical link by using a scheduling strategy that controls the link access. After transmission, the data are split up into VCs at the receiving router. The problem of unused bandwidth exists in both, packet and circuit switching networks. Therefore, the concept of virtual channels is applied to both switching strategies. The improved link utilization increases the performance (throughput and latency) of the NoC, but comes along with additional implementation cost. According to section 2.3.2.4, the complexity of the crossbar of an N-port router grows from $N\,x\,(N-1)$ to $N\,x\,[V\,x\,(N-1)]$ for a router with V virtual channels. Depending on the switching method, separate buffers for each VC are necessary, increasing the implementation cost.

In packet switching networks buffering is required, regardless. Consequently, virtual channels can be implemented by splitting up a single buffer to multiple VC

buffers with smaller size[17]. Figure 2.14 shows an output port and an input port of two neighboring packet switching routers, where the buffers per VC are located at the input port. The flits of different virtual channels share the physical link and are transmitted sequentially according to the scheduling strategy. The scheduling decision for the virtual channels is taken by the scheduling unit located at the output port of the sending router. Each router can take an independent scheduling decision based on priorities, deadlines or buffer fill levels. A very common scheduling strategy is round-robin scheduling; described in section 2.4.3.4. In the literature, the scheduling of the VCs is also referred to as link access arbitration scheme. The information about the scheduled VC that is used for transmission must be provided to the next router; as shown in figure 2.14. The receiving router takes this information to forward the data to the correct VC buffer. A VC only exists between a pair of neighboring routers. Consequently, different VCs can be used by a packet at different hops.

A virtual channel packet switching router with round-robin scheduling of the VCs is presented in [130]. The asynchronous MANGO NoC uses VC-based packet switching and a round-robin-like scheduling, as described in [24].

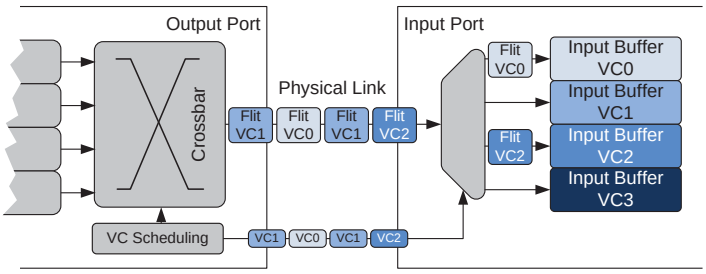


Figure 2.14.: Virtual channel realization with scheduling unit and separate buffers per VC.

If the virtual channel concept is applied to a circuit switching network a globally synchronized scheduling of the VCs is required to maintain circuit switching behavior[18]. Flits remain in the network adapter of the transmitting node tile the VC associated with the used end-to-end connection is scheduled. Afterwards, the flit traverses the network with the same speed compared to the circuit switching implementation without VCs. Due to the fact that the physical link bandwidth is shared between the VCs, each connection receives only a part of the physical link bandwidth. The end-to-end connections, which are multiplexed by the VC-

[17] In case of wormhole switching, the buffer size might be reduced if VCs are added. SAF and VCT still have the requirement of a buffer with a minimum size of an entire packet.

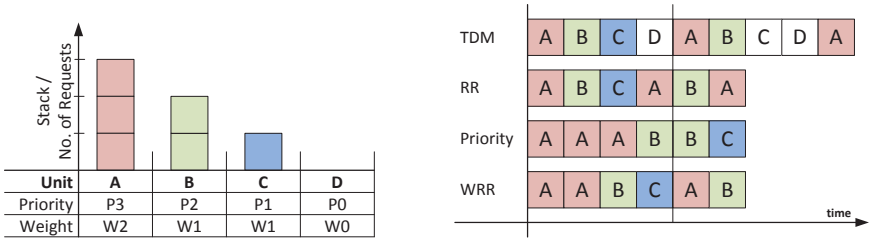
[18] In this context circuit switching behavior refers to the end-to-end connections without buffering and without or with single-cycle delay per hop.

scheduler to the physical link, are often named *virtual circuit* in the literature. For scheduling, *time division multiplexing (TDM)* is often used in a circuit switching NoC. TDM must be synchronized between all routers and NAs to enable end-to-end bandwidth and latency guarantees [87].

The concept of virtual channels is used in different circuit switching NoCs to increase the number of parallel communication streams and improve their flexibility. In the Nostrum NoC, a concept named looped containers uses globally synchronized TDM scheduling [177]. Lu and Jantsch [166] also use a synchronous TDM-based VC circuit switching NoC. In Æthereal [87, 92] contention-free routing is used to enable GS communication using VCs and circuit switching.

2.4.3.4. Scheduling and Arbitration

In the previous section, virtual channels were introduced. These virtual channels must be scheduled for data transmission. This scheduling is also commonly referred to as arbitration. In a NoC router, arbitration may be required at different positions. For example, an arbitration may be required between multiple packets that have entered the router through different inputs but have to use the same output. Depending on the router's implementation, the routing unit may be shared. Then, access to the unit must be managed by arbitration. However, there are many different positions or situations where arbitration is required. In the following, the most relevant scheduling strategies and arbitration schemes are briefly introduced.



(a) Situation (b) Schedules

Figure 2.15.: Popular arbitration/scheduling schemes used in NoCs: (a) gives a possible situation requiring arbitration between component *A*, *B*, *C* and *D*; (b) shows the resulting schedules for TDM, RR, priority and WRR scheduling.

TDM A *time division multiplexing (TDM)* scheduling or *time division multiple access (TDMA)* scheme uses fixed time slots that are statically assigned to each unit that is attached to the arbiter or scheduler. An advantage of TDM is its simple synchronization for the realization of distributed synchronous scheduling (e.g. synchronous scheduling of VCs for circuit switching [177]). Due to the fixed schedule order and scheduling cycle length, TDM only requires an initial synchronization. All components are scheduled with the same rate; resulting in a fair scheduling. A drawback of TDM is the scheduling of idle units. Due to the predefined fixed scheduling order, a TDM arbiter selects also component that do not request for arbitration. The selection of idle units results in an inefficient utilization of the available bandwidth, when TDM is used for VCs scheduling.

The example in figure 2.15(a) shows a situation with four components, each with a stack that must be processed by selecting the respective component. Figure 2.15(b) gives the resulting scheduling for different arbitration schemes. This example shows that compared to the other schedules, TDM requires more time to process the stacks of all the four units because of the scheduling of idle components *D* (right from the beginning) and *C* (in the second scheduling cycle).

Round-robin Another very common scheduling strategy is *round-robin (RR)*, also known as *asynchronous time division (ATD)*. Compared to TDM, round-robin does not consider idle components for scheduling. This results in an optimal scheduling with respect to the time required to process the stacks of all units. However, omitting idle components makes the implementation of the arbiter more complex and more costly. Like TDM, RR is a fair scheduling strategy where all components requesting for arbitration are selected for an equal time. The efficiency in terms of utilization and its fairness make round-robin a widely used scheduling strategy for NoCs and other integrated circuits. RR can be used to optimize the bandwidth utilization of a NoC with VCs [130]. Looking at the scheduling example provided in figure 2.15, it can be seen that RR results in an optimal schedule with respect to the total time. The requesting components are selected; alternating with the same frequency.

Priority TDM and RR can be referred to as fair arbitration schemes. However, there are situations where prioritization is desirable. Priority scheduling defines a priority for each component that is taken into account for arbitration. In the simplest implementation, the component with the highest priority is selected if it requests for scheduling. More advanced schemes have limitations for the maximum schedule length of each priority level to avoid starvation of lower priorities. Such a prioritization scheme is used in the QNoC [29] to enable QoS. Figure 2.15(b) shows a priority schedule for the priorities and stacks provided in figure 2.15(a). *P3* is the highest priority and *P0* the lowest. Because priority

scheduling is not a fair strategy, it enables to schedule the components with different rates. The drawback of priority scheduling is the potential blocking of components with low priorities by higher prioritized ones. Consequently, the predictability of priority scheduling is limited.

Weighted round-robin The *weighted round-robin (WRR)* arbitration scheme is an extension of RR scheduling. As in RR scheduling, it selects all components within one scheduling cycle. The time span or number of schedules per cycle for each component varies and depends on the weight that is assigned to the respective component. However, the length of a scheduling cycle is not fixed, but is limited by the sum of the assigned weights. To achieve predictability for WRR, the scheduling cycle length must be limited. This can be ensured by providing an upper boundary for the sum of all weights. WRR provides the capability to adjust the requirements (e.g. bandwidth or latency) at run-time by assigning weights dynamically. In figure 2.15(a), a weight is assigned to each unit. The given weights define the number of schedules per cycle. The resulting WRR schedule is provided in figure 2.15(b). From the example, it can be seen that WRR is fairer than priority scheduling. Compared to priority scheduling, the component *C* is not blocked that long, when WRR is used.

2.4.4. Flow Control

Flow control is necessary to ensure that no data are dropped unnoticed between sender and receiver. Data can either be dropped due to errors that appeared during transmission, such as the unavailability of buffer space in the receiving router. As an alternative, the communication system can be designed in a way that no data are dropped. In that case, flow control must ensure that buffer space is available at the receiver.

In a network on chip, flow control is applied at multiple layers. At the link layer it is applied between neighboring routers, known as *router-to-router flow control*. *End-to-end flow control* protects the communication between the transmitting and receiving node. Since the requirements for router-to-router and end-to-end flow control differ essentially, they are discussed separately.

2.4.4.1. Router-to-Router Flow Control

If a packet switching router (see section 2.4.3.2) forwards a flit or packet to a neighboring router, space must be available in the buffer of the following router. In case of wormhole switching, flow control must be carried out at the granularity of flits, whereas SAF and VCT only necessitate flow control at the granularity

of packets[19]. In case of circuit switching (section 2.4.3.1), acceptance of flits is ensured anyhow due to synchronous scheduling. Thus, explicit router-to-router flow control is not required for circuit switching.

In the following, three different router-to-router flow control mechanisms are described[20]:

STALL/GO: STALL/GO only requires two signals: The *REQ* signal is controlled by the sending router and indicates that valid data is transmitted. If the receiving router is running out of buffer space, it sets the *STALL* signal. This signal indicates that the sending router is not allowed to transmit data anymore. An implementation of STALL/GO is very lightweight. However, it can be critical with respect to timing due to the dependency between the *REQ* and the *STALL* signal. The *STALL* signal for the next cycle depends on the *REQ* of the current cycle, resulting in a combinational path through the receiving router and back to the sending router. This critical path prevents pipelining of the link and can limit the performance.

Credit-based: In the case of credit-based flow control [89], the sending router has a specific number of credits that represent the buffer space available in the neighboring router. Each time a flit is sent, one credit is consumed. If all credits are consumed by the router, it is not allowed to continue transmission and must wait until it receives new credits. New credits are forwarded from the receiving router to the transmitting router if buffer space becomes available again. Credit-based flow control can also be implemented with just two signals between the routers[21]. The first signal is again a *REQ* and the second signal, coming from the slave, is the *CREDIT* signal that indicates new credits. Credit-based flow control can be built using a counter at the output port of each router that indicates the number of available credits. This counter is decremented each time a flit is sent and incremented again if the *CREDIT* signal is set. Credit-based flow control enables pipelining and is widely used in NoCs.

ACK/NACK: The ACK/NACK protocol is based on explicit acknowledgment of each transmission. The sender sets a *REQ* signal if data are transmitted. As soon as the data have been received and processed correctly by the receiving router an *ACK* is returned. If data haven't been received correctly, a *NACK* is returned instead. ACK/NACK can be carried out in a credit-based manner

[19] In the following flit level flow control is assumed because wormhole switching is most popular in NoCs.

[20] In addition to the mechanisms discussed here, other router-to-router flow-control schemes, such as T-Error [241], exist. An overview is provided in [175].

[21] If virtual channels are used more signals are required to provide separate credits for each VC buffer.

to enable multiple outstanding *ACKs* and *NACKs*. In the case that a flit is not acknowledged (*NACK*), it must be transmitted again. Therefore, all flits that have not been acknowledged must be stored for potential retransmission in the buffer of the sending router. The retransmission scheme utilizing ACK/NACK comes along with higher implementation costs compared to the other two flow control mechanisms. However, it can be used to enable fault tolerance.

2.4.4.2. End-to-End Flow Control

End-to-end flow control is carried out between the sending and the receiving node. On one hand, it is used to ensure that erroneous or discarded packets [149] are retransmitted again. On the other hand, end-to-end flow control is required to ensure that buffer space or memory is available at the receiving node and packets are accepted. In packet switching networks, non-acceptance of packets at the receiver could result in back pressure or packet dropping. Back pressure could lead to congestion or even complete blocking of the entire communication system. Circuit switching also necessitates end-to-end flow control, because packet acceptance must be ensured at the receiving node due to the absence of buffers in the routers. End-to-end flow control is typically implemented within the network adapter by the use of flow control messages that are transferred between the involved network adapters. Due to the additional messages, end-to-end flow control comes along with a communication overhead that must be evaluated carefully.

According to [175], four different basic end-to-end flow control mechanisms exist[22]:

1. **Pre-allocation or reservation:** If the sender is aware of the buffer space available at the receiving node or can pre-allocate buffer space before transmission, acceptance can be ensured after reception. Credit-based flow control [89] can be used to ensure that buffer space is available at the receiver, as described in detail in section 2.4.4.1. For end-to-end flow control messages must be used. Pre-allocation only ensures packet acceptance. Fault tolerance is not addressed.
2. **Dropping and retransmission:** Another strategy is dropping of packets that cannot be stored. If a packet is dropped, retransmission must be initiated. Such a retransmission scheme can be carried out by ACK/NACK flow control (see section 2.4.4.1). It comes along with a high implementation overhead but can ensure fault tolerance.

[22] End-to-end flow control is not discussed in detail, due to the fact that it is done between the network adapters, which are not in the scope of this work.

3. **Rejection and return:** Packets that cannot be accepted by the receiver are rejected and returned to the sending node. This strategy can be used to enable fault tolerance. Depending on the packet size and the rejection rate, it can result in high communication overhead.
4. **Deflection routing:** The concept of deflection routing [75] is to send packets around in the network that cannot be accepted by the receiver. The strategy is only applicable to packet switching NoCs. Deflection routing cannot be used to enable fault tolerance with respect to erroneous packets.

2.4.5. Routing

Routing is the process of finding a path from a source to a destination node. In order to identify the respective source and destination node, each node within the network must have a unique identifier. Within the NoC itself, physical addresses in the form of sequential numbers or XY(Z) coordinates[23] are used for addressing. These physical addresses might be translated into logical addresses or memory address ranges by the network adapter. These addresses are then visible to the software. The physical source and destination[24] address of the two communicating nodes are processed by the routing algorithm.

The routing algorithm strongly depends on the topology. In the following, a regular topology (mesh or torus) is assumed because most existing NoC architectures use such a regular topology.

The choice of a routing algorithm impacts many different properties of a network on chip:

- **Power consumption:** In general, the length of the path between source and destination node has a noticeable impact on the energy required for data transmission. Minimal routing (e.g. dimension-ordered routing [185]) can improve the power consumption. Balancing of NoC traffic by the assistance of the routing algorithm can improve the power consumption [116].
- **Resource requirements:** Depending on the complexity of the routing algorithm, it might have a noticeable impact on the implementation cost or silicon area respectively. A complex routing algorithm might not only consume more area but also more power.
- **Protocol overhead:** The choice of the routing algorithm affects the protocol overhead of each packet. In case of source routing, described in section 2.4.5.1, the protocol overhead might be higher compared to a distributed

[23] Especially for regular topologies, such as mesh or torus networks, coordinates provide an intuitive way of addressing. The Z coordinate is only required for 3D topologies. In the following, 2D topologies are assumed for simplicity.

[24] Not all routing algorithms take the destination address into account.

routing algorithm, detailed in section 2.4.5.2. This protocol overhead impacts again the power consumption as well as the effective bandwidth of the network.

- **Performance:** Throughput and latency are also affected by the routing algorithm. The length of the path between source and destination node impacts the latency of the data transmission. A routing algorithm supporting load balancing can be used to improve the throughput between communicating nodes [117].
- **Robustness:** The robustness of a routing algorithm describes its capabilities to adapt to different load scenarios. XY routing [185] provides good performance for balanced load situations. However, adaptive routing schemes [15, 117] are more robust with respect to changing traffic patterns.
- **Fault tolerance:** If faults can occur in the NoC, the routing algorithm must be resistant against these faults. The routing decisions must be adjusted according to the current fault situation. Fault tolerant adaptive distributed routing [222], routing table based fault tolerance [76] and reconfigurable routing [281] have been proposed to achieve fault tolerance.
- **Deadlock and livelock freedom:** Deadlocks and livelocks could lead to situations where packets get stuck in the NoC and do not arrive at the destination node. In order to avoid such situations, deadlock-free and livelock-free routing algorithms, such as dimension-ordered routing [185] or adaptive odd-even turn [46], can be used. Otherwise, deadlock recovery techniques can detect and recover from deadlocks [12, 172].

According to [3], routing schemes can be classified into different categories. Table 2.1 summarizes different classifications and the alternative options for each of them. The most relevant realization options are discussed in the following subsections.

Classification	Options
Number of destinations	unicast, multicast, broadcast
Place of routing decision	centralized, source node, distributed, multiphase
Type of decision	lookup table, FSM
Implementation	software, hardware
Adaptivity	static, partially adaptive, fully adaptive
Minimality	minimal, non-minimal

Table 2.1.: Classification of routing schemes.

2.4.5.1. Source Routing

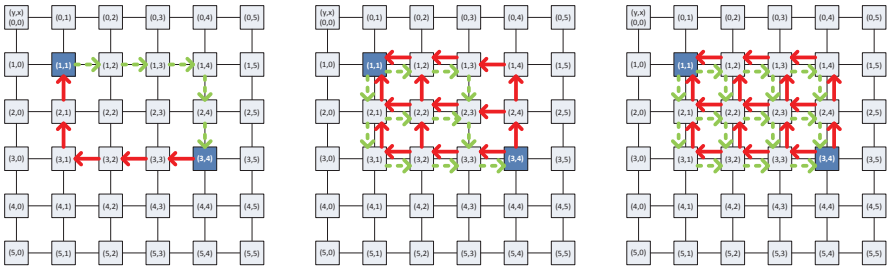
In the case of source routing, the sending nodes define the complete route of a packet and adds this information to the packet header. The information regarding the route is then carried throughout the network and processed by each router to forward the packet accordingly. The routing decision is either taken by accessing a pre-computed routing table that is located in the network adapter of each node or by an algorithm (FSM) calculating the route on demand. The routing information for each hop is then coded and inserted into the header of the packet to be processed by the routers. Because the routing is performed in the source node, the complexity of the NoC routers can be reduced. The drawback of source routing is additional protocol overhead resulting from the routing decision that must be carried around through the network. Moreover, the size or length of the routing decision that is carried throughout, is not fixed. The protocol overhead depends on the length of the path taken by the packet. This variability is hard to handle efficiently. Due to these drawbacks, the scalability of source routing is limited. However, Æthereal [92] as well as the MANGO NoC [25, 26] uses source routing for its BE router. A fault tolerant source routing method is presented in [137].

2.4.5.2. Distributed Routing

In contrast to source routing, distributed routing shifts the routing decision from the node or network adapter to the NoC routers. In distributed routing, the destination address is carried in the packet header. The source and destination addresses are mostly given as XY coordinates of the respective node. The routing decision is taken in a distributed way by the routers, which forward a packet. Each router decides independently, where to forward the packet. More precisely, it decides which output port to be used. The decision is either taken by a routing algorithm, implemented as an *finite state machine (FSM)* or by accessing a look up table that contains the output ports for given destination addresses. A very common example of a distributed routing algorithm used for meshed NoCs is XY routing [185]. It enables a lightweight FSM implementation when using geographical XY coordinates for addressing. For instance, distributed XY routing is used in Intel's SCC [113]. They pre-compute the route in the previous hop to allow fast output port identification on packet arrival. Compared to source routing, distributed routing offers better scalability due to its low and constant protocol overhead. However, due to the distributed decision making process in the routers, they become more complex.

2.4.5.3. Static Routing

Static routing, also known as deterministic routing, uses fixed or static paths between a pair of nodes. It does not provide any degree of freedom with respect to the path that is used for data transmission[25]. Thus, static routing does not enable to react on changing load conditions or faults by choosing the route accordingly. This makes static routing inflexible with respect to changing operation conditions of the architecture. Advantages of static routing algorithms are in-order arrival and its lightweight implementation. In-order arrival can easily be guaranteed since all packets take the same path[26]. The lightweight implementation results from the simple decision process, which does not necessitate to choose between alternative routes. Static routing can either be implemented as source or as distributed routing scheme. The most common static routing algorithm is XY routing [185] shown in figure 2.16(a). XY routing is a minimal routing strategy. Minimal means that the path between source and destination has the shortest distance referring to the Manhattan distance in a regular mesh or torus topology.



(a) XY routing (b) Odd even turn routing (c) Fully adaptive routing

Figure 2.16.: Possible paths for bidirectional communication between router *(1,1)* and *(3,4)* for three different minimal algorithms: (a) XY routing, (b) odd even turn routing and (c) fully adaptive routing.

2.4.5.4. Adaptive Routing

Adaptive routing is the counterpart to static routing. It is also knows as dynamic routing. An adaptive routing algorithm can provide different valid paths between

[25] Most static routing schemes allow only one path between source and destination. However, there are also static routing schemes with multiple fixed paths from source to destination.

[26] If virtual channels are used precautions must been taken to ensure in-order arrival. The reason is that packets can pass each other on different VCs.

a pair of nodes. Within the list of possible routes, the most suitable path can be selected for transmission. This additional degree of freedom compared to static routing can be used to optimize communication. The route can be selected in a way that load is balanced [117], power consumption optimized [15] or faulty routers bypassed [222]. Adaptive routing can either be implemented as source or as distributed routing scheme. The risk when using adaptive routing are deadlocks; as described in detail in section 2.4.5.5. However, adaptive routing algorithms that have been proven to be deadlock-free do exist. The most popular example is the odd-even turn model [46]. It is derived from the turn model [90]. Valid paths for minimal odd-even turn routing are shown in figure 2.16(b). Another deadlock-free adaptive routing scheme that can also be used to achieve fault tolerance is presented in [65]. Dally and Aoki [59] proposed an adaptive routing algorithm that uses VCs to eliminate cyclic dependencies and achieve deadlock-freedom. It can be used for fully-adaptive routing that allows each minimal path between source and destination [34], as shown in figure 2.16(c). A drawback of adaptive routing is the requirement for reordering to enable in-order delivery. When adaptive routing is used, different packets of the same message or with dependencies could take different routes and pass each other but must arrive in-order at the destination node. Thus, reordering must be performed at the destination network adapter [274] or in the network itself [146]. In contrast to static routing, adaptive routing is more complex to implement and can thus result in higher implementation cost and power consumption for routing. However, it offers better performance and can also be used to reduce energy consumption for data transmission. The trade-off between costs and benefits of rerouting has to be evaluated carefully depending on the NoC requirements.

2.4.5.5. Deadlock

Deadlock-freedom is a basic requirement for a reliable network. Deadlocks can block links permanently, resulting in back pressure and blocking of other links. Such a situation could result in a global blocking of the communication infrastructure. A deadlock occurs if a buffer on the way of a packet is blocked and can no longer be released. Figure 2.17 shows a typical example of a deadlock situation. Four packets are involved in this scenario. Each of the packets wants to turn right but cannot proceed to the next hop because the buffers are already occupied by another packet. This cyclic dependency results in a permanent deadlock.

Especially wormhole switching is susceptible to deadlocks because the packets are spread over multiple hops and buffers [78]. In order to avoid permanent deadlocks, different solutions have been proposed in the past. One feasible solution is the use of a routing algorithm that is deadlock-free by construction. The basic principle when constructing a deadlock-free routing algorithm is to

avoid certain turns[27]. A general concept for deadlock prevention in routing algorithms is the turn model [90]. The famous adaptive odd-even turn routing algorithm [46] is a derivation of the turn model and is therefore deadlock-free. Static dimension-ordered routing [185] also prevents turns and is deadlock-free by construction. Virtual channels can also be used to construct deadlock-free routing algorithms. In [60], VCs are grouped and the packets are assigned to the groups with respect to the taken routing decisions. This breaks up cyclic dependencies and ensures deadlock-freedom. However, if the routing algorithm is not deadlock-free by construction, deadlocks have to be avoided or recovery techniques must be applied at run-time. Virtual channels can be used to decrease the probability of deadlocks by providing multiple parallel paths per link [58]. In [136], a deadlock recovery scheme is presented that eliminates blocked packets and requests for a retransmission. In [172], packet injection rate is limited to avoid deadlocks. If a deadlock is detected anyhow, a recovery technique absorbs the deadlocked message at the current node and later re-injects it. Another strategy for deadlock recovery is the use of a recovery path [12, 202]. This recovery path is used to progressively forward one of the blocked packets to resolve the deadlock situation.

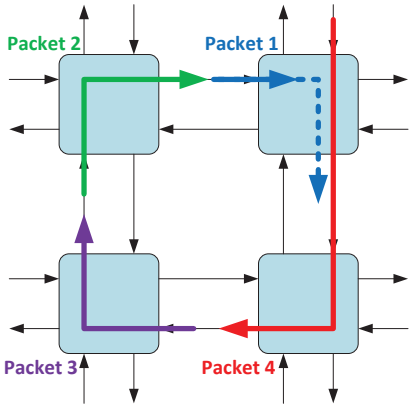


Figure 2.17.: Deadlock scenario with four packets waiting for channels with cyclic dependencies.

2.4.5.6. Livelock

The counterpart of a deadlock is a livelock [110]. A livelock is a situation where a packet is forwarded through the network without reaching its destination node.

[27] A turn is a change of direction performed by a packet. If a packet enters a router from west and leaves this router southwards, an east-south turn was taken.

In case of minimal routing algorithms, which forward the packet continuously towards the destination node, livelocks are not possible. Due to the fact that most static and adaptive routing algorithms are minimal, livelocks play a minor role. Thus, livelock-freedom must only be ensured in case of non-minimal routing. An example for a non-minimal routing algorithm is GOAL [231]. It is a load balancing adaptive routing algorithm for torus networks.

2.4.5.7. Multicast and Broadcast

Thus far, only point-to-point connections have been considered with respect to routing. However, situations exist, where the same information or message has to be delivered to various recipients. If the number of receivers is a subset of all nodes, this transmission is named *multicast* or one-to-many communication. If all members of the communication infrastructure are addressed, the transmission is referred to as *broadcast* or one-to-all. A broadcast can be considered as a special multicast and is thus not addressed separately here.

The simplest realization of a multicast data transmission is a separate transmission by the source node to each receiver using point-to-point communication. The drawback of this method is the high utilization of the network, resulting from the fact that the same message takes the same link multiple times. A more efficient multicast scheme for wormhole switched NoCs is presented in [168]. In this scheme, the multicast packet is forwarded from the transmitting node to the first destination node. The node replicates the message and forwards it to the next destination node and so on. Distributed XY routing is used to forward the packets between the nodes. Another multicast mechanism supporting wormhole-switching is proposed in [219]. It uses special header flits to configure routing tables used for the following broadcast. The payload that has to be broadcasted is then injected after setup. It is replicated and forwarded by the routers according to the entries in the routing table. A similar approach is named virtual circuit tree multicast [123].

2.5. Dependability

For future manufacturing technologies, fault tolerance and reliability concepts are expected to play an ever increasing role [105]. In this section, fundamental reliability principles and definitions will be provided. They are used as a basis for the reliability concept, which will be introduced in chapter 8.

In order to design a fault tolerant system, a cause-effect chain that extends over multiple layers has to be considered. At the lowest layer, physical failure mecha-

nisms cause transistors or wires to fail; as described in section 2.5.5. At the second layer, a failure is modeled as a fault; as summarized in section 2.5.2. The set of all modeled faults is denoted as fault model [210].

2.5.1. Terminology

A *fault* does not necessarily activate an error. An *error* denotes the deviation of information compared to fault-free operation. For example, a stuck-at-1 fault activates an error only if the correct logic value was 0. Otherwise, the fault does not become manifested as an error. An error that propagates to an output of the module causes a *failure* of the module. An error that does not propagate to the module's output is said to be masked or transparent.

2.5.2. Hierarchical Fault Modeling

The scheme of fault, error and failure can be repeated from layer to layer of the system. The failures of lower-level modules, which are instantiated as components and connections, are faults at a higher level. Consequently, at higher layers the fault model becomes more abstract and more remote from the original physical failure mechanism. Abstraction of fault models is necessary to reduce the complexity of dependencies and interactions [210]. The chain of faults, errors and failures is repeated upward the system's hierarchy until system failures are observed at the top level.

2.5.3. Redundancy

Redundancy is the fundamental technique to achieve fault tolerance. Depending on the class of fault, different redundancy techniques are used to cope with them:

- **Spatial/modular redundancy:** Several components perform the same task. Majority voting is used to determine the correct result.
- **Temporal redundancy:** A failed computation or data transmission is repeated (re-executed) with the same component until the result is verified by the consumer.
- **Information redundancy:** Additional information is added to the data being processed. Information redundancy can be used to correct errors directly on observation. This is named *forward error correction (FEC)*.

Diverse redundancy techniques are used to cope with the different fault classes, which are introduced next.

2.5.4. Fault Classes

According to [47], faults are widely grouped into transient, intermittent and permanent faults.

Transient faults appear randomly for one or several cycles. They typically result from radiation; as detailed in section 2.5.5. Their distribution is truly random because they result from neutron and alpha particle collisions. Due to this random distribution, their rate is constant during the lifetime of a chip.

Intermittent faults occur periodically. Thus, they are easy to confuse with transient faults. Three criteria are proposed in [47] to differentiate between transient and intermittent faults:

1. *"An intermittent fault occurs repeatedly at the same location"*
2. *"Errors induced by intermittents tend to occur in bursts when the fault is activated"*
3. *"Replacement of the offending circuit removes the intermittent fault, in contrast to transients, which repairs do not eliminate"*

Intermittent faults can result from crosstalk or self-coupling and their occurrence may depend on the operating conditions; such as temperature. Other intermittent fault might be caused by manufacturing variations and imperfections. Aging processes can first lead to intermittent delay or logic faults and later on to permanent faults; as described in detail later.

Permanent faults are faults that are always present after the first occurrence. Permanent faults can result from manufacturing process variations, aging effects or design errors. They can be divided into two classes:

- **Logic faults:** Transistors or wires are permanently open or shortcut
- **Delay faults:** Transistors or wires are too slow, causing setup and hold time violations that generate an incorrect logic value

If a permanent fault leads to a functional error only for specific patterns, it is still denoted as a permanent fault that is masked under certain conditions. A good example for a masked delay fault is crosstalk that may only lead to an error for certain logic transitions on neighboring wires.

According to [210], different redundancy techniques can be used to cope with transient, intermittent and permanent faults. Spatial redundancy is typically best suited for handling permanent faults. Temporal redundancy can cope well with transient and intermittent faults. Information redundancy in the form of

error-correcting codes (ECCs), such as hamming codes or turbo codes, can be used to protect a transmission against all types of faults.

2.5.5. Physical Failure Mechanisms

CMOS-based circuits can suffer from malfunctions due to a variety of reasons, rooted in the physics of devices and materials. They are affected by process variations and by dynamic variation of power consumption or temperature. According to [210], the causes of failures and errors during production and lifetime can be grouped into the following categories:

Radiation in the form of terrestrial cosmic neutrons and alpha particles that originate from radioactive impurities in the device and packaging material are the main source for soft errors. So-called *single event upsets (SEU)* can cause bit flipping in SRAM and DRAM memory cells. When a particle shifts the voltage level of a wire or a logic gate to an incorrect logic value, it is named *single event transient (SET)*. The probability of a SEU or SET event to happen depends on the critical charge, necessary to flip a bit. When shrinking technology nodes, the critical charge shrinks as well, hence the likelihood of an SEU or SET increases. Additionally, the number of transistors grows as well for future technologies, which results also in a higher *soft error rate (SER)*. Taking both aspects into account, a superlinear growth of the SER is expected for future technology nodes [229, 31].

Electromagnetic Interference mainly results from crosstalk between long parallel wires. In order to keep the delay and resistance of wires low, the height of the wires is not shrinked for technology scaling. This leads to a growing coupling capacitance and inductance between parallel wires. Consequently, signals of neighboring wires influence each other, resulting in glitches and higher signal delay. Self-interference increases with the frequency, due to skin effects, which are a growing concern for smaller wires [54].

Electrostatic Discharge can be caused by a strong electric current. The current is either induced by strong electric fields or by I/O pins. Technology scaling complicates the analysis and control of the underlying phenomena, such as PN junction, wiring or dielectric oxide breakdown. However, protection against these phenomena is done at the pins of a device. Internal circuits and components are typically not affected [88].

Aging is a generic term for various physical effects that lead to degradation of performance and function of a CMOS device over its lifetime. In the following, the most relevant aging effects are explained briefly.

Electromigration is the most important mechanism that leads to increasing delay and later on to malfunction of wires. It is a transport of metal atoms, induced by electric current. Due to higher current density, electromigration thins out a wire where it is already thinnest [47].

Negative bias temperature instability increases the threshold voltage of the PMOS transistor over time. The reason is charge migration into the insulating gate oxide. This effect is temperature-sensitive but is partially removed when the stress is over. The corresponding but less important effect on the NMOS transistor is termed *positive bias temperature instability* [278].

Hot carrier injection refers to an effect that changes the switching characteristic of a transistor. It is caused by carriers, electrons or holes that penetrate the insulating silicon oxide layer below the gate. Consequently, the threshold voltage increases, which leads to slower reaction times and performance degradation [239].

Time dependent dielectric breakdown is caused by crystal defects and heavy metal contamination in the dielectric material. This effect can be reduced by minimizing the contamination and crystal defects of the silicon [131].

Process Variability affects the occurrence and severity of all the aforementioned phenomena. Causes are variability of material impurities, doping concentrations and the size and geometries of the structures. Thus, minimizing process variability plays an important role when improving production quality and reliability. Unfortunately, scaling leads to an increased relative process variability [217].

Dynamic Temperature Variation during operation and high temperatures play an important role for aging effects. In addition, temperature impacts the power consumption and performance. Due to an unbalanced temperature distribution resulting from hotspots, the pace of wear out and faults is not distributed uniformly. Combined effects of static process variation and dynamic temperature variations lead to fluctuations of power consumption of more than 50 % and performance of up to 30 %, according to [31].

3. Context of Invasive Computing

The present work is intended to provide a general solution for communication in future many-core architectures. However, it refers also to a novel paradigm, named *invasive computing*, for designing and programming of future parallel computing systems. The invasive computing paradigm is investigated in a transregional collaborative research center funded by the Deutsche Forschungsgemeinschaft (DFG). This chapter will provide a brief introduction on the basic idea, communication requirements, software aspects and the hardware architecture addressed by the project.

3.1. Basic Principle of Invasive Computing

Invasive computing [243, 245] is motivated by the trend towards many-core architectures. As previously explained, it is visible and foreseeable that the number of processing cores will increase in future architectures. However, programming and management of such large architectures is still a huge challenge; addressed by invasive computing. Its main idea and novelty is to introduce resource-aware programming. A given program obtains the ability to explore and dynamically spread its computations to neighboring processors, taking into account the status of the underlying hardware. In [243], the principles of invasive programming are defined as follows:

> *"Invasive Programming denotes the capability of a program running on a parallel computer to request and temporarily claim processing, communication and memory resources in the neighborhood of its actual computing environment, to then execute in parallel using these claimed resources, and to be capable to subsequently release these resources again."*

Figure 3.1 shows the phases of an invasive program. An application requests for additional resources in a phase named *invade*. During the invasion phase, the run-time system, introduced in section 3.3.2, must search for resources that fit the application requirements. These requirements are provided to the run-time system in the form of constraints. These constraints describe the quantity, properties and type of resources required by the application, as described in detail in section 3.1.1.

During search phase, the OS takes the status of the underlying hardware into account to find suitable resources. If appropriate resources are found, they are provided to the application as a *claim*. A claim is a set of resources that are reserved for an application. These resources can then be used by the application to spread its computation. Therefore, a phase named *infect* is entered. During infection, the program starts its execution using the newly claimed resources. The actual application code is spread onto infected resources for subsequent parallel execution. This code is referred to as *i-let* (standing for invaslet)[1]. Once the program terminates or if the degree of parallelism decreases, the program may enter a phase named *retreat* to release resources and resume execution with reduced parallelism or even sequentially.

Figure 3.1.: State chart of an invasive program [104]. The phases *invade*, *infect* and *retreat* can be entered recursively or iteratively to adapt the degree of parallelism of the invasive program.

An invasive program is written in a way that it can adapt its degree of parallelism at run-time. This enables run-time optimization towards different objectives. An invasive application can be optimized at run-time towards low power consumption, high performance, low resource-utilization and other similar objectives. The idea of self-optimization is reflected in all layers of the hardware and software and it shall enable an efficient management of large many-core architectures.

The principles of invasive computing imply changes at the architecture, operating system and language level. An overview on each of these aspects is provided in the following sections.

3.1.1. Invasive Programming Language

For productive programming of parallel architectures, a programming language is required that allow to express and exploit parallelism. X10 [44], developed by IBM, allows to serve as a basis for invasive programming. It is designed specifically for parallel architectures and uses the partitioned global address space model, introduced in section 2.2.2.1. A computation is divided among a

[1] "This conception goes back to the notion of a servlet, which is a (Java) application program snippet targeted for execution within a web server." [104]

set of *places*. Each place is a set of cache-coherent processing elements[2]. It holds the required data and executes one or more *activities* that operate on those data. Activities are lightweight threads that can run in parallel on different cores. X10 enables object-oriented programming, provides dependent types, transactional memory, globally distributed array and uses a garbage collector for memory management. These promising new features make X10 the optimal candidate for invasive programming, compared to traditional languages, such as C++ or Java.

X10 was extended to implement the principles of invasive computing. The extended version of X10 is named InvadeX10 [282]. InvadeX10 supports the three basic constructs for invasive programming (`invade`, `infect` and `retreat`). It enables to specify the resource requirements for invasion by the use of the constraints; which will be introduced in section 3.1.1.1.

Figure 3.2 shows a simple, invasive program. The i-let function defines an action to perform in parallel on all allocated PEs. The invade function allocates resources under specific constraints in competition with other applications and returns the allocated resources as a claim object. Infect uses those resources by executing the previously defined i-lets. Retreat frees allocated resources after execution is completed.

```
1 val ilet = (id:IncarnationID) => {
2     do_something(id);
3 };
4 claim = Claim.invade(constraints)
5 claim.infect(ilet)
6 claim.retreat()
```

Figure 3.2.: Basic invasive program that defines an i-let, invades resources for execution and releases them subsequently [HZZ+14].

3.1.1.1. Constraints

Constraints are one of the basic principles of invasive programming. They are used to define the requirements of the application with respect to the resources that are invaded. The hierarchical constraint system, implemented in InvadeX10, is shown in figure 3.3 and further described in [282]. The most important constraint in practice is `PEQuantity`; which specifies the desired number of processing elements. In general, three additional basic classes of constraints exist: The first class of constraints are so-called *predicate constraints*; which specify a predicate

[2]The concept of places fits very well to the invasive architecture, where multiple CPUs are arranged in a cache coherent tile as described in detail later.

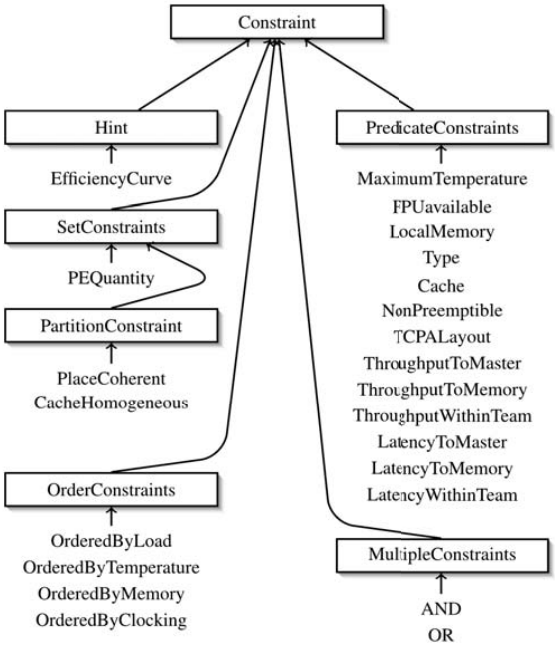


Figure 3.3.: Hierarchy of the invasive constraint system [282].

for processing elements. They place a constraint on the requested PEs, such as the amount of available local memory; a *floating-point unit (FPU)* being available or a certain bandwidth to the main memory. The second class of constraints are *order constraints*. They provide an ordering of processing elements and can be used to prioritize PEs selected for invasion with respect to their current load, temperature or available memory. The third class of constraints are *set constraints*, which includes the `PEQuantity` constraint. They specify conditions for a set of processing elements, e.g. a certain physical layout of the PEs or a cache coherency between the PEs (`PlaceCoherence`). Additionally, two operators AND and OR can be used to combine constraints. Furthermore, nonbinding *hints*, such as efficiency curves of parameters, are foreseen to handle complex information. The information is used by the run-time system for optimization. In [38], it has been shown that the InvadeX10 constraint system is applicable to create complex applications.

Figure 3.4 gives an example for the definition of constraints in an invasive program. The constraints request for invasion of one processing element of type RISC core. In the example, this PE should have a minimum communication bandwidth of 100 MB/s to the main memory.

```
1 val claim = Claim.invade(
2   new PEQuantity(1) &&
3   new Type(PEType.RISC) &&
4   new ThroughputToMemory(100)
5 )
```

Figure 3.4.: Constraining the required bandwidth or throughput to the main memory in X10 [HZZ+14].

Latency and throughput constraints As already indicated, a subset of the constraints is related to communication. These constraints have to be fulfilled by the communication infrastructure, as described in detail later. Currently, six different constraints can be used to specify the communication requirements within a *team*, to the *master* or to the *main memory*. The *team* is the set of processing elements, which were recently invaded and thus are in the claim. The *master* is the processing element where the invade call is executed. These are the available constraints for communication:

- `ThroughputToMaster`: A certain throughput is guaranteed between the team and the master.
- `ThroughputToMemory`: A certain throughput is guaranteed between the team and the main memory.
- `ThroughputWithinTeam`: A certain throughput is guaranteed within the team and thus between the individual cores of a team.
- `LatencyToMaster`: A certain latency is guaranteed between the team and the master.
- `LatencyToMemory`: A certain latency is guaranteed between the team and the main memory.
- `LatencyWithinTeam`: A certain latency is guaranteed within the team.

These communication constraints are used to manage the QoS connections that will be introduced in section 6.2. In addition to communication constraints, further language extensions relating to communication, are described in the following paragraph.

3.1.1.2. Block Prefetching

The InvasIC architecture uses distributed shared memory, as described in detail in section 3.2. To pin or cache data to a tile local memory for fast access, X10 was extended to support explicit block prefetching. Block prefetching initiates

copying of data between two memories. The data blocks are prefetched in parallel to the execution. This is enabled by a DMA unit located in the network adapter, as described in detail in section 3.2.2.1. This DMA unit pushes the requested data over the NoC.

Figure 3.5 shows how block prefetching and DMA is used and controlled in X10. First, the application must specify the amount of local memory. Although memory in X10 is managed by a garbage collector, there are explicit `alloc` and `free` methods for tile local memory to provide the application full freedom for management. In the example, the future concept is used to model the background activity of the DMA transfer. The `force`-call synchronizes and waits for the transfer to be completed.

```
1 val loc = TileLocalMemory.alloc[int](cs);
2 val offset = id.ordinal * cs;
3 val future = data.fetch(offset, loc);
4 ... // do something else, while the data is copied into tile
      local memory
5 val loc2 = future.force();
6 assert loc == loc2;
7 ... // use the tile local data in `loc`
```

Figure 3.5.: Example for block prefetching of data from main to tile local memory in X10 [HZZ+14].

3.2. InvasIC Hardware Architecture

An incarnation of a hardware architecture supporting invasive computing is referred to as *InvasIC*. It is a heterogeneous tile-based MPSoC architecture consisting of different types of processing, memory and I/O tiles. The processing elements include RISC processors, *Tightly-Coupled Processor Arrays (TCPAs)* and reconfigurable RISC cores (i-Cores). Details of the tiles are provided in section 3.2.1.

All tiles are connected by the *invasive network on chip (i-NoC)* in a 2D mesh topology. The *invasive network adapter (i-NA)* is the interface between the tile local bus and the i-NoC routers. Figure 3.6 gives an overview on the InvasIC architecture. A detailed overview is provided in [HHB+12].

All aspects of the architecture, especially the i-NoC, are designed with focus on scalability and decentralized resource management. This also applies to the memory organization. The architecture is implemented as a distributed shared

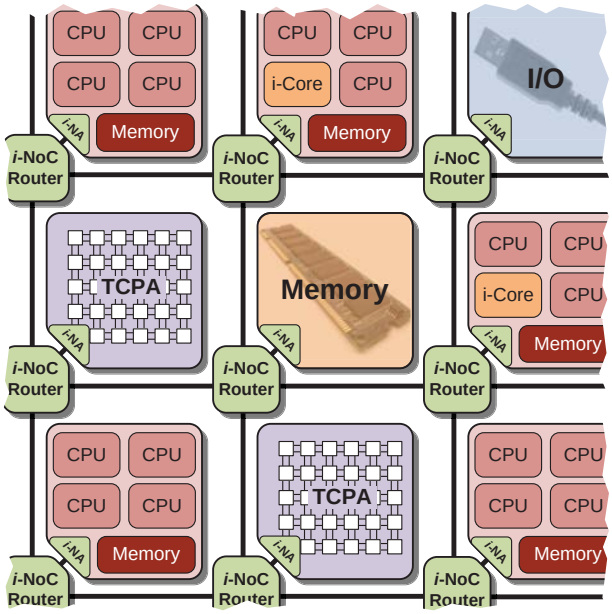


Figure 3.6.: Heterogeneous tiled InvasIC architecture with different computation and I/O tiles connected by the invasive network on chip.

memory architecture (see section 2.2.4): The PEs of a tile have coherent[3] L1 caches and a shared *tile local memory (TLM)*. However, there is no cache coherence between the tiles. The TLM of other tiles can be addressed from each tile directly. Additionally, all tiles can access the common shared main memory via the i-NoC.

Heterogeneity is another attribute of the InvasIC architecture; inspired by modern HPC systems. In today's HPC systems, FPGAs [68] and *graphics processing units (GPUs)*, such as NVIDIA Tesla [162], are recently used to improve performance and energy efficiency. Such accelerators form heterogeneous systems in combination with general-purpose CPUs. Along with this trend, the InvasIC architecture is not only equipped with standard RISC cores but also i-Cores and TCPAs.

3.2.1. Tiles

In the following, the basic concept and structure of all tile-types of the InvasIC architecture are introduced.

[3]Cache coherency is carried out by bus snooping in the LEON3 system.

3.2.1.1. RISC Core Tiles

The RISC core tiles currently consist of four[4] LEON3 SPARC V8 cores [83, 84]. However, LEON3 SPARC V8 cores were only selected due to their low complexity and open source availability as part of the GRLIB [85][5]. Conceptually speaking, other cores, such as ARM [82] or Intel cores [113] could also be employed. Another benefit from using the GRLIB is the availability of an AMBA bus system (see section 2.3.2) and numerous components, such as memory controllers or I/O interfaces, as well as debug support.

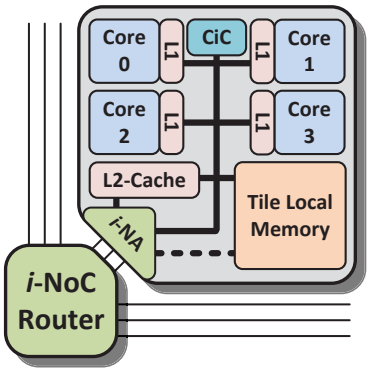


Figure 3.7.: Block diagram of a tile with four RISC core, caches and local memory.

Figure 3.7 shows the basic structure of a RISC core tile. Each of the four LEON cores contains a separate L1 data cache and instruction cache. The L1 caches of the CPUs are connected to the tile local front-side AMBA AHB bus. This bus is also used to attach the tile local memory. It is an SRAM-based on-chip memory with single cycle read and write latency, which is used to store frequently used data or even binaries. Another component attached to the tile local front-side bus is the L2 cache of the tile. It is shared between all cores and is used for holding tile-external data. These data are fetched transparently by the invasive network adapter in case of a cache miss. Therefore, the network adapter and the L2-cache are connected by a second AMBA AHB back-side bus. The i-NA has an additional interface to the front-side bus. It is used for message passing communication and configuration purposes. A direct connection between the NA and the TLM is planned. Another InvasIC-specific component in the RISC tile is the CiC.

[4]The LEON3 MP design provided with the GRLIB supports 1-16 CPUs. Four cores are a good choice with respect to the bandwidth of the AMBA AHB bus.

[5]The LEON3 SPARC V8 architecture is provided with the GRLIB [85] under GNU general public license.

The *core ilet controllers (CiC)* is a thread assignment unit that is implemented in hardware [207]. It is responsible for the assignment of i-lets to the CPUs of a tile. Incoming i-lets are filtered by the i-NA and forwarded directly to the CiC. The CiC uses a reconfigurable rule-system and monitors for CPU load, temperature or reliability in order to decide on which core an i-let should be executed. Subsequently, it triggers an interrupt for the selected core and hands over the i-let to the OS instance running on that core.

3.2.1.2. i-Core

The i-Core is a reconfigurable RISC core based on the LEON3 SPARC V8 architecture [83, 84]. It combines concepts for an adaptive microarchitecture [254] and fine-grained reconfiguration to instantiate application-specific accelerators [106]. The i-Core is implemented in a RISC tile by replacing one of the RISC cores.

In order to build an adaptive microarchitecture, the LEON3 design was extended to support reconfiguration of the processor pipeline [254], run-time configuration of cache-parameters and the branch prediction unit. This enables to adapt the microarchitecture to the requirements of the application using it.

The fine-grained reconfigurable accelerator of the i-Core relies on an FPGA fabric that is loosely connected to the processor pipeline [106]. The reconfigurable fabric has a dedicated high-bandwidth connection to a fast on-chip memory. In order to use the fabric for an accelerator, it is configured using a partial configuration bitstream from a library. This reconfiguration is triggered by an additional instruction implemented in the adapted LEON3 design. The accelerator is used after configuration by executing special instructions on the LEON3 core. However, in contrast to the adaptive microarchitecture, the reconfigurable fabric can only be used if this is supported by the binary of the application.

Depending on the application, significant speedups can be achieved by using the i-Core features, as described in [106]. The higher throughput of the application results in higher communication bandwidth requirements, which have to be fulfilled by the i-NoC. A specialized NoC router design will be introduced in section 5.5.5 to address these higher bandwidth requirements.

3.2.1.3. TCPA Tile

A *Tightly-Coupled Processor Array (TCPA)* [138] is a coarse-grained reconfigurable massively parallel architecture, consisting of an array of tightly-coupled processing elements. Each PE contains a VLIW processor and an invasion controller (iCtrl). This invasion controller enables hardware-supported decentralized invasion of TCPA PEs, as described in [151]. Communication in the TCPA is enabled

by point-to-point connections among PEs. Hierarchical power management is used to improve the energy efficiency of the TCPA [150].

Figure 3.8 shows the structure of a TCPA tile. The PEs are fed with data by the reconfigurable buffers. They are implemented as FIFOs, located at the borders of the array. A LEON3 core is used as *reconfiguration and communication processor*. It is responsible for controlling the execution of applications on the processor array and for reconfiguration management. The LEON core is also used to control the data flow to and from the buffers of the TCPA. Therefore, the TCPA has a direct connection to the invasive network adapter. An AMBA AHB bus is used to connect the TCPA, the network adapter and the LEON3 core.

TCPAs can process a huge amount of data in parallel. That implies that they have high bandwidth requirements with respect to inter-tile or main memory communication. This bandwidth must be made available by the i-NoC, as described in detail later. The high bandwidth router, introduced in section 5.5.5, can be used to increase the bandwidth of a TCPA tile.

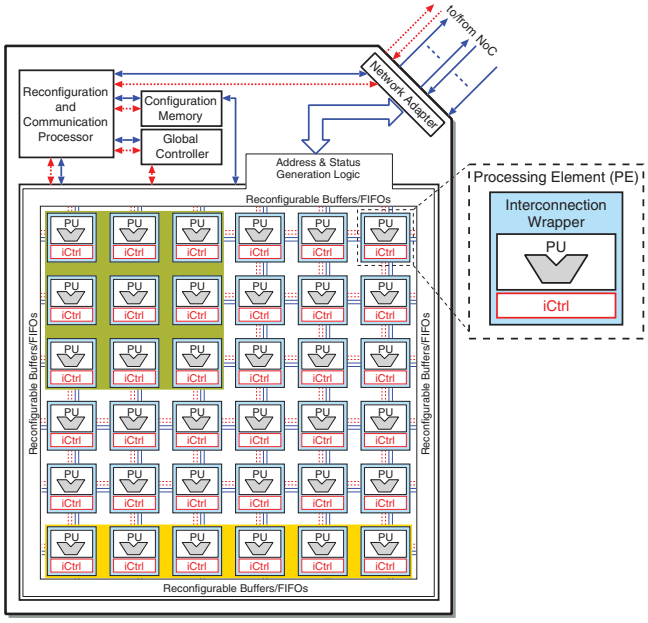


Figure 3.8.: Block diagram of a TCPA tile with network adapter [150].

Currently, a dedicated compiler named *LoopInvader* is used to generate code for the TCPA using a special programming language. However, the size of the array that is invaded by the application might not be known at compile-time. Symbolic

parallelization and symbolic scheduling [246] are used to enable generation and execution of the same code on different TCPA array sizes. In the long term, TCPAs shall be programmed in InvadeX10 and a common compilation flow shall be used for code generation; described in section 3.3.1.

3.2.1.4. Memory Tile

The main memory of the architecture is composed of multiple memory tiles. Each memory tile contains a DDR controller to interface the external memory. A network adapter is used to attach the memory tile to the i-NoC. Furthermore, a LEON3 core is used for management and control purposes. All components of the memory tiles, except the NA, are assembled from GRLIB IP cores. An AHB bus is used to connect all components.

From the communication perspective, the memory node might have increased bandwidth requirements, compared to other nodes. Multiple processing tiles might access the main memory in parallel. This results in a huge accumulated bandwidth at the memory tile. The heterogeneity of the architecture, in terms of throughput, is addressed by the concept presented in section 5.5.5.

3.2.1.5. I/O

An I/O tile is used to attach peripherals to the NoC-based architecture. Such peripherals include Ethernet[6], USB[7], I2C[8], DVI[9] and transactor interfaces (see section 3.4.2). A LEON3 core is located in each I/O tile. It is used by the OS to control the I/O components and the access to these components by other tiles. Similar to the memory tile, all I/Os, the LEON core and the network adapter are connected by an AHB bus.

3.2.2. Invasive Network on Chip

The *invasive network on chip (i-NoC)* [HZW+14], [HZZ+14] is one incarnation of the NoC design; presented in this work. The details about the general NoC template follow in the subsequent chapters. This section is limited to a very short overview on the i-NoC design and its functionality.

[6] Ethernet is used as the main interface for data exchange between the InvasIC architecture and a host system.
[7] USB is used in the InvasIC architecture to forward monitoring data to a host PC.
[8] I2C is used to visualize important system states by the use of LEDs.
[9] A DVI interface is used to feed a video stream to the architecture and to return a processed video for visualization.

The invasive network on chip is a wormhole packet switching network on chip with virtual channels. It has a meshed topology with bidirectional links. All aspects of the i-NoC are designed with focus on scalability and distributed management. Region-based management concepts, introduced in section 6.3, 6.4 and 7.3 are applied to enable distributed management of the communication resources. The i-NoC provides QoS support to enable the invasion of communication resources, as described in chapter 6. It has a monitoring infrastructure that is used to take proper invasion decisions at run-time [HZW+14]. The monitoring infrastructure is described in detail in section 5.5.3. According to the principles of invasive computing, hardware-based self-optimization features are used to improve the performance and to reduce the software management overhead of the large and scalable InvasIC many-core architecture. These features are described in chapter 7.

3.2.2.1. Invasive Network Adapter

In contrast to the i-NoC, the *invasive network adapter (i-NA)* is not in the scope of this work. However, it was developed in close collaboration with the invasive network on chip to support its features and transmission schemes. Due to the very close relation between invasive network on chip and the i-NA, the network adapter is explained in more detail, in the following paragraph.

The *invasive network adapter (i-NA)* [ZHW+15], [HZW+14] provides the interface between the tile local bus and the network routers. Currently, an AMBA AHB bus is used in all tiles. Thus, each type of tile can use the same NA equipped with AHB interfaces. In order to access data over the i-NoC, the network adapter performs a protocol translation from bus to network packets.

The network adapter has multiple functions. It enables direct access to all memories available in the InvasIC architecture. The i-NA provides message passing support including specialized OS messages. It has an interface to reserve communication resources and supports automatic QoS connection establishment and use of them. Each NA contains a configurable DMA unit and provides access to the status and configuration registers of the i-NoC routers. All of these aspects are described in detail, in the following paragraphs.

Basic structure The basic structure of the i-NA is shown in figure 3.9. It has a separate data path for outgoing and incoming packets. Each of these paths has two interfaces with different address ranges, connected to the bus of a tile. The *shared memory interface* is used for direct access of tile external memories. Message-based communication is enabled by the *message passing interface*. The i-NA has three functional pipeline stages named *tile interface*, *protocol translation* and *link interface*. They are implemented as a modular design. This modular design

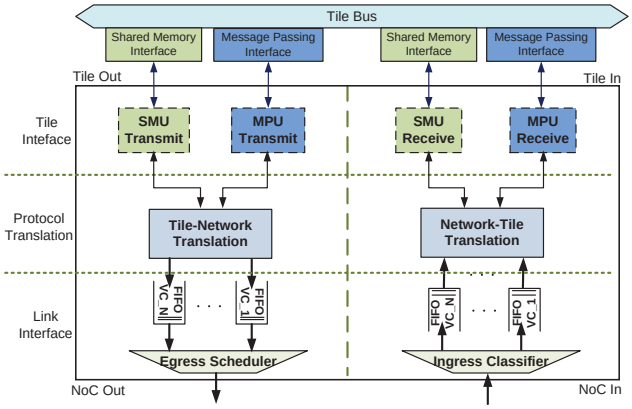


Figure 3.9.: Block diagram of the invasive network adapter with shared memory access and message passing support [ZHW+15].

enables transparent sharing of protocol translation and link interface services by the shared memory and message passing interface. At the tile interface, the *shared memory unit (SMU)* is used to process memory access requests and responses that enter (*transmit*) or leave (*receive*) the NoC. The *message passing unit (MPU)* at the transmit side is used to trigger the transmission of messages. Its counterpart at the receive side provides access to incoming messages. The protocol translation layer is used for packetization and depacketization. Packets are generated and stored in the output VC FIFOs. The packets are subsequently forwarded to the NoC. The link interface is used to inject data into the network (*NoC out*) and to receive data coming from the NoC (*NoC in*); implementing the flow control protocol between the routers and scheduling the transmission of the output VCs. On the receive side, it assigns packets to the receive buffers. The packets are fetched from the receive buffers for depacketization. The payload is forwarded to the SMU or MPU depending on the message type.

Distributed shared memory support The InvasIC architecture is a distributed shared memory architecture. This memory model is enabled by the shared memory interface of the invasive network adapter; shown in figure 3.9. It maps the global shared memory and the distributed memory of the architecture to the address space of each tile. Access to a tile external memory is received by the network adapter through its shared memory interface. Inside the transmit SMU, an address lockup is performed to determine the node where the respective memory is located. The address lockup implements a part of the memory map, provided in appendix A.3. Subsequently, a packet is generated and sent to the node where the memory is located. The NA at this node receives the packet, processes it and

performs an access to the memory using the receive SMU. In case of a read access, the requested data are packetized again and returned to the node which initiated the request. The unit initiating the request receives the requested data through the tile local bus after depacketization at the initiating node. In the case of a write access, no response packet is required.

More details about the address mapping of the InvasIC architecture are provided in appendix A.3. The address map defines the exact address ranges for access of remote TLMs and DDR main memory tiles.

Message passing The distributed shared memory support enables transparent communication between different nodes using normal `load` and `store` operations. However, the i-NA also supports message-based communication by the use of its message passing interface[10]. This interface is memory mapped and can be used to generate and transmit different types of messages. A message is generated by writing to the respective registers of the network adapter. The data to be sent are packetized and forwarded to the receiving node. After depacketization, the messages are available at the receive MPU and can be accessed through the message passing interface.

Currently, three different types of messages are supported by the network adapter:

- **i-lets:** i-lets or i-let messages are used to initialize code execution at a remote tile during infect phase of an invasive program (see section 3.1.1). The CiC at the destination tile is informed about the arrival of such i-let messages to process them subsequently.
- **System i-lets:** These are special messages used for OS internal communication. They support a limited amount of payload. After transmission, the payload is accessible via registers at the receive tile. In addition, an interrupt is triggered at core 0 of the receive tile to inform the system software about the arrival of a system i-let.
- **Lock message:** The remote lock message uses the test-and-set method to acquire a lock at the destination tile. This special lock message can be used to create a remote spinlock.

Reservation of communication resources In order to reserve or invade communication resources, a memory mapped interface is provided by the invasive network adapter. In figure 3.9, this interface is part of the *message passing interface*. It is used to trigger the reservation of an end-to-end connection. The MPU is then responsible for allocating a virtual channel for the connection and for injecting

[10] Message passing is basically a communication concept in computer science. In this work, it is assumed that message passing in the communication hardware is based on one-sided "put and get" method [113].

a special flit for connection setup, as described in detail in section 6.2. If the connection was established successfully, the NA, at the destination tile, sends an acknowledgement to the initiating node. After receiving the setup acknowledge message, the NA unblocks the allocated VC to make use of the established connection. Subsequently, traffic between two nodes with an established end-to-end connection is forwarded by the NA to the used VC. Thus, an end-to-end connection is used transparently by the source NA for all communication to the destination tile. Therefore, a VC reservation table is located in the transmission path of the i-NA.

Direct memory access In order to increase the data transfer efficiency, hardware support is provided by the i-NA. Therefore, a DMA unit is available in the network adapter to copy data between different memories of the architecture. It can be configured via memory mapped registers to perform linear DMA transfers. DMA units are restricted to only push data from local memory of the tile to a remote memory location. The restriction to push data and not pull from other tiles reduces the communication cost of the data transfer because the requests for data copying only utilize the local bus of the tile. A hardware pull DMA would cause additional load on the NoC and is thus emulated by the OS; using push DMAs.

Router management and status interface The routers of the invasive network on chip provide different capabilities for run-time configuration and status monitoring, as detailed later. However, this information must be accessed by the software running on the PEs of the tiles. Therefore, the i-NA provides an interface to access the registers of the i-NoC routers. A detailed definition of this interface is provided in appendix A.4. It is accomplished by mapping the registers into the local address space of each tile. Read and write accesses to these registers are forwarded by the NA to the router and vice versa. The i-NoC registers are mapped to the address range, which is remotely accessible. Thus, enabling access to registers of all routers from the architecture of each node. Details regarding the functionality of the i-NoC registers are provided in the following chapters.

3.3. Software

Invasive computing not only necessitates changes at the language and hardware level but also a compiler, which supports the InvadeX10 programming language. In addition, an operating system is required, which makes use of the available features of the InvasIC hardware. As such, the OS must be capable of executing invasive programs. The software perspective of invasive computing is briefly introduced in the following paragraphs.

3.3.1. Compiler

The heterogeneous invasive architecture and InvadeX10 necessitate a compiler that supports the new architecture and the programming language. An existing X10 source-to-source compiler, outputting C++ [100] or Java [240] code, was used as a basis. This compiler was extended to support the invasive principles and the constraint system, described in section 3.1.1. In addition to these frontend extensions, a backend was developed to support code generation for the invasive architecture.

For the LEON3 cores of the invasive architecture, code for the SPARC V8 *instruction set architecture (ISA)* must be generated. Therefore, the extended X10 compiler was equipped with a new backend [36]; targeting the graph-based intermediate representation known as *Firm*. This intermediate representation in turn is used in the *libFirm* compiler [102]. The libFirm infrastructure [36] comprises a collection of frontends, optimizations and backends that all operate on the Firm intermediate representation. From libFirm's point of view, the X10 compiler is just an additional frontend. A SPARC backend for libFirm was developed and used to generate code for the RISC cores[11] of the InvasIC architecture.

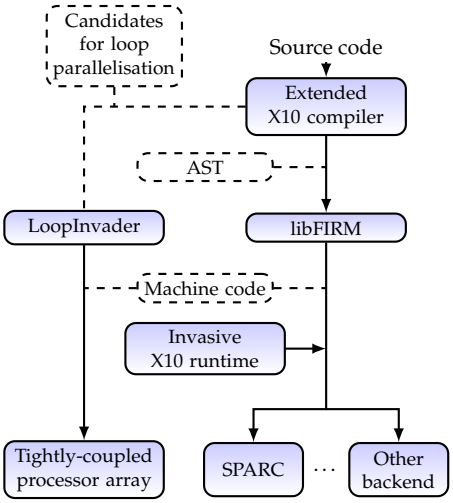


Figure 3.10.: Structure of the compiler framework for invasive computing architectures [244].

Plugging the extended X10 compiler and the libFirm compiler together enables continuous code generation from language to binary code. The flow of the

[11] i-Cores are specialized RISC cores, which also support the SPARC V8 ISA and can execute SPARC code generated by libFirm. Modified binutils are required to use the special i-Core features.

described compilation framework is illustrated in figure 3.10. During code generation in X10, the invasive program is linked against the *Invasive X10 runtime*. This runtime provides the OS support for resource allocations and execution of invasive X10 applications on the invasive architecture, as described in section 3.3.2.

Figure 3.10 also shows that it is planned to automatically select candidates of code to be executed on a TCPA. These candidates will then be forwarded to the LoopInvader to generate symbolic code for the TCPA architecture, as described in [246]. This would enable code generation for the entire invasive architecture in the same language, but remaining open for future work.

3.3.2. Operating System

The operating system for invasive computing is named *invasive run-time support system (iRTSS)*. It is a highly scalable native operating system implemented in a distributed way. One instance of this OS is executed on each core of the architecture. The single instances communicate with each other to form the distributed operating system.

A central constitution of iRTSS[12] is *OctoPOS* [188]. It is the link between higher-level software (application programs, X10 run-time system and agent system) and the invasive hardware (CiC and i-NoC).

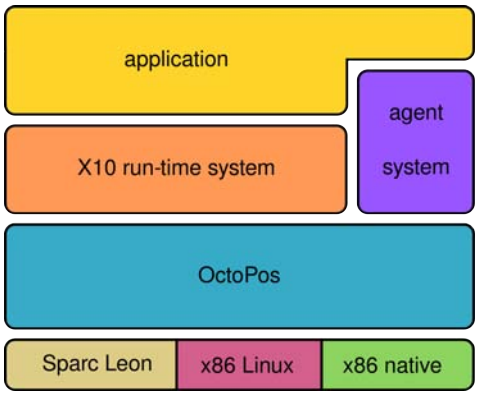


Figure 3.11.: OctoPOS system software acts as a broker between the X10 runtime system with its applications and the underlying hardware [244].

The role of OctoPOS is shown in figure 3.11. It provides an interface to the *X10 run-time system*. This interface includes all functions for memory management

[12]The iRTSS consists of the OctoPOS, the agent system and hardware abstraction layer. The hardware abstraction layer is not further described in this work).

(e.g. malloc and free), I/O support (e.g. printf or ethernet), i-let execution as well as communication primitives. The invasion process is managed by a special component of the iRTSS referred to as *agent system*. The agent system is responsible for resource management on behalf of the application. Therefore, *OctoPOS* provides another interface to the agent system. It is used by the agents to access status information (e.g. hardware monitors) and to allocate resources for an application during invasion phase. More details about the agent system are provided in section 3.3.2.1.

As shown in figure 3.11, OctoPOS support different hardware platforms. It can be used for the InvasIC SPARC architecture, an x86 architecture or as a user process within a Linux system[13]. With respect to the InvasIC architecture, OctoPOS has to support the special features of the hardware. Therefore, hardware drivers for the CiC and the invasive network on chip are provided. These drivers are needed in OctoPOS to make use of special hardware features, such as the invasion of communication resources or the infection of other tiles by the use of the CiC.

3.3.2.1. Agent-based Resource Management System

The agent system [140] is a subsystem of the iRTSS. It is a resource management system implemented in a distributed manner to enable scalability. The agent system consists of multiple agents (instances); each is either responsible for a region of the architecture and the respective resources or for an application and its hardware resources, as shown in figure 3.12. Agents communicate with neighboring or distant agents for bargaining resources.

The agent system plays a key role during invasion phase. An invasion request is handed over from the invasive application to an instance of the agent system (cf. figure 3.11). As described in section 3.1.1, this request also includes the constraints for invasion. The instance of the agent is now responsible for searching and allocating hardware resources that fit to the constraints. Therefore, it communicates with other agent instances and takes the status of the underlying hardware into account. The interfaces for communication and status information access are provided by the OctoPOS. When suitable resources are found, they are allocated on behalf of the application using interfaces of OctoPOS. Finally, a claim is assembled by the agent and returned to the application.

[13] The support of other platforms enables development and test of applications with the entire invasive software stack without having the InvasIC architecture available.

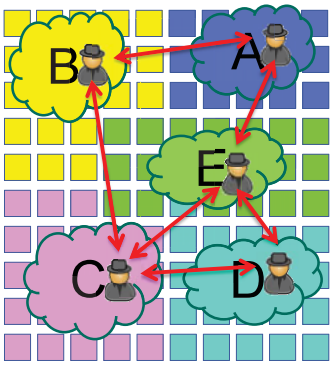


Figure 3.12.: Distributed agent system with five agents, each with a separate claim of resources [80].

3.4. Hardware Prototyping

On one hand, Prototypes are necessary for hardware testing. On the other hand, they are needed for early software development, as described in [FHMB14] and [FHB14]. Moreover, a prototype of the invasive architecture and the software stack (OctoPOS, agent system and invasive applications) is necessary to demonstrate the benefits of invasive computing. FPGAs are used for prototyping of the InvasIC architecture.

3.4.1. Single-FPGA Prototyping

In the first step, small prototypes of single components have been created using off-the-shelf FPGA boards, such as the Xilinx ML605 [272] equipped with a Virtex-6 LX240T FPGA. They are capable of implementing prototypes of the InvasIC architecture with up to four tiles. This is sufficient for hardware tests and initial software testing. However, an InvasIC architecture on a single FPGA board is too small to investigate concurrent applications, distributed management, scalability and additional aspects.

3.4.2. CHIPit Prototyping System

A professional prototyping platform, the Synopsys CHIPit Platinum Edition, is used for prototyping of the InvasIC architecture [BFH+12]. The platform mainly[14]

[14]Various other FPGAs are used for switching between the six main FPGAs and for maintenance.

consists of six Virtex-5 LX330 FPGAs. Extension boards, such as DVI, Ethernet, UART, DDR, SSRAM and others, can be plugged to the system and connected to the FPGAs. The prototyping system can be operated in standalone mode or connected to a host computer via the *Universal Multi-Resource (UMR)* bus. This bus can be used for transactor-based communication between the hardware design running on the CHIPit system and the software executed on the host computer[15]. In addition to the UMR bus, a *joint test action group (JTAG)* connection to the host computer is used for programming of the FPGAs.

The CHIPit system is used to setup a larger prototype. One limitation of the system is the available memory. FPGA block RAMs are used to model caches and small memories. The larger TLMs have to be created by FPGA-external *synchronous static random access memory (SSRAM)*. Thus, the number of available SSRAMs limits the number of tiles that can be instantiated.

Figure 3.13 shows a block diagram of the architecture built on the CHIPit system and its interfaces to the host computer. It is a 3x3 mesh architecture with six RISC tiles, one TCPA tile, one i-Core tile and one memory & I/O tile[16]. For this architecture three SSRAM, one Ethernet and one DDR extension board are used. Ethernet is used to visualize the system state (e.g. application mapping) and to transfer data, such as images or video streams. In addition, two UMR transactors enable code execution and debugging of the InvasIC architecture as well as hardware monitoring on the host computer. However, there are also other incarnations of the InvasIC architecture available for the CHIPit platform. A homogenous architecture with eight RISC tiles, one memory and I/O tile is also frequently used.

With respect to the NoC, multi-FPGA prototyping includes some difficulties and limitations. As shown in figure 3.13, the NoC links are used to bridge the gap between the FPGAs. The network is thus partitioned between the FPGAs. Currently, each signal between two FPGAs uses a separate I/O pin at each FPGA. This approach enables transparent partitioning from the design perspective, without additional delay in terms of clock cycles and without signal multiplexing being required. However, partitioning a design between multiple FPGAs presents two drawbacks: (1) the number of signals between two FPGAs is strictly limited and (2) there is an additional delay for the signals crossing the FPGA borders, which limits the clock frequency. These constraints have to be taken into account during design of the NoC.

[15] The UMR bus can also be used for co-simulation. In co-simulation mode, the prototype on the CHIPit system is coupled with a model or test bench running on the host computer. This was done in early phases of the network on chip development.

[16] DDR memory and I/O interfaces are connected to the same tile to have more resources available for compute tiles.

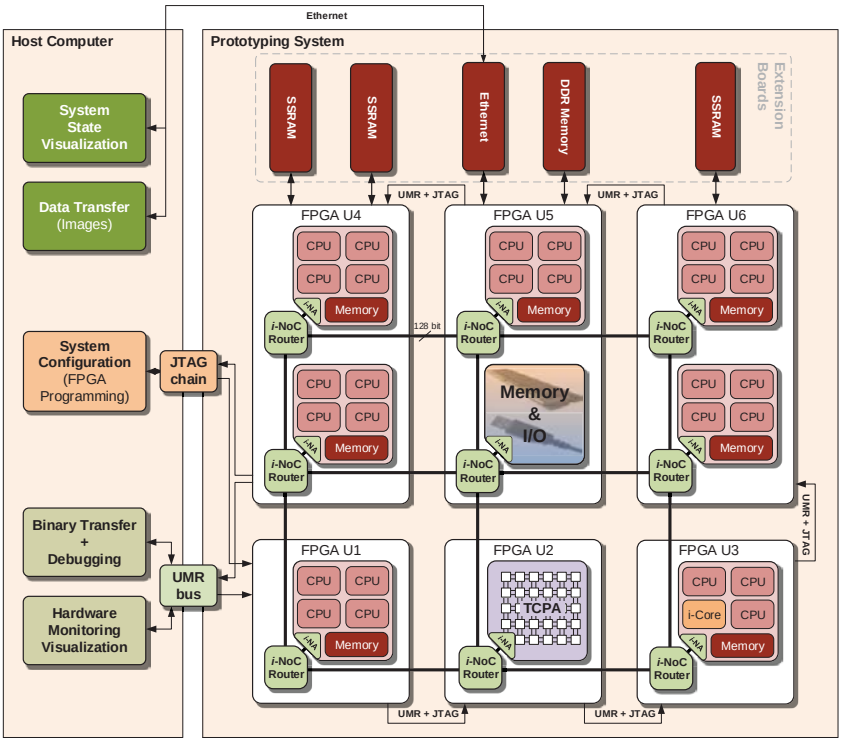


Figure 3.13.: Heterogeneous multi-FPGA prototype of the InvasIC architecture with 3x3 tiles.

Inspired from the drawbacks and limitations of the CHIPit platform, an alternative concept for prototyping of NoC-based architectures was investigated. It is briefly discussed in appendix A.1.

3.5. Summary

Parts of the present work have arisen in the context of the invasive computing research project. In this chapter, the basic principle of invasive computing and the invasive paradigm were first introduced.

Subsequently, an overview on the heterogeneous InvasIC MPSoC hardware architecture and its components was provided. The communication demands of

the different types of computation and I/O tiles were discussed and are used as a basis for the more detailed analysis; presented in section 4.1.4.

The software perspective and the distributed management concept, which is designed with a focus on scalability, were discussed in section 3.3. The distributed agent-based resource management system for the InvasIC architecture was introduced in section 3.3.2.1. Its region-based management concept is adapted for the NoC resource management schemes and the mechanisms presented in section 6.3, 6.4 and 7.3.

At the end of the chapter, multi-FPGA prototyping of the InvasIC hardware architecture was discussed. The special role of the NoC, which is partitioned between the different FPGAs of the prototype, was briefly explained.

The communication requirements of the InvasIC hardware architecture and the requirements of the distributed management concept are taken into account for the general concept of the proposed NoC design. This concept is generally presented in chapter 4, more specific aspects are discussed in the subsequent chapters.

4. Flexible NoC Architecture and Design Flow Concept

The design and implementation of a network on chip depends on many different constraints. This chapter is intended to identify and analyze the basic requirements on the communication infrastructure of a large, general-purpose, many-core architecture. The impact of these requirements on design decisions is discussed. In addition to these general requirements, the demands resulting from invasive computing are analyzed to substantiate the identified requirements. Subsequently, state of the art architectures are introduced and analyzed with respect to their suitability. Afterwards, the basic concept of the proposed network on chip[1] design is derived from the previous requirement analysis and the identified drawbacks of existing architectures. At the end of the chapter, the concept for a semiautomatic NoC design flow is presented to enable a flexible and comfortable usage of the proposed NoC design.

4.1. Communication Requirements and Constraints

There are various, different requirements for an on-chip communication infrastructure. They can be classified according to various criteria. One can distinguish between functional and non-functional requirements. Some requirements are design-specific or architecture-specific, others always apply. Some requirements refer to usability and performance, others result from technological constraints. Figure 4.1 gives an overview on general requirements for networks on chip. A distinction is made between functional and non-functional properties.

[1] In this and in the following chapters the term "network on chip" refers to a network of routers. The NI or NA as well as the nodes itself are not meant.

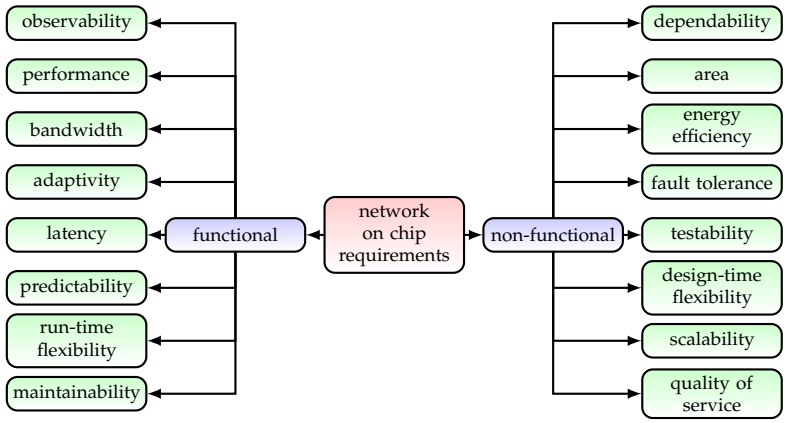


Figure 4.1.: Diverse functional and non-functional requirements to networks on chip.

4.1.1. General Communication Requirements

There are general communication requirements that apply to any communications infrastructure. The most fundamental requirement to a communication system is sufficient performance.

Definition. *The bandwidth and the latency of a communication system are defined as a measure of its performance.*

However, bandwidth and latency are not independent of one another. The bandwidth must be sufficient with respect to the demands of the connected components. Otherwise, latency can increase heavily due to bandwidth limitations and congestion. The correlation between bandwidth utilization and latency is shown exemplarily in figure 4.2. Bandwidth and latency requirements must be analyzed to scale the communication system properly. Especially for on-chip communication, a suitable dimensioning is essential for keeping the implementation cost in sight, as described in detail later.

Another general requirement of a communication system is its usability. The usability in turn depends on the purpose of the system. The supported communication modes, interfaces and protocols of the communications infrastructure play an important role. The support of point-to-point communication can be considered as a fundamental requirement of a network. Depending on the application, other communication modes, such as multicast or broadcast (see section 2.4.5.7), can improve the usability and performance. The protocol, which can be considered as the interface of the communication system, must be carried out in a way

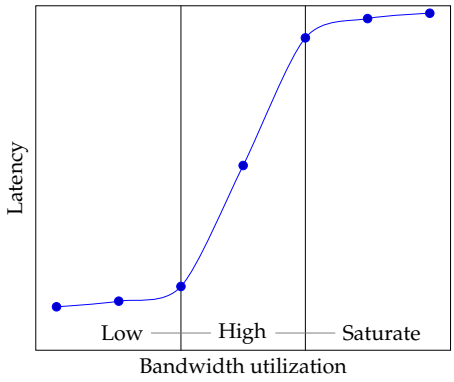


Figure 4.2.: Correlation between bandwidth utilization and latency in a packet switching network – *Low* utilization: very low latency, *High* utilization: latency is very sensitive with respect to changing load conditions, *Saturation*: very high latency due to congestion.

to balance between usability and flexibility. A flexible protocol may be more complicated to use from the software perspective, compared to a static protocol.

Flexibility of a communication's infrastructure can be identified as another basic requirement. A flexible communication system can be adapted at run-time, according to the current application. Thus, a flexible network may handle situations that have not been considered at design-time. However, the additional flexibility comes at a high price with regards to the implementation costs. The implementation costs play an important role especially for on-chip communication, as described in the following section. Adaptivity can be considered as a derivative of flexibility. An adaptive communication system is capable of self-adaptation, according to changing load conditions and other application-specific constraints.

Another requirement that applies to every system is the reliability or dependability. The fulfilling of reliability demands mainly depends on whether errors can occur during manufacturing or even at run-time. A design that can tolerate faults typically comes along with a significant implementation overhead, e.g. due to the necessity of redundancy. However, fault tolerance may be inevitable to ensure dependable communication.

Predictability, also known as quality of service, is required for timing (real-time) or safety critical applications but can also be used to improve the performance of other applications. In automotive or avionic communication systems [251] for instance, predictability is one of the most basic requirements. QoS requirements are met mostly via guarantees for bandwidth and latency. However, predictability is accompanied by an inefficient bandwidth utilization. The reason are pessimistic

assumptions that must be made to provide worst case guarantees for bandwidth and latency. Thus, efficiency aspects must be taken into account to limit the overhead for predictable communication.

4.1.2. On-Chip Communication Requirements

On-chip communication includes additional requirements or puts minor requirements in the focus. These requirements can result from economical, but especially from technological constraints that contain SoC design and silicon integration.

The limitation of the implementation costs is one requirement that becomes of great importance with respect to silicon integration. For an ASIC integration, the required area or the number of gates is used as a measure. The implementation costs are usually conditioned by the aforementioned requirements, such as performance, flexibility or reliability. To keep the implementation costs low, other requirements have to be limited and adjusted with respect to the application of the communication system. The template design, described in detail in section 4.3.5, facilitates balance between implementation costs and other demands. The implementation costs mainly include power and area.

The power budget and power density is another constraint that must be taken into account for on-chip communication. Today, the power budget of integrated circuits is not only limited in mobile devices due to the use of batteries. In deep submicron VLSI circuits, the power budget becomes more and more limited by physical constraints[2] as motivated in chapter 1. Compared to bus systems, networks on chip have a higher share of the total power consumption of the system due to their increased complexity. As such, low power consumption and power saving techniques are a basic requirement to NoCs. Power consumption is closely related to the complexity of a circuit or component. Thus, limiting or reducing the complexity is one way to ensure low power consumption. Another way is the use of power saving techniques, such as clock gating [271] or power gating [133].

Reducing the complexity of a system is a good way to save silicon area and to reduce the power consumption of a communication system. However, such a reduction is naturally accompanied by decreased performance or functionality. Thus, efficiency can be used as a measure. In order to estimate the efficiency of a system in general, its costs are compared with its performance. Referring to a network, the bandwidth BW can be taken as a good measure of the performance.

[2] Cooling of high power density devices is accompanied by ever-increasing challenges. In addition, power consumption must be reduced or limited due to economical and ecological reasons.

Using the bandwidth, the power efficiency (Eff_{Power}) of a communication system can be estimated as follows:

$$Eff_{Power} = \frac{BW}{P} \tag{4.1}$$

In equation 4.1, P is the power consumption of the system. Replacing the power consumption by the resource requirements R, the resource efficiency can be approximated as follows:

$$Eff_{Resource} = \frac{BW}{R} \tag{4.2}$$

Resource and power efficiency are good assessment criteria for the comparison of different implementation alternatives. Efficiency can be used to analyze dependent requirements in conjunction with one another.

Another technology and complexity dependent aspect referring to performance and throughput is the clock frequency at which the communication system can be operated. Generally speaking, the clock frequency is proportional to the bandwidth of a NoC or on-chip communication system. Thus, keeping the critical path in mind is important from the performance perspective.

In addition to these physical and technological requirements, architectural requirements that are specific to on-chip communication also exist.[3]. The main concern is the usage of the network on chip. – Is it only used to connect all cores of the system to a common shared memory? Is it used for core-to-core communication? Is it used in a homogenous or heterogeneous architecture? How do the communication requirements of the software running on the system look? What is the behavior of the software with respect to communication? – In answering these questions, precise architectural requirements could be derived.

Such architectural requirements impact the topology choice for the NoC and result in performance, usability, flexibility and scalability requirements, as described in section 4.1.3. Here are some examples for choices resulting from architectural demands: In case of frequent core-to-core communication a regular topology might be beneficial, whereas a tree topology might be used for communication with a common shared memory. Heterogeneous architectures might require an irregular topology. The software running on the system can have a huge dynamic, which would necessitate a flexible or adaptive network. If the processing cores have a high performance, this must be reflected by the communication infrastructure. Consequently, design-time adaptivity of a network on chip is desired to fulfill different architecture-specific requirements.

[3] Architectural requirements do not only exist for on-chip networks. The author decided to not refer to more general architectural requirements, as this would be too circumlocutory.

4.1.3. Communication Constraints of Scalable Architectures

All previously discussed requirements can be satisfied by conventional bus systems. However, in terms of scalability, NoCs are superior compared to bus-based interconnects, as previously analyzed and discussed in chapter 1 and 2. In many-core architectures, addressed in this work, scalability of the communication infrastructure is a fundamental requirement. To ensure scalability, different design decisions must be carefully considered. For instance, the topology plays an important role. In section 2.4.2, the suitability of different topologies with respect to scalability was already discussed. Thus, scalability limits the choice of a topology. Regular topologies, such as mesh or torus, are characterized by the fact that the complexity of a node is almost independent of the size of the overall architecture. Regular topologies can be scaled better compared to irregular topologies (e.g. tree or star). In addition to the topology, the implementation of the decision making procedures in the NoC has a significant impact on scalability. Decentralization and distribution of decision making is necessary to enable scalability. Consequently, routing and communication resource allocation (e.g. virtual channel allocation) must be carried out in a distributed way. Therefore, the necessity of scalability impacts many other architectural aspects, such as QoS support. In conclusion, scalability has an impact on basic design decisions of NoCs. A scalable NoC is accomplished by additional implementation cost due to the necessity of distributed management.

4.1.4. Communication Requirements of Invasive Computing

The previous subsections summarized general requirements on communication systems. However, some of these general requirements are contradictory. In order to derive basic design decisions for the NoC implementation, the requirements arising from invasive computing, detailed in chapter 3, are now analyzed.

The most basic idea of invasive computing is the invasion of resources by applications. Aside from memory and computation resources, the communication resources shall be invadable as well. An invasion is an exclusive allocation of resources by an application. However, due to the fact that a network on chip is normally shared by all nodes, it cannot be split up into separate regions for the application[4]. Thus, bandwidth allocation techniques must be used instead of exclusive spatial invasion for communication resources, as detailed in section 6.2.

Another requirement, resulting from the InvasIC architecture, is the support for memory communication. Since the main memory is connected to the processing elements by the NoC, transparent memory communication must be supported

[4]In contrast, invasion of processing cores or memories can be implemented straightforward by reserving regions of the address space or a set of processing cores exclusively.

for data and instruction fetching. It must be possible to access the memory from each node at any time. Connection-oriented communication cannot ensure memory communication in any situation (e.g. at start up no connection may exist, or the required connection cannot be established, due to resource limitations). Thus, connectionless best-effort communication must be supported to ensure memory communication in any situation. Best-effort communication can be used as a fallback solution, if exclusive invasion of communication resources is not possible.

Observability is another requirement of the NoC in the InvasIC architecture. The current status of the NoC is taken into account by the system software to determine the mapping of new application, as described in section 3.3.2.1. The monitoring information shall not only be used by the software, but also by the NoC hardware itself for self-optimization.

Self-optimization and self-organization shall be implemented in the InvasIC architecture to reduce the burden of management. Communication monitoring may be used to trigger and control self-optimization features. Load balancing, or the optimization of the resource allocation in the NoC, could be triggered based on monitoring. Self-optimization and self-organization should be carried out in a distributed way to ensure scalability. For such a distributed implementation, NoC internal communication may be required to orchestrate the self-optimization among the components of the network.

Another aspect of the InvasIC architecture, which must be taken into account for design of its communication system, is the heterogeneity of the architecture. The architecture consists of different tiles with different communication demands. Memory or TCPA tiles may be used by multiple RISC tiles in parallel, resulting in a huge aggregated bandwidth. These increased bandwidth requirements must be considered while dimensioning the NoC to avoid any bottlenecks resulting from heterogeneity of the communication.

Invasive computing addresses future many-core architectures with dozens of cores. Thus, scalability is an essential requirement to the invasive network on chip. As discussed in section 4.1.3, scalability impacts the choice of the topology, routing algorithm and resource allocation scheme. The demand for scalability must be taken into account for the bandwidth allocation scheme used to facilitate invasion of communication resources. Additionally, the aforementioned self-optimization and self-organization strategies shall reduce the management overhead of the architecture and improve its scalability.

Distributed management of the NoC is another requirements that is accompanied by scalability. The distributed operating system (see section 3.3.2) necessitates decentralized management of the hardware. This involves the allocation of communication resources during invasion phase and other configuration capabilities of the network. The requirement of distributed management must be taken into

account during design of the NoC itself and the resource allocation scheme required to invade communication resources. All interfaces for configuration and status information access must support the distrusted management scheme of the OS.

4.1.5. Summary

The previous discussion presented a huge diversity between the various requirements. Some requirements can be readily reconciled with each other (e.g. low power consumption and low implementation cost), while some are contradictory (e.g. high bandwidth and low implementation cost). Some requirements apply to any communication system (e.g. dependability), others are specific to the use case (e.g. observability, predictability or distributed management). Due to the diversity of the requirements, they must be analyzed for each new NoC design in order to derive proper design decisions. This applies especially for the basic characteristics of a network on chip, such as performance or implementation costs. An overdimensioning of the bandwidth would result in excessive implementation costs, whereas undersizing of the NoC bandwidth would result in communication bottlenecks, which could have a negative impact on the performance of the entire system. However, it is evident to only fulfill necessary requirements which limit power consumption and the complexity of a system. An evaluation of the selected design parameters is consequently necessary to verify their suitability. Such an evaluation can be very time consuming looking at the list of different requirements and the possible design choices. Different design choices must be compared against each other to find the optimal solution (e.g. with respect to power or resource efficiency). The short time to market, required for today's integrated circuit designs, necessitate a fast way for requirement analysis and design parameter evaluation. Once the design parameters are selected, the actual implementation has to take place, based on the previous analysis. During this design process, time plays an important role.

A framework would be desirable, enabling fast and easily evaluation and implement of architecture-specific networks on chip. The NoC design must be flexible and extendable to adapt it according to design-specific requirements, such as the extensions that may be required to fulfill the special requirements of invasive computing. Newly developed extensions of the existing NoC could become a part of the framework and used for future designs. The concept of a framework-based design process, which uses a flexible design kit, will be pursued in section 4.4. The framework shall reduce the time between an initial requirement analysis and the final hardware design.

4.2. State of the Art NoC Architectures

According to the previous section, a NoC design is desirable, which allows to fulfill alternating requirements depending on the target MPSoC architecture and the applications. Consequently, a flexible, adaptable and feature-rich NoC design is required. As a result of these findings, the suitability of existing NoC architectures as a basis for a holistic and flexible NoC design and evaluation framework are analyzed in the following paragraph. Such a framework necessitates design-time adaptivity and evaluation capabilities. The designs are also analyzed with respect to basic requirements of future general-purpose many-core architectures, such as scalability, distributed management, run-time flexibility and reliability. Additionally, it is discussed whether existing NoCs are suitable for the InvasIC architecture, as introduced in section 3.2.

4.2.1. SoCBUS

SoCBUS was originally introduced in 2000 as a circuit switching on-chip network [266]. It was extended by a packet switching component that is used to setup the circuit switching connections in a distributed way. Therefore, distributed minimal routing is used, as described in detail in [267]. SoCBUS supports the reservation of communication resources due to the use of circuit switching. In general, it may be deemed scalable due to the use of distributed routing and decentralized connection setup. The use of a 2D mesh topology enables scalability. However, Wiklund and Liu have stated in [267] that their system is not suitable for general-purpose computation platforms. As such, there is a high probability of blocking while circuit switching connection setup, if no static scheduling of the connections is used. If a static scheduling can be calculated at design-time, SoCBUS can be used efficiently. However, such a static scheduling is not possible in a highly dynamic system, addressed by this work. Consequently, SoCBUS is no option for a base communication infrastructure of a general-purpose many-core architecture.

4.2.2. Hermes

Hermes [180] is a wormhole packet switching mesh network on chip topology. It uses distributed XY routing and can therefore be considered to be a scalable communication system. An extension of the Hermes NoC is presented in [174]. Virtual channels are added to improve throughput and latency. Due to the use of packet switching, it has a higher run-time flexibility compared to SoCBUS. The Hermes NoC is used in the HeMPS architecture [39], which is an MPSoC

architecture available as a SystemC model, thus making the Hermes NoC suitable for evaluation purposes. The focus of the HeMPS framework is the evaluation of parallel software, e.g. the evaluation of task mapping alternatives. However, the customization capabilities of the Hermes model are very limited. It only provides very essential parameters, such as link size or number of VCs to be changed at design-time. The Hermes NoC uses a protocol that does not enable communication resource reservation[5]. Thus, Hermes cannot be used as a communication system for the InvasIC architecture, due to the missing capabilities for reservation.

4.2.3. SoCIN

The SoCIN network on chip [277] has similarities to the Hermes architecture. It is also a packet switched network with wormhole flow control. Due to the use of source XY routing, its scalability is limited. SoCIN claims to be parameterisable at design-time to build NoCs with different costs/performance ratios. Therefore, it supports different dimensions of the architecture, different link/channel sizes and buffer depths. SoCIN does not use virtual channels and therefore does not support explicit resource reservation; necessary for the invasive architecture and for QoS support. The limited scalability, due to the use of source routing, as well as the missing support for communication resource reservation make SoCIN an unsuitable candidate.

4.2.4. Æthereal

Æthereal [214, 87, 92, 93] is one of the most popular NoC architectures, designed and investigated at Philips Research since the year 2000. Æthereal combines a packet switching network with input buffers and a circuit switching network with VCs. The VCs are scheduled by the use of TDM. Both meshed networks share the same link. The circuit switching network enables connection-oriented guaranteed service communication. The wormhole packet switching network is used for best-effort communication [214]. Best-effort communication utilizes the link bandwidth that is not used by GS connections. The sharing of the link between BE and GS communication improves the link utilization. The pipelined TDM circuit switching network uses contention-free routing [87, 92]. Due to the pipelining of the circuit switching network, different time slots must be allocated for a connection in different routers. However, to plan a good time slot allocation for all circuit switching connections of a system that is using Æthereal global knowledge is required. Only a centralized allocation scheme is capable of calculating an

[5]The protocol used in the Hermes NoC requires fixed packet lengths that are known ahead of transmission.

optimal assignment of all connections to time slots. This assignment is named contention-free routing by the authors and can be completed at design-time or at run-time [87, 92]. On one hand, design-time allocation provides no flexibility at run-time. On the other hand, run-time assignment would necessitate a global view of the system state and a centralized decision making and management, which limits the scalability of the system. Consequently, Æthereal does not provide the run-time flexibility and scalability required for a general-purpose many-core architecture.

4.2.5. Nostrum NoC

The Nostrum network on chip [177, 178] uses packet switching and has a scalable 2D mesh topology. It supports best-effort and GS communication and uses hot-potato routing[6]. So-called virtual circuits are used to establish an end-to-end connection with guarantees. A virtual circuit is implemented by the use of TDM scheduling. A fraction of the link bandwidth is allocated for each connection [178]. In [177], virtual circuits are used to enable two concepts named *looped container* and *temporally disjoint networks*. The combination of these two concepts enables hard guarantees for bandwidth. However, the virtual circuits are set-up semi-statically, meaning that the route is calculated at design-time, but the bandwidth is variable at run-time. This semi-static establishment of virtual connections provides no run-time flexibility and limits the management capabilities. Thus, the Nostrum concept is inappropriate for general-purpose many-core architectures due to the missing flexibility and management capabilities.

Nostrum is available as an ASIC implementation. In addition, the Nostrum network on chip simulation environment [167] provides a SystemC-based simulation model of the Nostrum NoC. The simulation environment has a large set of parameters for topology, switching, routing algorithm and traffic generation. Thus making it suitable for design space exploration.

4.2.6. QNoC

The QNoC architecture is presented in [29]. QNoC is a wormhole packet switching NoC supporting regular mesh and irregular grid topologies of heterogeneous architectures. Routing is performed in a distributed way using minimal XY routing[7]. The novelty of QNoC is the support of different *service levels (SLs)* for data

[6] The packet transmission duration in hot-potato routing [73] is only one clock cycle, i.e. the length of packets is one flit.

[7] It remains unclear in [29] how irregular topologies, as claimed by the author, can be supported using XY routing.

transmission. Therefore, a service level is assigned to each packet according to its type. Four types of service levels are supported: signaling, real-time, read-/write and block-transfer. A priority ranking is defined for the four service levels, where signaling is given the highest priority and block-transfer the lowest. A separate static buffer for each SL is located at the input port of each router. Data transmission is scheduled according to the priorities, enabling low latency for the packets with higher SLs. However, it does not enable communication resource invasion or bandwidth guarantees. QNoC is only available as a simulation model for OPNET [43]. A synthesizable hardware implementation is not available.

4.2.7. Xpipes

Xpipes is a parameterizable NoC template supporting regular and irregular topologies [21, 57]. The XpipesCompiler tool is used to configure the template design. According to [57], Xpipes has been implemented in SystemC, at the cycle-accurate and signal-accurate level. Although the SystemC description is synthesizable, the tool support for SystemC *high-level synthesis (HLS)* is currently very limited. Consequently, the design can hardly be used for ASIC or FPGA realization. A simulation environment for design space exploration is not reported.

The Xpipes NoC uses wormhole switching and *look up table (LUT)*-based source routing; limiting its scalability. Due to the use of source routing, the run-time flexibility of the network is also limited. It does not provide any run-time adaptation capabilities. However, the network adapter design of Xpipes enables to manage the architecture in a distributed way from the software perspective, enabling distributed OS management schemes. Xpipes does not provide QoS support. Consequently, it is not possible to reserve communication resources.

Xpipes is the only NoC that is reported to provide fault tolerance, utilizing error-detection and correction scheme for transient errors. A sophisticated fault tolerance model and on-line testing procedure for Xpipes is presented in [126].

4.2.8. Kilo-NoC

Kilo-NoC [97, 96] is a scalable NoC architecture. It applies packet switching with virtual channels. Kilo-NoC uses elastic buffers [176] to reduce the implementation cost for buffering. Multidrop express channels [95], using point-to-multipoint channels, are applied to provide full intra-dimension connectivity. The Kilo-NoC concept assumes a heterogeneous architectures consisting of hundreds of processing nodes. These nodes share a limited number of memory nodes. The express channels are used in a topology-aware manner to reduce their number and thus, the implementation costs. Express channels are only used to access the

memory nodes from all other nodes with QoS. The architecture must be divided at design-time into non-overlapping domains to achieve topology-aware QoS. Therefore, the applications executed on the architecture must be known at design-time, resulting in a very limited run-time flexibility. All QoS requirements must be known at design-time, a run-time reservation of communication resources is not supported. This is not sufficient for general-purpose architectures, where communication patterns and QoS requirements are normally not known at design-time. Another drawback of Kilo-NoC is its availability. According to [97] the architecture is currently only available as a high-level simulation model. A cycle-accurate simulation model or a HDL model for FPGA or ASIC synthesis is not available. This reduced the evaluation capabilities of the design significantly.

4.2.9. Summary

The previous discussion of existing NoC architectures showed that relevant aspects for communication in future general-purpose many-core architectures are addressed. According to section 4.1 the most important communication requirements of such architectures are[8]:

- **Scalability**: Capability of the NoC architecture to be used for an MPSoC with a large number of nodes or cores.
- **Reservation/QoS**: Architecture supports allocation of communication resources to enable predictable communication and invasion of communication resources.
- **Run-time Flexibility**: The network on chip is flexible at run-time with respect to changing communication requirements of different applications.
- **Management**: Distributed management is supported from the software perspective to reduce the management overhead within a large architecture.
- **Reliability**: The architecture supports fault tolerance to handle transient and permanent faults, which are expected for future silicon technology nodes.
- **Design-time Flexibility**: The NoC architecture provides the possibility to be adapted with respect to the requirements of a specific incarnation of the architecture.
- **Availability**: A simulation model of the architecture must be available for an efficient evaluation of implementation alternatives. A synthesizable hardware model must be available for ASIC or FPGA implementation.

[8] General communication requirements, such as bandwidth or latency are not considered here. They can be fulfilled by each of the discussed architectures by adjusting basic parameters, such as link or buffer size.

	Scalability	**Reservation/QoS**	**Run-time Flexibility**	**Distrib. Management**	**Reliability**	**Design-time Flexibility**	**Availability** (Simulation)	**Availability** (Hardware)
SocBUS	✓	✓	♦	✓	✗	✗	✓	✗
Æthereal	♦	✓	♦	✗	✗	✗	✗	✓
Hermes	✓	✗	✓	✓	✗	♦	✓	✓
Nostrum	♦	✓	♦	✗	✗	✓	✓	✓
QNoC	✓	♦	✓	✓	✗	✗	✓	✗
SoCIN	✗	✗	♦	✓	✗	♦	✗	✓
Xpipes	♦	✗	✗	✓	✓	✓	✗	♦
Kilo-NoC	✓	✗	♦	✓	✗	♦	✓	✗

Table 4.1.: Comparison of existing NoC architectures with respect to different fundamental requirements of future general-purpose many-core architectures – Requirement fully satisfied (✓), Requirement partially satisfied (♦), Requirement not satisfied (✗).

Table 4.1 summarizes the fulfillment of the identified requirements by existing NoC architectures. Utilizing the table, it is evident that none of the existing architectures can fulfill all requirements, as identified for a scalable general-purpose many-core architecture. Most architectures lack in flexibility at design-time. However, design-time flexibility is necessary to adapt the NoC rapidly to specific requirements of an architecture. Another aspect that is not supported by most of the existing architectures is the run-time flexibility. In order to enable efficient execution, it is required to adapt the communication at run-time according the requirements of the applications, which are executed concurrently. With respect to the communication sub-system of the InvasIC architecture, an exclusive reservation of communication resources by application is required. However, the run-time allocation of communication resources is also not supported by the majority of existing NoC architectures. Table 4.1 shows that none of the existing architectures can meet all of the identified requirements. Consequently, none of the existing architectures can be used directly. Thus, different aspects of existing architectures must be combined to create an architecture that fulfills all necessary requirements. According to table 4.1, the Hermes NoC was identified to meet most of the requirements. Hermes was analyzed in more detail as part of a student

work [Ker13] to clarify whether it can be used as a basis for this work. However, the detailed analysis showed that the basic concept and implementation of Hermes does not enable the necessary extensions. Especially the router architecture of the Hermes NoC was not suitable because of its sequential packet processing. It was therefore decided to not use an existing architecture. Instead, selected aspects and concepts that have proven their efficiency in existing architectures will be combined to create a new fundamental architecture concept.

4.3. Fundamental Architecture Concept

The fundamental concept and goal of this work is a flexible NoC-based communication infrastructure for future general-purpose many-core architectures. In order to adapt and use a network on chip for different architectures under different constraints, it must be flexible at design-time. The NoC can be either used for FPGA prototyping or for an ASIC implementation. The NoC must be scalable to use it for large architectures, which can be expected in the near future [33]. If the NoC provides functionality, which must be configured or controlled by the operating system at run-time, distributed management is required to ensure scalability. Self-optimization features could reduce the burden for management by the OS and applications. In contrast to application-specific systems, general-purpose architectures must be flexible at run-time to deal with changing requirements of different applications. This flexibility must be reflected by the communication infrastructure. It must cope with changing load conditions and may need to adapt itself to the current communication requirements. Some applications could profit or necessitate guarantees for bandwidth and latency, which can be carried out by exclusive allocation of communication resources. Such an allocation strategy is also required to enable the invasion of communication resources, as discussed in chapter 3. Despite all of these requirements, the efficiency must not be forgotten. In particular, the energy efficiency plays an ever increasing role, especially in mobile battery-powered system. Therefore, the system must not be oversized. In order to meet these divergent requirements for different architectures, a flexible base design is necessary. The design must support an easy design-time adaptation and configuration. Evaluation capabilities are required to verify the fulfillment of architecture-specific requirements for a designated configuration.

In the following subsections, the fundamental concept and design decisions of such an architecture will be identified and discussed.

4.3.1. Switching Scheme and Quality of Service

An on-chip network of a general-purpose many-core architecture must efficiently support both pre-scheduled and dynamic traffic. For numerous applications, large data flows are relatively static and demand high-bandwidth with low latency and low jitter. For example, a flow of video data from a camera input to an MPEG encoder is entirely static and requires high bandwidth with predictable delay. However, dynamic communication, which is not known prior to its occurrence, must be supported. Examples for such dynamic communication are: memory communication resulting from cache misses or communication between OS instances for resource management. Pre-scheduled communication can efficiently be enabled by circuit switching networks, as explained in section 2.4.3.1. The connections could be established prior to execution, using static information about the communication of the application. However, managing circuit switching in a distributed way includes additional design challenges, as discussed in section 2.4.3.1. The drawback of circuit switching is that a connection setup phase is required. Connection setup must be successful before data transmission can take place. This connection setup is associated with a high latency which leads to a poor performance from dynamic communication. Communication can be completely blocked if connection setup fails (e.g. due to over utilization). This can also result in low performance or even lead to deadlocks. In order to prevent such deadlocks and to avoid the overhead of connection setup for dynamic communication, best-effort communication is required. Packet switching networks are predestined for BE communication, which is to be expected in general-purpose architectures. Thus, packet switching is preferable for dynamic communication. Consequently, the flexibility of packet switching is inevitable for the desired communication infrastructure. Packet switching is the means of choice for the network presented in the following chapters.

In addition to best-effort packet switching communication, a concept is presented in section 6.2, which enables quality of service and end-to-end connections with adjustable bandwidth for packet switching networks. This strategy is also used to allocate communication resources in the InvasIC architecture. In general, it can be used to provide hard guarantees for safety, or timing critical applications by the use of virtual channels [58]. In addition, virtual channels are an efficient way to increase the maximum utilization of a network, as shown in section 5.4. This is another aspect that justifies the use of VCs. However, it is worth noting that VCs include a substantial implementation and power dissipation overhead. Therefore, in section 7.4, a power saving mechanism for packet switching NoCs with virtual channels is presented.

Due to the irresistible advantages of circuit switching, such as high energy efficiency [269], a circuit switching extension for the primary packet switching network is introduced in section 5.5.2. This extension utilizes energy efficient

circuit switching communication for pre-scheduled communication and packet switching for dynamic traffic. In case of an over-utilization of the circuit switching network, packet switching may be used as a fallback solution to avoid blocking and deadlocks.

4.3.2. Scalability and Distributed Self-Optimization

Scalability is necessary from the hardware perspective to enable large architectures, addressed by this work. A simplistic design decision with respect to scalability is the topology. As discussed in section 2.4.2, regular topologies, such as meshes or tori, provide better scalability compared to irregular ones. Therefore, the router design presented in the following is mainly designed to be used in a regular topology. Nevertheless, it can be used with slight adjustments for other topologies.

The topology is closely related to the routing strategy, which must be selected according to the topology of the network. In fully connected regular topologies, there might be multiple candidate paths between two nodes that could be used for data transmission. Therefore, routing plays an important role with respect to load balancing and performance. The implementation of the routing strategy affects the scalability of the system. Centralized routing does not scale from the software management perspective. Source routing does not scale from the protocol perspective and also necessitates global knowledge hardly available in large distributed architectures. In order to ensure scalability, the burden of routing must be borne by the hardware in a distributed way. Local status information can be consulted to take the routing decision in each router. The proposed NoC design uses distributed FSM-based routing for packet and circuit switching to enable scalability. Different minimal static and adaptive routing schemes are implemented, as described in section 5.1.4. Hardware load monitors are used to take proper decisions in case of adaptive routing, as described in detail in section 5.1.4 and 5.5.3. These load monitors are also used to enable a self-optimization strategy, named rerouting. It is used to relocate existing end-to-end connections to balance the load of the NoC, as described in detail in section 7.1.

Self-optimization capabilities of the hardware, such as the aforementioned rerouting strategy, is one basic idea of invasive computing. Therefore the general hardware design shall enable self-optimization to reduce the software management overhead for improving the scalability of an architecture. The aforementioned monitoring infrastructure is one key concept for distributed self-optimization. It enables decision making based on local up-to-date status information. In addition to a distributed monitoring infrastructure, NoC internal communication is necessary to implement distributed self-optimization strategies, such as the rerouting strategy introduced in section 7.1. Thus a light-weight NoC layer is

used, as detailed in section 5.5.1. Self-optimization strategies for performance and power optimization, which fit into the overall concept, are presented in chapter 7.

4.3.3. Region-based Distributed Management

In order to support a scalable software management concepts, such as the OS and agent subsystem presented in section 3.3.2, the NoC must provide capabilities for decentralized management. A scalable operating system management concept necessitates a NoC hardware, which can be managed in a distributed way.

Access to status information of the NoC hardware is required by the software in order to make appropriate decisions for resource allocation or task mapping. The monitoring units, used by the hardware for self-optimization, shall be reused for this purpose. Thus, each node can independently obtain all necessary state information from the hardware monitors without involving other operating system instances. For efficient access and collection of the required monitoring data, hardware supported data collection is introduced in section 7.3. It is deemed a region-based concept because it enables to divide the NoC architecture into regions, which are used and managed independently of each other. This region-based strategy is inspired and supports the agent-based software management concept; used in invasive computing (see section 3.3.2.1). The *virtual network (VN)* support, introduced in section 6.4, is another region-based management scheme for networks on chip. It allocates a portion of the NoC bandwidth of a cohesive region for the application, executed in this region. VNs enable an application in an environment that provides bandwidth guarantees for the application in its entirety. Virtual networks can be established at run-time according to the mapping and shape of the application. This necessitates run-time management by the system software. The management and configuration of virtual networks is carried out in a distributed way in order to enable distributed run-time management.

The other quality of service capabilities of the presented architecture, introduced in chapter 6, also must be managed and established at run-time. A distributed management scheme for these quality of service features is necessary to support the use of distributed system software. The concept of such schemes is detailed in section 6.2.1.5, 6.3 and 6.4.1.1 for the supported QoS mechanisms.

The fault tolerance scheme, introduced in the following and described in detail in chapter 8, necessitates a software controlled reconfiguration of the NoC hardware in case of hardware defects. The reconfiguration interface enables to perform the reconfiguration from each node without requiring a central control instance. Each node of the architecture can be used for reconfiguration, in order to respond to errors. A central authority is not needed.

4.3.4. Fault Tolerance and Dependability

Taking the technological perspective into account, networks on chip must be fault tolerant to handle transient and permanent faults, which are expected for future technology nodes [203]. The applied strategies for fault tolerance are constrained by power, area and performance limitations. A more detailed motivation and overview on state of the art is provided in chapter 8.

In order to deal with transient faults, well-known retransmission schemes [9] can be used independently of architectural details. In regards to permanent faults, architecture-specific fault tolerance schemes are necessary. Thus, a strategy for dealing with permanent faults in the proposed NoC architecture is discussed and investigated in chapter 8. It includes a strategy for detection and localization of permanent errors in the complex VC-based packet switching network with QoS support. When faulty routers are localized, a configurable lightweight second layer network is used to bypass these routers transparently and to substitute their communication bandwidth. The second layer network shall substitute the bandwidth, which is lost, when disabling the faulty router. Alternatively, the proposed concept for deactivation of faulty routers can be used to disable parts of the network in case of low utilization. This enables the usage of the reliability concept for power saving, as described in section 8.3.1.2.

4.3.5. Design-time Flexibility and Adaptability

The proposed NoC design shall combine different aspects and mechanisms for flexible communication, QoS support, self-optimization and reliable, fault tolerant communication. Not all of these mechanisms are required by each MPSoC architecture where the network on chip is used. Regardless, different architectures have different fundamental communication requirements, such as bandwidth and latency. The power and area budget of the NoC also depends heavily on the desired architecture. The proposed NoC architecture shall be suitable for a broad range of multiprocessor systems on a chip. Therefore, it should be flexible enough to use it for different architectures with different requirements. This necessitates a NoC design that is flexible at design-time. Such a design could be adapted according to the specific requirements of the architecture.

In order to enable the design-time flexibility that is necessary to adapt the design to changing requirements, a template-based approach shall be used. The base architecture, introduced in chapter 5, and the extensions, presented in chapter 6, 7 and 8, are designed in a modular and parameterizable way. This approach shall enable and simplify the necessary design-time adaptations of the network on chip. The base-router and the functionalities for QoS support, self-optimization, reliability or the circuit switching extension are designed as separate modules. Each

of these modules has multiple configuration options for design-time adaptation. This modular approach, which is denoted as a design template, enables to omit functionality, which is not required by the current implementation. Therefore, the different modules are designed in a way that they can be easily combined to fulfill the specific demands of the desired architecture. In addition, this modules approach shall enable a simple extension of the architecture by new features. The template-based design approach shall enable fast and easy creation of architecture-specific incarnations of the NoC. It is intended to meet the requirements of a short time to market for NoC-based MPSoC architectures.

The modular design template is available as a simulation model and as a synthesizable hardware implementation. The simulation model, introduced in section 5.2, enables a fast design space exploration to select a configuration, which fulfills the stated functional requirements, such as latency and throughput. The synthesizable HDL model, presented in 5.3, subsequently can be used to evaluate the fulfillment of non-functional requirements, such as power and area consumption but also to implement the NoC in the desired target technology. Both, the HDL and the simulation model, are used in the semi-automated design flow, introduced in section 4.4.

The design template concept enables fast creation of highly diverse networks on chip. On the one hand, it can be used to build complex networks, such as the i-NoC [HZW+14] presented in section 3.2.2. On the other hand, it can also be used to implement very lightweight communication infrastructures, such as the liteNoC. The liteNoC incarnation can beat other existing NoC implementations, which claim to be resource efficient for FPGAs, in terms of performance and resource requirements, as shown in [PQHW+13].

4.4. Semiautomatic NoC Design Flow

Bus systems, introduced in section 2.3.2, and standardized interfaces have proven to enable efficiency realization of complex systems. They are used to connect *intellectual property (IP)* blocks to form complex systems on a chip, as described in section 2.1. This design methodology enables a short time to market, necessary in the fast-paced semiconductor industry. For future systems, it is expected that bus systems will be replaced by NoCs. Therefore, a NoC implementation flow must also fulfill the requirement of a short time to market; staying competitive in the industries fast pace. However, a network on chip is accompanied by an increased complexity which in turn includes an increased design space. This design space has to be evaluated to make optimal design decisions with respect to the requirements of the architecture being built. An efficient way for evaluation of different implementation alternatives is necessary to enable a short time to

market and to reduce the development costs of a new system. Therefore, a design with flexibility at design-time and run-time is necessary to adapt it to different requirements. In order to quickly analyze whether a particular configuration meets the requirements of the architecture, a rapid evaluation method is required.

4.4.1. Evaluation and Implementation Flow

This work proposes a semiautomatic design flow, which enables fast and accurate evaluation, followed by semi-automated generation of a synthesizable NoC architecture. It targets homogenous and heterogeneous general-purpose many-core architectures. Such architectures come along with additional difficulties during design phase; compared to specialized architectures, where the communication behavior might be known at design-time. For a general-purpose architecture, the communication behavior depends on the current application and can have a huge variability at run-time. These unpredictable communication properties are accompanied by additional challenges during the design process. Different corner cases must be evaluated at design-time with respect to communication, ensuring that the architecture can fulfill diverse requirements of different applications.

A simulation framework is used to enable a fast comparison of different implementation alternatives under different load situations and traffic scenarios. The simulation environment is cycle-accurate to ensure accuracy of the simulation results with respect to the hardware implementation. It uses different traffic generation models to evaluate the NoC architecture under different corner cases, which can appear at run-time. The simulation framework is described in detail in section 5.2. It is mainly used to analyze the fulfillment of the functional communication requirements, summarized in section 4.1. Therefore, the NoC simulation model is initially configured with a seed configuration that is expected to meet the requirements. After simulation, the measurements of different traffic scenarios are analyzed and verified as to the fulfillment of the functional requirements. If this verification fails, the previous configuration is modified and the simulation-based evaluation is restarted. This process is continued until a valid configuration is found.

Once a valid configuration has been found, which fulfills the derived functional requirements, the fulfillment of non-functional requirements (e.g. area and power consumption) must be analyzed. Therefore, a model-based approach, such as ORION [125], may be used. However, such a model-based approach naturally lacks accuracy. Due to the fact that the proposed framework is capable of generating synthesizable HDL code, accurate target-technology-specific area, power and clock frequency estimations are possible. The HDL model, introduced in 5.3, is used to evaluate the fulfillment of the non-functional requirements of the current configuration. The HDL model is configured according to the final result

of the preceding functional requirement analysis. The generated model is then verified automatically for accuracy by the use of an HDL simulation. Afterwards, the generated design is synthesized for the target ASIC or FPGA technology. The synthesis process provides accurate values for the implementation cost of the design. Post-synthesis simulation is used to verify the final netlist of the design and to obtain accurate numbers for the power consumption of the design. The outcome of the synthesis and the following netlist-simulation are then compared against the non-functional requirements of the design. If the analysis of the synthesis results show that the non-functional requirements are not met, other parameters must be chosen and evaluated. If the requirements are satisfied, a valid configuration is found and the generated HDL model can finally be used as an outcome of the flow. The generated HDL model is suitable for FPGA and ASIC implementation.

Figure 4.3 provides an overview of the semi-automated design flow, described above. It is currently composed of automated and manual procedures. The deduction of configuration parameters, based on the requirements and the feedback of previous iterations, must be performed manually. Additionally, the fulfillment of functional and non-functional requirements is analyzed manually. All remaining steps of the flow are fully automated.

4.4.2. Application of the Evaluation and Implementation Flow

The proposed evaluation and synthesis flow is used extensively hereafter. It is suitable for fast and accurate *design space exploration (DSE)*. As a proof of concept, a case study is presented in section 5.4. It analyzes the impact of different basic design parameters.

The framework and the associated HDL implementation are used for evaluation of the concepts, which are presented in chapter 6, 7 and 8. It was also used to generate the i-NoC for the InvasIC architecture, described in section 3.2. In addition, it was used to evaluate and generate appropriate communication infrastructures for two other architectures. The first architecture is the MOLEN architecture [258]. For MOLEN, a lightweight NoC was required to extend the existing bus-based communication infrastructures, as described in [PQHW+13]. A comparison to other resource-saving NoC implementations showed the superiority of the NoC, generated with the proposed implementation flow [PQHW+13]. The third architecture, which uses a NoC that was generated by the use of the proposed flow, is the KAHRISMA architecture [141][KSHB11]. The extension of the KAHRISMA architecture by a network on chip is described in [37].

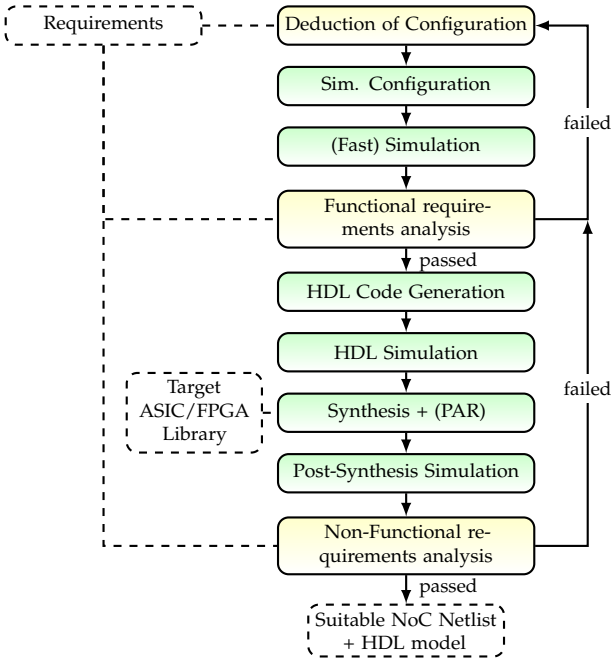


Figure 4.3.: Semi-automated, iterative, requirement driven NoC evaluation and implementation flow – Green node: fully automated step, Yellow node: manual step, Dashed node: input/output of the framework.

4.5. Summary

In this chapter, general requirements for networks on chip were identified and discussed. As elementary functional characteristics, the performance of the communication system was identified. The most important non-functional properties were identified to be the power and area consumption, but also the reliability of the network. In addition, scalability, distributed management, self-optimization and quality of service support are demands on the communication infrastructure of future MPSoCs. These demands apply also for the InvasIC architecture, introduced in chapter 3.

Based on these findings, state of the art NoC architectures were discussed and analyzed in section 4.2. The analysis, summarized in section 4.2.9, showed that none of the existing architectures meet the sum of identified requirements. The limited flexibility at design-time, which existing architectures have in common, is taken as a motivation for the subsequently presented concept.

The fundamental concept of this work, introduced in section 4.3, is motivated by the limitations of existing NoC architectures. It shall meet the sum of requirements, such as scalability, predictability, self-optimization, distributed management or energy efficiency, which were identified to be relevant for the communication infrastructure of future general-purpose MPSoC architectures. In the concept section, elementary design decisions were discussed, which are necessary to create an architecture that can meet the sum of requirements. However, a specific architecture may only meet a subset of the requirements. Consequently, a flexible design is desired and proposed, enabling to adapt the network on chip at design-time according to the requirements of the architecture to be implemented.

In the final part of this chapter, a NoC design flow was introduced. It uses the flexible and configurable future-rich NoC design, proposed previously, to enable semi-automated generation of suitable NoC incarnations for specific MPSoC architectures. The proposed design flow uses the simulation framework and the HDL model of the NoC, described in detail in chapter 5. The design flow enables to evaluate and generate a suitable network on chip with a low effort for the system designer. This methodology shall meet the demands of a short time to marked, which is an important factor of today's and future MPSoC design processes. The proposed methodology is used extensively for the evaluations presented in the following chapters. In addition, it has already been applied successfully to generate networks on chip for three different MPSoC architectures.

5. Basic Architecture Realization

This chapter describes the basic network on chip router architecture, its HDL implementation and the simulation framework. A case study is presented, showing the feasibility of the proposed evaluation and design flow by the use of the HDL and the simulation model. In addition, basic extensions of the router are discussed. The components and methods described in this chapter are the basis for the concepts and extensions introduced in the following chapters.

5.1. Scalable Basic Router Design

The basic router design is inspired by state of the art networks, as discussed in the previous section. It implements the basic design decision, as identified in section 4.3.1. The resulting router architecture is nearly identical to the router design used in the Single-chip Cloud Computer, which was described in section 2.2.5.2. Thus, it can be said, that the basic router design represents the state of the art. Its concept and implementation is described in detail below.

Switching The router design relies on packet switching to enable best-effort communication. In general, packet switching includes high implementation cost, which in turn are associated with high power consumption. There are two reasons for the relatively high power consumption of a packet switching network. First, the arbitration mechanisms, required within the router, consume a noticeable share of resources and power. Second, the buffers used to store data that cannot be forwarded immediately have high resource requirements and power consumption. Therefore, arbitration and buffering must be investigated in detail to limit their implementation costs. Optimized building blocks are used to optimize buffer and arbitration resources efficiently, as described in section 5.3. In addition, conceptual optimization is taken into account to improve router costs. The choice of the switching scheme has a significant impact on the buffer requirements, as described in section 2.4.3.2. Compared to the other switching schemes, wormhole switching has the lowest buffer requirements. In order to reduce the implementation overhead of packet switching, which is typically dominated by the size of the buffers, wormhole switching is applied. Another

benefit justifying the use of wormhole switching, is the facilitation of end-to-end connection [27], utilized for the QoS concept introduced in section 6.2. The utilization of wormhole switching affects the protocol of the network and impacts its functioning, as described in section 5.1.2.

Buffering Packet switching requires buffering of packets which cannot be forwarded directly, as detailed in section 2.4.3.2. The buffers can either be placed at the input [130] or at the output port [25] of the router. However, placing the buffers at the input port is beneficial from the implementation perspective. The reason, which justifies the placement of the buffers at the input ports, is that the buffer output ports are accessed more frequently by the router's logic. The buffer output port is not only accessed in case of data transmission, but also in case of routing, reservation or arbitration decisions. When input buffers are used, no inter-router communication is necessary to access the buffer output port. Thus, buffers located at the input port may be used to reduce the amount of inter-router communication. Additionally, input buffers make up the critical path of the router. It starts normally at the output port of the buffer and continues into the arbitration unit. Splitting this path between two routers would impact the maximum clock frequency, due to increased wire lengths. A reduced clock frequency in turn would increase the latency and reduce the bandwidth of the NoC. As previous discussed, the buffers are placed at the router input ports, implemented as FIFOs. In the proposed router design, there is not only one buffer per port, but multiple buffers are used to develop virtual channels, as detailed in section 5.1.1.

Structure and Functioning Figure 5.1 gives a rough overview of the structure of the basic router, briefly describing the functioning of the router. A more detailed explanation of the functioning, as well as the implementation, is provided in section 5.1.5. As discussed earlier, the separate *buffers* of the VCs are located directly at the input ports of the router. The number N of router ports can be changed at design-time to support different topologies, such as 2D mesh or 3D torus. Once the flits of a packet arrive at a VC buffer of the input port, the destination address is forwarded to the *routing unit*, described in detail in section 5.1.4. The routing unit determines the output port to be used by the packet. According to the determined output port, a reservation request is forwarded to the *reservation table* of the respective port[1]. The reservation table selects a free VC and reserves it for the corresponding input port and VC. If all virtual channels are occupied, the reservation process is delayed until a VC becomes available. All entries in the reservation table are taken into account for scheduling

[1] At this point, figure 5.1 is very inaccurate for simplification. Between the routing unit and the reservation table, a transition from the input zone of the router to its output zone takes place. The reason for this is that the routing unit corresponds to an input port and the reservation table to an output port. Section 5.1.5 provides an accurate description of the implementation.

via *transmission control* unit. It controls the access to an output port by arbitrating between the transmission requests of the VCs, which are currently assigned to the respective output port. Therefore, the transmission control unit accesses the reservation table and verifies if data is available in the buffers for transmission. A round-robin scheduling scheme is used for transmission control, as described later in section 5.1.1.1. According to the scheduling decision, the transmission control unit switches the crossbar to forward the selected data from the input buffer to the output port. The used unidirectional crossbar is classically implemented as a set of multiplexers. The output port of each router is directly connected to the input port of the neighboring router or a network adapter. Credit-based flow control is used to ensure availability of buffer space in the neighboring router. As described in section 2.4.4.1, credit-based flow control makes it possible to pipeline the links between the routers without appreciable performance degradation[2].

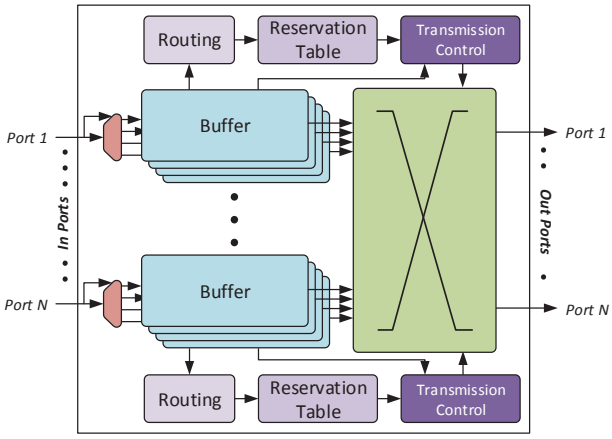


Figure 5.1.: Simplified structure of the basic NoC router with virtual channels.

5.1.1. Virtual Channels

The concept of *virtual channels (VCs)* [58] improves the effective bandwidth of a network, as described in detail in section 2.4.3.3. In conjunction with wormhole switching, it enables end-to-end connections in packet switching networks. Thus, virtual channels are necessary to enable the QoS concepts, as presented in chapter 6. The implementation of VCs necessitates separate buffers. Thus, a separate buffer must be available per virtual channel.

[2]The performance degradation resulting from pipelining depends on the buffer size. The larger the buffer, the lower the performance degradation.

In addition to the buffers, the use of VCs impacts the implementation of other router components. The reservation table, which contains the assignment of a virtual channel at the input port to a virtual channel at the output port, must be extended compared to a router without VCs. This is necessary to manage multiple packet transmissions in parallel, according to the VC concept. Therefore, the reservation table of each output port must be equipped with one separate entry per virtual channel. These entries are used to manage the data transmission for each output port.

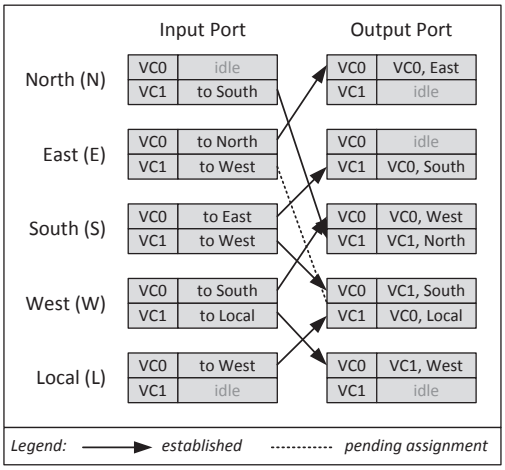


Figure 5.2.: Example of a virtual channel assignment for a router incarnation with two virtual channels and five ports.

Figure 5.2 shows an example of a virtual channel assignment for a five port router[3] with two virtual channels. The left column in the figure shows the occupancy of the input ports or buffers, respectively. For each VC buffer, the routing decision is given as the ID of the output port, which will be used to forward the packet. In the example, some of the VCs are idle, meaning that they do not contain data. According to the routing decision, each occupied VC of the input port must be assigned to a virtual channel at the output port, calculated by the routing unit. The assignment is managed by the reservation table of the respective output port; containing an entry for each virtual channel. These entries are shown in figure 5.2 on the right. The routing table stores the input port and an identifier for the VC that is currently assigned. The information stored in the routing table is used by the transmission control unit to manage the data transmission separately for each

[3] A five port router is typically used in meshed topologies. The four ports, connecting the router to its neighboring routers are typically named by their cardinal directions. The fifth port is named *local* port. It is connected to the NA to attach the node or tile.

output port. The input VC and port identifiers, selected for data transmission, are forwarded by the transmission control unit to the crossbar to set the multiplexers accordingly. This enables to forward the data from the input buffers to the output port. In addition to the data, the identifier of the output VC is passed to the neighboring router, or NA attached to the port. There it is used to store the data in the respected buffer of the input port. Consequently, a virtual channel for transmission is always determined by the output port of a router. Thus, the transmitting router controls the input buffers for each VC located at the input port of the neighboring router or network adapter. A VC allocation is negotiated again in each router by the reservation table.

The example, provided in figure 5.2, contains established and pending assignments. Established assignments represent successful reservations of a virtual channel at the output port. Pending assignments couldn't be utilized, due to an occupancy of all VCs at the corresponding output port. In the example provided, such a pending assignment exists for the packet located in the input buffer of *VC1* at the east input port. It waits for a reservation of a virtual channel at the west output port. At the west output port all VCs are already reserved for other transmissions. Thus, the transmission from the east port is delayed until one of the VCs of the west output becomes available. Such a delay of a transmission is often referred to as blocking. The higher the number of available virtual channels, the lower the risk of blocking. However, due to the implementation overhead of VCs, their number must be chosen with great care, as demonstrated in section 5.4.2.2.

5.1.1.1. Arbitration

The implementation of virtual channels necessitates an arbitration or scheduling process for data transmission, which wouldn't be required without VCs. This arbitration is performed by the *transmission control (TC)* unit, shown in figure 5.1. It decides which virtual channel is selected for transmission. Therefore, only the virtual channels, which meet the conditions for transmission, are taken into account. These conditions are:

- **Reservation:** A reservation must exist for the respective VC in the reservation table of the output port.
- **Data available:** Data must be available for transmission in the VC input port buffer. This information is retrieved from the input buffer status signals.
- **Credits available:** Buffer space must be available in the downstream router that will receive the data to be transmitted. This is tested by accessing the credit counters available for each output VC.

These conditions are evaluated for each virtual channel by the transmission control unit. If all conditions are met for more than one VCs of an output port, an

arbitration must take place. To enable a fair access to the shared link by all VCs, a round-robin arbitration is used. The evaluation of the constraints in advance to the arbitration decision enables a most efficient link utilization.

The decision making process is implemented within the transmission control unit. It evaluates the previously discussed conditions. The arbitration process implies a noticeable implementation complexity. This complexity results in the emergence of the critical path in the router design, limiting its clock frequency. Pipelining is used to break up this path, as detailed in section 5.1.3. However, a high number of supported virtual channels can limit the clock frequency of the router, as analyzed in section 5.4. Additionally, VCs can improve the throughput of the system, which will be investigated in section 5.4.

5.1.2. Network Layer Protocol

The router design utilizes wormhole-switching. Thus, packets are divided into equal sized flits, as described in detail in section 2.4.3.2. The router is capable of transmitting one flit per cycle. Consequently, a packet consisting of N flits, can be transmitted within N cycles in best case[4].

A typical packet consists of a head flit, several body flits and a tail flit, as shown in figure 5.3. The usage of tail flits is necessary to enable "infinite" packet sizes necessary for end-to-end connections.[5] Each flit contains a *control flag* prefix that indicates whether a flit must be interpreted by the router or not.

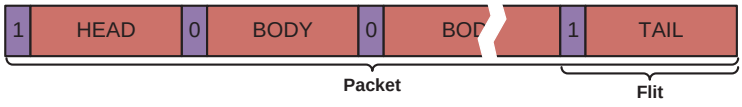


Figure 5.3.: A typical packet consisting of a head, several body and a tail flit – Each flit contains a *ctrl. flag*.

Figure 5.4 shows the general structure of a flit. If the control flag is set to 0, the flit is a body flit that does not require interpretation by the router. The entire payload of the flit can be used to transmit payload. The flit and payload size P depends on the link width W_{link} and can be configured at design-time according to the

[4] The best case is a situation where no arbitration between multiple VC is required. The pipeline delay of the router is not taken into account.

[5] Another protocol concept, where the head flit contains the packet size, does not allow end-to-end connections. Such a concept is implemented in the Hermes [180] NoC for example.

bandwidth requirements of the system. Taking the control flag into account, the payload size of a body flit can be calculated as follows:

$$P_{body} = W_{link} - 1 \tag{5.1}$$

However, a payload size is most efficient if it is a power of two, according to the data word size of the architecture. Therefore, the link width W_{link} is normally set to $2^n + 1$ with an arbitrary value of n, which suits the requirements of the architecture. If the control flag is 1, the flit contains control information, consuming part of the space of a flit.

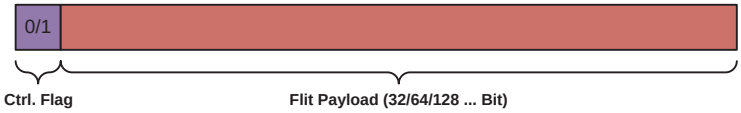


Figure 5.4.: General structure of a flit, which can either be a body flit (*ctrl. flag*=0) or a *control flit* (*ctrl. flag*=1).

A control flag equal to 1 indicates the necessity of extracting information from the flit in each router and processing it. The general structure of a control flit is shown in figure 5.5. It can either be a head or a tail flit, but also indicates a special *short packet*. A short packet consists of a single flit and therefore contains both, the control information of a head and a tail flit. It enables to transmit very short messages, which fit into the payload field of a single control flit. A short packet is indicated by a head and a tail bit set to 1. For head and tail flits either the *head* or the *tail bit* is set, respectively.

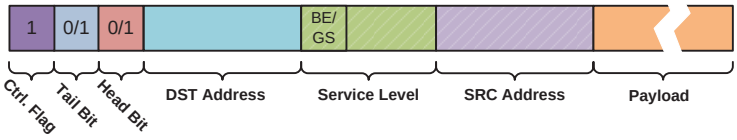


Figure 5.5.: Structure of a *control flit*, which could be either a head or tail flit or a *short packet* (consisting of only one flit).

A normal head flit is indicated by setting the head bit to 1 and the tail bit to 0. The head bit then triggers the reservation process within the router. As shown in figure 5.5, these two bits are followed by the destination address, which is given as X and Y coordinates[6] in the current implementation for mesh and torus

[6]For a 3D integration, the destination address field is extended to include Z- coordinates in addition to X and Y.

topologies. The destination address is forwarded within the router to the routing unit to calculate the output port, which will be taken by the packet. This field is mandatory, but may be followed by additional optional fields. These fields are necessary for guaranteed service communication and are only required if QoS support is enabled at design-time. In that case, the *BE/GS* bit of the *service level* field is mandatory. It indicates whether the head flit is part of a conventional BE packet or used to establish a GS connection. If this bit is set to 1, it denotes a GS header and as such, further fields are necessary. The SL field contains further information for bandwidth allocation, as described in detail in chapter 6. Additionally, the address of the node initializing the GS connection is encoded in the *source address* field of a GS header. The source address is necessary for end-to-end flow control and for some adaptive routing schemes, such as odd-even turn (see section 5.1.4.2). The remaining space of the head flit can be used for payload. For a BE header of a NoC incarnation without QoS support, the payload size can be calculated as follows:

$$P_{head,BE} = W_{link} - \lceil \log_2(DIM_x) \rceil - \lceil \log_2(DIM_y) \rceil - 3 \tag{5.2}$$

The dimensions of the network, DIM_x and DIM_y, determine the size of the destination address field. Referring to equation 5.2, the size of the payload field in the head flit depends on different constraints. For large link sizes or in case of very small network diameters, noticeable space for payload is available in the header. This space can be used to reduce the number of flits of a packet.

A tail flit triggers the deletion of the reservation table entry for the respective packet within the router. For a tail flit, only the tail bit is required.[7] The remaining space in the flit can be used for payload. Thus, the payload size of a tail flit is as follows:

$$P_{tail} = W_{link} - 2 \tag{5.3}$$

Equation 5.3 indicates the very low protocol overhead resulting from the use of tail flits. The payload fields of a head and a tail flit provide noticeable space. The space may be used for payload in order to reduce the number of required body flits.

Table 5.1 summarizes the fields shown in figure 5.5. These fields represent the protocol interpreted by the routers of the network.

[7] The head bit is not required, due to the fact that the state machine in the router is aware of an ongoing transmission. So the flit with tail bit set to 1 could not be a short packet, which would necessitate a head bit in addition.

Field	Description	Size [*bit*]
Ctrl. Flag	Indicates a control flit (head or tail) if set to 1.	1
Tail Bit	Triggers release of the resources reserved for transmission.	1
Head Bit	Indicates a head flit and triggers the reservation and resource allocation process within each router.	1
DST Address	Is used by the routing unit to calculate the direction and output port of the packet.	$\lceil \log_2(DIM_x) \rceil + \lceil \log_2(DIM_y) \rceil$
BE/GS bit	Used in case of QoS support to specify the type of header.	1
Service Level	Provides information for bandwidth allocation (see section 6.2).	$\lceil \log_2(SL_{max}) \rceil$
SRC Address	Used to identify the source node of a transmission (e.g. for end-to-end flow control).	$\lceil \log_2(DIM_x) \rceil + \lceil \log_2(DIM_y) \rceil$
Payload	Is used for the payload of the packet.	variable

Table 5.1.: Fields of the network layer protocol encoded in head flits.

5.1.3. Pipeline Model

Pipelining is utilized in the router design to improve the timing and consequently, the clock frequency. The number of pipeline stages is configurable at design-time according the requirements of the architecture. Optional pipeline stages can be omitted to reduce the processing latency in terms of clock cycles. However, this might reduce the achievable clock frequency and must be traded off carefully.

The router design can have up to five pipeline stages that are summarized in table 5.2. The first pipeline stage results from the input buffers, which have a delay of one clock cycle. This pipeline stage is mandatory taken by all flits independent of their type. The second and the third pipeline stage are only necessary during reservation phase. Thus, they are only taken by head flits. The first of these two pipeline stages is carried out by an arbitration unit, which is necessary to arbitrate between concurrent reservation requests for the same output port. This unit was not previously mentioned, but will be described in detail in section 5.1.5. The pipeline registers within this arbitration unit are optional and can be disabled at design-time to reduce the additional reservation latency in terms of cycles. The third pipeline stage, which is the second in the reservation process, is located in the reservation table. Adding a new entry to the reservation table, according to the previous routing decision and arbitration decision, requires one cycle. Since the reservation table must be implemented by the use of registers, this pipeline stage

cannot be omitted. The last two pipeline stages of the router must be traversed by all flit types. The fourth stage results from pipeline registers in the arbitration unit of the transmission control unit. These registers are optional and can be omitted on demand. The fifth and final pipeline stage is also optional and originates from registers located at the output ports of the router. These registers are used to relax the timing of the links connecting the routers. Table 5.2 summarizes the pipelining of the router.

Stage	Description	Flit Type	Optional
Input Buffer	The input buffers have a natural latency of one cycle.	all	no
Routing	The routing and arbitration process ahead of the reservation table is pipelined.	header	yes
Reservation	Adding an entry to the reservation table takes one cycle.	header	no
Scheduling	The scheduling process, implemented in the transmission control unit, can be pipelined.	all	yes
Output	Pipeline registers can be enabled at the output port to relax timing for the links of the network on chip.	all	yes

Table 5.2.: Mandatory and optional pipeline stages of the router. Two stages are taken by head flits during the reservation process.

The five pipeline stages of the router result in a per hop latency of five clock cycles for best-effort packet switching communication. If optional stages are omitted, the latency is reduced accordingly. The pipelining is implemented in a way that each router port can process a new flit every cycle. Thus, a throughput of one flit per cycle and port is achievable. The processing latency of one router for a single packet of size S, given as the number of flits, can therefore be calculated as follows, for the best case[8]:

$$L_{router,best\_case} = P_{total} + S - 1 \tag{5.4}$$

P_{total} is the total number of pipeline stages of the router, which can vary between 2 and 5 for the current implementation. Due to the application of the wormhole switching concept, the processing of packets is also pipelined between neighbor-

[8]The best case refers to a situation where the reservation process is successful immediately and where no other VCs of the respective output port are used.

ing routers. Therefore, the latency of an entire transmission path of H hops can be calculated as follows:

$$L_{path,best\_case} = (P_{total} \cdot H) + S - 1 \leq \sum_{i=0}^{H-1} L_{router,best\_case} \tag{5.5}$$

Equation 5.4 and 5.5 show that the best case transmission latency is dominated by the pipeline delay for packets with short length and by the packet sizes S in case of long packets. In addition to this best case latency analysis, worst case latency is examined in section 5.1.6.

5.1.4. Modular Distributed Routing

The usage and implementation of a routing strategy can impact performance, scalability and flexibility of a network, as described in section 2.4.5. Different requirements may necessitate the choice of different routing algorithms. To supporting various routing algorithms, the routing strategy is implemented modularly in the proposed router design. It is carried out as a separate module, which can easily be replaced, to use different distributed routing algorithms.

In case of distributed routing, the routing module is responsible for determining an output port to be taken by the respective packet, based on its destination address. The packet is subsequently forwarded according to the routing decision to the next router. In each router, the routing process is repeated accordingly until the packet arrives at the destination node. At the destination node, the packet leaves the NoC through the local port. The hardware implementation, described in section 5.3, currently provides a routing module for static XY routing and adaptive odd-even turn routing. However, the simulation framework, presented in section 5.2, supports a wide range of adaptive routing algorithms.

The routing module is located at the output port of the input buffers of each port. It is shared between the VCs of an input port by the use of an arbitration unit, as described in section 5.1.5. The routing module uses the destination address of a packet to calculate which output port shall be used. For routing, the destination address is extracted from the head flit, located in the input buffer of the router, and forwarded to the routing unit of this input port. In order to simplify the implementation of the routing algorithm, the destination address is encoded in the form of Cartesian coordinates in the packet header, as detailed in section 5.1.2. The routing algorithm uses the destination address and the address of the router itself. Optionally, status information can be considered to determine an output port for a packet. The routing decision is then used to trigger the reservation of a VC in the reservation table of the respective output. The implementation for XY routing and odd-even turn routing is explained in the following paragraphs.

5.1.4.1. XY Routing

XY routing [185] is a static routing algorithm for 2D mesh or torus topologies[9], which ensures deadlock-freedom and minimal paths. It can be implemented with low overhead in a distributed way; ensuring scalability. Consequently, XY routing is used in many existing NoC implementations. Its good scalability and low cost of implementation make XY routing a suitable method for this router design. Hence, a XY routing module was created for the router design, shown in figure 5.1. It uses the XY coordinates of the destination node and the coordinates of the current router[10] as inputs. A combinational circuit is used to determine the output port. First, with the highest priority, it is determined whether the packet is at the destination node. If this is not the case, the X coordinates are taken into account. If the X coordinate of the router matches the destination coordinate, the Y coordinate is considered. The routing decision is emitted in the form of a port ID. The ID is then used to forward the reservation request to the reservation table of the respective output port.

5.1.4.2. Adaptive Odd-Even Turn Routing

Different adaptive routing algorithms have been evaluated as part of a supervised student's work [Bul12][11]. As a result of this investigation, odd-even turn routing [46] emerged as the most suitable adaptive routing scheme with respect to performance, implementation complexity and deadlock-freedom. Thus, it was implemented as an HDL module for the proposed router design. In contrast to static routing, odd-even turn may offer different valid paths between a pair of nodes. With respect to the routing decision, taken by a single router, alternative paths may result in multiple valid output ports for a single packet. Thus, at run-time it must be decided which port is to be used. Therefore, a selection strategy must be used within the router, as discussed in section 2.4.5. Such a selection strategy was also implemented in [Bul12], as now described in detail.

The adaptive odd-even turn routing module has the the same interfaces as the XY routing module aside from a few additional inputs. It also uses the destination address of the packet and the coordinates of the router. In addition, the source address of the packet is used to make a routing decision. In order to make the routing decision, two rules must be observed according to [46]:

[9] 3D topologies can also be easily supported by extending the topology, protocol and routing strategy by a third dimension. In the following 2D topologies are assumed for simplicity.
[10] The router has the same XY coordinates as the tile/node attached to its local port.
[11] In the following, aspects investigated and implemented as part of supervised student work, are always marked accordingly.

- **Rule 1:** A packet is prohibited to take an east-north turn at any nodes located in an even column, and it is not allowed to take a north-west turn at any nodes located in an odd column.
- **Rule 2:** A packet is not allowed to take a south-west turn at any nodes located in an odd column, and it is prohibited to take an east-south turn at any nodes located in an even column.

In addition to these general rules, the source address is taken into account. This enables the creation of a minimal odd-even turn routing. The routing decision is taken by a six stage combinational circuit. The first stage, with the highest priority, determines whether the packet is at the destination node. If this is the case the local port is the only valid output port. Otherwise, the second routing stage is entered. It verifies whether a packet is already in its destination column. If this is not the case, it is decided in a third stage whether a packet is eastbound or westbound with respect to its destination node. Westbound packets can always be forwarded through the west output port, but might also use the north or south port, if the current coordinates of the router are even. This is verified by stages number four and five of the westbound branch and might result in two candidate ports for a packet. The eastbound branch of the routing decision tree includes not only two, but three stages. In the first sub-stage (stage four), it is analyzed whether a packet already reached the destination row. If this is the case, it must be forwarded eastwards. Otherwise it could take the north or south port in every odd column of the NoC or if the packet is still in the source column. If the destination column is odd or if the distance to the destination node is greater than one it also must be forwarded eastwards.

The previous discussed minimal odd-even turn router algorithm calculates either one or two valid output ports for each packet. If there are two valid output ports for a single packet, it has to be decided, which of the two ports shall be used. This decision is taken by a selection strategy implemented as a combination circuit. It is attached to the output of the odd-even turn routing module. The selection unit uses the two candidate ports as inputs. In addition, utilization information, obtained from status information and hardware monitors, are used to select the output port. For decision making, the current VC utilization and the link utilization of the latest monitoring period is taken into account by the selection unit. The used monitoring infrastructure is described in section 5.5.3. The VC utilization is evaluated by the selection unit with the highest priority. The candidate port with lower VC utilization is selected to balance the VC usage of the ports. If both candidate ports have the same number of available VCs, the link utilization is taken into account. Then, the port with the lower link utilization within the current monitoring period is selected. In the unlikely event of an identical load at both ports, the first port is used.

Figure 5.6 shows the realization of odd-even turn routing with a monitoring-based selection strategy. Odd-even turn routing can be easily replaced by another routing unit. The monitoring infrastructure, introduced in section 5.5.3, is used for port selection based on the current utilization of the router ports.

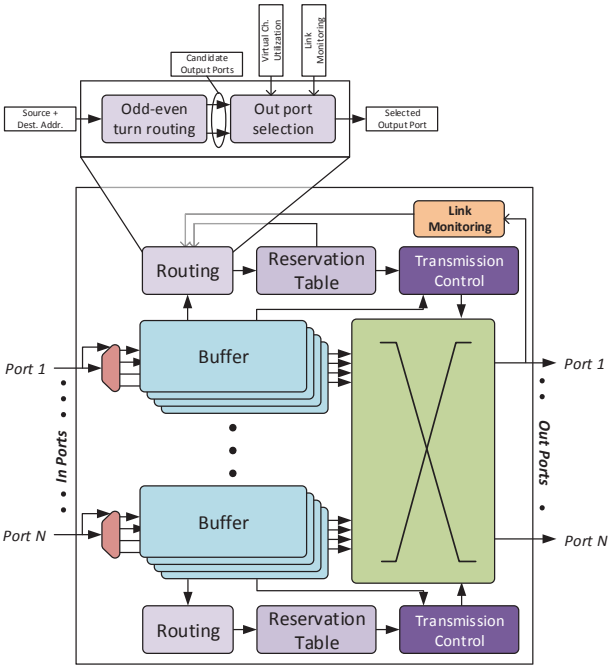


Figure 5.6.: Modular adaptive routing support for the router design – monitors are used to select an output port.

The detailed investigation in [Bul12] shows that the proposed adaptive routing can compete with existing routing schemes, such as DyAD [117], in terms of performance. In contrast to DyAD, the proposed selection strategy takes virtual channels into account. Aside from the cost of implementing the monitoring infrastructure[12], the overhead for employing the adaptive routing scheme is very low, as investigated in [Bul12].

[12]Monitoring might be necessary anyhow for self-optimization or observability. Consequently, it must not be counted as implementation costs for adaptive routing.

5.1.5. Implementation and Functioning

In the previous sections, the basic characteristic and the functioning of the router have been described. Some details have been deliberately left out for simplicity. These neglected aspects of router designs will now be explained. The structure and functionality of the router, described in the following paragraphs, reflects the corresponding hardware implementation precisely.

Figure 5.7 shows the detailed structure of the proposed NoC router design. It consists of an *input*, *crossing* and *output* zone. The input and output zone are present once per input and output port respectively, with the crossing zone existing once. It is located between the input and output zones; connecting them flexibly. The physical interface of each router input and output ports comprises four signals, as described in section 5.1.5.1. At the input port, the `req`, `data` and `vc` signal are used to sort the flits into the input buffers. The `vc` signal serves as a select signal for the respective *VC input buffer* and controls the *demultiplexer (DEMUX)*. The request signal (`req`) is used as an enable signal, triggering the write access at the input buffers. A separate *first in - first out (FIFO)* is used per virtual channel as input buffer. The output of this FIFO always carries the oldest flit in the buffer. When the buffer contains a new packet, the header is first visible at the output port of the router. This is necessary to extract the destination address, service level or the source address for routing and virtual channel reservation. Detection and decoding of head flits is performed by the *header decoding* unit. It is available once per VC to detect whether a header is available at the respective output. If a header is present, without an existing reservation, the header decoding unit raises a reservation request. The *input reservation arbitration (IRA)* unit arbitrates between parallel reservation requests. It performs a round-robin arbitration[13] between all the virtual channels of a port which requests a reservation. This arbitration is necessary to enable sharing of one *routing* unit between all VCs of a port. The motivation for sharing the routing unit between the virtual channels of a port is the saving of resources. In general, this sharing should not impact the performance of a router because only one header can arrive at maximum per clock cycle and port. Since the routing unit is capable of processing one packet or header per cycle, it should be capable of processing all routing requests directly. However, if the following reservation request fails, the routing and reservation requests must be repeated. This could lead to parallel reservation and routing requests at one input port of the router. However, this should only happen if an output port is overloaded. In such a situation, the additional reservation delay, caused by arbitration, should not be noticeable.

According to the routing decision, taken by the routing unit, a reservation request is forwarded to the *output port reservation arbitration (OPRA)* unit of the determined

[13] The arbitration decision is taken by combination logic. In case of a single request, this request is forwarded immediately.

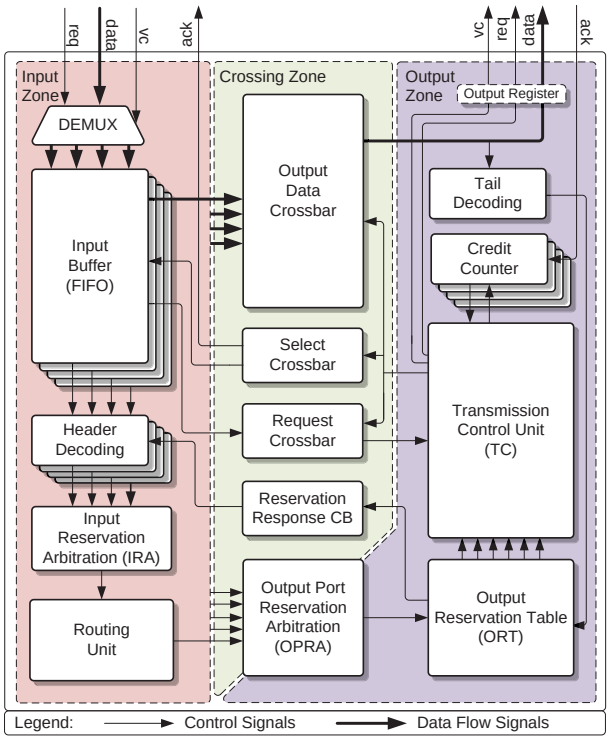


Figure 5.7.: Detailed structure of the NoC router with virtual channels.

output port. More specifically, the reservation request is always sent to all OPRA units of all ports, but masked if the output port does not match the routing decision. This method simplifies the implementation and reduces the cost. The OPRA unit can be on the one hand interpreted as the crossing zone for reservation requests. On the other hand each, OPRA unit is assigned to a specific output port. This ambiguous association is illustrated in figure 5.7 accordingly. OPRA first analysis the desired destination port. Subsequently, it performs a round-robin arbitration between the requests from different input ports. Therefore, the routing unit of each port is connected to the OPRA unit of all other output ports[14]. In contrast to the arbitration logic within the IRA unit, which is purely combinational, the arbiter of the OPRA unit can be configured to be sequential to serve as a pipeline stage or purely combinational. The reservation request that

[14] An input and output port pair is not connected by the crossing zone components. Thus, it is not feasible that a packet is returned to the router where it comes from. This restriction saves a lot of resources in the crossing zone.

was granted by the arbitration process[15] is forwarded to the output reservation table.

The *output reservation table (ORT)* plays a central role. Each output port has a separate ORT, simply named *reservation table*. It is responsible for processing the reservation requests and for assigning the output VCs accordingly. In case of a reservation request, the output reservation table verifies whether an output VC is available. Therefore, status bits for the reservation of each VC are analyzed. For each VC, there is a *reservation status flag* that indicates a valid reservation if it is set to 1. If a VC is available, the reservation request is granted and the actual reservation procedure takes place. The reservation process is granted by setting the respective output signal to 1. This acknowledgment is returned to the *header decoding* unit of the input port and VC that contains the header raising the reservation request. In order to return the reservation acknowledgment to the input port, the *reservation response crossbar* is used. It consists of a set of demultiplexers, which are controlled by the ORT using the input port and VC identifiers. The acknowledgment feedback is used to set a *reservation flag* in the header decoding unit. It is used to prevent repetition of the reservation request for the respective header[16]. The input port ID and the input VC ID are forwarded to the output reservation table by the output port reservation arbitration unit. The IDs are on the one hand used to control the reservation response crossbar. On the other hand, they are necessary for the VC assignment. This assignment process is performed by the ORT, as described in detail in section 5.1.1. According to figure 5.2, it contains a table with one entry per virtual channel. Each entry is used to store the ID of the input port and VC that are currently assigned to the respective output VC. The reservation status flag indicates a valid entry. Another combinational round-robin arbitration logic is used within the ORT to select a free VC to be used for a new reservation request. The selected output VC entry is then updated with the input port and VC ID coming from the output port reservation arbitration unit. The reservation table is accomplish by a set of registers. These registers represent an obligatory pipeline stage of the router's reservation process. By adding a new entry to the reservation table, the reservation process is completed. The output reservation table also plays a key role during release of an existing reservation, triggered by a tail flit. The release procedure is described later in this section.

All entries of the output reservation table are transferred to the subsequent *transmission control (TC)* unit. It is responsible for the scheduling of the virtual channels of the respective output port. The transmission control unit takes all VCs into account, which fulfill the conditions described in section 5.1.1.1. Therefore, it uses the output reservation table entries to determine whether a reservation is valid for the respective VC or not. If an entry is valid, the content of the entry (input

[15] In most cases only one single request exists at each OPRA unit, which is for sure granted directly.
[16] The feedback to the header decoding unit is necessary to avoid double reservations.

port and VC ID) is used to control the *request crossbar*. The request crossbar is used to forward transmission requests from the input buffers to the transmission control unit. As soon as data are available in the input buffer, the corresponding transmission request signal is set. It is derived from the fill-level status of the FIFOs[17]. A valid reservation and transmission request represent two of the three conditions to be met for scheduling. The third condition is the availability of buffer resources at the input port of the following router. The *credit counters* of the credit-based data-link layer flow control (see section 2.4.4.1) are used to evaluate this condition. They represent the available credits for each virtual channel. If the counter value is greater than 0, buffer slots are available for the respective VC. The conjunction of all three conditions, each represented as a Boolean value, is used as an input for the arbiter located in the transmission control unit. The arbitration unit has a separate input for each VC that is set to 1, if all three conditions are met for the respective VC. A round-robin arbitration is performed, enabling a fair access to the physical link for all VCs. The arbitration unit of the transmission control unit can either be configured to have registers within, or as a combinational circuit. If the registers are enabled, the transmission control unit represents a pipeline stage of the router. This pipeline stage can be used to improve the timing of the design by relaxing the critical path. This path results from the constraint evaluation and arbitration process that is located within the transmission control unit. The output VC, which is selected by the arbiter for transmission, is used to control the *select crossbar* and the *output data crossbar*. The select crossbar is a demultiplexer network that is used to pop a flit from the input FIFO of the VC, which was selected for transmission[18]. The pop signal of a FIFO is also used as an `ack` signal. It is used to return flow control credits to the previous router. The neighboring router uses this `ack` signal to update (increment) the credit counters of the respective VC at its output port. The flit taken from the input buffer then traverses the multiplexer network of the output data crossbar. The crossbar forwards the flit to the appropriate output port. At the output port, the flit is optionally stored in the *output register*, or transmitted directly to the neighboring router, if the register is disabled. The corresponding `req` signal and `vc` signals are set by the transmission control unit. Once a flit is transmitted, the credit counters at the output port have to be updated. The corresponding counter is decremented each time a flit is transmitted.

After the reservation process, triggered by a header, all flits are treated equally during transmission process. However, the transmission of a tail flit is detected by the *tail decoding* unit. It analyzes the *ctrl. flag* and the *tail bit* and triggers the release of the VC assignment. Therefore, a release request, consisting of an enable

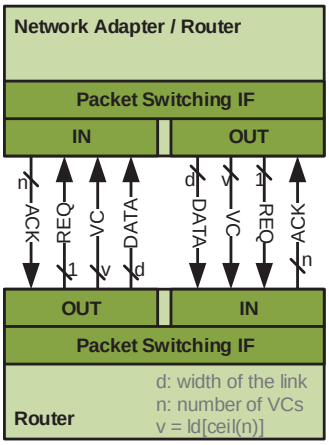
[17] As soon as the input FIFO is not empty, a transmission request is raised.

[18] Thanks to the separate transmission control for each output port, it may occur naturally that several flits from a single input port (but different VCs) are transferred simultaneously to different outputs ports.

signal and the ID of the output VC, is forwarded by the tail decoding unit to the output reservation table. The VC ID is used in the output reservation table to invalidate the entry of the assignment where the tail flit belongs. Subsequently, the corresponding virtual channel is available for other transmissions.

5.1.5.1. Physical Router Interface

The physical interface of the router reflects the data-link layer protocol and the flow-control mechanism of the router design. The interface consists of a symmetric full-duplex link. It is used to connect two neighboring routers or to connect the local port of a router to the network adapter.



(a) Interface

Name	Width $[bit]$	Default
req	1	1 bit
vc	$v = 1, 2, 4, 8, ...$	2 bit
data	$d = 8, 16, 32, ...$	128 bit
ack	$n = 2^v$	4 bit

(b) Signals

Figure 5.8.: (a) Physical interface of the router with a full-duplex link. (b) Data-link layer and flow-control signals that connect neighboring routers.

Figure 5.8(a) shows the physical interface of a router port with individual signals. Four signals are used per direction, as described in figure 5.8(b). The `req`, `vc` and `data` signal leave an output port while the corresponding `ack` signal enters the output in the opposite direction. The request signal (`req`) is used to indicate a transmission. Thus, it is set to 1 for a single clock cycle. Due to the use of credit-based flow control, it is ensured that a transmission request is only raised if the data can be stored in the input buffers. Thus, no direct feedback is necessary. The `vc` signals indicate the used virtual channel. The `data` signals are used to transfer the actual flit, according to the network layer protocol, defined in section 5.1.2. A separate one-bit acknowledge signal (`ack`) is used for each virtual channel to implement credit-based flow control. It is set to 1 for the corresponding VC

for one clock cycle, if a buffer slot becomes available. Thus, a set `vc` signal thus indicates the return of a credit for the corresponding VC. Separate `ack` signals are necessary for the individual VCs because multiple credits may be returned for different VCs, in the same clock cycle.

The link, connecting a router pair in the opposite direction, of course consists of the same signals and works identically. However, the two directions are fully independent of each other in order to establish a full-duplex link.

5.1.6. Latency and Bandwidth Analysis

The previously presented router design enables so-called best-effort communication. This means that all communication flows or packets obtain best-effort service, meaning that they obtain unspecified variable data rates and latencies, depending on the current load. The current load is usually unknown due to the fact that it is hard to obtain in a distributed system. Therefore, bandwidth and latency guarantees can barely be given for best-effort communication. Nevertheless, in this section worst case timing and throughput assessments for best-effort communication will be made under certain boundary conditions. It is used as a motivation for the QoS mechanisms, presented in chapter 6.

The necessary boundary conditions for an analysis are defined and justified as follows:

Back pressure: Back pressure originates from an over-utilization of a link or port in the network. If an overload situation exists for a long period of time, the backlog continues to spread in the network. Transmission scenarios arise from such a backlog, which can be barely analyzed even for a single router. For this reason, it can hardly be predicted, from the perspective of a single router, when a blockage elsewhere in the network will be dissolved. Consequently, it is assumed that there are no blockages exist in the network. This in turn means that the neighboring routers can process the data, with at least the same data rate as the router to be analyzed[19]. Thus, from the router's perspective, there are always credits available for each output port and VC to be used. In practice, the potential of back pressure is decreased by end-to-end flow control mechanisms, as described in detail in section 2.4.4.2.

Starvation: If a blockage exists ahead of a router's input, this could lead to starvation of the respective input port. If such a bottleneck occurs during the transmission of a packet, it is impossible to estimate its latency, due to the additional unknown latency resulting from the starvation. It must therefore be assumed

[19] In practice, this assumption is not tenable or given, because there can always be temporary overload situations.

for the following analysis that the neighboring routers are always capable of delivering outstanding flits of a packet.

Packet length: The time period for which a link or VC is blocked by a packet depends on its length. In order to estimate how long a reservation will persist, the packet length must be known. Consequently, a maximum packet size S_{max} is assumed hereafter; although the protocol enables infinite packet lengths.

Buffering: Another aspect, which refers to the packet size S and to the buffer size B is the number of packets per buffer. If the packet is smaller than the buffer size, parts of subsequent packets may already be in the same buffer prior to the transmission of the first packet. Such a situation may result in a huge latency for the second packet, due to the dependence on the first packet. To prevent this situation, the minimum packet size may be defined to be larger than the buffer size ($S_{min} > B$) or multiple packets per buffer must be prohibited[20]. For the following example, it is assumed that only parts of one packet are stored at the same time in any virtual channel buffer.

Fair access: A fair access to each output port via each input virtual channel is required. Otherwise, VCs with data could potentially block each other. This could result in unpredictable behavior where single VCs stall others infinitely. In such an implementation, the worst case latency for each VC would be infinite, with the worst case bandwidth being zero. Thus, no meaningful estimations for latency and throughput are feasible in such a design. The fair round-robin arbitration for reservation and transmission scheduling of the proposed router design enables a fair data processing. This fair scheduling and arbitration enables balanced throughput and delay. In section 5.5.4, an extension of the basic router design, enabling in-order processing, is introduced. In-order processing is also assumed for the following analysis.

Based on the given boundary conditions, worst case estimations for the throughput and latency of a single router are made.

5.1.6.1. Bandwidth Analysis

For a worst case bandwidth analysis of a router, it must be assumed that each VC buffer of the router contains a packet. In the worst case, all these packets must be forwarded through the same output port. Assuming that a packet always leaves the router through a different port, as it has arrived, $N-1$ competing ports have to be considered in case of a router with N ports. Consequently, the bandwidth of the respective output port is shared between $N-1$ input ports. A fair sharing for concurrent reservation requests is enabled by round-robin

[20] The router design provides an option that restricts routers to store parts of multiple packets in parallel in the same virtual channel buffer.

arbitration in the output port reservation arbitration unit. Thus, the worst case bandwidth $BW_{in\_port,wc}$ of an input port of the router is as follows:

$$BW_{in\_port,wc} = \frac{BW_{link}}{N-1} \tag{5.6}$$

BW_{link} is the bandwidth of one link or port, respectively. The bandwidth of each input port is in turn fairly shared between the VCs of each port. The sharing is managed by the input reservation arbitration unit, using a round-robin arbitration scheme. Thus, the worst case bandwidth $BW_{in\_vc,wc}$ of a single virtual channel, which is equal to the bandwidth for the packet using this VC, can be estimated as follows:

$$BW_{packet,wc} = BW_{in\_vc,wc} = \frac{BW_{in\_port,wc}}{\#VC} = \frac{BW_{link}}{\#VC \cdot (N-1)} \tag{5.7}$$

According to equation 5.7, each packet of a five port router with four VCs would have a worst case bandwidth of $\frac{BW_{link}}{16}$. The quality of service scheme, presented in section 6.2, provides much higher bandwidths.

5.1.6.2. Latency Analysis

The latency could be analyzed for a single flit or a complete packet. However, virtual channel allocation and end-to-end flow control are performed on the granularity of packets. Thus, packet transmission latency of a single router is analyzed now.

Definition. *The packet transmission latency of a router is defined as the number of cycles elapsed between the arrival of the first flit of a packet in one router and the transmission of the last flit of this packet by the same router.*

The delay for packet transmission depends on two aspects, the reservation latency L_{res} and the latency for data transmission L_{trans} after successful reservation:

$$L_{total} = L_{res} + L_{trans} \tag{5.8}$$

The reservation latency in turn depends on the time L_{res,vc_free} elapsed until one VC is available for transmission and the time $L_{res,pipeline}$ required to reserve this VC subsequently. Thus, the reservation latency is composed as follows:

$$L_{res} = L_{res,vc\_free} + L_{res,pipeline} \tag{5.9}$$

When estimating the worst case latency $L_{res,wc}$ for reservation, the worst case delay must be known. It is dictated by the latency until a virtual channel becomes available. This delay in turn depends on the transmission latency of other packets, which allocate the VCs of the output port to be used. Taking the boundary conditions described above into account, the worst case situation, with respect to the latency, is equal to the worst case scenario for the bandwidth. In that scenario, VCs at all input ports have to use the same output port; receiving packets at the same time[21]. Assuming there is no back pressure and starvation, the transmission latency for M packets through the same port only depends on the bandwidth[22] of this port. The packet size S and the total number of pipeline stages P_{total} of the router are:

$$L_{trans}(M) = M \cdot S + P_{total} \tag{5.10}$$

In the worst case scenario, equation 5.10 may now be used to estimate the time elapsed until a virtual channel becomes available for data transmission. For the worst case, S can be assumed to be the maximum packet size S_{max}. Assuming a fair access scheme, M must reflect the number of packets that can allocate the VCs before the considered packet. The total number of packets to be transmitted via the same port is $(N-1) \cdot \#VC$. Due to the availability of $\#VC$ virtual channels at the output port, the number of packets, which have to be transmitted completely before the last reservation takes place, is $(N-2) \cdot \#VC$. Therefore, the worst case latency until a virtual channel becomes available for transmission is:

$$L_{res,vc\_free,wc} = M_{wc} \cdot S_{max} + P_{total} = (N-2) \cdot \#VC \cdot S_{max} + P_{total} \tag{5.11}$$

In order to estimate the worst case reservation delay, equation 5.9 and equation 5.11 may be used. However, $L_{res,pipeline,wc}$ must be known es well. Due to the pipelining of the router, there is no additional reservation delay after a virtual channel becomes available. Thus, $L_{res,pipeline}$ may be assumed to be 0 for the worst case, where the pipeline is always busy. Using equation 5.11 and the constant reservation delay, equation 5.9 can be noted as follows:

$$L_{res,wc} = L_{res,vc\_free,wc} = (N-2) \cdot \#VC \cdot S_{max} + P_{total} \tag{5.12}$$

In order to estimate the overall latency for a packet, per hop or router, the worst transmission latency must be determined. For the transmission latency, the worst case exists, if all VCs of the reserved output port are occupied. Thus, the bandwidth is equally shared between all VCs. Meaning that one flit per VC is transmitted within an arbitration round of length $\#VC$. Consequently, the worst

[21] Packet headers arrive consecutively at the input buffer of each VC until all VCs are occupied.
[22] In this section, the bandwidth is always assumed to be '1'. This means 1 $Flit/Cycle/Port$.

case transmission latency of a packet of size S_{max} and a router incarnation with $\#VC$ virtual channels is[23]:

$$L_{trans,wc} = S_{max} \cdot \#VC \tag{5.13}$$

Finally, the total worst case latency for forwarding a single packet under the given boundary conditions may be estimated by inserting equation 5.11 and equation 5.13 into equation 5.8:

$$\begin{aligned} L_{total,wc} &= L_{res,wc} + L_{trans,wc} \\ &= (N-2) \cdot \#VC \cdot S_{max} + P_{total} + S_{max} \cdot \#VC \\ &= (N-1) \cdot \#VC \cdot S_{max} + P_{total} \end{aligned} \tag{5.14}$$

According to equation 5.14, each packet of a five port router would have a worst case latency of 261 cycles[24]. However, it must be taken into account that the given number is only the latency for a single router. The worst case transmission latency grows linear with the number of hops. Depending on the communication patterns, back pressure or starvation may not always be prevented. Consequently, the estimated values could even be exceeded. However, the quality of service schemes, proposed in chapter 6, may be used to produce better bounds for worst case throughput and especially latency.

5.2. Simulation Framework

A simulation framework is desired to investigate different implementation alternatives and new features for the proposed router architecture. Compared to an HDL implementation, the simulation framework shall reduce the time that is necessary to investigate new features and compare different router configurations with respect to suitability and performance. The simulation environment is used as part of the semi-automated design and implementation flow, introduced in section 4.4. It shall enable fast evaluation of different implementation alternatives under different load conditions.

[23] In contrast to equation 5.10, which gives the transmission latency for multiple packets using the same link, equation 5.13 provides the worst case latency for a single packet using one VC.

[24] A router with five pipeline stages and four VCs is assumed. Furthermore, a maximum packet size S_{max} of 16 flits is assumed.

5.2.1. State of the Art NoC Simulators

Essentially, the question arose whether an existing, state of the art simulator should be used or a new simulator should be developed. Thus, existing NoC simulation environments are now analyzed.

The open-source *On-Chip Communication Network (OCCN)* [49] is a framework for network on chip modeling and simulation based on an object-oriented C++ library built on top of SystemC. However, OCCN does not address detailed communication analysis. It is meant to provide NoC models for efficient high-level system simulation. Thus, OCCN is not suitable for accurate and detailed investigation of router and NoC designs. Moreover, OCCN is not freely available.

The *Nostrum NoC Simulation Environment (NNSE)* [167] is a simulator for the Nostrum NoC architecture [178]. It is based on SystemC. However, NNSE offers very little configuration capabilities of the topology, routing and switching. Its traffic generation capabilities are limited to synthetic traffic. The limited configuration capabilities of NNSE would necessitate substantial extension to fulfill the desired requirements. Taking the training period into account, which would be necessary to extent NNSE accordingly, a decision against the use of NNSE resulted.

The most common, existing NoC simulation environment, that is freely available, is named *NIRGAM* [121]. It is a SystemC-based cycle-accurate simulator for packet switching NoCs. NIRGAM addresses the evaluation of NoC designs in terms of routing algorithms and applications on various topologies. Thus, NIRGAM appears to be well suited as a basis for the desired simulation framework. However, on closer inspection, it seems that the simulator can indeed be configure well, but is very difficult to extend. Thus, the necessary novel NoC features, introduced by the work, can hardly be implemented in NIRGAM. Consequently, the decision was made to build a simulator from scratch, which better suites the identified requirements and can be efficiently extended.

An additional existing simulator is known as *DARSIM* [163]. DARSIM is a parallel, configurable, cycle-level network on chip simulator that uses a wormhole router architecture. It offers cycle accuracy as well as periodic synchronization to balance between accuracy and simulation speed. However, DARSIM was published in mid of 2010, after the development of this NoC simulator was almost finished. Hence, DARSIM couldn't been taken into consideration, but in general, DARSIM is a promising simulation environment.

Noxim [71] is another SystemC-based simulator, providing a simple[25] packet switching router model with several parameters. In particular, the user can customize the network size, buffer size and the routing algorithm. In addition, the packet injection rate and the traffic distribution can be configured. Compared

[25]The router model of Noxim is denoted as "simple" due to the use of very simple REQ/ACK flow control and the lack of virtual channel support.

to the previously discussed simulators, Noxim configuration capabilities are even more limited. An advantage of Noxim is its simple expandability. Although Noxim is not suitable, due to its limited configurability, its flexible implementation inspired the simulator design; presented in the following paragraphs.

After precise analysis of existing NoC simulators, none of them are suitable to be used as a basis for the desired simulation framework. Most of the simulators do not enable a simple way to model the proposed router design. Moreover, it would be too complex to extend them to support the features, introduced and investigated hereafter. Thus, it seems to be more efficient to implement a novel simulation framework that reflects the desired router design. Flexibility and expandability are elementary requirements of this novel simulator. This simulation framework is introduced and described now.

5.2.2. Simulator Concept

The simulation framework shall be used to derive a specific configuration of the NoC design. For a fast evaluation, different implementation alternatives must be efficiently compared against each other. An accurate modeling of the behavior of the NoC, which reflects the real hardware implementation, is one of the basic requirements for a successful evaluation. For a meaningful comparison of different implementation alternatives, realistic traffic generation is required. Moreover, a good observability of the NoC's behavior is desired to compare the potential implementation alternatives.

However, the simulation framework is not only used to analyze the impact of existing NoC features. It shall also be used to implement and evaluate new features, such as self-optimization or quality of service support. Therefore, the simulation framework and router model must easily extend. This requirement of simple expandability must be taken into account, especially during the initial implementation phase of the simulation framework.

All previously summarized requirements must be fulfilled by the simulation framework. The NoC model must be accurate and flexible in terms of configurability and expandability. It must be embedded into a simulation environment that enables generic and application-specific traffic generation. Additionally, the simulation framework must be capable of analyzing the NoC behavior and performance for evaluation.

A modular design of the simulation framework enables to decouple simulator control, traffic generation, the NoC model, statistic generation and analysis of the simulation results. The simulation framework uses SystemC [160] for cycle-accurate modeling of the NoC routers and network adapters. SystemC is a C++ library which provides an event-driven simulation interface in C++. It enables

accurate hardware simulation with relatively high simulation speed. However, for further increase of the simulation speed, only the NoC itself is modeled cycle-accurate using SystemC. The other components of the simulator, such as traffic generation, control and analysis are implemented using conventional C++.

5.2.2.1. Structure of the Simulation Framework

Figure 5.9 gives an overview of the modular simulation framework and its most important components. The *simulation controller* is the heart of the simulator. On the one hand, it is used to configure all other components according to the selected parameters. On the other hand, it controls the other components by generating the clock signal and cycle counter. The clock signal is used by the SystemC components. The cycle counter provides a notion of time for the other components, built by the use of conventional C++. The data flow in the simulator starts at the *traffic generation* module. It is responsible for traffic generation in the form of abstract transmission requests. The traffic generation module consists of different sub-modules, each realizing another traffic generation method. The selection and configuration of the traffic generation module is controlled by the simulation controller. It is described in section 5.2.3. The *packetization layer* is used to decouple traffic generation from the used switching scheme and network layer protocol. It converts the abstract transmission request, originating from the traffic generation module, into packets. These packets are enqueued into the *transmission queue* of the *network adapter*. The packetization layer enables conversion from BE to GS communication and vice versa. This allows a fair comparison of GS and BE communication, used for the evaluation of QoS mechanisms. With respect to QoS, the packetization layer is also in charge of GS connection management. It triggers setup and release of GS connections and can be configured to use BE communication as a fallback. The packetization layer also enables an easy replacement of the packet switching NA and NoC router model by circuit switching components, because the other modules of the framework can be reused without modification. However, a circuit switching model is not yet implemented.

The *network adapter* fetches the packets from its *transmission queue* and converts them into individual flits, according to the network layer protocol. These flits are subsequently forwarded to the output queues of the network adapter. A separate queue is instantiate for each virtual channel. The NA is also responsible for reserving virtual channels for guaranteed service connections according to the concept presented in section 6.2. The NA assigns GS communication to the respective VC. The network adapter back end, described in section 2.4.1.1, dictates the data transmission process. It schedules the transmission of flits from the individual VCs and utilizes the hop-to-hop credit-based flow control as well as end-to-end flow control for guaranteed service connections.

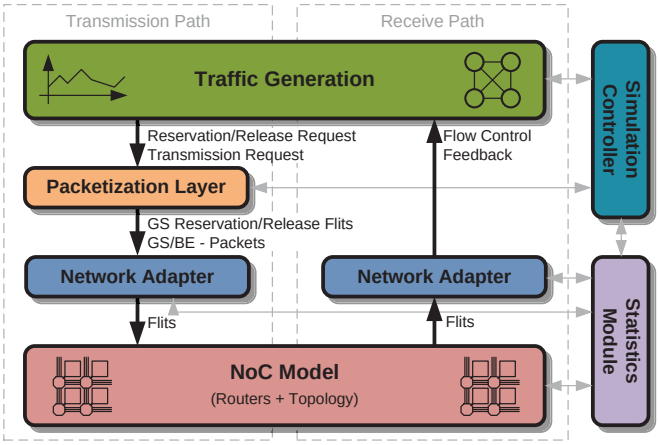


Figure 5.9.: Overview on the NoC simulation framework for evaluation and design space exploration.

According to figure 5.9, the network adapter injects the flits into the NoC. The *NoC model* is carried out in a hierarchical manner. On the highest hierarchical level, the topology is defined. According to the topology, the routers are instantiated and connected. The instantiated router model in turn consists of many subroutines. These subroutines fulfill different functions of the router, such as routing, virtual channel allocation and scheduling. The NoC model, including the router model, is described in section 5.2.4. When the flits leave the network at their destination address, they are passed to the respective network adapter, which has an input queue for each virtual channel. The flits are then fetched from the input queues and analyzed by the NA receive path. The traffic generation module is informed of the packets' arrival or about established or released connections on demand.

The *statistics module* is also informed about ejected flits. It stores accumulated statistics for utilization of links, buffers, virtual channels and latency information. In order to collect accurate data, not only the transmission and receive path of the NA is connected to the statistics module, but also the routers. Depending on their necessity, different statistics sub-modules may be disabled to improve the performance of the simulator. At the end of the simulation, which is defined by a command-line parameter, the statistics module prints a report. This report is written to a text file for manual or automatic processing, as described later.

The simulation framework is built in a way to be easily extendable. Thus, the simulator is dedicated to enable exploration of new features and fast performance evaluation under various load conditions. Consequently, it is frequently used for the investigations, presented in the following chapters. In addition, efficient

exploration of implementation alternatives is supported by a rich set of configuration options for the existing modules of the simulator. A highlight with respect to configurability of the simulator is surely the variety of traffic generation options.

5.2.3. Traffic Generation

The communication patterns used in simulation should correspond as closely as possible to real traffic. However, the type of traffic occurring at run-time, is often unknown and difficult to predict at design-time. Consequently, a NoC for general-purpose many-core architectures must be evaluated at design-time under various load conditions. In the simulation framework, traffic generation is done by the traffic generation module. It supports various traffic generation methodologies for testing a NoC design under different load conditions.

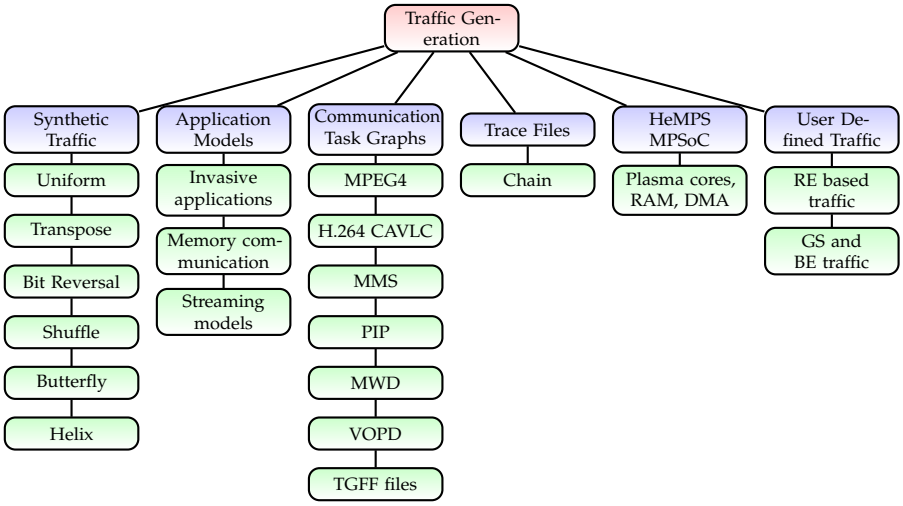


Figure 5.10.: Overview on the traffic generation capabilities of the NoC simulation framework. Currently, six traffic generation classes *(blue)* with different communication patterns *(green)* are supported.

Figure 5.10 provides an overview on the traffic generation methods, which are currently supported. The six traffic generation classes *(blue)* are instantiated within the traffic generation module. Each of the modules can be configured to generate different communication patterns *(green)*. Various traffic generation methods are used for the following evaluations. The traffic generation models will now be explained in detail. Their configuration parameters are summarized

in table A.10 and A.11. Due to the modular structure of the simulator, new traffic generation modules may simply be added.

5.2.3.1. Synthetic Traffic

Synthetic traffic is widely used for the performance evaluation of networks. The most common synthetic traffic pattern is *uniform* random traffic. In case of *uniform* random traffic, each node sends packets with the same rate to all other nodes of the architecture. In contrast, non-uniform traffic is characterized by an uneven load distribution [81]. Such an imbalanced load typically results in hot spots. The use of hot spot scenarios is very important for investigating the limitations of a network. Currently, five different non-uniform synthetic traffic patterns can be generated by the simulation framework. They include *transpose* traffic, *bit reversal*, *shuffle*, *butterfly* and *helix*. Thanks to the packetization layer, the data can either be transferred using best-effort or guaranteed service communication.

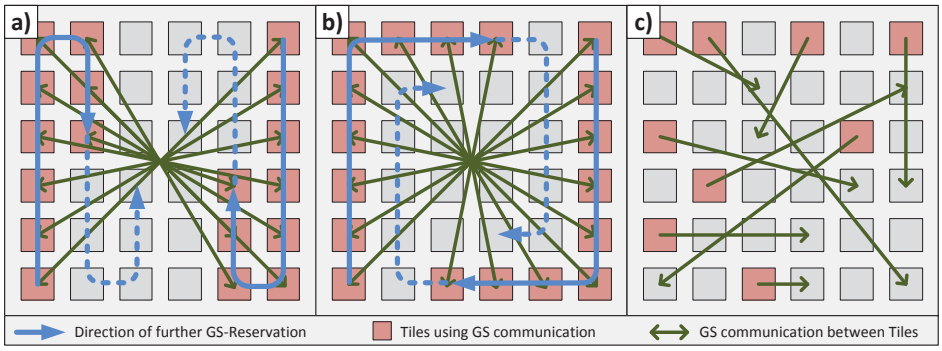


Figure 5.11.: Different synthetic traffic types: (a) transpose, (b) helix and (c) uniform.

Figure 5.11 shows how three of the synthetic traffic generators work in the case of GS communication. In figure 5.11(a), the connection setup and communication for transpose traffic is shown. New GS communication channels are established in a meandering shape. The connections are established bi-directionally between opposite cores. Helix traffic, as shown in figure 5.11(b), also connects opposite cores bi-directionally. In contrast to transpose traffic, the direction of establishing new connections gives a helix. Thus offering additional degrees of freedom while routing and allowing to establish more connections, compared to transpose traffic. Transpose and helix traffic both result in hot spots at the center of the NoC, as shown in figure 5.11. Hot spots are necessary to demonstrate the gain of routing algorithms in terms of hot spot avoidance. In contrast to these hot

spot scenarios, uniform random traffic is used to generate traffic with uniform distribution; shown in figure 5.11(c).

5.2.3.2. Application Models

The application-model-based traffic generator is composed of an abstract synthetic MPSoC-model and an application communication model. The MPSoC architecture model connects different types of *resource elements (REs)* to the network on chip. Currently, processing and memory resource elements are supported. The tasks and processes of the application model are assigned to processing elements. A shared memory architecture can be modeled by the use of memory nodes. Such an architecture is shown in the figure 5.12(a). For shared memory access, the processing elements communicate with the memory REs.

The application communication model is meant to be an abstract model, imitating the communication of a distributed application. This synthetic model offers many configuration options. The size of each application in terms of the number of used PEs may be defined manually or chosen randomly. The same applies for the number of applications and their mapping. A nearest neighbor heuristic [41] is used to map communicating processes automatically to neighboring PEs. In order to imitate the behavior of malleable invasive applications, invade and retreat rates can be defined. This results in dynamic applications, which allocate and release REs during simulation. The data rates and patterns for communication between the single processes of the application can be defined, as well as the communication type. Currently three types are supported:

1. Random communication between all processes of the same application and communication with a shared memory node by all other nodes.
2. All processing elements, executing the same application, communicate with each other in the form of a clique. Each of these PEs communicates with a shared memory node.
3. All PEs communicate with a master (master-slave communication) and with memory nodes.

Independent of the communication type, it can be chosen whether best-effort or guaranteed service traffic is used. Burst communication patterns can be configured separately for inter-PE and memory communication. Figure 5.12(b) illustrates examples of applications with communication type 1. The communication between the processes is represented by the directed arrows. Figure 5.12(c) shows the final mapping of the processes to the architecture model, when the nearest neighbor heuristic is used for mapping.

Application-model-based traffic generation is very helpful for the investigation of guaranteed service connections. The traffic model can be configured to behave

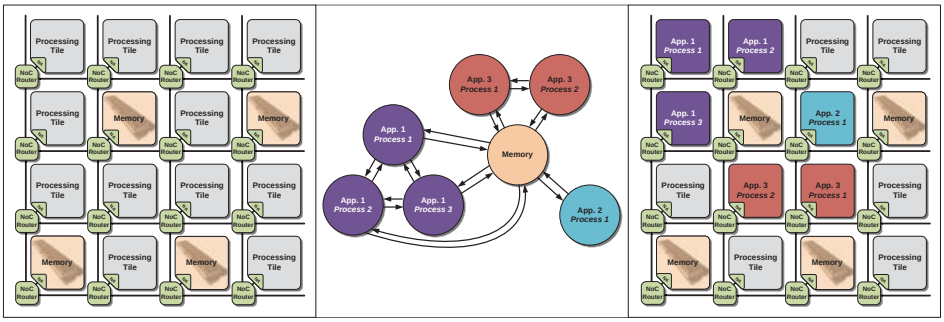


Figure 5.12.: Example of a synthetic application model which is mapped to a NoC architecture with two types of resource element for memory and processing nodes.

dynamically, representing the dynamic behavior expected for invasive computing. This results in continuing reservation and release requests for GS connections. Such a stressful scenario assists in evaluating adaptive QoS mechanisms.

5.2.3.3. Communication Task Graphs

The use of communication graphs or communication task graphs is a very efficient way to generate communication; reflecting the behavior of real applications. Communication graphs represent communicate between the individual tasks of a distributed application. They are obtained by analyzing the communication of a distributed applications. Such an analysis is provided in [252] for a distributed MPEG4 decoding platform. The task graph obtained from this analysis is provided in figure 5.13. It is one of the task graphs used in this simulation framework. The task graph of the MPEG4 decoder consists of 12 nodes of 4 different types[26]. The nodes or tasks are connected with 13 edges, each of them representing bidirectional communication[27]. The bandwidth requirements for each connection are given as a weight of the respective edge in the task graph.

The simulator is capable of reading task graphs from files, using the widely accepted *task graphs for free (TGFF)* format [62], which is also used by the NoCTweak simulator [255]. Due to the use of TGFF, many task graphs are available and can be added very easily. In addition to the task graph itself, a mapping for

[26] The four different types of nodes in the MPEG4 decoder can be taken into account by the simulator while mapping.

[27] Within the task graph specification used in the following, each bidirectional connection is represented by two unidirectional connections.

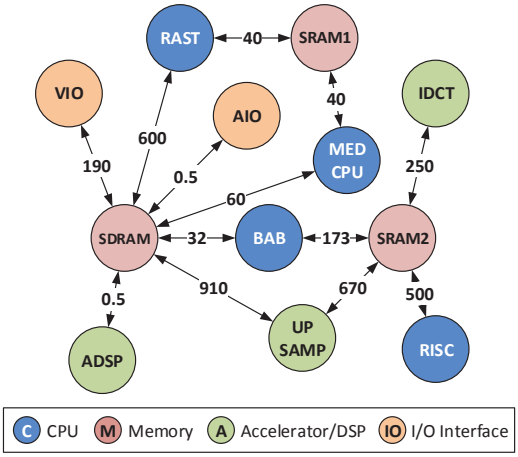


Figure 5.13.: Communication task graph of an MPEG4 video decoder with 12 tasks – Bandwidth requirements are provided in MB/s [22].

each graph can be defined, using a second input file. This file is naturally referred to as a *mapping file*. For the following investigations, six task graphs of different applications are used. They are summarized in table 5.3. The MPEG4, VOPD, PIP and MWD task graphs are obtained from [22], H264 and MMS are provided with NoCTweak simulator [255].

Abbr.	Details	Nodes	Edges
MPEG4	MPEG4 video decoding	12	26
VOPD	Video Object Plan Decoding	12	15
PIP	Picture-In-Picture	8	8
MWD	Multi-Windows Display	12	13
H264	H.264 CAVLC encoder	16	23
MMS	Multimedia System	25	33

Table 5.3.: Task graphs of the applications used in the simulation framework.

5.2.3.4. Trace Files

In addition to communication task graphs, trace files are another way to generate application-specific NoC load. In order to generate a trace file, each message, sent by a distributed application, is logged in a trace file. Each entry of the trace file specifies source, destination, size and time of occurrence of the respective message.

More advanced trace files also capture the processing time of each core after arrival of a message, enabling more precise generation of communication patterns. Such an advanced trace file format, termed *chain*[28] is supported by the proposed simulator. The chain-format can trigger a computation task after the arrival of messages. In addition, it supports triggering of multiple transmissions after a computation task is finished. This enables modeling of complex computation and communication dependency graphs.

However, obtaining such trace files is very challenging, as it requires detailed observation of the execution and communication of an application. Therefore, trace files are typically unavailable. It is envisaged to generate trace files for the simulation framework by probing the FPGA prototype of the invasive architecture, introduced in section 3.4.

5.2.3.5. HeMPS MPSoC

The HeMPS [39] is an MPSoC architecture, available as a SystemC model. It uses Plasma CPUs, each of them equipped with a *random-access memory (RAM)* and a DMA unit. The Plasma CPU is a small 32-bit RISC microprocessor, which is available for free. The Plasma core and its peripherals are attached to a network adapter that is used to connect the single-core tiles to a network on chip. In the original HeMPS architecture, the Hermes [180] NoC is used. The HeMPS SystemC model executes compiler generated binary code of real parallel applications on an MPSoC architecture. Thus, real applications written in C++ or other programming languages can be executed to generate NoC traffic.

As part of a bachelor's thesis [Ker13], the Hermes [180] NoC was replaced by the SystemC NoC model, created in the current work. This was achieved by developing a new network adapter, which serves as a wrapper between the Plasma compute tiles and the present SystemC NoC simulator. In order to use the QoS features of the proposed NoC model, software interfaces were created to establish GS connections. The implementation of these interfaces necessitated changes in the micro-kernel of the processors, as described in detail in [Ker13].

The incorporation of the HeMPS MPSoC model into the proposed simulation framework enables to investigate NoC architectures under realistic load, generated by the applications executed on the Plasma cores[29]. In [Ker13], a case study with two applications was performed to show the capabilities of the coupling between the present simulation framework and HeMPS. This study demonstrates that the coupling enables to determine the impact of different NoC configurations

[28] The name "*chain*", is derived from the fact that computation and communication logs of the trace file constitute a chain of events.

[29] The coupling between the proposed NoC simulation framework and the HeMPS MPSoC model is currently implemented in a separate development branch.

on the execution time of parallel applications. The analysis of a distributed MPEG application shows the superior performance of the proposed NoC compared to Hermes [180], as well as the benefit of the QoS scheme, presented in section 6.2.

5.2.3.6. User Defined Traffic

In addition to the plurality of pre-built traffic generation methods of the SystemC simulation framework, it is possible to define specific traffic patterns for more advanced investigations manually. Therefore, the simulation framework provides a specific interface, enabling a comfortable definition of user specific traffic. Depending on the requirements, communication can be defined at the granularity of packages or in a more abstract way.

Particularly in this abstract, coarse grained traffic generation enables efficient definition of complex traffic scenarios. It utilizes the primitives of the application model, presented in section 5.2.3.2, for point-to-point traffic generation. For each generator instance, the source address and destination address, the communication type (BE or GS), the communication pattern (burst, streaming, random), packet size and bandwidth must be defined by the user. Subsequently, the generators produce traffic during the entire simulation period according to these definition.

5.2.4. NoC Model

The router design, introduced in section 5.1, is built as a cycle-accurate SystemC model in the simulation framework. It is built modularly, enabling easy extension by new features. Two SystemC processes are used to model the sequential behavior of the NoC in a cycle-accurate way. One of these processes is responsible for reception of incoming flits; referred to as the *receive process*. The other process is in charge of managing the reservation and transmission. It is the most complex process of the SystemC router model; referred to as *transmission process*. Various C-functions are called by these two processes to perform routing, VC selection, reservation or release of virtual channels. In order to clarify the functional principle of the SystemC router model, the receive process and the transmission process are briefly discussed in the following sections.

The router model is implemented as a SystemC module. It is instantiated in the form of a 2D array to create a mesh or torus topology. The routers are connected via the SystemC signals to form the desired topology. The local port of each router is connected to the instantiated network adapter model.

5.2.4.1. Router Receive Process

The receive process of the router carries out the functionality that is required to write the data to the respective input buffer of the router. For this purpose, the control signals of the respective input port are processed first. The ID of the VC, used by the received flit, is encoded in the control signals. It is used to forward the flits to the respective queue of the buffer. The buffer is implemented as a separate module, to simply replace the buffer implementation. The current implementation supports different buffering schemes, where buffers are either shared between the VCs or used exclusively. More details about the available configuration options for the buffering schemes are provided in appendix A.5. The buffers of the router link the receive process and the transmission process.

5.2.4.2. Router Transmission Process

The transmission process is the most complex process of the router implementation. It is responsible for routing, VC allocation and release, as well as scheduling the VCs. Thus, the transmission process carries out all pipeline stages of the router, except the input buffers. According to table 5.2, the number of pipeline stages is configurable in the SystemC model.

In the first phase of the transmission process, the buffers are checked for flits awaiting transmission. Subsequently, it verifies that one of the flits is a header. If multiple headers are available, which have yet to be assigned to an output port, an arbitration decision is made to determine which header is processed first. Afterwards, the routing decision is taken for the respective flit to determine the output port for the packet. Currently, four different routing schemes and two selection strategies for adaptive routing are supported. Routing is implemented in a modular way using a hierarchy of functions. This modular implementation enables the simple replacement of single components (e.g. the selection strategy of an adaptive routing scheme). Once the output port is determined, a VC of the respective port is selected for reservation. According to the routing decision, the VC selection and reservation is also built as separate functions to enable expandability. The virtual channel selection and reservation functions represent the functionality of the *reservation table*, shown in figure 5.7.

The second part of the transmission process implements the functioning of the *transmission control* unit, the *crossbar* and flow control. It checks the buffers for valid transmission requests using a round-robin arbitration scheme. It is also built as a separate function with respect to modularity. The function verifies the reservation status, the availability of data and the availability of flow control credits. Only those VCs are taken into account for scheduling, which fulfill said conditions. The flit, which is selected for transmission, is than fetched from the

respective input buffer. The output signals of the router port are set according to the selected VC and the data fetched from the buffers. The last step of the transmission process determines whether a tail flit is sent. If this is the case, the release function is called to free the allocated virtual channel. [30]

5.2.5. Analysis and Evaluation Capabilities

The simulation framework can be used by manually configuring the desired parameters and executing the simulation. Subsequently, the output generated by the *statistics module*, can be validated and analyzed. However, such a manual analysis can be very time consuming if many different parameters or implementation alternatives must be compared. Hence, the simulation framework supports a script-based mode, which configures, simulates and aggregates the simulation results for different implementations. This enables an automated design space exploration; used for the following evaluations. The automated simulation is also used in the design flow, introduced in section 4.4.

The automated script based simulation mode utilizes Unix shell scripts. The first script sets and modifies the desired parameters for simulation. A description of the available parameters of the simulation framework, which are defined in the *parameter file*, is provided in appendix A.5. After parameter configuration, the script generates the simulator binary and begins execution. The script-based simulation is capable of performing multiple simulations with equal or different parameters, in parallel. This parallel simulation reduces the simulation time drastically by utilizing multiple cores of the host computer used to execute the simulation. At the end of the simulation phase, the simulation results are written into a set of output files.

After simulation phase is complete, a second script is used to process the simulation results. It processes the output files containing the simulation results, generated by the statistics module. This script can be configured to extract various simulation results from the complex statistics. The desired values, such as latency, bandwidth, buffer-utilization or link-utilization, are extracted. Subsequently, the values are inserted into a *comma-separated values (CSV)* file, generated by the script. The CSV file can subsequently be used to generate diagrams representing the performance impact of the investigated parameters. It is also used for the requirements analysis in the semi-automated design flow, described in section 4.4.

The script-based simulation enables a very fast design space exploration, which is frequently used for the investigations, presented in the following chapters.

[30] The transmission process contains additional functionality, such as a recognition for partially established GS connections, which are not discussed here in detail.

5.3. HDL Model and Implementation

The previously presented simulation framework and SystemC model of the NoC enables accurate performance measurements with respect to bandwidth, latency or utilization. However, an HDL implementation is also necessary to enable investigation of the router design for a desired target technology[31]. A synthesizable HDL implementation is required to obtain accurate numbers for resource requirements, power consumption and the achievable clock frequency. These numbers required, in order to compare different implementation alternatives accurately with respect to energy and resource efficiency, as explained in section 4.1.2. Such a comparison of technology-specific design properties is used for the following investigations, but also as part of the design flow, proposed in section 4.4. If an appropriate implementation is found, which fits the given functional and non-functional requirements, an HDL implementation enables an FPGA-based prototype or an ASIC.

SystemVerilog [238, 236] was used for the HDL implementation of the proposed router template design[32]. The HDL implementation of the router design corresponds to the structure, described in section 5.1.5. The structure and the modules, shown in figure 5.7, were almost identical implemented in the form of SystemVerilog modules. According to the previously discussed SystemC implementation, the modularity shall enable an easy extension of the basic router design. This was required for the following investigations, where the basic router is extended, but also to use the design as a template for future architectures. In order to achieve the desired flexibility of the design, the HDL implementation is highly parameterizable. The available parameters of the HDL implementation are summarized in appendix A.6. With the assistance of the parameters, the router can be configure at design-time, according to the given requirements.

Building blocks[33] are used for the HDL implementation to improve the performance of the design. In the present work, the *DesignWare* library building blocks from *Synopsys* are used. For each of the building blocks used, an alternative HDL implementation is available to ensure portability of the design and to enable synthesis without the availability of the DesignWare library. Prominent building blocks, used in the design, are FIFOs for the buffers and a round-robin arbitration unit, which is instantiated in various modules. A parameter is used at design-

[31] At the point in time where the HDL implementation of the router design was started, there were no reliable tools for the synthesis of SystemC models available. However, in recent times HLS tools have been introduced, which enable synthesis of SystemC models (e.g. Xilinx Vivado). At that point in time, the development of the HDL models was already very far advanced. Consequently, their suitability was not investigated.

[32] Compared to older HDLs, such as VHDL or Verilog, SystemVerilog offers additional design features [238] and extensive verification capabilities [236]. These novel features have advantageously been used for implementation and verification of the design.

[33] A building block is a prefabricated component provided by a third-party vendor.

time to decide whether building blocks are instantiate or the alternative HDL implementation of the respective components is used.

5.3.1. HDL Test Environment

The substantial verification capabilities of SystemVerilog are utilized to build a complex simulation environment. This environment is used to test and verify the HDL implementation using an HDL simulator[34]. Test and verification of individual sub-components was carried out by module tests using SystemVerilog test benches. For verification of a complete router, or an entire NoC, a flexible SystemVerilog test environment was designed. The test environment provides multiple functions for generation of best-effort or guaranteed service communication. For automated verification, multiple checks for flit loss, packet ordering or deadlock detection are implemented. The test environment also provides capabilities for performance measurements, generating statistics for throughput and latency distribution. This sophisticated test environment allows efficient testing of new features and enhancements of the HDL router implementation.

In addition to test and verification, the test environment is also used for accurate power estimation of ASIC implementations. Therefore, a single router is synthesized, as described in the following section. The resulting netlist, consisting of ASIC standard cells, is instantiated in the test environment and stressed by the use of its traffic generation capabilities. During simulation, the switching activities in the netlist are captured and stored in a *value change dump (VCD)* file. The VCD files are subsequently used to calculate the power consumption.

5.3.2. Synthesis

The main purpose of the HDL implementation is the synthesis of the design for the desired target technology. The HDL template of the NoC is designed to support different target technology. Therefore, it does not contain any technology-specific components, such as FPGA block RAMs or ASIC memories. The aforementioned building blocks can be disabled, if unavailable or unsupported, by the synthesis tools. Thus, a technology independent HDL model is available, which can be synthesized with any tool supporting the SystemVerilog standard. This also applies to the extensions, presented in the following chapters.

In this work, two synthesis flows have been set up and utilized. An ASIC synthesis flow is used to estimate the design properties for a silicon implementation. However, an FPGA synthesis flow enables prototyping of the NoC design and

[34] *Mentor Graphics ModelSim* is used for simulation and functional verification.

the InvasIC architecture, introduced in section 3.2. Both flows will be introduced briefly now as a background for the evaluations presented later.

5.3.2.1. ASIC Synthesis

The main objective of the proposed NoC design is an ASIC implementation. Thus, an ASIC synthesis is desired to obtain accurate numbers for area and power consumption. Another important aspect, obtained from the synthesis results, is the achievable clock frequency. The clock frequency determines the bandwidth and consequently the performance of the implementation.

In order to enable the semi-automated design flow, the ASIC synthesis flow is automated by the use of Unix shell scripts. In the first step of the flow, the design parameters, provided in appendix A.6, are set. Subsequently, the design is elaborated by the synthesis tool. Therefore, the *Synopsys Design Compiler*[35] is currently used. After the elaboration phase, the actual synthesis takes place. At that point, the standard cell library for the desired target technology is required. Any available library can be used. However, in the following, a *TSMC* 45 nm library[36] with worst-case operation conditions[37], is used. The assumption of worst-case conditions ensure that the synthesis results will not be too optimistic with respect to actual implementation. In addition to the target library definition, the design must be constraint for synthesis. Therefore, the clock frequency and external delays for all input and output signals must be defined to obtain realistic numbers[38]. Subsequently, the actual synthesis process takes place[39]. Retiming and clock gating is enabled for synthesis to optimize clock frequency and power consumption. The final netlist is verified after synthesis by performing a netlist simulation, using the HDL test environment. During this step, switching activities are captured for power estimation. In the last step, the *Design Compiler* is used again to calculate the power consumption of the design. Therefore, the netlist, the captured switching activities and the standard cell library are used. Finally, power consumption, timing and area reports are generated. Optionally, multiple synthesis runs with different input parameters can be performed automatically by using the scripted flow.

[35] For the results presented in the following, *Synopsys Design Compiler* G-2012.06-SP4 was used.
[36] More recent technology nodes than the used *TSMC* 45 nm library were not available at that time.
[37] The exact name of the *TSMC* 45 nm library used in this work is *tcbn45gsbwpwc*.
[38] An external delay of 50 % of the clock period is pessimistically assumed for all I/O signals.
[39] The *compile_ultra* command is used for synthesis. It flattens the design to achieve optimal results.

5.3.2.2. FPGA Synthesis and Prototyping

Field programmable gate arrays are a powerful means for prototyping of digital integrated circuits. Thus, FPGA-based prototyping is also used to verify the proposed network on chip and its features. The FPGA prototype used later on, contains not only the NoC itself, but also other components of an MPSoC. Therefore, components of the InvasIC architecture have been used. The invasive network adapter, presented in section 3.2.2.1, and the RISC core tiles, introduced in section 3.2.1.1, are used to complete the architecture. Details about the FPGA prototype used later on for evaluation, are provided in appendix A.1.

Synopsys Synplify[40] is currently the only FPGA synthesis tool with reliable SystemVerilog support. Hence, it is used for synthesis of the networks on chip components because they are designed in SystemVerilog.

In order to synthesize a complete architecture prototype, *Xilinx PlanAhead*[41] had to be used[42]. The netlists, resulting from the previous *Synplify* synthesis of the NoC, are processed by *PlanAhead* together with the VHDL and Verilog source files of the other components. Synthesis and place-and-route are triggered by *PlanAhead*, which calls the respective *Xilinx* tools. Finally, a bitstream is generated, which can be used to program the desired FPGA. Similar to the ASIC synthesis flow, FPGA synthesis is completely automated via scripts.

5.4. Case Study

The semiautomatic design flow, introduced in section 4.4, enables fast generation of a requirement-specific network on chip. The design flow utilizes the simulation framework, introduced in section 5.2, and the previously introduced HDL implementation. This enables fast evaluation of implementation alternatives with respect to performance and implementation costs.

In this section, an exemplary case study will be presented. This study shall show the possibilities of the proposed automated evaluation and implementation methodology. In addition, it is used to determine appropriate settings for basic parameters of the router design. In the following, the appropriate number of VCs and buffer slots will be evaluated.

[40] *Synopsys Synplify Premier* version H-2013.03-1 is used in the work for FPGA synthesis.
[41] Version 14.4 of *Xilinx ISE* tools have been used exclusively in the context of this work.
[42] The DDR3 controller, which is part of the prototype, couldn't been synthesized successfully using *Synopsys Synplify* due to a bug in the tool.

5.4.1. Performance

As a measure for the performance of a communication system, latency and bandwidth are mainly used. Sufficient bandwidth is the most basic requirements, which must be fulfilled to avoid communication bottlenecks. In case of overload, bandwidth limitations lead to high latencies, according to section 5.1.6.2. However, even if the communication system is not overloaded, various design decisions may impact the latency, as shown by the following investigations.

5.4.1.1. Buffer Size

The simulation framework, introduced in section 5.2.5, and the evaluation methodology from section 5.2.5 are used for the following exploration of the design space. In order to investigate the impact of the virtual channel buffer size, it is varied between 2 and 10 slots in steps of 2 slots[43]. Multiple simulation runs are performed for each setting[44]. Uniform random traffic, defined in section 5.2.3, is injected with different rates[45]. This enables measuring the throughput for different load situations, with the packet size set constantly to eight flits.

Figure 5.14(a) shows the throughput for different buffer sizes[46] for an 8x8 mesh NoC. The results show that before the network becomes saturated, all configurations can satisfy the requested injection rate.

Definition. *The saturation point of a network is the point where the network becomes congested. At this point the packet latency increases rapidly and the throughput cannot be further increased.*

As shown in figure 5.14(a), the saturation point depends on the buffer size. The larger the buffers, the more throughput can be achieved by the NoC. If the buffer size is increased from 2 to 4 slots, the throughput increases by 8.4 %. The best results, with respect to throughput, can be achieved for the largest buffer. With a size of 10 slots, the throughput is 16.4 % higher than for the smallest buffer. Thus, increasing the buffer size is in general beneficial from the throughput perspective. However, an additional increase above a certain size is barely profitable from the throughput perspective.

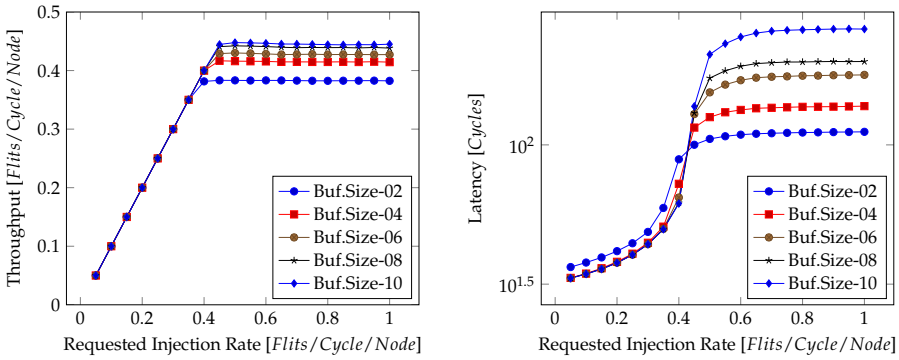
Figure 5.14(b) shows the latency measurements, obtained from the same simulations used for the throughput measurements. Below the saturation point, the

[43] A slot is a single entry in the FIFO, capable of storing one flit.

[44] For each setting 10 independent simulation runs are performed and the results are averaged. Each run simulations 10^6 cycles. This evaluation scheme is also used for all subsequent studies.

[45] The injection rate is defined as the number of flits which are injected per cycle and node. Thus, it can naturally vary between 0 and 1, if each link has a capacity of 1 flit per cycle.

[46] For the analysis of the buffer size, the number of virtual channels is set to a constant value of 4.



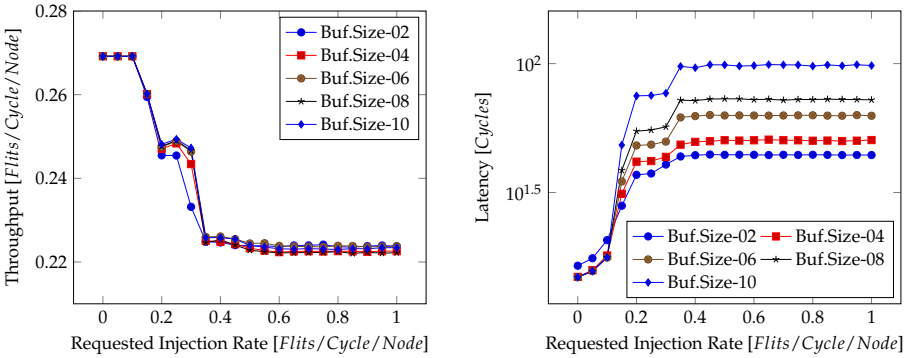
(a) Throughput (b) Latency

Figure 5.14.: Impact of buffer size on throughput and latency of an 8x8 mesh NoC for uniform random traffic.

latency is very low. Larger buffers assist in reducing the latency, if the NoC is unsaturated. At the point of saturation, the latency increases sharply. Above the saturation point, an increased buffer size has a negative impact on latency. Thus, the highest latency is experienced for the configuration with the largest buffers. The increased latency is reasoned by the higher number of flits, which are simultaneously on the way through the NoC if the buffers are large. More flits in the network result in a higher transmission latency for each flit due to the sharing of the available bandwidth. The latency is increase by a factor of 2.38 for high injection rates, if 10 buffer slots are available as opposed to 2.

Figure 5.15 also shows throughput and latency analysis for different buffer size configurations of the NoC routers. In contrast to the results provided by figure 5.14, the communication task graph of the MPEG4 video decoder, introduced in section 5.2.3.3, is investigated. The respective task graph is mapped in the center of the 8x8 mesh. The other nodes, which are not used by the MPEG tasks, generate uniform random traffic again. However, the random traffic is only used to generate additional load. The obtained results only measure throughput and latency of the MPEG4 application. Throughput and latency of uniform random traffic is not measured.

Figure 5.15(a) shows the throughput of the MPEG4 application for different uniform random traffic injection rates. Compared to the previous scenario, the throughput is almost independent of the buffer size. Only around the saturation point, larger buffers assist in slightly reducing the throughput. Once the network becomes saturated by additional random traffic, the throughput of the MPEG4 decoder drops. A throughput reduction of 17 % is experienced at maximum.



(a) Throughput (b) Latency

Figure 5.15.: Impact of buffer size on throughput and latency of an MPEG4 video decoding application, mapped to an 8x8 mesh NoC, burdened with uniform random traffic with varying injection rates.

Figure 5.15(b) shows the latency measurements for the MPEG4 decoder communication. In contrast to the throughput, the latency of the application is impacted by the buffer size of the NoC routers. The results again demonstrate, that an increased buffer size has a negative impact on latency, if the network becomes saturated. As previously mentioned, this results from the higher number of flits traveling in parallel through the network and sharing its bandwidth.

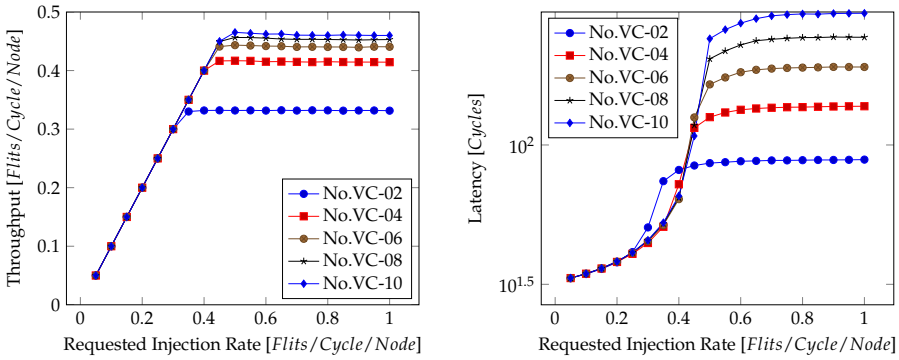
5.4.1.2. Virtual Channels

Virtual channels are proposed in [58] to improve the performance of a packet switching network. This statement will now be reviewed. Therefore, similar analysis are performed, as previously for various buffer sizes[47].

Figure 5.16(a) shows the throughput of NoC configurations with different numbers of VCs, with uniform random traffic being utilized. Compared to the buffer size, the influence of the number of VCs on throughput is greater. The maximum throughput in case of 10 VCs is 38.8 % higher than for the variant with only 2 VCs. However, increasing the number of VCs from 2 to 4 has the highest impact, thus further increasing the number of VCs results in a smaller performance gain.

The latency analysis for a NoC with different numbers of VCs under uniform random traffic load is shown in figure 5.16(b). For very low injection rates, there is nearly no impact on latency with respect to the number of VCs. In case of

[47] For analyzing the impact of different numbers of VCs, the buffer size is kept constant to 4.



(a) Throughput

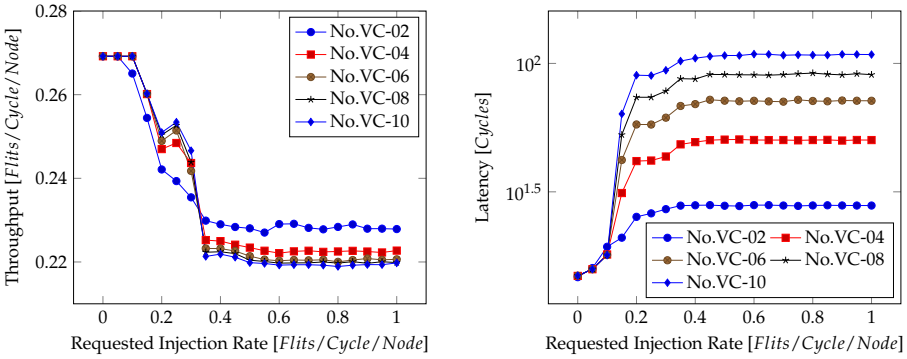
(b) Latency

Figure 5.16.: Impact of the number of virtual channels on throughput and latency of an 8x8 mesh NoC for uniform random traffic.

high load, near to the saturation point, additional VCs have a negative impact on latency. The reason is the increased buffer capacity, resulting from additional VCs. It leads to a higher number of flits in the network, which in turn leads to a higher latency. The correlation between the number of flits in the network and the latency is confirmed by the theoretical analysis, presented in section 5.1.6.2.

Next, the impact of additional VCs is also investigated for the MPEG4 decoder, previously used. Figure 5.17(a) shows the throughput analysis. In contrast to the buffer size, the number of VCs has a noticeable influence on performance of the MPEG application. Additional VCs have a positive impact on throughput for low injection rates. In case of high load, close to the saturation point, the performance is negatively impacted by additional VCs. However, operating a NoC close to the saturation point is undesirable, due to the high latency. Thus, injection rate control is proposed in [193] to limit the injection rate.

Figure 5.17 shows the latency analysis for the MPEG4 application, mapped to the center of the 8x8 mesh. A low latency is experienced by the MPEG4 decoder as long as the injection rate of the random traffic is below the saturation point. However, the MPEG application cannot profit from additional VCs with respect to latency. Once the NoC becomes congested by uniform random traffic, additional virtual channels have a negative influence on the latency. The reason is again the increased number of buffer slots and the resulting higher number of packets and flits in the network. This high number results in an increased delay for each single packet, due to bandwidth sharing.



(a) Throughput (b) Latency

Figure 5.17.: Impact of the number of virtual channels on throughput and latency of an MPEG4 video decoding application, mapped to an 8x8 mesh NoC, under uniform random traffic with varying injection rates.

5.4.2. Implementation Costs

The previous performance analysis, which investigated the impact of the buffer size and number of VCs, clearly showed a benefit of additional VCs and buffer slots for moderate load situations. However, raising these parameters necessitates additional hardware resources. This may impact the achievable clock frequency and the power consumption of the network on chip. In order to evaluate these costs, the ASIC synthesis flow, introduced in section 5.3.2.1, is used in combination with a 45 nm ASIC library. All settings of the router design are chosen according to parameters used for simulation. This allows to match the performance analysis with the following cost analysis, according the evaluation flow, outlined in section 4.4.

5.4.2.1. Buffer Size

Buffers are built by the use of FIFOs. In order to increase the buffer size in the HDL model, the size of each FIFO is increased.

Figure 5.18(a) shows the impact of the buffer size on the area consumption of a single five-port router with a link size of 32 bit. The buffer size is varied between 2 and 16 slots[48]. The results clearly exhibit the increased area consumption, resulting from larger buffers. A linear dependence between the buffer size and the area consumption of the router can be obtained from the synthesis results.

[48] For this analysis, the number of virtual channels is set again to a constant value of 4.

Increasing the buffer size from four to eight slots, results in a raise of the area consumption by 51.7 %.

Figure 5.18(b) provides the achievable clock frequency of a router with different buffer sizes. The clock frequency for synthesis was constrained to be 1 GHz. The synthesis results clearly indicate the observance of the defined frequency. Thus, the clock frequency is not affected by an increase of the buffer size[49]. This indicates that the size of the buffers does not impact the critical path of the NoC router design.

In addition to clock frequency and area consumption, the power consumption of an integrated circuit is another important aspect. The high power density of current and future technology nodes necessitates a limitation of the power consumption. Additionally, mobile devices have to deal with a limited power budget. Consequently, power consumption is a crucial aspect of many systems. Figure 5.18(c) shows the minimum and maximum power consumption[50] of a single router for different buffer sizes. The results exhibit that the minimum and maximum power consumption increases linearly with the buffer size, according to the area utilization. The gap between the minimum and the maximum power nearly remains constant, independent of the buffer size. If the buffer size is increased from 4 to 8 slots, the minimum power consumption grows by only 17.5 %. Compared to the area consumption, the power consumption is less impacted by larger buffers. Consequently, there is no proportional relation between area requirements and power consumption.

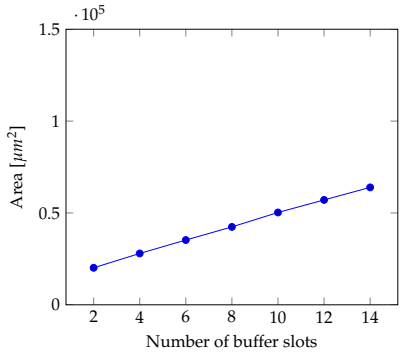
5.4.2.2. Virtual Channels

The virtual channels are deeply embedded into the router design. According to section 5.1.5, the implementation complexity of many components, such as the input reservation arbitration, output reservation table, transmission control and crossbars, are affected by the number of VCs. Consequently, increasing the number of VCs raises the complexity of many router components. The impact of additional VCs, with respect to implementation costs, will now be investigated.

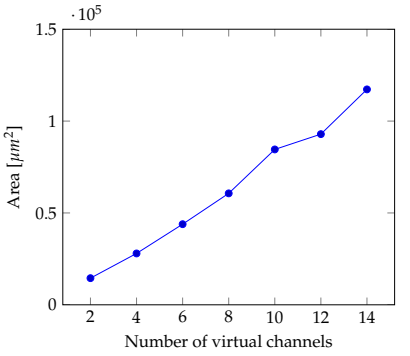
Figure 5.18(d) shows the relation between the number of virtual channels and the area, consumed by a five-port router. Compared to an increased buffer size, additional VCs have a greater impact on the area consumption. Increasing the number of VCs from 4 to 8, raises the area utilization by 117 %. This super-linear increase results from the crossbar implementation. According to equation 2.3, its complexity grows quadratic with the number of VCs. However, a router design with 8 VCs and 4 slots per buffer consumes only 60659 µm of area.

[49] It is expected that a further increase of the buffer size would affect the clock frequency.

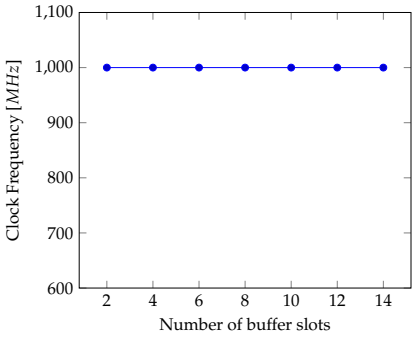
[50] The minimum power consumption is the power consumption of an idle router. The maximum power consumption results from a traffic scenario with very high load at all router ports.



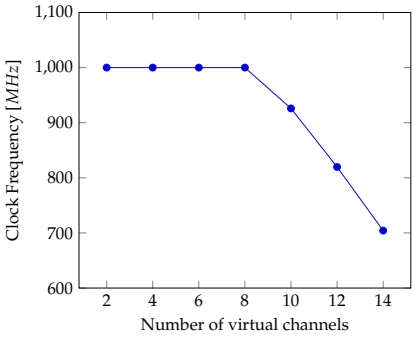
(a) Area, Buffer size



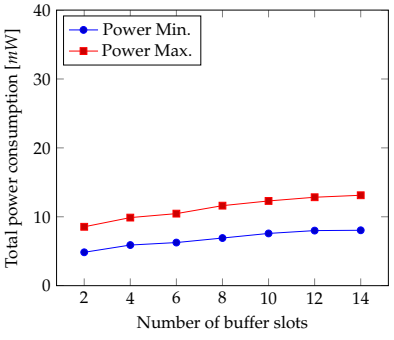
(d) Area, No. of VCs



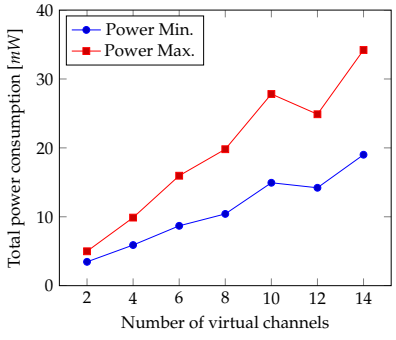
(b) Clock frequency, Buffer size



(e) Clock frequency, No. of VCs



(c) Total power, Buffer size



(f) Total power, No. of VCs

Figure 5.18.: Impact of buffer size and number of virtual channels on area (a), (d), clock frequency (b), (e) and power consumption (c) (f).

Figure 5.18(e) shows the achievable clock frequency for different numbers of virtual channels. It is clearly visible that a large number of VCs impacts the desired frequency of 1 GHz. The reason is the critical path of the router, which emerges in the transmission control unit for higher numbers of VCs. Therefore, 10 or more virtual channels reduce the desired clock frequency and thus the performance of the NoC. Considering these results, the number of VCs should be chosen carefully with respect to the desired clock frequency.

Figure 5.18(f)[51] summarizes the minimum and maximum power consumption of different router configurations with varying numbers of virtual channels. Compared to additional buffer slots, more virtual channels have a greater impact on the power consumption. The gap between the minimum and maximum power consumption increases with a raising number of VCs. The main reason is the transmission control unit, which has a high power consumption if all VCs are occupied, as shown in the traffic scenario used to estimate the maximum power. Increasing the number of VCs from 4 to 8, increases the minimum or idle power consumption by 77 % and the maximum power consumption by 100 %.

5.4.3. Comparison to State of the Art NoC Implementations

In the previous sections, different variants of the NoC and router design have been compared against each other. The intention of this paragraph is to compare the router design against a selection of other existing designs. Unfortunately, the number of publications, presenting detailed implementation results for comparison is incredibly limited. However, the proposed router design is compared against other FPGA implementations and against an ASIC implementation.

5.4.3.1. Comparison to State of the Art FPGA NoCs

FPGAs are typically used for prototyping, systems produced in small quantities or reconfigurable systems, which exploit the reconfiguration capabilities of the FPGA. However, the implemented system must cope with the limited resources of the FPGA, regardless of the reasons for using an FPGA. Thus, a lightweight NoC is desired for FPGA implementation.

A resource-efficient, delay-optimized NoC configuration is required for FPGA implementation. The HDL template design, introduced in section 5.3, can be configured accordingly. The resulting, lightweight incarnation of the network, is termed *liteNoC*. LiteNoC does not use virtual channels to reduce its resource

[51] The drop of the power consumption, when the number of VCs is increased from 10 to 12, results presumably from the reduced clock frequency. However, this was not analyzed in detail.

requirements. The router is optimized to decrease transmission delay. Therefore, the number of pipeline stages is reduced to two, according to section 5.1.3.

The following investigations shall prove the efficiency of the lightweight incarnation of the proposed NoC template. Therefore, the performance and resource utilization of a *lite*NoC router is compared with two other packet switching router designs without VCs. Both reference designs, Hermes [180] and LiPaR [225], claim to be lightweight for FPGA implementation. For a fair comparison, different versions of liteNoC have been synthesized, using the flow presented in section 5.3.2.2. For synthesis, the FPGA devices and parameters of liteNoC are set according to the results presented for Hermes and LiPaR in [180] and [225].

Parameter	**Hermes**	***liteNoC***	***liteNoC* Freq.**
Latency (1Flit,1Hop)	12 cycles	3 cycles (-75 %)	3 cycles (-75 %)
Latency (4Flit,2Hop)	28 cycles	8 cycles (-71 %)	8 cycles (-71 %)
Used FPGA Device	Xilinx XC2V1000		
Clock Freq. (*MHz*)	25	80 (+220 %)	135 (+440 %)
Throughput (*Mbits/s*)	500	3200 (+540 %)	5400 (+980 %)
Slices	278	275 (-1 %)	330 (+18 %)
LUTs	555	459 (-17 %)	547 (-1 %)
Flip Flops / BRAMs	172 / 0	134 (-22 %) / 0	197 (+15 %) / 0

Table 5.4.: Comparison of a *liteNoC* and a Hermes 5-port router implementation on a Xilinx XC2V1000 FPGA.

Table 5.4 shows the comparison to the Hermes NoC. A 2x2 meshed NoC with a link size of 8 bit and a buffer size of 8 slots was used. For a moderate clock frequency of 80 MHz liteNoC outperforms Hermes in all aspects. Once the clock frequency is tuned to 135 MHz, liteNoC resource utilization is slightly higher compared to Hermes[52].

LiPaR is another NoC optimized for an efficient FPGA implementation. In [225], synthesis results of a standalone LiPaR router are presented. These results are summarized in table 5.5 as well as the results of a liteNoC router with equal parameters[53], synthesized for the same Xilinx XC2VP30 device. For a standalone router LiPaR outnumbers the liteNoC router resource utilization. However, liteNoC outperforms LiPaR regarding latency as well as clock frequency and thus has a much better performance and throughput. Additional synthesis results presented in [225] show that liteNoC outnumbers LiPaR in terms of slice utilization for a 2x2 NoC, where each router has only three ports.

[52] The reason are timing optimization during synthesis, such as register duplication and retiming. These techniques result in additional area consumption.

[53] The LiPaR NoC uses a flit size of 8 bit and a buffer size of 16 slots.

Parameter	LiPaR	*liteNoC*	*liteNoC* Freq.
Latency (1Flit,1Hop)	10 cycles	3 cycles (-70 %)	3 cycles (-70 %)
Latency (4Flit,2Hop)	28 cycles	8 cycles (-71 %)	8 cycles (-71 %)
Used FPGA Device	Xilinx XC2VP30		
Clock Freq. (*MHz*)	33.33	65 (+95 %)	115 (+245 %)
Throughput (*Mbits/s*)	1333	2600 (+95 %)	4600 (+245 %)
Slices	352	583 (+66 %)	706 (+101 %)
LUTs	772	957 (+24 %)	1283 (+66 %)
Flip Flops / BRAMs	478 / 10	240 (-50 %) / 0	358 (-25 %) / 0

Table 5.5.: Comparison of a *liteNoC* and a LiPaR 5-port router implementation on Xilinx XC2VP30.

The previous results show that the flexible NoC design template, proposed in this work, is also very efficient for building lightweight communication infrastructures for FPGAs. The instantiated liteNoC is superior to NoC implementations, such as Hermes and LiPaR, which are dedicated for FPGAs. LiteNoC was used to extend the Molen architecture [258], as described in detail in [PQHW+13].

5.4.3.2. Comparison to State of the Art ASIC implementations

Intel's SCC architecture [113] is one of the most advanced many-core architectures, as described in detail in section 2.2.5.2. The router design in the NoC of the SCC also uses packet switching with virtual channels. Thus, it can be fairly compared against the basic router design, introduced previously.

In [112], area and power consumption of the SCC router are presented for a 45 nm CMOS implementation. The router has a four-stage pipeline, uses 8 virtual channels and has a link width of 16 byte. The HDL template of the router design, proposed in this work, was configured accordingly to enable a fair comparison. Subsequently, it was also synthesized using also a 45 nm CMOS library.

Table 5.6 summarizes the comparison of the two router implementations. The largest advantage of the SCC router is its higher clock frequency, which is stated to be 2 GHz[54] in [112]. However, the proposed router design achieves only 1.15 GHz clock frequency under worst case operation conditions. Thus, throughput of the SCC router is almost twice as high as the presented design. This substantial difference is justified by worst case assumptions and by the use of another ASIC library. Comparing the area requirement of both routers, there is a significant advantage for the router design of this work. It requires less than half of the area needed to implement an SCC router with the same parameters. The lower area

[54]The NoC of the SCC prototype is only running stable at a much lower frequency of 800 MHz [265].

requirements are also reflected in a significantly lower power consumption of the proposed router. The much lower power consumption is on the one hand explained by the lower clock frequency. On the other hand, the lower area requirements also result in less power consumption compared to the router of the Single-chip Cloud Computer.

Parameter	SCC Router [112]	Proposed Router
Frequency (MHz)	2000	1150 (-43 %)
Latency ($cycles$)	4	4
Link Width ($Byte$)	16	16
Throughput (GB/s)	64	36.8 (-43 %)
Area Requirements (mm^2)	1.100	0.457 (-58 %)
Power Consumption (mW)	500	30.0 (-94 %) - 37.4 (-93 %)

Table 5.6.: Comparison of the proposed basic router implementation against a router of Intel's SCC architecture.

The comparison between the SCC router and the proposed router design shows the advantages of both implementations. If a high clock frequency is desired, the SCC router has clear advantages. However, if the NoC should not operate at frequencies higher than 1.15 GHz, the proposed router is beneficial due to its lower area and power consumption.

5.5. Extensions

Different extensions of the basic router are described in this section. Most of these extensions are necessary to carry out the concepts presented in the following chapters.

5.5.1. Control Network Layer

A so-called *control network layer* is used to enable NoC-internal communication. NoC-internal communication includes NA-to-router, router-to-router, router-to-NA or NA-to-NA communication. NA-to-NA communication is typically required for end-to-end flow control. However, in existing NoCs it is not envisaged that a router transmits or receives packets. It is only responsible for forwarding packets. In order to implement some of the self-optimization strategies, presented in chapter 7, router-to-router and router-to-NA communication is required. To enable packet injection by the router itself, an additional router port is required

or one of the existing router ports must be shared. However, both of these solutions have serious drawbacks: (1) adding an additional ports to each router results in a super-linear increase of the area and power consumption, according to equation 2.3[55]. (2) Sharing an existing router port leads to a disadvantage of the connected component due to increased latencies and reduced bandwidth. (3) Both solutions lead to additional load of the main NoC. Because of these disadvantages, it was decided to use a separate lightweight NoC for NoC-internal communication. Moreover, a separate NoC enables to completely decouple NoC-internal communication from inter-tile communication[56]. This is one important aspect that underlines the decision to use a separate NoC. The low implementation overhead of the control network, investigated later in this section, is another benefit of a separate NoC layer.

The *liteNoC* variant, introduced in section 5.4.3.1, is utilized as a lightweight control network. Due to the overhead of virtual channels, previously analyzed, it does not use VCs at all; resulting in reduced implementation costs. The reduction of the complexity enables to merge pipeline stages to reduce the transmission latency[57]. Depending on the bandwidth requirements of the control network, its link width might be small to keep the additional area consumption low.

Figure 5.19 shows the embedding of the control network. The control layer is also referred to as a control channel. If the respective feature of the NoC template is enabled according to appendix A.6, one *control router* is instantiated in each router. It is connected to the four adjacent routers and to the processing tile by using separate *control channel links*. The sixth port of the control router is used to connect a router's internal components, which must communicate to other routers or tiles. An example of such a component is one of the FSMs, used for the rerouting implementation, as detailed in section 7.1. In figure 5.19, such components are exemplary represented by the *FSM* module, attached to the router-internal port.

Figure 5.20 illustrates the head flit format, used in the control network. An additional bit in the head flit is used to address a router instead of a tile[58]. Consequently, a flit is forwarded to the router internal components, if the respective bit is set. However, depending on the configuration of the design, multiple components may exist in the router, which must use the control network. Another field, named *FSM ID*, is used to identify the component or FSM, in this case. This ID is then used to forward incoming packets to the appropriate component.

[55] The reason is the super-linear growth of the crossbar complexity with respect to the number of input and output ports.

[56] Decoupling of communication can be used to prevent deadlocks. End-to-End flow control between two NAs can e.g. be decoupled from other communication by using a separate network.

[57] Instead of 4-5 pipeline stages, used for the main network, only 2-3 pipeline stages are required in *liteNoC* to achieve the same clock frequency.

[58] In order to address the sixth port of the router, the same address as for the tile/NA attached to the router is used.

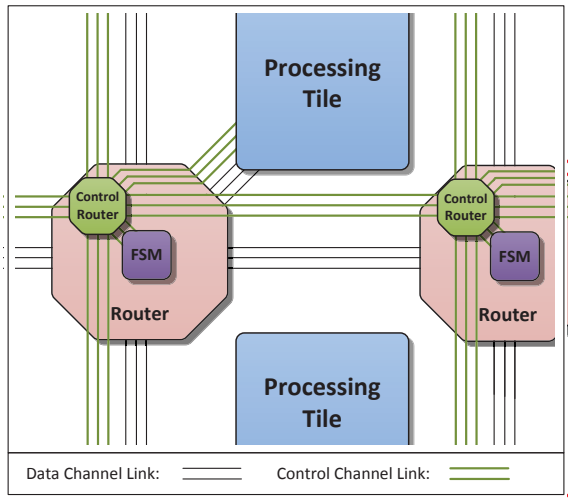


Figure 5.19.: NoC with separate data and control channel. The control network has an additional port for the router itself.

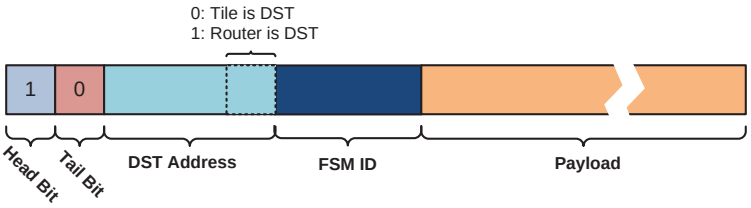


Figure 5.20.: Head flit with additional fields to address router-internal components in the control network.

However, the control network does not only enable to receive data by the router of the primary NoC, it also enables to generate messages in the router and transmit them to other routers or network adapters. An arbitration unit is used within the router to manage the transmission process in case of multiple components with a capability to generate and transmit packets. The arbitration unit enables a fair access to the control network by all components. Therefore, a round-robin arbitration is utilized.

Thus, the implementation overhead of the control network will be quantified. Therefore, a reference design with the same parameter setting as in section 5.4.3.2, is used. A single router is synthesized with and without a control channel. The control channel is configured to have a link width of 16 bit. The results show, that

the control router does not impact the clock frequency of the design. The area consumption increases by only 4.0 %, if a control network is used. The power consumption of the idle router rises by 2.3 %, when adding a control channel. These results show the small additional cost resulting from the implementation of the control channel.

5.5.2. Circuit Switching Extension

Compared to packet switching, circuit switching typically has a lower power and chip area consumption [269]. Moreover, its latency is typically much lower than for packet switching. A detailed comparison of packet and circuit switching is provided in section 2.4.3. However, a pure circuit switching NoC cannot be built in a distributed way. The reason is the lack of best-effort support, which is necessary to manage the circuit switching connection setup in a distributed way. Consequently, a circuit switching network is typically combined with a packet switching network to enable scalability. Prominent examples are the Æthereal NoC [92] and the Tilera iMesh NoC [18]. Both architectures combine packet and circuit switching to make use of the advantages of both concepts.

As part of a student's work [Sch13], a circuit switching extension for the proposed router design, was developed. It combines the packet switching router and a circuit switching router to form a hybrid NoC. A schematic block diagram is provided in figure 5.21. As shown, the packet switching router (white) is coupled with the circuit switching extension (green). Currently, the liteNoC router, introduced in section 5.5.1, is used as packet switching router[59]. Optionally, link sharing can be enabled, which necessitates an input and output multiplexer, as well as the *link share control* unit. Link sharing allows the same link to be used for packet and circuit switching communication. For example, the optional sharing of links can be used to reduce the number of TSVs, in case of 3D integration. If the number of wires between neighboring routers plays a minor role, separate links may be used for packet and circuit switching; improving the throughput.

The circuit switching router uses a TDM scheduling scheme, introduced in section 2.4.3.4, in order to enable multiple parallel circuit switching connections at each router port. If link sharing is activated, free TDM time slots are used by the packet switching router to transmit flits. Link sharing improves the link utilization, as shown in the following evaluation. The reason for the relatively low link utilization of a circuit switching network is the restriction for the time slot assignment due to the synchronous scheduling, as described in detail in

[59] The liteNoC router was used to keep the initial circuit switching design simple. However, it can be replaced by the more complex router design with VCs, with minimal modification of the circuit switching components.

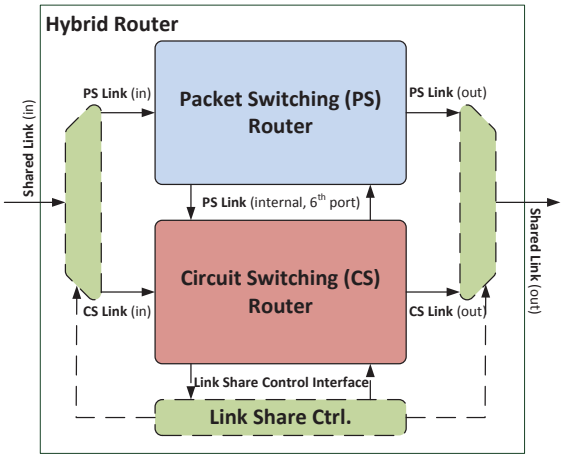


Figure 5.21.: Schematic structure of the hybrid router. The packet and circuit switching sub-router share the same link optionally.

section 6.1.2. The search of a valid time slot assignment at each router, passed by a new connection, is very challenging to implement in a distributed way.

Figure 5.22 shows a scenario where some time slots are currently in use. Assuming that a new connection shall be established between node 0x0 and node 2x2, only one valid path and *time slot (TS)* assignment exists[60]. This path begins at the local port of router 0x0 with TS 3, uses TS 0 between router 0x0 and 0x1 and continues between router 0x1 and 1x1 using TS 1. Subsequently, TS 2 is used between router 1x1 and 1x2, followed by TS 3 at the link to router 2x2. At the local port of router 2x2, TS 0 is again used.

The previous example should give an impression of the complexity of a circuit switching connection setup, including time slot assignment. A depth-first-search routing strategy is utilized to search for a valid path and TS assignment in a distributed way. For connection management, the packet switching network must be used. A request packet is sent from hop to hop towards the destination node, containing a list of the valid time slots. If a TS is not available in one of the routers, the respective entry in the *request packet* is set to 0. When the list of valid time slots is empty before the packet reaches the destination node, the path cannot be used. Then, a *fail response packet* is sent back on the same way. Subsequently, an alternative route is taken by another request packet originating from the source router. This process is continued until a valid route is found or all

[60] For the example, it is assumed that only those links are taken into account, where the time slot utilization is given. Each router has a transmission delay of one cycle. All routers schedule the same TS synchronously.

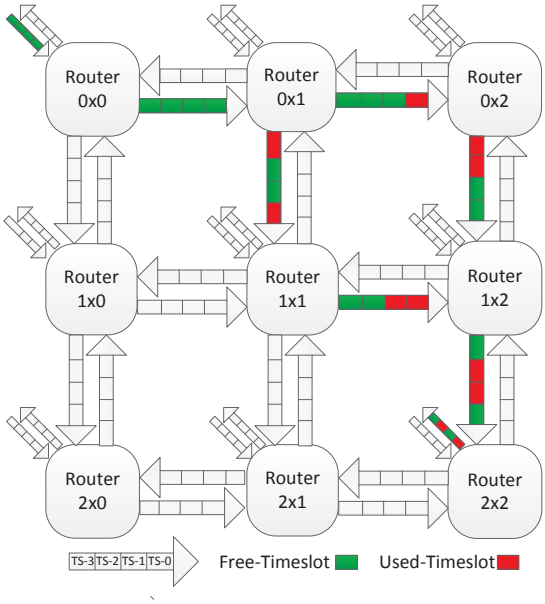


Figure 5.22.: Example of a 3x3 circuit switching NoC with free and used time slots [Sch13].

possible alternative paths have failed. If a valid path and time slot assignment is found, a *set packet* is used to activate the route and the pre-allocated time slots. Release of an established circuit switching connection is again performed by the use of BE packet switching communication. A *release packet* is sent from router to router to close a connection hop by hop.

Within the circuit switching router component the *circuit switching control* unit, shown in figure 5.23, enables distributed connection management. It is attached to an additional port of the packet switching network, similar to the control network, introduced in section 5.5.1. The *circuit switching control* unit consists of a four stage pipeline. The first and the last stage may be occupied by a packet for multiple cycles, depending on its size. The first stage is the *input-FSM*. It is used to extract the data, such as the valid time slots from incoming packets. The *packet selector* is used to select a packet for processing from the input-FSM or *waiting-queue*. The waiting-queue is used to store requests, which cannot be processed directly. Requests, selected by the packet selector for processing, are forwarded to the *execution unit*. It is used to process, request, fail, set and release packets as well as controlling the reservation table of the circuit switching network. The execution unit is also used to trigger the generation of new packets by the *output-FSM*.

According to the depth-first-search routing strategy, reservation request packets are generated and injected into the packet switching network by the output-FSM.

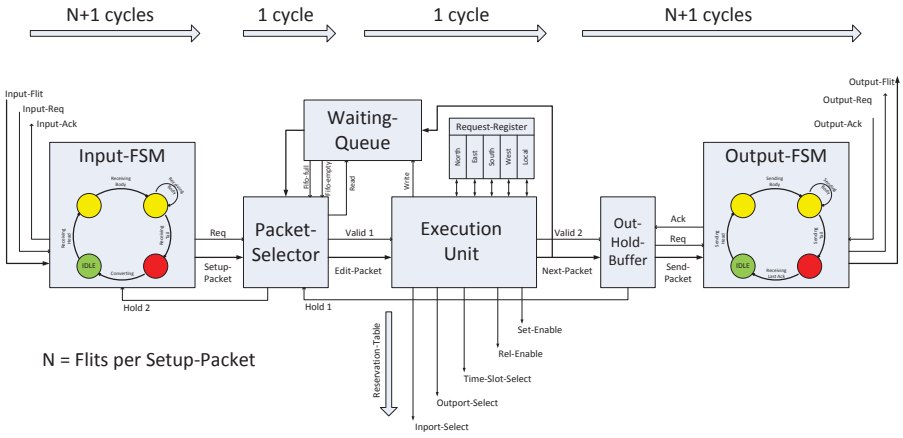


Figure 5.23.: Structure of the circuit switching control unit. It enables distributed setup and release of circuit switching connections [Sch13].

The reservation table of the circuit switching network performs the TDM scheduling. It controls the crossbar of the circuit switching sub-router. Therefore, the reservation table entry of the selected time slot is elaborated. When link sharing is activated, the reservation table is also responsible for triggering packet switching transmission.

A so-called *pass-signals* enables to reuse time slots, reserved for circuit switching, for packet switching transmission. Therefore, the pass-signals is set at the start-point of a connection as long as the connection is idle. An active pass-signals is taken into account by the TDM scheduler of each router. The use of the pass-signals can increase the link utilization, as shown by the following results.

More details about the functioning of the distributed circuit switching connection management and the link sharing concept as well as implementation details are provided in [Sch13]. However, circuit switching is only considered an optional extension of the packet switching NoC, within the scope of this work.

5.5.2.1. Evaluation

The circuit switching extension is evaluated by comparing a pure packet switching and a pure circuit switching network with two variants of the proposed hybrid NoC design. A 4x4 mesh topology is used to evaluate the achievable throughput

or bandwidth. The circuit switching NoC and the hybrid NoC design have 4 TDM time slots. NoC load is generated by setting up 20 random circuit switching connections and best-effort packet switching traffic with uniform random distribution. The results are shown in figure 5.24. The *CS+PS* variant represents the hybrid router design with shared links; used for packet and circuit switching. The *pass* value represents the amount of cycles where the *pass-signal* is active[61]. *Only PS* is a packet switching NoC without virtual channels. *Only CS* in turn is a pure circuit switching implementation. A hybrid router with separate links for packet and circuit switching is represented by *CS+PS, 2Links*.

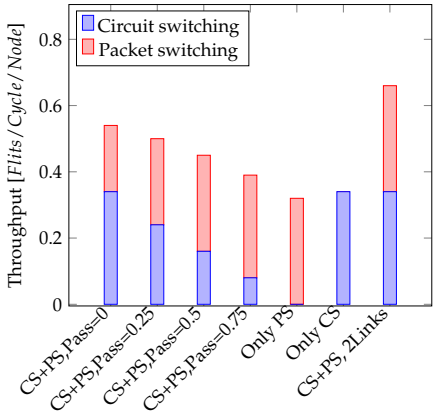


Figure 5.24.: Achievable throughput for packet switching (*PS*), circuit switching (*CS*) and for the proposed hybrid network (*CS+PS*) with shared (*Pass*) and separate links (*2Links*) [Sch13].

The investigations, provided in figure 5.24, show that the hybrid router design *CS+PS* achieves a higher throughput compared to a pure packet or circuit switching variant. If the workload of the circuit switching decreases, indicated by active pass-signals, the total bandwidth utilization also decreases. However, the bandwidth for packet switching communication increases due to the bandwidth stealing mechanism. Thus presenting the benefit of the pass-signals. When the packet and circuit switching router use separate links (*CS+PS, 2Links*), the bandwidth is of course higher. However, the performance penalty resulting from link sharing is only 18 % for the given scenario. Thus, link sharing may be used with a small bandwidth penalty at positions in the NoC, with the number of wires playing an important role. Other links can be built separately for packet and circuit switching. A more detailed performance evaluation is provided in [Sch13].

[61] An active *pass-signal* reduces the bandwidth of circuit switching traffic but increases the bandwidth for packet switching.

The implementation costs for an ASIC realization of the proposed hybrid router design are also analyzed in detail in [Sch13]. Compared to a packet switching router with two virtual channels, it consumes 20 % less area, while achieving approximately equal throughput.

To summarize, the hybrid router design offers promising results with regards to performance and cost. It offers a lower latency and power consumption as well as an improved link utilization compared to pure packet or circuit switching. In future versions of the InvasIC architecture, the hybrid router design can be used to offer packet and circuit switching communication on demand.

5.5.3. Monitoring Infrastructure

Monitoring is a key enabler for debugging, performance analysis and optimization. With respect to communication, monitoring can be used to identify hotspots and bottlenecks, subsequently react on them. In the context of invasive computing, monitoring is used to enable optimization at the software layers of the system. Therefore, the monitoring information is taken into account while allocation of resources and for task mapping, as described in detail in [HZW+14]. Furthermore, monitoring is also used by the NoC hardware itself to enable self-optimization based on the current utilization status of the network. Thus, monitoring is also necessary to implement some of the self-optimization strategies, presented in chapter 7. In addition, monitoring information is used by the adaptive routing scheme, discussed in section 2.4.5.

A modular monitoring infrastructure is created in the NoC routers. The monitoring units can be configured at design-time and run-time to meet the system requirements. Three different types of monitors are available:

- ***Link utilization (LU)*:** Monitor the utilization of all outgoing links to neighboring routers and to the network adapter at the local port. LU monitors are used during resource allocation to obtain the available bandwidth.
- ***Virtual channel utilization (VCU)*:** Providing information about the utilization and availability of virtual channels. They are used to estimate if additional GS connections, used for invasion of communication resources, can be established. This scheme is described in detail in section 6.2.1.5.
- ***Buffer utilization (BU)*:** These monitors capture the buffer fill level. They can be used to detect back pressure and overload of existing GS connections. The buffer utilization monitoring is only used for detailed analysis of the NoC performance.

All monitors measure the respective utilization for a predefined period. Counters are used to sample the respective signals in the NoC routers, as shown in fig-

ure 5.25. A separate monitor is used for each port or VC respectively. The values, captured by the monitors, provide the average utilization within the measurement period P. The length of the period P can be defined at run-time between 1 and P_{max} by using the memory mapped configuration registers of the router. Details about the monitoring registers are provided in appendix A.4. The run-time configuration enables the adjustment of the monitoring period according to the application behavior. In addition to this average monitoring, the VCU and BU monitors measure the peak utilization. This is the highest number of used VCs or buffer slots being utilized at the same time. A history of the last N periods is stored in the monitoring units. The history enables to obtain information about the evolution of the utilization during the past periods. The size N of the history can be defined at design-time by the use of the parameters, defined in table A.14.

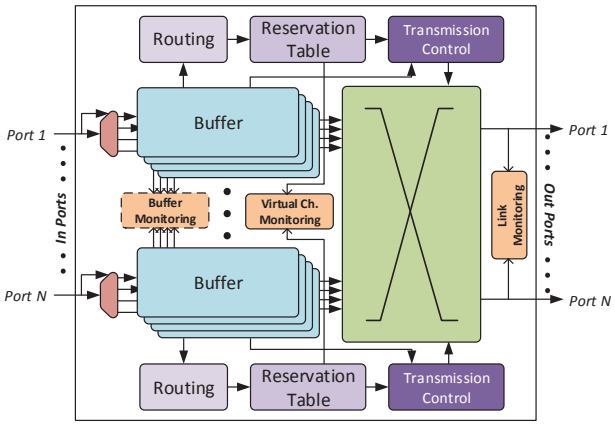


Figure 5.25.: Packet switching router with monitoring extensions.

There are ways for accessing the monitoring information. The most common way is the access of monitoring data by the use of memory mapped registers. This approach is applied in the InvasIC architecture, where the monitoring information are access via memory mapped registers, located inside the invasive network adapter. These registers are explained in appendix A.4. Furthermore, a finite state machine was built to enable access of the monitoring data through the aforementioned control network. The FSM also supports periodic transmission of monitoring data to distribute them automatically. More details about the FSM-based monitoring data transmission are provided in [Kar12]. The aforementioned methods are used to enable monitoring data access by the software or operating system managing the hardware. A concept, describing the usage of monitoring data by the OS, is presented in section 6.2.1.5. However, the monitors are also used as a basis for self-optimization in the NoC. Therefore, the monitoring data

is forwarded router-internally to the components making the decisions for self-optimization. A detailed description is provided in chapter 7.

5.5.3.1. Overhead Analysis

The implementation overhead of the monitoring infrastructure is investigated now. Therefore, two baseline versions of the NoC router are used: The *Base32* variant has a link size of 32 bit. It is the router version used for the FPGA prototype, presented in appendix A.1. LU and VCU monitoring is used in the prototype. However, that link size and the resulting bandwidth would be too small for an ASIC implementation. Thus, a second variant (*Base256*) with 256 bit link size is used to estimate the overhead of a router. This variant adheres to the requirements of an ASIC design, which contains a large number of cores. A link size of 256 bit is reasonable for a large many-core architecture, considering Intel's Single-chip Cloud Computer [113], which has a link size of 144 bit to connect 24 tiles with only 48 cores.

Router Version	**Frequ.** (MHz)	**Area** (μm^2)	**Power** ***(Min. - Max.)*** (mW)
Base Router (32 bit)	1500	50039	7.9 - 12.7
Base32 + LU,VCU Mon.	1500	53777 (+7.5 %)	9.3 (+16.3 %) - 14.2 (+12.3 %)
Base Router (256 bit)	1500	250728	23.9 - 28.1
Base256 + LU,VCU Mon.	1500	255992 (+2.1 %)	25.2 (+5.1 %) - 29.3 (+4.3 %)
Base256 + LU,VCU,BU	1500	261376 (+4.2 %)	27.9 (+16.3 %) - 32.1 (+14.1 %)

Table 5.7.: Implementation overhead of the monitoring infrastructure for two different router variants [HZW+14].

The synthesis results for a 45 nm ASIC implementation of a single 5-port router with four virtual channels are provided in table 5.7. The results for the small *Base32* router show that link utilization and virtual channel utilization monitoring increases the area consumption by 7.5 % and the power consumption between 12.3 % and 16.3 %. The critical path and thus the clock frequency is not affected by the monitoring infrastructure. As such, the desired clock frequency of 1.5 GHz is achieved for all router variants[62].

For a real ASIC implementation of a large NoC-based architecture, the synthesize results of the router with a link size of 256 bit are more meaningful. Such a link size better reflects the bandwidth requirements of a large and scalable architecture. The synthesis results in table 5.7 show that the *Base256* ASIC implementation,

[62] A clock frequency of 1.5 GHz can be achieved with four virtual channels. If the number of virtual channels is higher, the critical path in the transmission control unit limits the clock frequency. This is the case in the variant used in section 5.4.3.2.

equipped with link utilization and virtual channel utilization monitoring, has a 2.1 % higher area consumption compared to the baseline. Power consumption is increased by 5.1 % at maximum. The achievable clock frequency of 1.5 GHz is not affected.

Buffer utilization monitoring has a much higher implementation overhead compared to LU and VCU monitoring. In contrast to LU and VCU monitoring, BU monitors are not required to make decisions at run-time. Both the operating system and the self-optimization mechanisms necessitate only LU and VCU monitoring. Thus, BU monitors should only be included within a design if detailed back-pressure analysis for individual applications is required.

In general, it can be summarized that link utilization and VCU monitoring have an acceptable implementation overhead for a realistic ASIC implementation with a 256 bit NoC. More details about the monitoring infrastructure and its use in the context of invasive computing are provided in [HZW+14].

5.5.4. In-Order Packet Processing Support

In-order delivery or in-order arrival is an important requirement of a communications system. For instance, it is necessary to prevent race conditions during memory access, which could result from data hazards[63]. In order to prevent such race conditions, it must be ensured that packets, transmitted between two nodes, are delivered in the order of transmission[64].

In the case of adaptive routing, where packets from the same source to the same destination can take different paths, the data may arrive in the wrong order. The reason is different load situations and consequently varying delays on different paths. Consequently, a reorder mechanism is necessary in the network adapter to deliver the data in the original order for prevention of race conditions. Such an NA implementation with reorder buffers, for in-order delivery, is presented in [67]. However, such reordering mechanisms typically include a huge implementation overhead because of the large and complex reorder buffers. Alternatively, deterministic routing may be used to ensure in-order arrival. When all packets with the same source and destination node take the same route through the NoC, in-order delivery can be ensured by in-order processing within each router. If no virtual channels are used, in-order processing is typically done implicitly. However, the proposed router design uses virtual channels. Thus, multiple packets may be

[63] There are three situations that can lead to a data hazard: read after write (RAW), a true dependency; write after read (WAR), an anti-dependency; write after write (WAW), an output dependency.

[64] In-order delivery can only be ensured for one sending and one receiving node. If multiple nodes are involved in a communication scenario, synchronization mechanisms must be used to prevent race conditions.

processed in parallel, which could lead to a situation where they pass each other. Consequently, a special mechanism is required to ensure in-order processing.

In the proposed router, in-order processing is ensured by modifying the reservation process in the *output reservation table*. Therefore, the output reservation table is extended to determine whether there is an existing reservation for a packet that has the same destination address. If this is the case, the reservation request is rejected to avoid parallel reservation of multiple VCs for packets with the same destination address. This mechanism ensures that packets, which are transferred between the same pair of nodes, are processed sequentially and thus in-order.

The implementation has been verified in simulation by examining the packet-order in each router for multiple traffic scenarios. In order to investigate the implementation overhead for in-order processing, a reference design with the parameter setting from section 5.4.3.2 is used. Compared to reorder buffers in the NA [67], the modification of the *ORT* is very lightweight; with the area consumption only being increased by 1 %. As such, neither the idle power consumption nor the clock frequency is affected.

With respect to performance, it could be expected that the restrictions, which are necessary to ensure in-order processing, have a negative impact on delay and throughput. Latency and throughput of an 8x8 mesh with in-order processing support are compared against the reference implementation in order to investigate the performance impact. The results show that latency is slightly reduced, by an average of 1.1 % for random best-effort traffic, with different injection rates. Astonishingly, throughput is increased by an average of 2.6 %, when in-order processing is enabled in the router design. This increase results from the restriction to only use one VC at each router for packets to an overloaded destination node. This restriction reduces the probability that other packets experience head-of-line blocking.

In summary, it can be stated that in-order delivery can be ensured by the proposed extension of the routers with very low additional costs and with negligible impact on latency and throughput.

5.5.5. High Bandwidth Router

The InvasIC architecture, introduced in section 3.2, is a heterogeneous architecture, consisting of different types of tiles. In contrast to homogenous architectures, where all tiles have typically the same bandwidth requirements, varying demands may exist for different types of tiles. The TCPA or the memory tile of the InvasIC architecture are tiles, which are expected to have higher bandwidth demands than others, as described in detail in section 3.2.1.

Figure 5.26 shows an example of a heterogeneous architecture with a memory node being accessed by other nodes. The example illustrates memory communication arriving at different input ports of the router attaching the memory. All communication must be forwarded through the local port to the memory. Vice versa, the requested data is transferred through the local port, split up inside the router and subsequently forwarded to the requesting processing nodes. As shown in the example, the local port of the router connecting the memory can become a bottleneck from the bandwidth perspective.

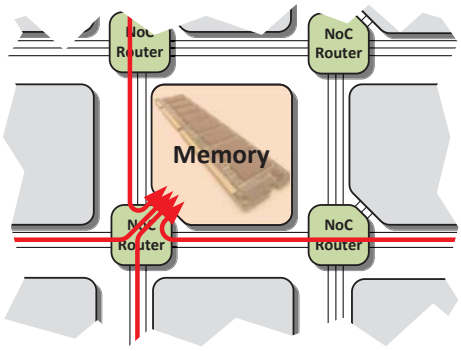


Figure 5.26.: Memory node surrounded by processing nodes. Memory communication meets at the local port.

A more specific scenario, which shows that the local port of the memory nodes can become a hot spot, is provided in figure 5.27. The link utilization of each router port in a 6x6 mesh with 4 memory tiles and 32 processing tiles is analyzed. Each memory tile (M) is surrounded by 8 processing tiles (P). The measurements are obtained from the simulation framework. The application model from section 5.2.3.2 is utilized to model memory communication as well as core-to-core message passing communication. It can be observed that the local ports, connecting the memory tiles, have a utilization around 100 %, whereas other ports in the NoC have a much lower utilization.

The previous examples showed that the local port of a normal 5-port router, as shown in figure 5.28(a), can become a bottleneck for memory communication. The reason is that aggregated bandwidth is required at the local port. The total throughput TP_{out} of an output port p_{out} can be calculated as follows:

$$TP_{max} \geq TP_{out}(p_{out}) = \sum_{i=0}^{N-1} TP_{in}(p_{out})(i) \tag{5.15}$$

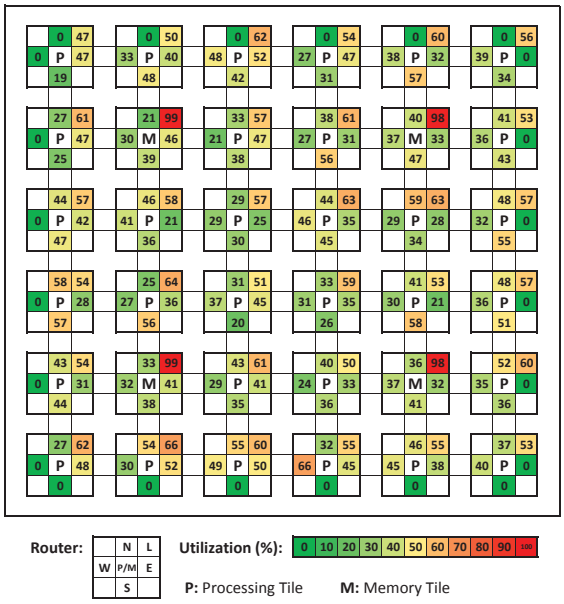


Figure 5.27.: Output port utilization of each router in a 6x6 mesh with 4 memory tiles (M) and 32 processing tiles (P) [HBRB13].

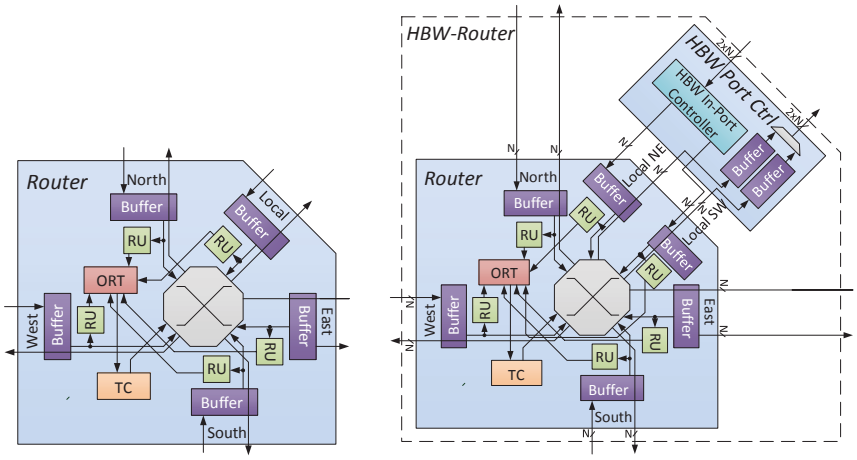
In equation 5.15, TP_{max} is the achievable, maximum throughput per port. N is the number of ports per router. TP_{in} is the incoming throughput at input i, which will be forwarded to p_{out}. According to equation 5.15 the output of a 2D meshed NoC could be over-utilized by a factor of four[65]. In other words, if all ports must use the same output port with a quarter of TP_{max}, TP_{out} is utilized to 100 %. This is the case for the memory tiles in figure 5.27.

In order to avoid frequently used ports becoming a bottleneck, the general bandwidth of the network may be increased by increasing the link size. However, this would reduce the relative utilization of all ports in the network, but results in huge implementation overhead, as later analyzed. Thus, it is more efficient to increase the bandwidth only where it is required. Therefore, a router with a higher bandwidth at the local port is desired and consequently designed. This router is named *high bandwidth (HBW) router*. The HBW router shall forward packets from the four cardinal-direction input-ports (N, E, S, W) with increased bandwidth to

[65] Assuming an equal bandwidth at all ports and data that do not leave the router through the same port as they have entered it.

the connected node and vice versa. The router is used to attach nodes with high bandwidth demands[66].

In general, two options exist for increasing the bandwidth of the local port, assuming a constant clock frequency: (A) increasing the link capacity by increasing the width of the link or (B) increasing the number of links. It was decided to use multiple links from the router perspective, thus increasing the number of links. This has two main advantages: (1) it enables a homogeneous router design with equal capacity and size of all links[67] and (2) the number of virtual channels, which could be used by GS connections to the tile, increases if the number of local ports is increased[68].



(a) Normal 5-port router (b) High bandwidth router

Figure 5.28.: Structure of a normal 5-port router (a) and a high bandwidth router (b) with two internal local ports [HBRB13].

The high bandwidth router, resulting from the previous deliberations, is shown in figure 5.28(b). Compared to a standard 5-port router, similar to the example shown in figure 5.28(a), it has a second local port to double the bandwidth to the tile. Typically, the two ports must be merged to form a single port with double

[66] In case of a memory tile, this implies that the attached memory and the memory controller support this higher bandwidth. However, this is required anyhow if the memory shall not become a bottleneck. The bandwidth of the memory controller and the memory itself is out of the scope of this work and thus assumed to be sufficient.

[67] Increasing the capacity of one out of five ports would make the router implementation very complex, due to the heterogeneity resulting from different link sizes of the ports.

[68] Additional VCs also increases the number of possible GS connections that can be used for predictable memory communication. Thus, more nodes can profit from GS memory communication.

capacity to connect it to a single tile. This is necessary to attach a single memory or memory controller respectively. A module referred to as *HBW port controller* is used to merge the two local ports, also shown in figure 5.28(b). The *in-port controller* is used to distribute the communication flows from the tile to the two local ports. Memory access requests, arriving at the memory node, are inserted into the output buffer. The requests of the buffers are processed using a fair round-robin scheduling.

Increasing the number of router ports from five to six results in an increase of the crossbar complexity, as described in section 2.3.2.4. In order to reduce the complexity and implementation overhead of the crossbar, each regular router input (N, E, S, W) is assigned to one of the two local ports. This reduces the number of inputs of the crossbar multiplexers, which are used for forwarding data to one of the local ports, from 4 to 2. The routing unit (*RU*), shown in figure 5.28(b), must be slightly adapted to select one of the two local ports for data to be forwarded to the tile. In the current implementation, the north and the east port share one output and the south and the west port share one. Consequently, the two local ports are named *Local_NE* and *Local_SW* respectively. In order to further optimize the crossbar, it is required to allocate packets entering the router from the local ports, according to the used output port. This enables to reduce the size of the multiplexers of the four cardinal directions from 5 to 4 inputs. The in-port controller is used to ensure that incoming packets from the memory node are injected into the correct local input port. Therefore, it contains separate buffers for each of the two local input ports. They are used to keep both ports busy. The in-port controller is a wrapper, which forms one single router port with a doubled link width from the tile perspective. This enables a straight forward implementation of the network adapter attached to the NoC. More details about the implementation are provided in [HBRB13] and [Bis12].

5.5.5.1. Evaluation

Performance The performance improvement of a high bandwidth router is investigated by analyzing the memory communication of different applications. Therefore, it is assumed, that L2 cache misses result in memory communication through the NoC, according to the InvasIC architecture. The *independent execution time (IET)* is defined as the number of cycles between two L2 cache misses. The analysis of different *SPEC CPU2000* benchmarks [51], described in [HBRB13], exhibits large differences between various applications. For example, a *gcc* kernel has high communication demands. It has an IET of 59, which means that one cache miss occurs every 59 cycles. In contrast, a *gzip* application requires minimal main memory bandwidth with an IET of 2922 cycles. In order to generate the

memory communication according to the given IET, a memory communication model was built.

In the following paragraphs, two memory placement variants named *border* and *center* are investigated. For border placement, the memory nodes are attached to the left and right border of the mesh[69]. Center placement distributes the memory in a way that reduces the average distance to the processing nodes.

Figure 5.29(a) shows the load at the memory node for different cache miss rates, represented by the independent execution time. As expected, the average memory load increases if the independent execution time decreases. The reason is that a higher L2 miss rate, which is inversely proportional to the independent execution time, results in more packages in the NoC and thus in higher load. The results show that the load characteristic of the normal 5-port router, referred to as *low bandwidth (LBW)* router, is almost independent of its placement. Using the proposed HBW routers to connect the memory improves the bandwidth to the memory node by 15 % for border placement. Placing a memory node at the center of the NoC increases the possible throughput even by 25 % compared to the reference LBW router.

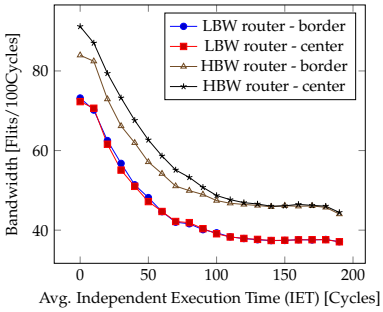
Figure 5.29(b) shows the impact on memory access latency. Placing the memory nodes centric, reduces the latency compared to a placement at the borders. The reason is the reduced average distance from the CPU to the memory nodes. At higher load and consequently smaller independent execution time, the latency increases for all configurations. Replacing a normal router with centering placement with the proposed HBW router reduces the memory communication latency by up to 26.7 %.

Next, the impact of the buffer size of the HBW port control module is investigated for central placement. These buffers are required to supply data to both input ports of the HBW router in parallel. Thus, the buffer size affects the performance. The impact on latency is rather small when changing the buffer size. The results for different buffer sizes do not differ significantly from the latency measurements of the centric HBW router, shown in figure 5.29(b). As such, the respective results are not presented here. Figure 5.29(c) shows the usable bandwidth for a normal router and the HBW router with 4, 8 and 16 buffer slots[70]. When the IET is low, the utilization of the memory tile is obviously high. A buffer size of 16 increases the bandwidth by up to 26 % compared to a HBW router with only 4 buffer slots. Compared to the LBW variant, the HBW router with 16 buffer slots has a 59 % higher throughput in case of high L2 miss rates.

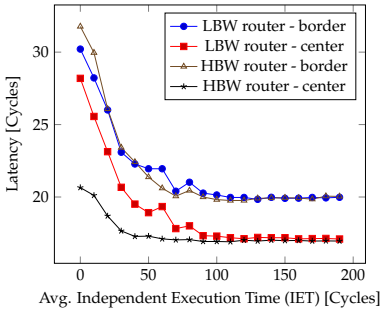
Further performance evaluations are provided in [HBRB13]. There, it is also shown that a HBW router can be used to double the number of GS connections

[69] In Intel's SCC architecture [113], the memory controllers are placed at the borders of the mesh.

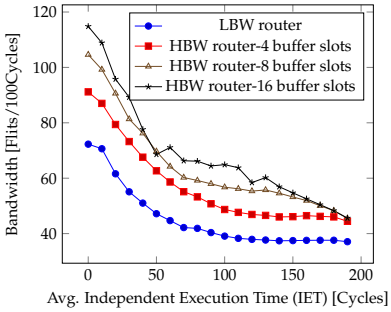
[70] For the previous studies, shown in figure 5.29(a) and figure 5.29(b), HBW routers with 4 buffer slots were used.



(a) Usable NoC bandwidth to the memory for different placements and routers



(b) NoC latency for memory access for different placements and routers



(c) Usable NoC bandwidth to the memory for various in-port control buffer sizes

Figure 5.29.: Performance evaluation for different routers, placements and HBW in-port control buffer sizes [HBRB13].

to a memory node. This can be very beneficial when using the QoS scheme, introduced in section 6.2, for memory communication.

Implementation costs The SystemVerilog implementation of the HBW router, which is derived from the basic router architecture presented earlier in this chapter, is used to investigate the implementation costs. Therefore, the synthesis flow for the 45 nm ASIC implementation, presented in section 5.3.2.1, is used.

The synthesis results of a normal 5-port router (LBW) and a HBW router with a link width of 32 bit are compared. The impact of the buffer size of the HBW in-port controller is also investigated. The results, presented in [HBRB13] in detail, show that the area utilization of the HBW router depends heavily on the size of the buffers. For a small buffer size of 4 slots, the HBW router has an area overhead

of only 32 % compared to the LBW variant[71]. For a buffer size of 16, the area overhead is around 60 %. However, it must be taken into account that only a small number of routers in the NoC have to be replaced by HBW routers. In case of 4 memory nodes in a 64 tile architecture only 4 HBW routers are required. Thus, the area consumption of the NoC grows by only 3.8 %, if HBW routers are used. The router clock frequency is not impacted by the HBW implementation, since the depth of the crossbar is not increased.

As an alternative to the proposed HBW router, the memory bandwidth could be improved by increasing the general link size of the NoC. A normal router with 64 bit link size instead of 32 bit was synthesized to compare the implementation overhead to the HBW router approach. The doubling of the link size would result in an additional area consumption of 53.2 % for the entire NoC, which is huge compared to the 3.8 % overhead of a NoC with 4 HBW routers. Consequently, HBW routers are an efficient way to avoid communication bottlenecks in heterogeneous tiled many-core architectures, such as the InvasIC architecture.

5.6. Summary

In this chapter, the design of the best-effort packet switching network on chip was described. The implementation of the basic router design with virtual channels and its pipeline model were described. The use of VCs plays a key role for the realization of the QoS concepts, presented in chapter 6. Subsequently, the concept and implementation of the network layer protocol was presented. Two different distributed routing schemes were introduced in order to ensure scalability. A worst case analysis for the throughput and latency of best-effort communication was presented in order to motivate the necessity of guaranteed service communication, enabled by the concept presented in section 6.2.

A SystemC-based simulation framework for the proposed network on chip was introduced in section 5.2. It is used for the following evaluations and as part of the design flow, presented in chapter 4. The concept of this framework and a distinction from existing simulation environments was provided. Subsequently, different traffic generation methodologies were described. They enable an analysis of a specific NoC configuration under various traffic scenarios. This is very important for the design of general-purpose communication infrastructures, where different communication patterns are expected to occur at run-time. The realization of a cycle-accurate simulation model of the NoC was then described. The evaluation

[71] When taking the crossbar complexity, provided by equation 2.3, a 6-port router has an overhead of 50 % ($\frac{5 \cdot 6}{4 \cdot 5}$) compared to a 5-port router. Taking into account that the HBW port controller comes along with additional implementation costs, the implementation overhead is relatively low.

and analysis capabilities of the simulation framework were described at the end of the section.

The HDL model of the proposed NoC design was described in section 5.3, building the second part of the design and evaluation flow that was introduced in section 4.4. The HDL model and the synthesis flow, presented in section 5.3.2, are used in the design and evaluation flow and in the following chapters to estimate clock frequency, power consumption and area requirements.

In section 5.4, a case study was presented in order to demonstrate the capabilities of the semiautomatic design flow. The previously introduced simulation framework and the HDL model are used to analyze the impact of the number of VCs and buffer slots on performance, power and area consumption. The case study is also used to find appropriate settings for the following investigations. In section 5.4.3, a comparison to state of the art router designs is presented, showing the resource and power efficiency of the proposed design as well as its increased performance compared to existing NoC architectures.

At the end of the chapter, different extensions of the basic router design were introduced. In particular, a control network layer, a circuit switching extension, a monitoring infrastructure, in-order packet processing support and a high bandwidth router extensions were proposed. The extensions are designed in a way that they can be added to the base design on demand. They are used to enable the additional concepts, presented in chapter 6, 7 and 8 and to fulfill the requirements of specific architectures. For instance, the control network is used to enable the self-optimization scheme, presented in section 7.1. The monitoring infrastructure is also necessary in order to implement the self-optimization scheme, which will be introduced in chapter 7. The high bandwidth routers can be used to fulfill the demands of a heterogeneous many-core architecture, just as the InvasIC architecture.

The NoC design, the simulation framework and the HDL model, as well as the extensions, presented in this chapter, are the basis for the concepts of chapter 6, 7 and 8.

6. Quality of Service

An important aspect of a communication infrastructure is *quality of service (QoS)*. In general, quality of service is the overall performance of a network, particularly the performance seen by the users of the network. Different aspects, such as error rates, availability, bandwidth, throughput or transmission latency, are considered to quantify QoS capabilities of a communication system. However, in the context of networks on chip, quality of service can be defines as follows:

Definition. *Quality of service is the bandwidth or latency experienced by a component using the network on chip.*

QoS is often used in the context of certain guarantees for bandwidth or latency. The reason is that quantitative assertions about latency or throughput are usually only possible in case of guarantees. Such guarantees are typically named guaranteed QoS. In the context of networks on chip, guaranteed QoS is defined as follows:

Definition. *Guaranteed quality of service is the capability of a network on chip to provide certain promises for throughput and/or latency. If only throughput is guaranteed, this is referred to as guaranteed throughput (GT). If throughput and latency are guaranteed, this is referred to as guaranteed service (GS).*

Quality of service is particularly important for the transport of traffic with special requirements. Typically, this is the case for traffic or communication that is part of an application with real-time, safety or security demands. One example of an application with soft real-time requirements, is the MPEG4 video decoder, introduced in section 5.2.3.3. When this distributed realization of the decoder is used as part of a video conference system or media player, a shortfall of the required bandwidth between the individual tasks would lead to a juddering picture. Another example for QoS demands are architectures, used as part of a safety-critical or security-critical system in the embedded domain. A typical example of such an embedded system is a flight management system of an aircraft or an electronic control unit within a car.

In general, packet switching networks do not support quality of service. In case of congestion, the bandwidth may not be sufficient for all best-effort packets.

In order to avoid congestion in a best-effort network, the bandwidth may be over-provisioned, so that the bandwidth is sufficient for the expected peak traffic load. The resulting absence of network congestion eliminates the need for QoS mechanisms. However, an over-provisioning typically has a strong negative impact on the NoC's implementation cost and power consumption. Thus, QoS mechanisms are typically used to enable guaranteed service communication. Essentially, exclusive resource reservation mechanisms or prioritization schemes can be used to guarantee a certain level of performance to a data flow. Existing QoS schemes, using resource reservation or prioritization, are discussed in this chapter.

6.1. State of the Art

The implementation of QoS schemes essentially affects the scheduling and resource allocation mechanism of a network on chip. As a basis, different schemes are discussed in section 6.1.1 and 6.1.2. The related work will be addressed afterwards with focus on scheduling mechanisms and resource allocation strategies.

6.1.1. Scheduling Mechanisms

Providing quality of service in the NoC domain requires a predictable scheduling of parallel data transmissions. Common QoS supporting scheduling mechanisms for NoCs are: *time division multiplexing (TDM)*, *priority* and *round-robin (RR)*. These QoS scheduling mechanisms are introduced and compared now, by the use of a traffic scenario, in a router with four virtual channels.

In a **TDM** NoC[1], the same VC_n is scheduled synchronous (in the same cycle) in all routers. This strategy is also known as circuit switching, as described in section 2.4.3.1. Data, arriving in cycle n in one router, is forwarded in cycle $n+1$. The assignment of VC_n to a connection requires VC_{n+1} in the following router. Figure 6.1(a) compares the reservation phase for synchronous and asynchronous scheduling schemes in VC routers. In the example for synchronous scheduling, only *VC2* at the input port can be used currently because *VC1* would have to use *VC2* at the output port. However, *VC2* is already occupied. This limitation reduces the degrees of freedom during the reservation phase of a circuit switching connection. The advantage, resulting from the synchronized scheduling, is the low latency compared to other scheduling policies. Furthermore, synchronous scheduled circuit switching avoids buffering within the routers; which reduces their implementation cost, as already discussed in section 2.4.3.1. Figure 6.1(b)

[1] In the NoC domain, TDM is also known as synchronous TDM or *time division multiple access (TDMA)*.

compares different scheduling policies for a given utilization scenario of a VC router port shown at the lower left. Synchronous TDM must schedule all VCs, even if they are idle. This can reduce the achievable throughput of the NoC. The given bandwidth and latency guarantees are equal for all VCs.

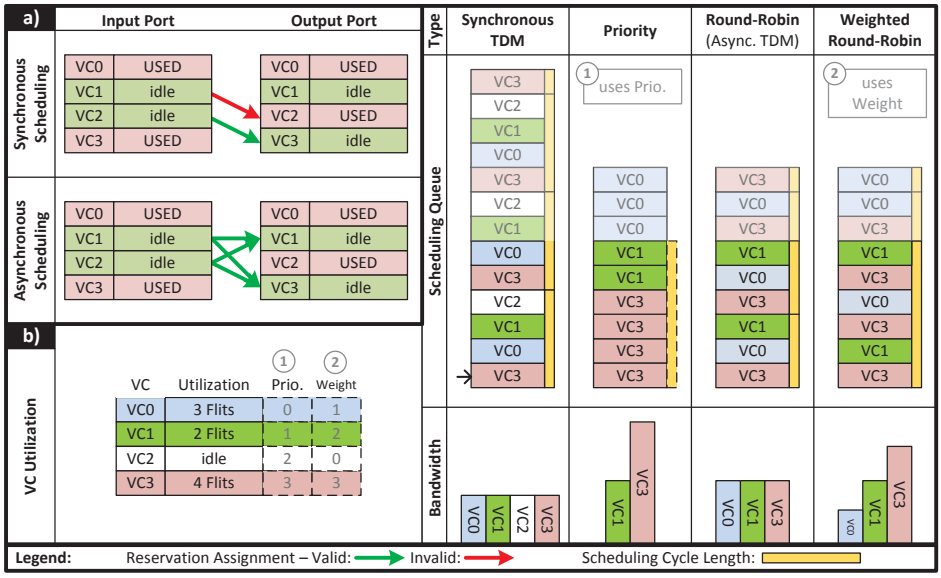


Figure 6.1.: (a) Degrees of freedom while VC assignment, (b) Scheduling policy comparison.

In contrast to time division multiplexing, used in circuit switching networks on chip, the following scheduling mechanisms are asynchronous.

Priority scheduling always selects the virtual channel with the highest priority for data transmission. VCs with lower priority are preempted. This preemption requires buffering in the NoC. Priority scheduling is an asynchronous scheduling scheme. Compared to synchronous scheduling, it has a higher degree of freedom during virtual channel assignment, as shown in figure 6.1(a). The assignment of communication flows to VCs is independent of their scheduling as long as priorities are not assigned statically to VCs. Static assignment of priorities to VCs can simplify the implementation and avoid blocking of priorities, but it reduced the flexibility of VCs assignment. Priority scheduling does not require resource reservation in advance. Throughput and delay for a given flow depend on the utilization of the used priority class and higher, due to preemption. In figure 6.1(b), *VC0* is blocked by *VC1* and *VC3* because of their higher priorities (given by ①).

Consequently, independence of different communication flows, also known as isolation [99], cannot be guaranteed.

Round-robin (RR) is a widely used scheduling policy for asynchronous scheduled NoCs. It enables a fair bandwidth distribution between the virtual channels carrying data. Idle VCs are not taken into account for scheduling. This increases the bandwidth compared to synchronous TDM, as shown in the example in figure 6.1(b). *VC2* that is idling, is not scheduled when using RR, thus the bandwidth of the other VCs increases. After reservation of communication resources, RR provides isolation of independent communication flows. Just as other asynchronous scheduled policies, buffering is required within routers. The advantage is additional degrees of freedom during the reservation phase, as shown by the example in figure 6.1(a).

Weighted round-robin (WRR) scheduling is an extension of the previous described round-robin scheduling. A weight is assigned to each flow or VC. VCs can be scheduled more frequently according to their weights. In figure 6.1(b), a weight ② is defined for each VC. The weight represents the number of schedules per scheduling cycle[2]. As shown in the example, *VC3* with a high weight of three experiences the highest bandwidth. Since it is scheduled more frequently, the transmission delay is reduced. WRR provides different hard guarantees regarding throughput and latency. It has the same drawback as other asynchronous scheduling schemes: buffers are required in the router, increasing their implementation cost, as well as the typical transmission delay.

6.1.2. Resource Allocation

The number of communication flows, with service guarantees per router port, is normally limited to the number of available virtual channels or *time slots (TSs)*. As such, exclusive resource reservation is required (aside from priority scheduling). Best-effort communication can be used as a fallback solution once this resource limitation is reached or if guarantees are not required. Therefore, most of the existing NoCs support GS and BE communication. They can be distinguished according to their resource allocation policy:

Static Allocation: The resources for guaranteed service and best-effort data transmission are separated. Dedicated resources, e.g. buffers, VCs and scheduling policies, are used for GS and BE traffic or for different service levels of a GS communication scheme. The run-time flexibility is limited.

Dynamic Allocation: The resources are assigned dynamically to GS and BE communication flows during run-time according to their communication require-

[2] A scheduling cycle is defined as the number of clock cycles until the state machine is again in the initial state.

ments. This scheme offers a higher flexibility but may have the drawback of increased implementation cost.

The classifications for scheduling and resource allocation, introduced previously, will be used for the following discussion of related work.

6.1.3. Related Work and Existing QoS Architectures

Providing quality of service guarantees for on-chip communication has been an attractive topic for research in recent times. Many of the NoCs, introduced in section 4.2, provide QoS mechanisms. The QoS schemes of these NoCs, as well as other existing mechanisms will now be discussed.

Bjerregaard and Sparsø propose the MANGO NoC, which offers GS and BE support through two separately implemented routers [25]. The GS routers forward data streams on statically programmed point-to-point connections. BE routers are responsible for connectionless packet-based traffic. The allocation of resources to GS and BE traffic is done statically. BiNoC [148] is a QoS-aware NoC that prioritizes GS packets. It uses a special inter-router arbitration scheme to increase the channel utilization for GS packets.

Millberg et al. present the TDM-based Nostrum NoC providing guaranteed service support in addition to BE traffic delivery [178]. A concept named looped containers is applied to provide QoS guarantees. The VCs that are used for the looped containers are set up semi-statically, enabling limited run-time flexibility. Therefore, it is barely applicable to general-purpose MPSoCs with varying bandwidth and latency requirements of the executed applications. Lu and Jantsch [166] also use a synchronous TDM strategy. They apply the looped containers concept and introduce logical networks to fulfill QoS guarantees. The used VC configuration algorithm is complex and requires an overview of the NoC status. Hence, it has to be implemented in software, thus limiting the scalability.

Goossens et al. present the Æthereal NoC architecture, which offers both GS and BE traffic support [92]. GS support requires resource reservation to provide worst case bandwidth guarantees with bounded latency. BE traffic exploits the NoC capacity that is not utilized by GS traffic.

Aside from other proposals, [166] [171] [262] Nostrum [178] and Æthereal [92] use synchronous TDM-based circuit switching in order to provide QoS. None of the discussed designs support different GS service levels for throughput and latency, in contrast to the QoS scheme, which will be presented in section 6.2.

Multi-path routing has been proposed in [237] to increase the bandwidth by the use of multiple parallel connections between source and destination. Each path has an equal bandwidth. Hence, multi-path routing increases the throughput

of a connection, but latency cannot be decreased. Multi-path routing has two main drawbacks: (1) in-order arrival cannot be guaranteed, (2) each additional path occupies VCs or TDM time slots. Although in-order arrival can be achieved through re-ordering [146] or more advanced mechanisms [182], the drawback of increased resource consumption remains. Moreover, multi-paths can only be used to increase bandwidth, latency cannot be adjusted or reduced.

Bolotin *et al.* present QNoC providing QoS guarantees with four *service levels (SLs)* of different priorities [29]. Communication resources (e.g. buffers) are assigned statically to the SLs. Each priority level uses a separate queue at each router port. *Globally-synchronized frames* [152] is another method which allows different SLs regarding throughput and latency guarantees. It uses a global barrier network to distribute and assign priorities to communication flows dynamically. The complex task of prioritizing packets to provide guaranteed QoS is shifted to the source node, requiring global knowledge in each node to take concurrent communication into account. *Preemptive virtual clock* [99], applied in the Kilo-NoC [98], is another scheme allowing different bandwidth guarantees per flow by utilizing priority scheduling. Compared to globally-synchronized frames it has a better bandwidth utilization and a lower area overhead due to reduced input buffer sizes. QNoC, globally-synchronized frames and preemptive virtual clocks all use priority scheduling. Traffic of higher priority preempts communication flows with lower priority. Hence, isolation between independent flows cannot be guaranteed. This can lead to negligence of lower priority traffic, as explained in section 6.1.1.

Weighted round-robin scheduling has been proven to be an efficient scheduling policy in ATM [128] and IP networks [260]. The authors of [209] compared weighted round-robin to priority-based scheduling on a theoretical basis. They proposed router service analysis models and automated delay bound calculations. The outcome of their investigation shows that WRR scheduling is fairer and more flexible compared to priority-based scheduling. An architecture concept of a WRR-based NoC and for its use by system software was not available. Consequently, a resource efficient decentralized weighted round-robin scheduling implementation for on-chip networks is presented in section 6.2. To the author's knowledge, this is the first NoC router architecture giving different hard guarantees per connection for bandwidth and latency. WRR is combined with a dynamic resource allocation scheme. Resources for connectionless BE and connection-oriented GS traffic of different SLs are assigned on demand at run-time to VCs, increasing the utilization compared to QoS NoCs with static resource allocation. An algorithm for GS connection weight assignment, based on application requirements, is also introduced.

6.2. Run-time Adaptive End-to-End Connections

End-to-end connection or end-to-end QoS are the preferred means to provide GS support for point-to-point communication. In the following sections, a concept is presented, which provides different levels of throughput and latency guarantees for point-to-point connections. The concept targets homogeneous and heterogeneous MPSoCs with varying workload and communication requirements. It was published in [HKKB13] and [HKB12][3].

Existing QoS strategies, which support different SLs, use prioritization. Delay and throughput for one service level depends on the number of packets of the same SL and higher prioritized SLs, which are transmitted at the same time. Consequently, no hard guarantees can be given. In contrast, the proposed concept enables multiple levels of hard guarantees with respect to throughput and latency. Higher SLs have greater weights with respect to scheduling, but do not preempt lower ones as priority-based approaches. Therefore, weighted round-robin scheduling is used. In addition to QoS supporting GS data transmission, the proposed scheme offers connectionless best-effort communication. BE and GS data transmission flexibly share the same resources depending on the current utilization by BE and GS communication flows. The BE packet switching NoC, introduced in chapter 5, is used as a basis. The virtual channels are used to enable efficient resource sharing and exclusive resource allocation for GS connections. GS and BE transmission flows are assigned to the VCs dynamically during run-time. Multiple service levels allow to adjust the given guarantees according to the application requirements, enabling a more efficient utilization.

6.2.1. QoS Communication Concept with Hard Guarantees

The proposed concept provides GS connections with different SLs enabling hard guarantees. A higher service level results in higher bandwidth guarantees and lower latency. Applications can select the used SL for communication, according to their requirements. Established GS connections are isolated[4] from other flows, since they exclusively reserve communication resources. Hence, hard guarantees regarding throughput and latency can be given. GS end-to-end connections from one node to another are reserved as a chain of virtual channels. All flits of a packet are injected contiguously for BE transmission, whereas a header flit is injected solely to setup a GS end-to-end connection. On its way from source to

[3] Extracts from [HKKB13] and [HKB12], which are completely written by the author of the work in hand, are used verbatim in this section without identification.

[4] The isolation refers to the hard guaranteed for throughput and latency. The used dynamic allocation scheme, described in section 6.1.2, does not enable isolation with respect to security and side channels.

destination, the header flit performs the required reservations of resources (i.e. virtual channels and buffers). This distributed reservation of resources ensures the scalability of the proposed QoS concept. One of the distributed routing schemes, introduced in section 5.1.4, is used to calculate the route hop by hop. Once the GS header flit arrives at the destination, a flow control acknowledge message is sent back to the source node. GS flow control is carried out by the network adapter, introduced in section 3.2.2.1. After the source node has received the acknowledge message, the connection can be used for data transmission. Body flits are used to transfer data on an established GS connection. A reservation can fail if the VC limit for GS communication is already reached, as described in detail in section 6.3. In this case, the source node releases the partly established connection after a timeout by injecting a tail flit. In the router, where the reservation failed, the header and tail flit are deleted[5]. The application or operating system instance requesting the reservation is informed about the failed reservation.

In order to allow different service levels, the routers' VC scheduling policy is extended by a weighted round-robin arbitration scheme, instead of a fair round-robin arbitration. In the proposed concept, each VC can be assigned to multiple time slots. The number of time slots a VC is assigned to represents its weight. This number is equal to the service level requested for the connection. The round-robin arbitration between concurrent data transmission at the output port is no longer performed between the VCs but between the time slots the VCs are assigned to. Once a VC is assigned to multiple TS, it is scheduled multiple times per scheduling cycle, as shown in figure 6.1(b). Since it is an asynchronous scheduling scheme, the assignments of the time slots for a connection within different routers are fully independent of each other. Hence, a connection can use different VCs and time slots in different routers. This enables flexible resource allocation on a per hop basis. Since no global knowledge resp. synchronization is required, the concept offers good scalability.

Figure 6.2 shows an example of a data transmissions scenario in a NoC with the proposed GS support. The routers in the example have three VCs, which are illustrated as parallel connections for clarity reasons. Due to the dynamic resource allocation scheme, no dedicated BE or GS VCs exist. Each VC can be assigned to BE or GS communication flows. This increases the flexibility during virtual channel reservation compared to statically assigned virtual channels.

The connection *C1* (red) shown in figure 6.2, is an established GS connection with an SL of 3. The assignment of the connections to the VCs and time slots are shown for each router output port. The time slots and VC used by *C1* in different routers are fully independent of each other, increasing the flexibility compared to a synchronous TDM-based scheduling. *C2* is a head flit transmission that initiates a GS connection of SL 1. Once the GS header arrives at a router, the output port

[5]GS head and tail flits can be easily dropped or deleted because they do not contain any payload.

is determined by the decentralized routing algorithm, followed by a VC being reserved at the output. According to the requested service level, the reserved output VC is assigned to the appropriate number of TS. A connectionless BE packet transmission, enabled by the router design from chapter 5, is represented by *C3*. BE head flits are treated similar to GS headers. However, they are not allowed to reserve multiple time slots. In contrast to GS communication, body and tail flits are injected directly after the head flit, forming a complete packet.

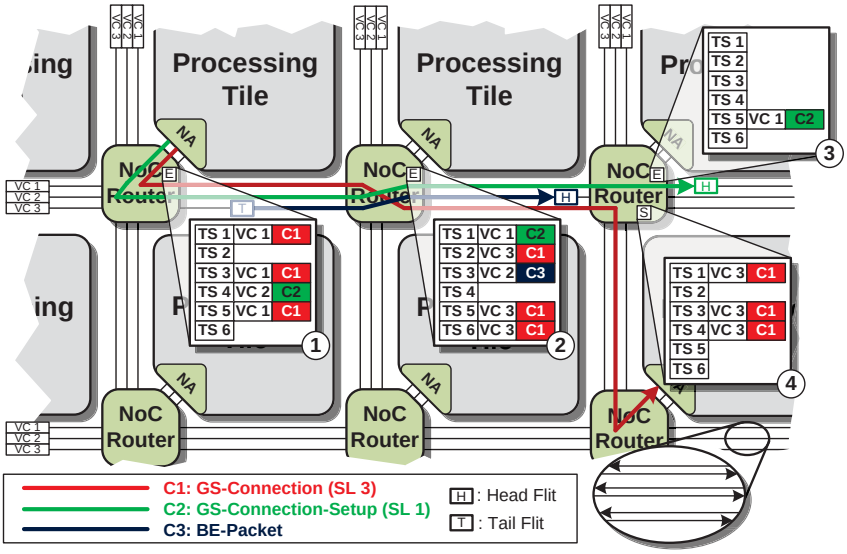


Figure 6.2.: Communication scenario illustrating the time slot allocation for guaranteed service connections and best-effort communication.

In order to avoid the blocking of BE packets by GS transmissions, a predefined number of VCs and TSs is always reserved for BE transmissions, as described in section 6.3. Thus, best-effort communication is always possible. It can be used as a fallback solution when the GS reservation fails due to resource limitations.

6.2.1.1. Latency and Throughput Guarantees

Once a GS connection has been established, latency and throughput can be guaranteed and hard boundaries can be calculated. $TS_{util,max}$ is defined as the maximum time slot utilization on the GS connection path given as:

$$TS_{util,max} = max\left\{TS_{util,i}, \forall i \in \{0,1,...,k\}\right\} \tag{6.1}$$

In equation 6.1, $TS_{util,i}$, with i representing the ports *0* to k used by an established GS connection, is the current port utilization during transmission. $TS_{util,max}$ is typically independent of the number of reserved time slots due to the used arbitration scheme. Only VCs are taken into account, which are reserved and carry data at the moment. Taking just the used time slots into account results in increased throughput compared to synchronous TDM-based approaches, where all time slots must be scheduled, independent from their reservation status or data availability. Taking figure 6.2 as an example, $TS_{util,max}$ for *C1* and *C2* is equal 5, since this is the maximum of reserved TS on their path in router port ②.

The bandwidth for a given service level $SL \in \{0, 1, ..., SL_{max}\}$ can be given as:

$$BW_{GS}(SL) = \frac{SL}{TS_{util,max}} \cdot BW_{link} \tag{6.2}$$

where BW_{link} is the bandwidth of the physical link. The transmission latency L_{GS}[6] per packet for an established GS connection of a particular SL is given as:

$$L_{GS}(SL) = (TS_{util,max} - SL + 1) \cdot (H \cdot L_{pipeline} + S_{pkt} - 1) \tag{6.3}$$

H represents the number of hops, taken by a packet from source to destination. $L_{pipeline}$ is the flit processing delay per hop, which is reflected by the number of pipeline stages taken by a body flit. S_{pkt} is the packet size, given in flits. Strict pipelining of the transmission, described in section 5.1.3, keeps the latency low.

In order to calculate the worst case bandwidth and latency that can be guaranteed, $TS_{util,max}$ must be replaced by the number of existing time slots TS_{total}, because $TS_{util,max}$ is usually not known and can change during run-time. All other parameters are known at design-time. Once the bandwidth and latency requirements of a connection are known, the required service level for a specific guarantee can be calculated at design time by using the equation 6.2 and 6.3.

Compared to the worst case latency and throughput for best-effort communication, estimated in section 5.1.6, the guarantees for GS connections are significantly lower. For a best-effort communication, taking 8 hops in a NoC with 4 VCs, a worst case bandwidth of $\frac{BW_{link}}{16}$ is estimated. In contrast, a throughput of $\frac{BW_{link}}{2}$ can be guaranteed for a GS connection using 4 out of 8 time slots. This is an increase by a factor of 8 compared to BE. The worst case latency under the same boundary conditions is 2088 cycles for BE communication, if the packet size is assumed to be 16 flits. For the GS connection, a worst case latency of 195 cycles can be guaranteed for the same transmission scenario.

[6]The transmission latency L_{GS} and the pipeline delay per hop $L_{pipeline}$ are given in clock cycles.

6.2.1.2. Time Slot Assignment

In the proposed concept, virtual channels can only be used if they can be assigned to one or several time slots. Otherwise, the VC cannot be scheduled and used for data transmission. Hence, at least one TS must be available per virtual channel. This could be achieved by having one dedicated time slot per VC. Dedicated time slots would have additional implementation overhead because the time slots would no longer be uniform. Consequently, another strategy is used to ensure the availability of one time slot per VC at any time. One time slot is available per VC as long as the following equation is fulfilled:

$$TS_{free} \geq VC_{free} \tag{6.4}$$

TS_{free} gives the number of free time slots and VC_{free} the number of unused virtual channels. Once a new reservation request arrives, it must be ensured that the number of requested time slots will not lead to a violation of equation 6.4. Therefore, the following equation is used:

$$TS_{free} \geq (VC_{free} - 1) + TS_{req} \tag{6.5}$$

Equation 6.5 takes TS_{req} into account. TS_{req} represents the number of time slot of the current reservation request. For the provided equation, it is assumed that VC_{free} still includes the VC used for the current reservation process. If equation 6.5 is violated during the reservation process, the reservation request must be rejected. This strategy ensures that all VCs can be used at any time. Rejection of reservation requests is only possible for GS connections that use service levels higher than one. By default, BE packets use only one time slot. Thus, equation 6.5 cannot be violated for best-effort reservation requests and consequently BE communication is never blocked due to unavailable time slots.

6.2.1.3. Protocol Extension

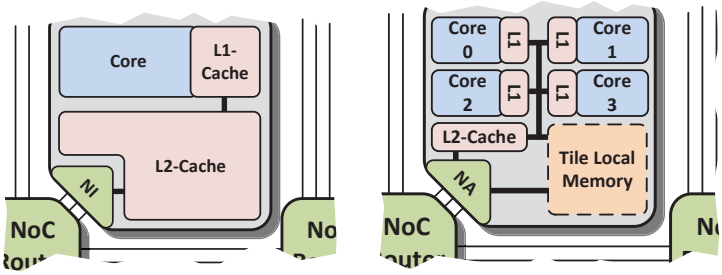
The proposed concept requires the extension of the packet protocol; presented in section 5.1.2. Figure 5.5 shows the header format used in the packet switching NoC. In contrast to BE packet headers, GS head flits contain additional information tagged with cross lines in figure 5.5. This information is used for setup of end-to-end connections. A GS header is indicated by setting the *BE/GS* bit to 1. Compared to a BE header, a GS header contains additional fields for the *service level* required for time slot reservation and the *source address* required for end-to-end flow control[7]. Because the additional GS fields are only available in

[7] End-to-end flow control is currently only supported for GS connection setup.

GS headers, the protocol overhead of BE packets is not increased by enabling guaranteed service support. Body and tail flits are equal for BE and GS traffic. However, in the case of GS communication, only body flits are used to constitute a packet or message. A tail flit is used to release a GS connection.

6.2.1.4. End-to-End Guarantees

In many-core architectures, such as the InvasIC architecture described in section 3.2, the processing cores normally use buses to connect the components located within the tile. Figure 6.3(a) shows a single-core processing tile with two levels of cache. In single-core tiles, the communication infrastructure within the tile is not shared and thus controlled exclusively by one core. Therefore, it is not required to provide guarantees for the communication infrastructure within the tile, since bandwidth and latency are known and invariable. Thus, no additional effort is necessary to provide end-to-end guarantees.



(a) Single-core tile (b) Multi-Core tile

Figure 6.3.: Tile internal communication between cores and caches.

Figure 6.3(b) shows a multi-core tile with four cores and two levels of cache, similar to the RISC core tile, presented in section 3.2.1.1. The cores are connected to a shared bus via their L1 caches. The L2 cache is also attached to this bus. The back-end of the L2 cache is connected to the network adapter, enabling access to the NoC. In such a multi-core tile, the bus must support different guarantees for bandwidth and latency to enable end-to-end QoS with the proposed scheme. The arbitration mechanism of the bus must support these guarantees. In [35], a bus arbiter for hard real-time guarantees with respect to bandwidth and latency is proposed. The use of such an arbiter is necessary in multi-core tiles to enable end-to-end guarantees[8].

[8]Since tile internal bus communication is not in the scope of this work, it is assumed and ensured for the following that there is always sufficient bandwidth available at the tile-local bus. Consequently, the bus communication is not affected by bandwidth limitations.

The network adapter also must support WRR scheduling to enable end-to-end guarantees in case of multi-core tiles or multiple connections per single-core tile. The implementation of WRR QoS support for an NA is discussed in [HKKB13].

6.2.1.5. Task Mapping

The proposed QoS concept necessitates an extension of the task mapping scheme. Existing mapping algorithms and strategies for NoC-based architectures, such as [218], [232] or [155], may be used as a basis. QoS requirements must be considered during mapping, in order to establish GS connections successfully after mapping. It must be decided whether GS connections are used for the communication. When GS connections are used, service levels must be assigned, according to the bandwidth and latency requirements.

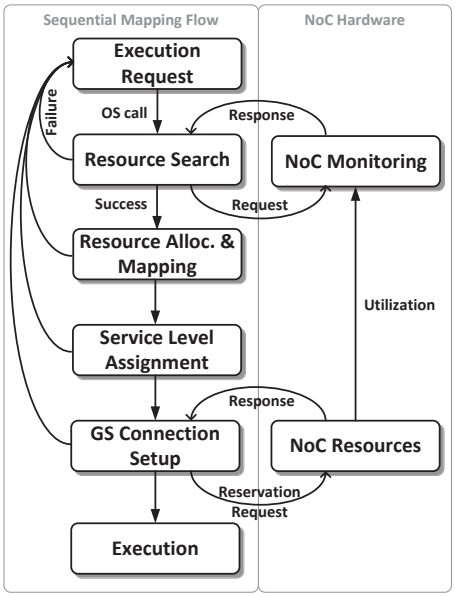


Figure 6.4.: Phases of the extended mapping flow with service level assignment and GS connection setup. Communication monitoring information is taken into account during mapping and allocation phase.

Figure 6.4 shows an extended task mapping flow used to meet the requirements of the proposed concept. An arising execution request necessitates a description of the computation and communication requirements of the application:

- *Computation Requirements:* Number and type of processing elements

- *Communication Requirements:* Communication partners, bandwidth and latency requirements

These requirements may be described by the use of a communication graph, as shown in figure 6.5(a). In the context of invasive computing, these requirements are described at the language level by the use of constraints, as further explained in section 3.1.1.1. According to the computation requirements, appropriate computation resources must be explored. The utilization of the communication infrastructure, in the surrounding of the desired computation resources, must be taken into account during the search phase. This is necessary to meet the communication requirements of the application. If the utilization is too high, it is very likely that a GS connection setup will fail later on. Thus, monitoring information must be taken into account while searching for appropriated resources, as shown in figure 6.4. Therefore, the monitoring infrastructure, introduced in section 5.5.3, can be used. It can be accessed by the operating system that is typically responsible for task mapping, resource management and reservation. Therefore, the memory mapped registers, summarized in appendix A.4, are used.

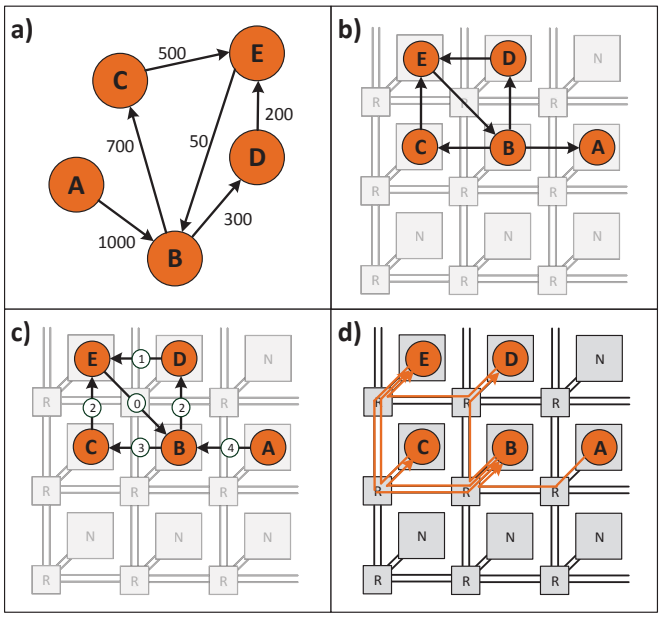


Figure 6.5.: Four steps of the QoS mapping flow: (a) application with communication requirements, (b) mapping of the tasks, (c) service level assignment and (d) establishment of GS connections between the mapped tasks.

The mapping describes the process of assigning the tasks, shown in figure 6.5(a), to processing elements. During this phase, the communication requirements must once again be taken into account to minimize NoC utilization, power consumption and transmission latency by optimizing the distance between communicating nodes. The outcome of the allocation and mapping phase is shown in figure 6.5(b). Each node is now assigned to a specific processing element. Pairs of nodes, with high communication requirements, are mapped next to each other to minimize the distance for communication and thus the communication resource utilization and power consumption. Energy and performance aware mapping flows for regular NoC-based architectures have been proposed in [115] and [261]. These mapping algorithms may be used for the proposed mapping flow. Thus, they are not discussed in detail here.

After the mapping phase, execution could be initiated in ordinary packet switching NoCs. The proposed QoS concept requires two additional steps before execution can take place. The third step, shown in figure 6.4, is named *service level assignment*[9]. During this phase, service levels are assigned to each edge or connection of the task graph. The SLs are assigned according to the bandwidth or latency requirements. The outcome of this phase is shown in figure 6.5(c). In the provided example, the SLs are assigned according to the bandwidth requirements of the task graph; presented in figure 6.5(a), whereas *SL 0* represents BE communication. It is used in the example for communication flows with very low bandwidth requirements[10]. SLs from 1 to 4 represent GS connections using 1 to 4 time slots. In section 6.2.2.2 an algorithm for service level assignment is presented.

Once the service levels for all communication flows are defined, the GS connections may be established accordingly. From the software perspective, this is done by writing to memory mapped registers of the NAs in order to trigger the reservation process of GS connections. The respective source and destination addresses for GS connection setup can be obtained from task mapping. The service level for each connection results from the previous SL assignment phase. In figure 6.4, it is indicated that the reservation of GS connections affects the NoC utilization and thus the monitoring information; taking into account for future application mappings. Figure 6.5(d) shows the architecture after mapping and GS connections establishment. Once the setup of the GS connections is successfully completed, the execution can be initiated.

However, each of the steps of the mapping flow, presented in figure 6.4, can fail for various reasons. The *resource search* phase can fail due to the unavailability of computation or communication resources. *Resource allocation & mapping* fails if no valid mapping is found. The *service level assignment* phase may also fail;

[9] The phase of service level assignment could also take place before task mapping, because the phases are independent of each other.

[10] If real-time or safety requirements exist, only GS must be used.

described in detail in section 6.2.2.2. *GS connection setup* fails if one of the desired GS connections cannot be successfully established. Such a failure will lead to an abortion of the execution request. In managing such a situation, malleable applications may be used and could continue processing with less resources. If an application does not have any resources, execution must be delayed. Taking the NoC monitoring information into account increases the chance of successful mapping and execution.

More details on the usage of the proposed QoS scheme in the context of invasive computing, including the presented mapping flow, are provided in [HZZ+14] and [HZW+14].

6.2.2. Hardware and Software Implementation

The proposed concept for QoS is applicable to all asynchronous scheduled VC-based routers and not limited to the one presented in chapter 5. It is independent of the topology and dimension of the NoC and is applicable to exotic networks, such as optical networks [279, 280] or 3D networks [273]. All distrusted routing schemes can be used. In the following sections, the necessary extensions for the best-effort packet switching router design, introduced in chapter 5, are discussed. A more general and more detailed described is provided in [HKKB13].

6.2.2.1. Router

The block diagram of the router design with QoS support is shown in figure 6.6. A detailed explanation of the general operation of the router is provided in section 5.1.5. The components of the QoS router, which differ from the basic router design, are the *output reservation table* and the *transmission control* unit, as shown in figure 6.6. A detailed implementation of these units is given in figure 6.7 for a router with 4 virtual channels and 6 time slots. However, the router HDL template and the simulation framework supports all values for the number of VCs and TSs, as described in detail in appendix A.5 and A.6.

There is one *output reservation table (ORT)* per output port, as shown in figure 6.7 on the left side. It contains the reservation table that assigns a pair, consisting of input port and input VC, to an output VC of the respective output port. This part is equal to the basic router design from section 5.1.5. However, the *ORT* unit contains a second reservation table with one entry per time slot. Each time slot contains the ID of the output VC assigned to it. Reservation requests arrive at the *reservation control FSM*. The requested number of TSs to be reserved is part of the reservation request. This FSM validates the number of available time slots and grants the reservation once the parameters, provided in the following paragraph,

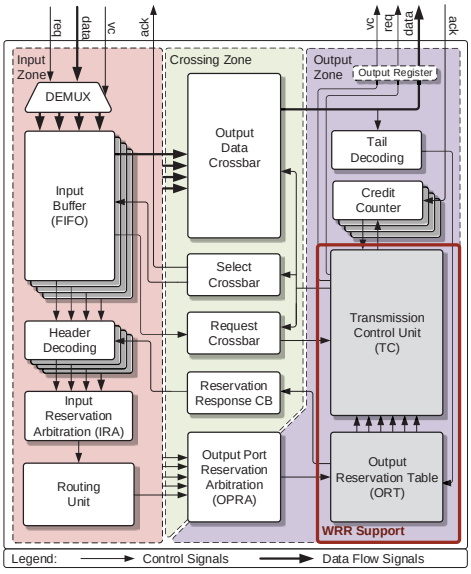


Figure 6.6.: Basic router design with modifications for WRR support.

are fulfilled. Reservations of multiple TSs are done sequentially. The number of cycles, required for reservation, corresponds to the requested service level. The FSM limits the number of GS connections, ensuring the availability of VCs for BE communication by using the policies; described in detail in section 6.3. The reservation process for BE traffic is similar to GS connections requesting a single time slot.

The request signals of the reserved time slots and the assigned output VCs are forwarded to the *transmission control* unit, shown in figure 6.7 on the right. The request signals are used for a fair round-robin arbitration between the time slots. The weighting of weighted round-robin scheduling is carried out by assigning one VC and its request signal to multiple time slots. Consequently, one VC can be selected by the round-robin arbiter multiple times per scheduling cycle. For each request signal it is verified whether data and credits are available for transmission, as described in section 5.1.5. This is indicated by the *valid request* signal.

All input and output signals of the *ORT* and the *TC* unit are equal to the basic router design from section 5.1. They are the only units that require modification, in order to enable the proposed adaptive QoS scheme.

GS Reservation Management A set of policies must be adhered in order to grant a reservation request within the output reservation table. These policies are

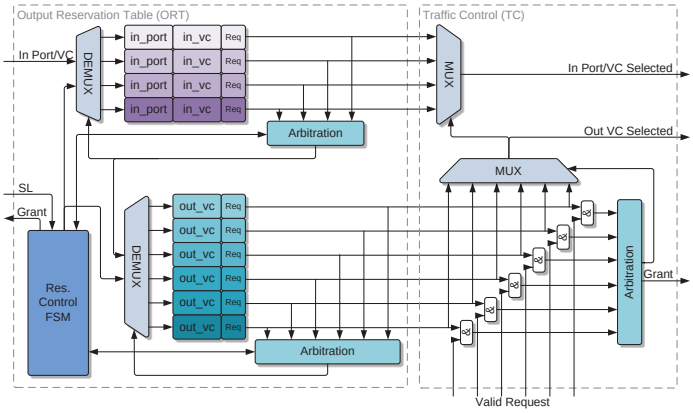


Figure 6.7.: Extended output reservation table and transmission control unit for weighted round-robin scheduling support.

implemented inside the *reservation control FSM*, shown in figure 6.7. The policies are validated in the same cycle the reservation process is requested; independent of the type (BE or GS). This allows to grant or reject a reservation request within one cycle. Two counters inside the reservation control FSM are used to log the VC utilization. One counter contains the number of free VCs (VC_{free}). The second counter contains the number of VCs currently utilized by GSs connections (VC_{GS}). As long as VCs are available, BE reservation requests are granted:

$$VC_{free} > 0 \tag{6.6}$$

For GS connection requests, two additional rules are required. The following policy limits the number of virtual channels used for GS connections:

$$(VC_{total} - VC_{GS} > VC_{BE,excl}) \tag{6.7}$$

VC_{total} gives the total number of VCs and $VC_{BE,excl}$ the number of VCs exclusively reserved for BE packets. For $VC_{BE,excl} > 0$ this policy ensures that VCs are always available for BE transmissions. Hence, BE communication can never be blocked by established GS connections. Another counter, within the reservation control FSM, monitors the number of free time slots (TS_{free}). This counter and the VC_{free} counter are used to evaluate equation 6.5, provided in section 6.2.1.2. GS connections are acknowledged if equations 6.5, 6.6 and 6.7 are fulfilled.

The set of policies ensure that the discussed TS and VC limits are observed for BE and GS traffic. Thus enabling the use of uniform time slots and virtual channels.

In section 6.3, a concept for run-time reconfiguration of these policies is presented. It enables to adjust the number of VCs to be used for GS and BE communication at run-time.

6.2.2.2. Service Level Assignment

In section 6.2.1.5, a QoS-aware task mapping flow was introduced. This flow contains steps for service level assignment and GS connection reservation. In order to calculate an optimal service level assignment for a given task graph, the bandwidth and delay requirements of all communication flows must be taken into account. According to the requirements of each communication flow, the optimal service levels can be calculated by using equation 6.2 and 6.3.

Algorithm A.1, in appendix A.2, calculates the optimal service levels for all connections using the specified communication requirements of a distributed application. These requirements could either be specified by communication constraints, as described in detail in section 3.1.1.1, or in the form of task graphs, as shown in figure 5.13. Algorithm A.1 takes equation 6.2 into account to select the service level according to the bandwidth requirements of each connection. In order to take delay requirements into account, only minimal modifications are required.

Detailed investigations exhibited that the number of used virtual channels and time slots is typically highest at the local ports of the routers, if neighboring mapping is used. This can also be seen in the example shown in figure 6.5(d). All connections coming from different neighboring routers meet at the local port. The algorithm, further described in appendix A.2, ensures that equation 6.4 and 6.5 are met for each local port of the nodes used by the application to be mapped.

The algorithm does not consider concurrent communication of other applications. Hence, the reservation of the calculated GS connections can fail subsequently. In order to reduce the probability that GS connection setup fails, the load of the communication infrastructure must be taken into consideration during resource allocation. This is done by taking the monitoring information into account during resource allocation, as described in section 6.2.1.5.

The final task graph, generated by algorithm A.1, includes the type (BE or GS) for each communication flow as well as the service level assignments for all GS flows. This task graph can be used by the operating system to set up the GS connections accordingly in order to optimize QoS on behalf of the application.

6.2.3. Evaluation

In the following, the performance benefit, the compliance of the given guarantees, the number of parallel connections and the implementation cost are investigated. Therefore, simulation results are presented in section 6.2.3.1. The FPGA prototype, described in appendix A.1, is used for additional performance analysis. Subsequently, ASIC synthesis results are presented in section 6.2.3.3.

6.2.3.1. Simulation

The proposed QoS support was implemented in the SystemC simulation framework and introduced in section 5.2. It is used for the following investigations. A meshed NoC of 10x10 routers with XY routing is instantiated[11]. The routers are configured to have 4 virtual channels. The number of time slots (TS_{total}) is set to 8. The SL or weight respectively was varied between 1 and 4. BE traffic, used for some scenarios, is generated as uniform random traffic. [12]

Round-robin scheduling is dominant in existing packet switching NoCs. The NoC presented in [129] applies round-robin scheduling and offers GS communication. It is used as a reference. In the proposed scheme, GS connections with SL 1 have the same performance as in [129], when no connections with higher SLs are established. Hence, GS connections with SL 1 are also named *Ref*.

Analysis of single GS connections The delay and throughput of GS connections with different SLs are analyzed first. BE traffic, with different injection rates, is used to emulate concurrent load on the NoC. Therefore, single short distant (2 hops) and long distant connections (19 hops) were established.

Figure 6.8(a) shows the achievable throughput per connection for a short distant connection with a hop distance[13] of 2 nodes. If BE injection rate is low, a GS throughput equal to 1 could be achieved for all SLs. A rising BE injection rate reduces the GS throughput because the links are shared between BE and GS communication flows. Once the saturation point of the NoC is reached, the difference in throughput for the various service levels is highest. Compared

[11] For the following investigations, each router is configured to have a transmission delay of 2 cycles and a reservation delay of 1 cycle.

[12] The size of BE and GS packets is chosen randomly between 2 and 8 flits with an average packet size of 5 flits.

[13] The hop distance represents the number of routers to be taken by the packets for transmission.

to SL 1, which is equal to common RR, the throughput of SL 4 is increased by 123 %[14].

Figure 6.8(b) summarizes the measured throughput of a long connection taking 19 hops. Due to the longer transmission distance, the probability that arbitration decisions must be taken is higher. This influences the throughput and leads to an earlier throughput saturation at a lower BE injection rate compared to figure 6.8(a).

Figure 6.8(c) shows the delay analysis for short connections. For low BE traffic injection rates the GS packets with an average size of 5 flits were transmitted with a delay of 8 cycles for all SLs. Once the NoC utilization rises, due to higher BE injection rates, the delay differences between the SLs grows. At the saturation point of the BE traffic (0.4 $Flits/Cycle/Node$), the delay of the different SLs is significant. SL 4 has an average packet delay of only 10.2 cycles. This is a latency reduction of 44 % compared to SL 1 with an average packet delay of 18.2 cycles.

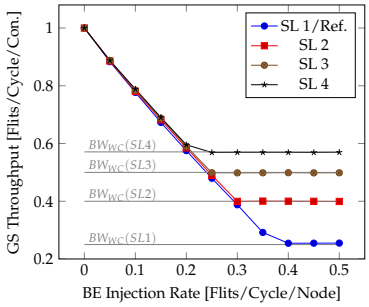
Figure 6.8(d) shows the mean packet latency of the investigated SLs for a connection, making again 19 hops. The latency for low injection rates is again close to the theoretical minimum of 42 cycles for all service level. At the saturation point of the BE traffic, the delay for long distance GS transmissions is highest. The delay difference between the different service levels reaches its maximum at this point. The latency reduction of SL 4 compared to SL 1 is 25 %.

Maximum number of connections Since GS connections require exclusive reservation of communication resources in the proposed scheme, their number is limited. The maximal number of connections is determined by randomly establishing GS connections of different SLs and distance. Figure 6.8(e) shows the maximum number of connections for different communication distances and service levels, with $VC_{BE,excl}$ being set at 2. The results for SL 1, 2 and 3 are nearly identical, because there are enough time slots available to occupy 2 VCs with GS connections. SL 4 is affected by the limited number of TSs per link.

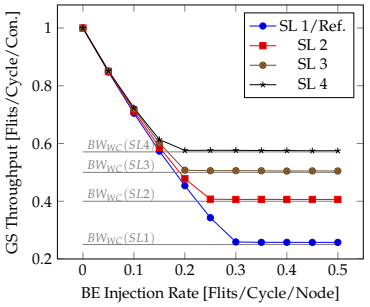
Figure 6.8(f) shows the maximum number of connections once $VC_{BE,excl}$, the number of VCs reserved for BE communication, is reduced from 2 to 1. SL 1 and 2 profit from the additional available GS VCs. For short distances, up to 65 % more connections can be set up due to the additional available VC. SL 3 and 4 cannot profit from additional VCs because the number of connections is limited by the available TSs.

Idle GS connection analysis Due to the dynamic WRR scheduling, idle GS connections do not occupy bandwidth. However, GS connections utilize router

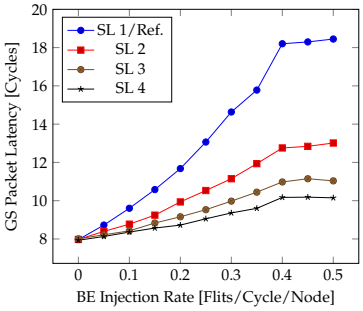
[14] SL 1 uses $\frac{1}{4}$ of the available bandwidth because 4 VCs are used (3 additional by BE), whereas each VC is assigned to 1 TS. In contrast SL 4 uses $\frac{4}{7}$ of the total bandwidth. 4 TSs are assigned to the GS connection and 3 are used by the other 3 VCs carrying BE communication.



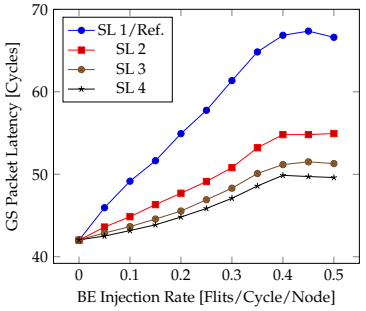
(a) Throughput of short distant con.



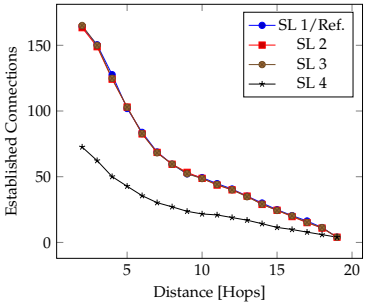
(b) Throughput of long distant con.



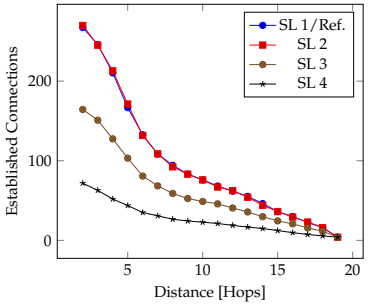
(c) Average latency of short distant con.



(d) Average latency of long distant con.



(e) Number of parallel connections per distance with $VC_{BE,excl} = 2$



(f) Number of parallel connections per distance with $VC_{BE,excl} = 1$

Figure 6.8.: GS throughput for a (a) short distance and (b) long distance connection. Average packet delay for a (c) short distance and (d) long distance connection. Number of parallel connections related to distance and SL for (e) 2 VCs and (f) 3 VCs usable for GS connections.

resources (e.g. VCs and buffers) even if they are idle. Thus, the influence with respect to BE throughput and latency, in case of idle GS connections, was analyzed in [HKKB13] and [HKB12]. The throughput and latency in case of idle GS connections was compared against a reference scenario. The reference scenario represents measurements of a purely BE packet switching NoC, where no virtual channels are occupied by GS connections. The results show a low impact of idle GS connections. The throughput reduction is at maximum 7.2 %, if a high number of 50 connections of SL 4 and a distance of 4 hops are established.

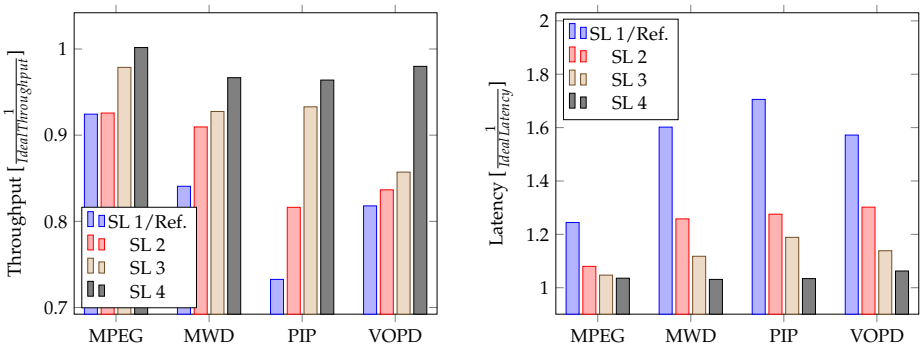
Due to the fact that reserved VCs cannot be used for BE transmission, the number of BE packets in the NoC is reduced when GS connections are established. This leads to a throughput reduction, but also reduces the average packet delay for BE traffic. Under heavy load, a delay reduction of 15.2 % could be obtained, if 50 connections with SL 4 and a distance of 4 hops are established. The same effect was investigated in section 5.4.1.2 – a reduced number of available VCs led to a delay degradation, but also reduced the achievable throughput.

Video processing applications Finally, the MPEG4, MWD, PIP and VOPD applications, introduced in section 5.2.3.3, are used for delay and throughput investigation. The communication graphs of the applications are mapped according to the flow presented in section 6.2.1.5. The SL assignment for the connections was performed according to the required bandwidths using the assignment strategy, presented in section 6.2.2.2. Connections with very low bandwidth requirements use BE communication if the number of available virtual channels or time slots is exceeded. Throughput and latency are investigated under heavy load, generated by random GS and BE communication. The given results are relative to an ideal reference scenario without additional traffic on the NoC.

Figure 6.9(a) shows the throughput of the most communication intensive connection, since they limit the performance of such streaming-like application. For PIP, the throughput can be improved by 31 %, compared to the reference with standard round-robin scheduled GS communication.

Figure 6.9(b) gives the average latency. Delay reductions from 13 % to 40 % could be achieved compared to GS communication in a NoC using non-weighted round-robin scheduling. Once SLs up to 4 are utilized, the throughput and delay for all applications is close to the ideal scenario.

The previous results clearly show the benefit of adjustable GS connections by the use of weighted round-robin scheduling. In contrast to existing QoS approaches, the throughput and latency can be adapted at run-time by the use of different service levels. More detailed investigations can be found in [HKKB13]. Among other things, the service level assignment algorithm is investigated in detail.



(a) Application throughput (b) Application latency

Figure 6.9.: (a) Throughput and (b) latency of video processing applications relative to ideal communication.

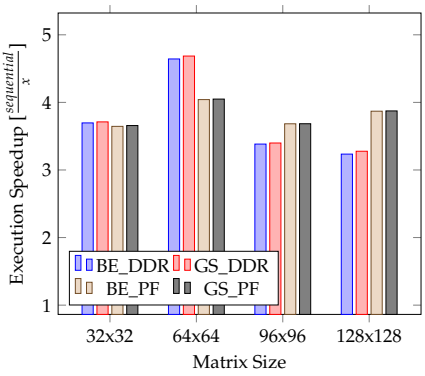
6.2.3.2. FPGA Prototype

In addition to the simulation results, the FPGA-prototype of the InvasIC architecture, described in appendix A.1, is used to investigate the gain of GS communication. The architecture consists of four single-core tiles. A parallel version of an integer matrix multiplication was built in C++ to investigate the impact of GS communication on execution time, NoC-utilization and NoC power consumption. The results are provided in figure 6.10. Four different variants of the matrix multiplication were investigated: two versions use *BE* communication, the other two versions use *GS* end-to-end connections. Each communication variant is investigated as *DDR*, where the source matrices are located in the main memory and *PF*, where the required parts of the source matrices are prefetched from the DDR to the tile local memory. The *BE* variants are used as a reference.

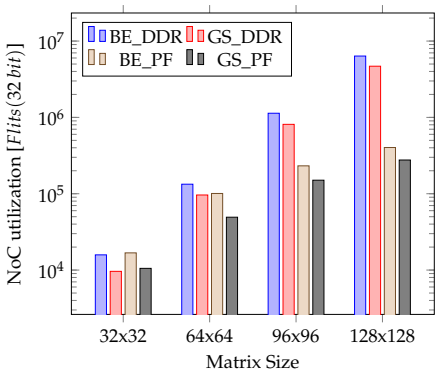
However, it was not possible to investigate the impact of different service levels due to the small size of the prototype. The architecture is too small to generate concurrent load, which would be necessary to see the impact of different SLs. Without additional load the performance of GS connections with different service levels do not differ, as shown in the previous section.

Figure 6.10(a) compares the acceleration of the different variants; relative to a single core variant. However, due to the unavailability of additional background traffic, the impact of GS communication on performance is very small. A performance improvement of 1.3 % is achieved for the *DDR* variant and a matrix size of *128x128*, if GS communication is used instead of best-effort. Prefetching of data to the tile local memory has a much greater impact on performance. At maximum,

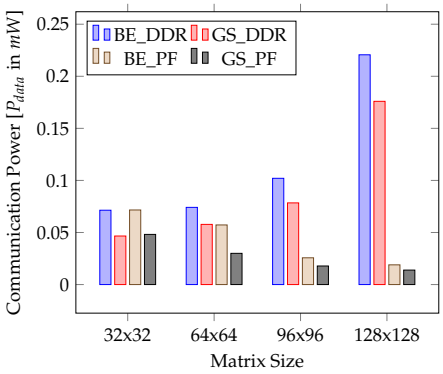
26 % performance improves are experienced by the use of prefetching for a matrix of *128x128* elements.



(a) Application speedup



(b) NoC utilization



(c) NoC power consumption

Figure 6.10.: Two variants of a parallel matrix multiplication executed with and without GS support on a 4 tile architecture prototype[15].

Figure 6.10(b) shows the NoC utilization caused by executing the four variants of the matrix multiplication. In order to obtain these numbers, the link utilization monitors from section 5.5.3 have been used. As expected, the amount of communication increases with the matrix size. For larger matrix sizes, the amount of flits can be reduced by up to 26 % if GS communication is used instead of BE.

[15]The results show that prefetching has no benefit with respect to execution time for small matrix sizes due to the fact that all data fit into the L2 cache.

This significant reduction of the NoC utilization results from the reduced protocol overhead of GS communication[16].

Finally, the power consumption that is directly related to data transmissions is analyzed for the ASIC implementation of the NoC, as described in section 5.3.2.1. For accurate power estimation, the monitoring information from the FPGA prototype was used for netlist simulation to derive accurate toggle rates for power analysis. Figure 6.10(c) summarizes the results. The benefit of prefetching, with respect to power consumption, increases with the size of the matrix. If prefetching is not utilized, GS communication can assist in reducing the power consumption for communication by up to 20 % depending on the matrix size.

6.2.3.3. Implementation Costs

The proposed QoS concept with WRR scheduling was implemented in the HDL template of the router design, introduced in section 5.3. It was synthesized using a 45 nm ASIC standard cell library, as described in section 5.3.2.1. Since the HDL template is widely parameterizable, it is possible to investigate the impact of various design parameters on area, timing and power consumption.

For the following design space exploration, the number of VCs and time slots is varied. The investigated router has buffers with 8 slots. The link width is 128 bit in order to provide a reasonable bandwidth for large many-core architectures.

Figure 6.11(a) shows the area consumption of a single 5-port router for various VC and TS configuration. The number of time slots must be greater or equal to the number of VCs, because at least one TS is required to use a VC. Doubling the number of TSs from 4 to 8 increases the area consumption only by 3.1 % for a router with 4 VCs. However, doubling the number of VCs and TSs from 4 to 8 increases the area consumption by 99 %. Compared to additional VCs, which would be occupied in multi-path routing [237], the area consumption of additional time slots is very low. The small area overhead is an essential benefit of the proposed QoS concept, in comparison to existing approaches.

Figure 6.11(b) shows the impact of additional VCs and TSs with respect to the achievable clock frequency. The router configurations are the same as previously mentioned. Additional TSs and VCs reduce the clock frequency since the transmission control unit, which is part of the critical path, becomes more complex. This is not specific to the proposed router design. The complexity of the scheduling unit limits the clock frequency in many heavily pipelined packet switching NoCs

[16] A usage of the payload fields of head and body flits (see section 5.1.2) can reduce the protocol overhead for best-effort communication. However, this is currently not supported by the invasive network adapter.

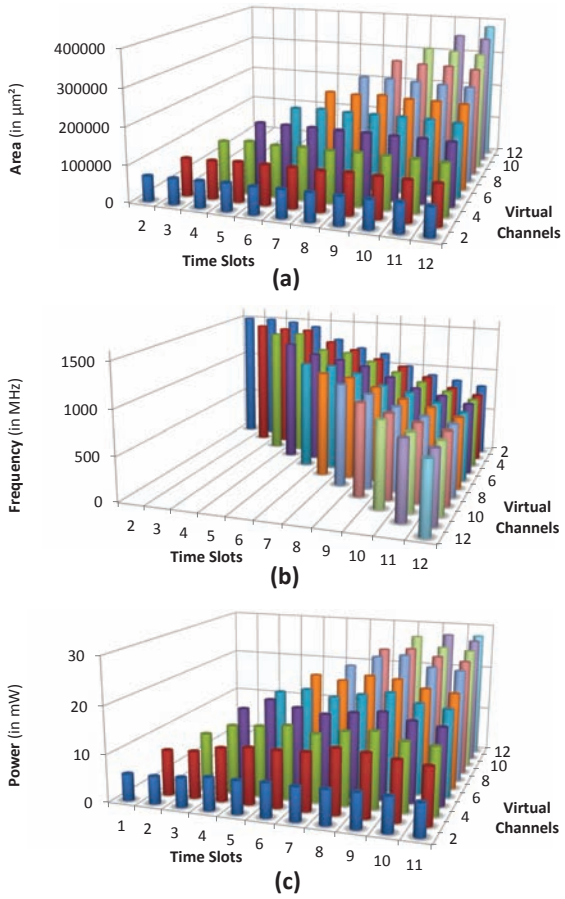


Figure 6.11.: Implementation overhead of the QoS concept: (a) area consumption, (b) achieved clock frequency and (c) power consumption for various VCs and TSs settings.

using higher numbers of VCs. The impact on clock frequency can be reduced by adding pipeline registers to the transmission control unit.

In figure 6.11(c), the impact of additional time slots and virtual channels on the power consumption is shown. The sums of static and dynamic power consumption of a single 5-port router are shown. The scenario used for dynamic power estimation includes best-effort communication as well as GS connections crossing the router. The results show that doubling the number of time slots from 4 to 8 increases the power consumption of the router by 18.7 %. In comparison, doubling

the number of VCs[17] from 4 to 8 increases the power consumption of the router by 95.5 %.

The presented synthesis results show that increasing the number of time slots has a much lower impact on area, frequency and power consumption than increasing the number of virtual channels.

A router with 4 VCs and 8 TSs, as used for simulation, requires 139093.9 μm^2 chip area and reaches a post synthesis clock frequency of 1150 MHz. It consumes 14.3 mW power on typical load conditions with BE and GS traffic.

6.3. Adaptive QoS Policy Management

In the previous section, a QoS concept was introduced that enables dynamic sharing of communication resources between GS and BE communication flows. Static policies are implemented in the *reservation control FSM*, ensuring the availability of VCs for BE communication, as described in section 6.2.2.1. This ensures that BE traffic is never completely blocked by GS communication, which would be the case if all VCs are occupied by GS flows. In order to avoid this scenario, the number of GS connections per router port must be limited as follows:

$$VC_{total} > VC_{GS,max} \geq VC_{GS,cur} \tag{6.8}$$

VC_{total} gives the number of virtual channels per router port. The maximum number of VCs per router port, which can be occupied by GS connections, is represented by $VC_{GS,max}$. $VC_{GS,cur}$ provides the number of virtual channels currently used by GS connection. By placing a limit on the number of VCs, which are reservable for GS connections, blocking of BE traffic by GS connections can be avoided, because a specific number of VCs is always available for BE traffic. The number of VCs used by BE transmissions $VC_{BE,cur}$ must not be restricted. However, a similar policy can be given for best-effort traffic:

$$VC_{total} \geq VC_{BE,max} \geq VC_{BE,cur} \tag{6.9}$$

The limit $VC_{BE,max}$ is normally equal to VC_{total}. Thus, all VCs can be occupied by BE traffic.

The resource allocation policies for GS and BE traffic are defined statically in existing router designs. This also applies for the concept, presented in section 6.2, where the policies for VC assignment are defined at design-time. These static policies cannot be adapted to the application requirements. This in turn can

[17] Doubling the number of VCs requires to also double the number of TS to have one TS per VC.

lead to degradation of performance, as shown later. In the following paragraphs, a concept for run-time configuration of the VC allocation policy for GS and BE communication is presented. $VC_{GS,max}$ and $VC_{BE,max}$ can be adapted for each router port according to the current communication requirements. This concept can be used to pre-allocate resources for an application. The concept also allows to control global communication[18] by restricting the use of certain regions. A detailed description of this concept is provided in [HZW+12][19]. In the following, a summary of this work is provided.

Figure 6.12 shows a 10x10 mesh NoC. The adaptive QoS policy management enables reconfiguration of virtual channel allocation policies for individual routers, in a distributed manner in order to ensure scalability. Such policies are referred to as *resource management policy (RMP)*. In the example, three different scenarios for RMP reconfiguration are provided. Example (a) in figure 6.12 illustrates the reconfiguration of the policy by modifying $VC_{BE,max}$ within the respective NoC region. This benefits the GS communication within this region, as the resource utilization by BE traffic is limited. The regions can be defined during run-time through distributed policy adaptation according to the application requirements.

Example (b) in figure 6.12 shows a region where the number of GS connections crossing the border is reduced. This is done by modifying the $VC_{GS,max}$ policy at the borders of the region. However, the number of GS connections within the region may still be higher because the policies for the routers within are not modified. This increases the performance for GS communication within the region and improves the probability of setting up connections successfully, if an application is mapped into the region. The results, presented in section 6.3.3, show how an application, which is mapped into such a region, can benefits from region-based policy adaptation.

The task mappings of an application may lead to a region which is not square. Example (c) in figure 6.12 shows the configuration of a non-square region. Compared to example (a), where the policy for BE traffic is adapted for all routers of the region, the modification of $VC_{BE,max}$ policy at the regions' borders allows to improve BE performance within the region. The strategy in example (c) can be used to enhance the performance of the application by improving the BE communication[20]. Adaptive routing schemes may be necessary in case of non-square regions to avoid crossing of borders for intra-region communication.

The investigations, summarized in section 6.3.3, show that the applications within the region benefit from such policy reconfiguration. However, the policies may harm the communication crossing the regions. Adaptive routing, introduced in

[18] More details about global and local communication are provided in section 6.4.1.

[19] Parts of the publication [HZW+12] are used verbatim in this section without further identification.

[20] BE might be necessary if the number of communication partners within a distributed application exceeds $VC_{GS,max}$. In such a case GS communication cannot be used for all communication flows.

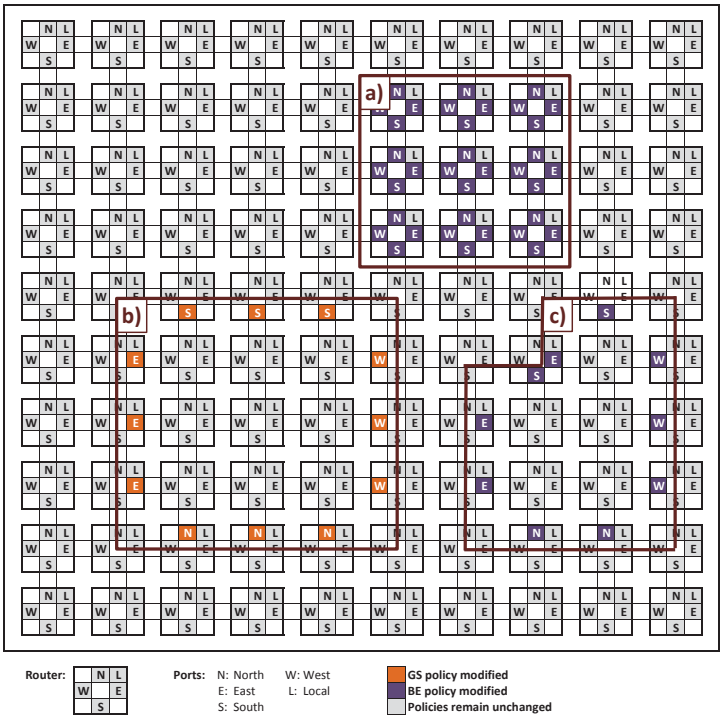


Figure 6.12.: Different policy configurations for routers in NoC regions: (a) BE-policy reconfiguration within a region, (b) GS-policy and (c) BE-policy reconfiguration at regions' borders [HZW+12].

section 5.1.4.2, can be used to increase the performance outside the regions. It allows certain BE packets or GS connections to bypass a region.

6.3.1. Run-Time Mapping and Policy Configuration

The concept for run-time policy configuration targets a distributed software management, such as the agent system, introduced in section 3.3.2.1. The agent system divides the architecture into disjoint regions. Each of these regions is managed by an instance of the agent system. This concept of non-overlapping regions is also reflected by the proposed policy configuration scheme.

Figure 6.13 illustrates how an application mapping flow works in conjunction with the proposed concept. Figure 6.13(a) shows a part of an architecture where two applications are already mapped. To optimize the communication for the

respective application, the RMPs have been reconfigured to fence the applications in their respective regions. Figure 6.13(b) shows an application graph which is now requested to be mapped to the architecture. The operating system typically performs a decentralized search to find a suitable region for application execution, as explained in section 6.2.1.5. Once a region is found, the policy is reconfigured to fence the region as shown in figure 6.13(c). Figure 6.13(d) illustrates the mapping of the application to the selected region.

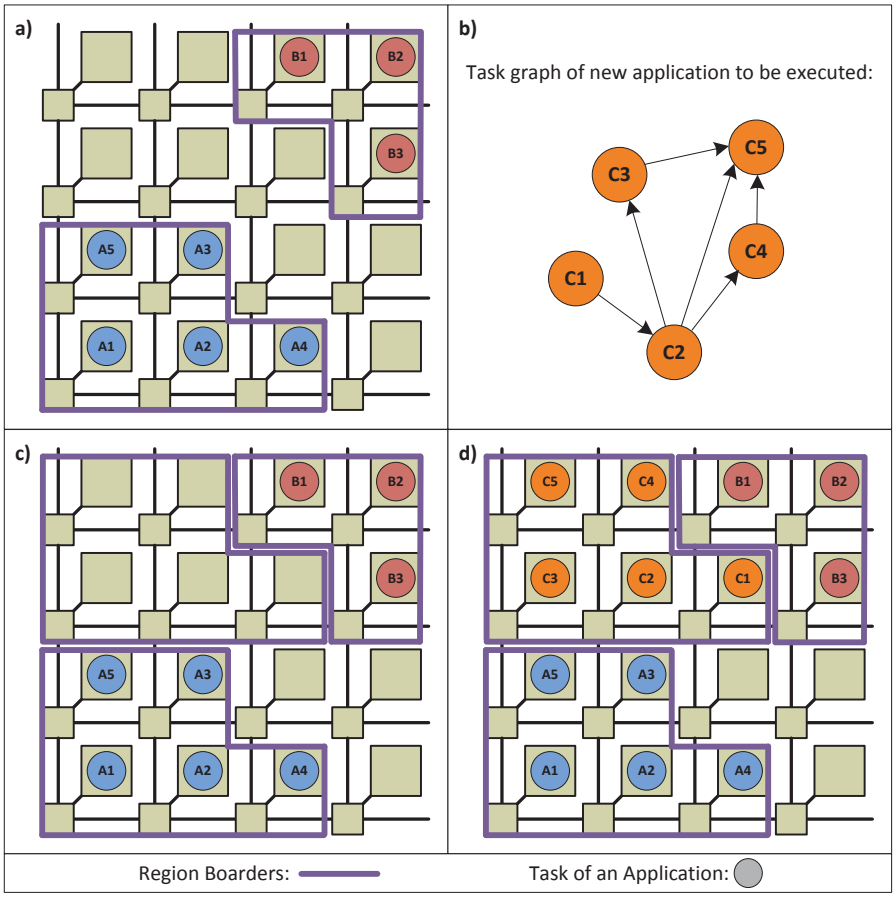


Figure 6.13.: Policy reconfiguration for application mapping: (a) two applications are running, (b) new application needs to be mapped, (c) the region for new application is prepared and (d) application is mapped into new region and executed.

The GS or BE virtual channel allocation policy can be configured for each router port. This policy reconfiguration is triggered by the OS as mentioned previously. In [HZW+12], three different mechanisms for policy reconfiguration are discussed. Interestingly enough, memory mapped registers are the most efficient way to enable run-time configuration. Consequently, memory mapped registers are used, as defined in table A.6. They can be accessed directly from each node of the architecture, enabling a configuration of an entire region by a single node or core. Thus, a single instance of an operating system is able to configure a complete region.

6.3.2. Hardware Implementation

The *reservation control FSM* manages the reservation process within the *output reservation table*, as described in section 6.2.2.1. It is extended to support dynamic policies for GS and BE traffic. Therefore, the reservation control FSM is extended by a so-called *resource management unit (RMU)*. The block diagram of the resource management unit is shown in figure 6.14. It supports separate policies for GS and BE traffic. The RMU is responsible for ensuring that the resource management policies are not violated for the respective router port.

Once a reservation request arrives at the output reservation table, it is processed by the reservation control FSM. During this reservation phase, the RMU is requested to acknowledge the reservation. Therefore, the RMU contains separate interfaces for GS and BE reservation requests. A reservation request is acknowledged in the same cycle, if the existing policy is not violated. Once an RMU does not acknowledge the reservation request, it is rejected. The counters of the RMU contain the numbers of VCs occupied by GS and BE traffic, respectively. Whenever a virtual channel is reserved or released, the corresponding signals are set and the counters inside the RMU are incremented or decremented accordingly.

The RMU contains a separate interface, which is used to reconfigure the GS and BE policies during run-time. The signal, named *policy level*, provides the rules in terms of $VC_{GS,max}$ and $VC_{BE,max}$ parameters. The configuration is performed by writing to memory mapped registers, as previously described.

6.3.3. Evaluation

A detailed investigation of the proposed concept for policy reconfiguration is provided in [HZW+12]. A summary of the most important results is now provided.

The SystemC simulation framework, introduced in section 5.2, is used for evaluation. A 10x10 NoC with mesh topology is instantiated. The MPEG4, MWD, PIP and VOPD applications, introduced in section 5.2.3.3, are used for evaluation.

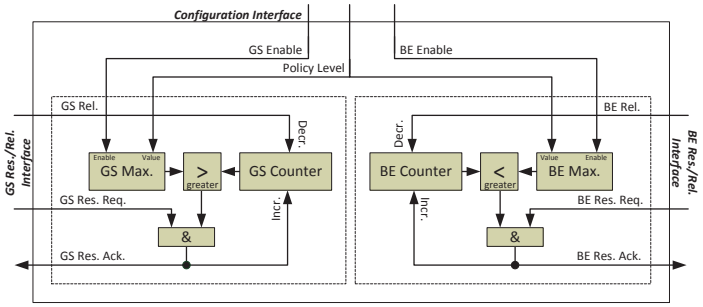


Figure 6.14.: Block diagram of resource management unit with separate resource management policies for GS and BE traffic.

Each application is mapped separately to a region, located in the center of the mesh. Uniform random BE communication and randomly established GS connections are used to mimic global communication[21] through the region of the application. The VC assignment policies at the borders of the region are modified according to the proposed concept. Subsequently, throughput and latency of the application are measured. For comparison an *ideal* and a *baseline* scenario are used. The *ideal* scenario represents a situation without any concurrent communication on the NoC. In contrast, the *baseline* provides measurements with concurrent communication but without application of the proposed concept. It can be seen as a reference with respect to state of the art NoCs.

In the initial scenario, BE traffic is used as additional load in order to impact the investigated applications. The BE policy ($VC_{BE,max}$), at the borders of the region is decreased incrementally, as shown in figure 6.12(c). By reducing the $VC_{BE,max}$ to 1, the overall throughput for MPEG4 could be improved by 12 % compared to the *baseline*. For PIP and VOPD the policy reconfiguration can improve the throughput near the *ideal* scenario. Altering the BE resource management policy has a much larger influence on packet delay; compared to throughput. PIP is the application that profits the most from the modified RMP. Its delay can be reduced by up to 53 %; compared to *baseline* scenario.

In the second scenario, random GS connections are setup between the tiles outside the region. These connections may cross the region, where the investigated application is mapped. The policy for GS communication, at the regions' borders, is modified by reconfiguring $VC_{GS,max}$, as shown in figure 6.12(b). Compared to the *baseline*, MPEG4 achieves a throughput increase of 17 %, if the policy for $VC_{GS,max}$ is set to 1 at the borders. For PIP, a delay reduction of 36 % is experienced

[21] Global communication typically results from communication to chip-external memories or communication between operating system instances.

by reconfiguring the $VC_{GS,max}$ from 3 to 1 at the regions' borders. VOPD, MPEG and MWD packet delay is reduced by 25 %, 23 % and 13 %, respectively.

As previously mentioned, reducing the $VC_{GS,max}$ at the regions' borders reduces the number of connections crossing this region. Hence, the proposed concept can increase the probability of successfully establishing a GS connections within an existing region. Limiting the number of GS connections through the region improves the rate of established GS connections by up to 21 % for PIP. A $VC_{GS,max}$ limit of 1 at the regions' borders leads to a GS setup success rate of 98 % for all investigated applications. Thus, the proposed concept can be used to ensure successful mapping of applications with QoS requirements.

The implementation overhead of the proposed concept is investigated, in detail, in [HZW+12] for a *Xilinx Virtex-5 LX 330* FPGA. Instantiating one resource management unit per router port increases the LUT utilization of a router by 2.5 %. Clock frequency is unaffected if memory mapped policy reconfiguration is used[22].

6

6.4. Virtual Networks

Networks on chip enable the creation of large scalable on-chip architectures. In such architectures, with hundreds of cores, centralized management of resources is inefficient due to higher communication overhead. Distributed resource management is proposed to reduce the management and communication overhead [140]. The architecture is logically divided into regions and the computational resources of each region are administrated by an OS instance which is responsible for the respective region. In addition, each instance is responsible for the application using the region. Such a decentralized management scheme is also used in invasive computing, as described in section 3.3.2. The computational resources are divided into disjoint regions.

Managing the underlying communication infrastructure according to the application requirements is challenging. The partitioning of communication resources into regions is complicated, compared to partitioning of computational resources. The reason is the existence of local and global communication[23]. Typical local communication is communication between tasks, which belong to the same application and are thus mapped neighboring one another. However, global communication may on the one hand be necessary to access specialized components, such as off-chip memories or specialized tiles. On the other hand, global

[22] In [HZW+12], local reconfiguration is used. This necessitates additional I/O pins and decreases the clock frequency by 7 % compared to the baseline router.

[23] Communication of an application that is internal to a region is referred to as "local", while communication crossing regional borders is named "global".

communication is necessary for OS-internal communication. As such, the communication resources could be shared and used concurrently by both local and global communication or partitioned between the two communication types. If the resources are shared and used concurrently, the application's local communication latency and throughput depends on the amount of global communication. On the other hand, an exclusive partitioning of communication resources into regions would prohibit global communication. Consequently, a more flexible concept is necessary to fulfill the requirements of global and local communication for large-scale many-core architecture.

As an extension of the policy management scheme, presented in section 6.3, a distributed communication resource management strategy for packet switched networks on chip with virtual channels, published in [HZW+13], is explained in the following[24]. This strategy enables the establishment of so-called *virtual networks (VNs)* at run-time in arbitrary regions of the architecture. Virtual channels can be assigned dynamically to a virtual network in a region. The application executed within the region of the virtual network can exclusively use the assigned VCs. Whereas the remaining virtual channels are available for global communication.

The following example shall motivate the necessity of virtual networks: Some distributed applications can be organized in a way that the number of communication partners per node is low. Guaranteed service connections, introduced in section 6.2, can then be used to decouple local communication of the application from global communication. Other applications, such as the distributed MPEG application, shown in figure 6.15, have nodes with high fan-out and fan-in. These nodes (*SDRAM* and *SRAM2*) have too many communication partners to use GS for all connections[25]. Consequently, the use of dedicated point-to-point connections is not always applicable to enable QoS guarantees.

The concept of virtual networks enables run-time adaptation of the communication infrastructure, according to the requirements of the current application. This enables QoS and can be used to improve communication performance of any application. In contrast to GS point-to-point connections, where the number of communication partners per node is limited, VNs can always be used. A virtual network can be dynamically established for a region. A part of the physical bandwidth of the communication infrastructure is assigned exclusively to a VN, according to the requirements of the application. The bandwidth, which is not allocated to virtual networks, is available for global communication. A VN can be seen as an independent, exclusively used communication infrastructure from

[24] Parts of [HZW+13] (own work) are reused in this section without further identification.

[25] Assuming that four virtual channels exist and one is reserved for global best-effort communication, only three VCs remain and this only allows to establish three connections per node at maximum.

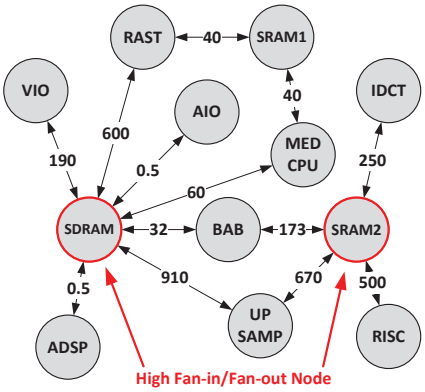


Figure 6.15.: MPEG video processing application with high fan-out and fan-in nodes [22].

the perspective of the software using it. Thus, virtual networks are a step toward visualization of MPSoC architectures using NoC-based interconnects.

6.4.1. Concept of Run-time Adaptive Virtual Networks

In a packet switching network with virtual channels, just as the NoC introduced in section 5.1, each packet is assigned to a virtual channel on a per hop basis. Thus, a packet can use an arbitrary VC at each router. This flexible VC assignment maximizes the communication resource utilization and is a noticeable advantage of packet switched networks. In order to enable virtual networks, the virtual channel assignment strategy must be controlled by the use of policies[26]. A separate policy is defined per virtual network i:

$$VC_{util}(i) \leq VC_{max}(i) \leq VC_{total} \tag{6.10}$$

$VC_{util}(i)$ gives the number of virtual channels, currently utilized by VN i. It is compared to $VC_{max}(i)$ each time a virtual channel assignment is required in the router. $VC_{max}(i)$ is the maximum number of VCs usable by VN i. $VC_{max}(i)$ is a policy and is configured at all routers of a region when establishing a virtual network. VC_{total} gives the total number of virtual channels in the network.

When all virtual channels are assigned to the same virtual network, as shown in figure 6.16(a), and all packets use this network, the network behaves identically

[26]The assigment policies presented in this section are independent of the policy management, introduced in section 6.3. It is possible to apply both concepts in common.

to a conventional network. This configuration may also be used as a default setting of the network after reset. Figure 6.16(b) shows the simplest virtual network with a static assignment. Two of the virtual channels are assigned to *VN0* and the other two to *VN1*. In order to assign each packet to a virtual network, it contains the ID of the VN. With respect to the example presented in figure 6.16(b), the policies must be set to $VC_{max}(0) = 2$ and $VC_{max}(1) = 2$ according to equation 6.10. Figure 6.16(c) shows a dynamic virtual network. *VN1* has two exclusively used virtual channels but can also use the other two VCs, thus two of the VCs are dynamically shared between *VN0* and *VN1*. In order to configure the assignment policies for the example provided, $VC_{max}(0)$ is set to 2 and $VC_{max}(1)$ to 4. Partial dynamic virtual networks are established in the example provided in figure 6.16(d). In partial dynamic VNs, some of the VCs are exclusive to each VN and some are shared.

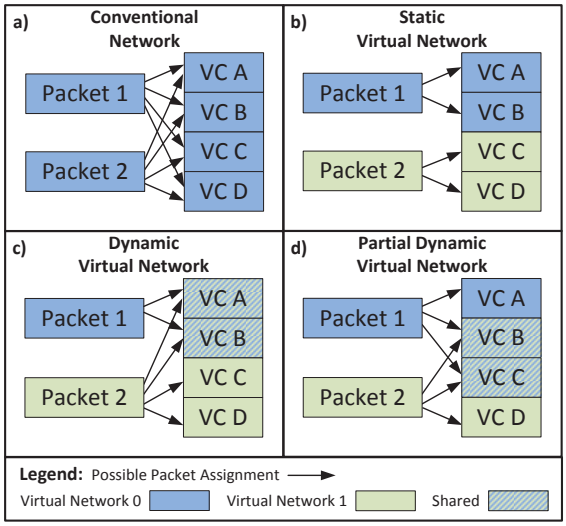


Figure 6.16.: Assignment of packets to virtual channels for (a) conventional NoCs as well as (b) static and (c,d) dynamic virtual networks.

A part of an architecture using virtual networks is presented in figure 6.17(a). Most of the cores are assigned to one of the three regions *A*, *B* and *C*. Each region is used by a separate application and has a separate virtual network. The configured virtual networks are not required to be rectangular in shape, as shown by region *C*. The VNs are disjoint for this example. In contrast, figure 6.17(b) shows an example for overlapping virtual networks. Two cores are shared between region *A* and *B*.

Figure 6.17(c) shows the virtual channel assignment for one row of routers from the example provided by figure 6.17(a). It can be seen that VCs of each router

are assigned to two VNs, at most. The number of virtual networks, supported per router, is limited by the number of hardware-modules managing the VC assignment, as described later, in detail. Figure 6.17(d) shows the VC assignment for the scenario provided by figure 6.17(b). In addition to the virtual networks, used by the application and the one used for global communication, another virtual network is established for external memory communication. The router in the center must manage four different VNs. Consequently, the router hardware must be support the management of four virtual networks in parallel.

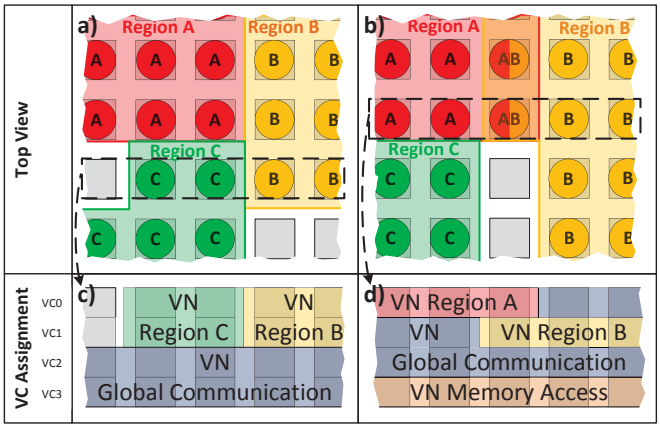


Figure 6.17.: (a) Three applications A, B and C are mapped to different separated regions. (b) Regions A and B are overlapping. The VC assignment for a row of routers is shown by (c) and (d) respectively.

When a virtual network is established, hard guarantees for the overall throughput within the sub-network can be given:

$$BW_{VN}(i) = \frac{VC_{excl}(i)}{VC_{total}} \cdot BW_{total} \tag{6.11}$$

BW_{total} is the total bandwidth of the network, VC_{total} gives the total number of virtual channels and $VC_{excl}(i)$ is the number of VCs assigned exclusively to VN i. $VC_{excl}(i)$ can differ from $VC_{max}(i)$ if VCs are shared between virtual networks, as shown by figure 6.16(c) and (d). $VC_{excl}(i)$ can be calculated as follows:

$$VC_{excl}(i) = min\left\{VC_{max}(i), max\left\{VC_{total} - \sum_{j=0, j\neq i}^{N} VC_{max}(j), 0\right\}\right\} \tag{6.12}$$

N is the number of virtual networks supported by the architecture. $VC_{excl}(i)$ depends on the policies ($VC_{max}(i)$) of all VNs.

6.4.1.1. Software Management and Configuration

Software support is required to establish and use virtual networks according to the requirements of the application. A request to execute a new application arises from an ongoing execution at some node of the architecture and is forwarded to an operating system instance. The OS searches for appropriate resources to execute the application. Once sufficient resources are found, they are reserved for the application. Figure 6.18 shows how an application mapping flow works when using the proposed virtual network concept.

Figure 6.18(a) shows a part of an architecture in which two applications are mapped. These applications use the same virtual network or no VN respectively. A new application, represented by the task graph in figure 6.18(b), must be execution. The new application C is assumed to be communication-bound[27]; hence, a virtual network is requested. A virtual network setup request is passed to the OS instance, managing the communication resources of the region; selected for task mapping. Subsequently, the responsible OS instance reconfigures the virtual channel management policies at the routers in the region, as highlighted in figure 6.18(c). Therefore, a reconfiguration packet is sent to each router, which must be configured. Memory mapped registers are the most efficient way to enable run-time configuration. They can be accessed directly from each node of the architecture, enabling a decentralized configuration of the entire region in order to ensure scalability. After the virtual network is successfully established, the application is mapped to the designated region of the VN, as shown in figure 6.18(d). During execution of the application, the OS in collaboration with the network adapter is responsible for assigning the correct *VN ID* to the packets using the virtual network, as described in section 6.4.2.1.

6.4.2. Implementation

Virtual network support requires additional components inside the network routers and an extension of the network layer protocol of the NoC. From the protocol perspective, a *VN ID* is required to assign each packet to the respective VCs of the virtual network. It is processed at run-time in each router during virtual channel assignment. The corresponding implementation is discussed now.

[27] The performance of a communication-bound application is limited by the performance of the communication infrastructure and memory subsystem. In contrast, the performance of a computation bound application is typically limited by the computation power of the cores.

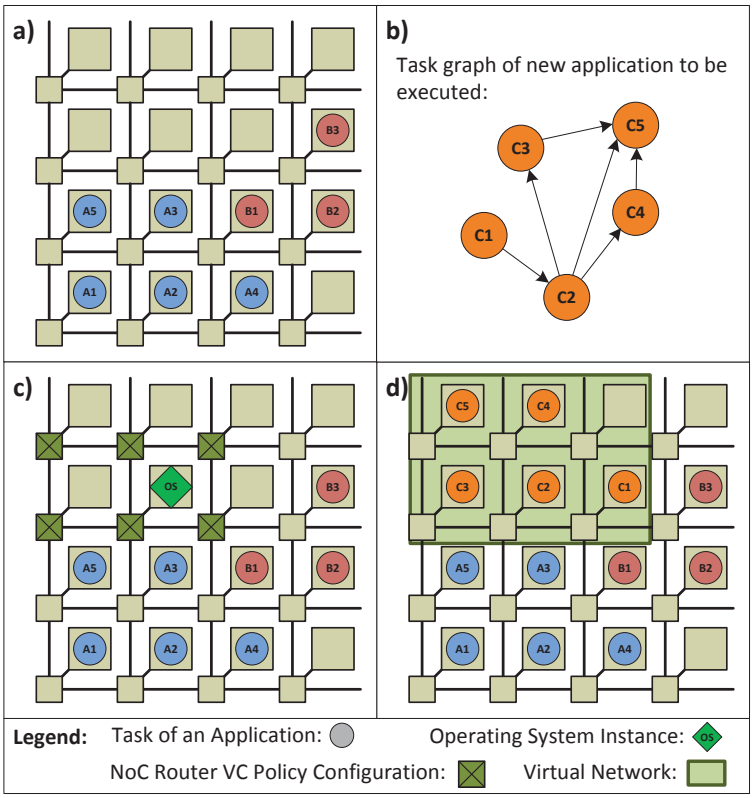


Figure 6.18.: Establishment of a virtual network while task mapping: (a) two applications are running, (b) new application requests for resources, (c) virtual network for new application is established and (d) application is mapped into the virtual network region and executed.

6.4.2.1. Flit Format Extension

The VN concept requires the assignment of each packet to a virtual network. Thus, the index of the virtual network, the so-called *VN ID*, must be contained in each packet. The VN ID is required during the reservation phase and is contained in the head flit. Figure 6.19 shows the fields of a header supporting VNs.

The VN ID, is the additional field used as an index for VC assignment and must be set by the network adapter. The VN ID is the only protocol extension required to support VNs, compared to the basic protocol introduced in section 5.1.2. Thus, the protocol overhead for VN support is very low. For an architecture supporting two VNs per router, only one bit is required for the VN ID.

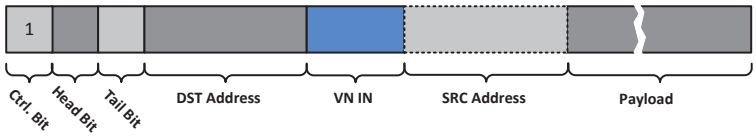


Figure 6.19.: Head flit supporting virtual networks by *VN ID* field.

6.4.2.2. Virtual Network Management Unit

The *virtual network management unit (VNMU)* controls the VC assignment policies, provided by equation 6.10. It is involved in the reservation and release process of the virtual channels. Reservation and release of virtual channels is performed by the reservation logic embedded in the reservation table, described in detail in section 5.1. One VNMU is required per router output port, since the virtual channel management is independent for each router port.

The structure of the virtual network management unit is shown in figure 6.20. It contains a separate control unit for each virtual network. The number of supported virtual networks is defined by the number of *virtual network control units (VNCUs)*. This number must be defined at design-time. The choice of the number of virtual network control units and thus the number of supported virtual networks depends on the requirements of the architecture. The number of VNCUs limits the number of virtual networks, overlapping in the same router, but not the overall number of VNs. The example shown in figure 6.17(d), would require 4 virtual network control units per router port, since the router in the middle manages 4 different VNs. The number of VNs, used by a single application, is independent of the application size. One VN is sufficient for a large application to decouple its internal communication from other communication.

Manner of Functioning Once a head flit arrives at a router input port, the routing unit calculates the output port. Subsequently, the reservation of an output VC is performed by adding an entry to the reservation table of the output, calculated by the routing unit. During this reservation phase, the VNMU is requested to acknowledge the reservation. Therefore, the VN ID, shown in figure 6.19, is forwarded to the VNMU. The responsible VNCU of the virtual network management unit is selected according to the VN ID. The VC counter within each control unit contains the number of VCs currently occupied by the respective virtual network. The maximum number of VCs for each VN is defined by the value stored in the *VC max.* register, representing $VC_{max}(i)$ in equation 6.10. A reservation request is acknowledged in the same cycle, if the defined policy is not violated. The VC counter is incremented each time a reservation request is acknowledged. Release requests, which are raised when a tail flit is transmitted,

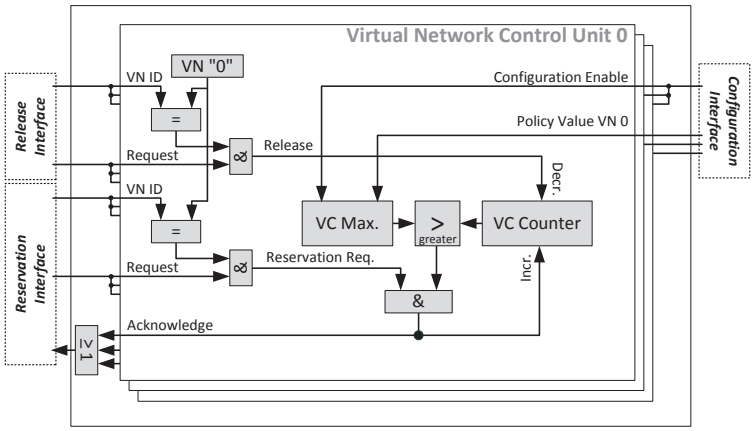


Figure 6.20.: Block diagram of the virtual network management unit.

6

are also forwarded to the virtual network management unit. Consequently, the VC counter of the VN to which the tail flit corresponds is decremented. In addition to the interface for request and release, the RMU contains a configuration interface. The configuration interface is used to store a new value in the VC max. register when the configuration port is enabled. This interface is connected to the memory mapped configuration registers, used for virtual network configuration, as described in section 6.4.1.1.

6.4.3. Evaluation

The evaluation section consists of two parts. The performance evaluation for different traffic scenarios, using the simulation framework introduced in section 5.2, is discussed in section 6.4.3.1. Subsequently, section 6.4.3.2 provides ASIC synthesis results, analyzing the implementation costs and power consumption for virtual network support.

6.4.3.1. Simulation

An 8x8 mesh NoC with XY routing is used for the following investigations. The routers are configured to have 4 virtual channels and a buffer depth of 4 flits. The software mechanisms described in section 6.4.1.1 are implemented in C++ and integrated into the simulation framework.

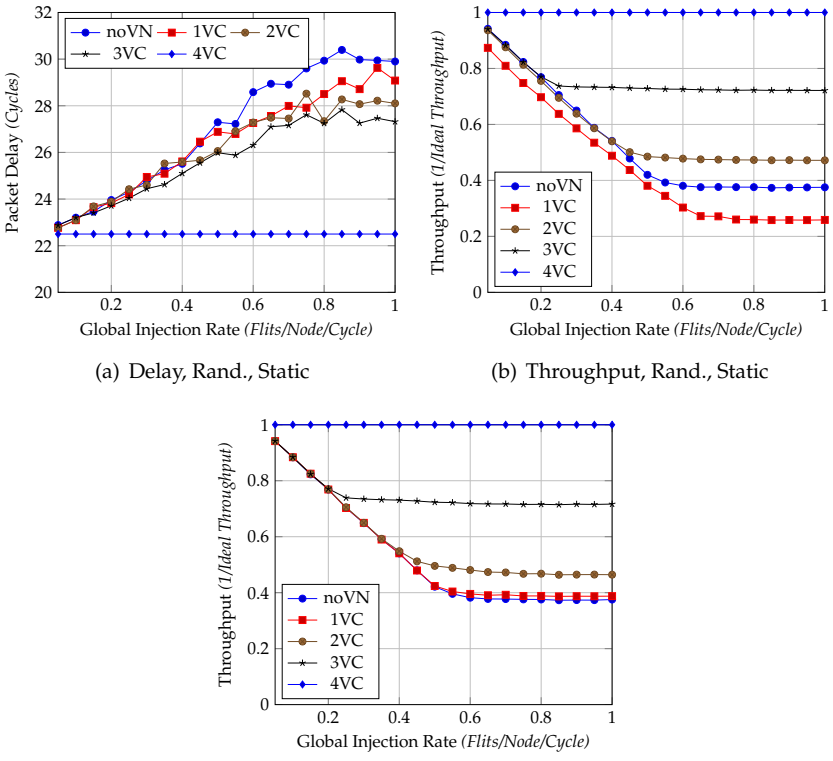
Different configurations of virtual networks are investigated in the following paragraphs. The reference is a network without VN-support, as shown in figure 6.16(a). For this configuration, all communication flows share the VCs. It is referred to as *noVN* hereafter. The *noVN* configuration is compared to configurations where virtual networks with different bandwidths (1-4 VCs) are established. Static virtual networks and dynamic virtual networks, as shown in figure 6.16(b) and 6.16(c) respectively, are used for investigation.

Different communication scenarios are analyzed. The investigated communication flows represent applications that are mapped to a predefined region. In addition to these flows, synthetic traffic is used as global communication crossing the regions used by the applications. Uniform random traffic (*Rand.*) and transpose traffic (*Transp.*), introduced in section 5.2.3.1, are used for global communication. Due to the characteristics of the synthetic traffic, its impact on region-internal communication is maximized when the evaluated regions are centered. Thus, all investigated flows and applications are mapped to the center of the NoC. If a virtual network exists, it is configured according to the shape of the application. The VN is used exclusively by the investigated communication flows.

Single Connection In the first scenario, a single data transmission across a region of 4x4 nodes, located at the center of the architecture, is analyzed. The injection rate of the synthetic global communication varies between rates of 0 and 1 $Flits/Cycle/Node$. In figure 6.21(a), the delay of a data transmission is investigated. As shown by the results, the latency can be reduced if a virtual network is established. Assigning all 4 VCs to the VN of the region, completely blocks global communication through the region. This leads to a low packet delay that is independent from global communication. Virtual networks may decrease delay by up to 25 %.

Figure 6.21(b) shows the achievable throughput of a single data transmission. Assigning only 1 VC to the virtual network results in reduced performance compared to an architecture without VNs. The reason is the limited bandwidth of a single VC. Assigning two or more VCs to the VN increases the throughput compared to the reference architecture, named *noVN*. When 3 VCs are assigned to the VN, the throughput is nearly doubled; compared to the reference. Dynamic VNs are permitted to use the VCs of other virtual networks in addition to the exclusively assigned VCs. This improves the throughput for low numbers of VCs, as shown in figure 6.21(c).

Synthetic Region Traffic For the second test case, uniform random traffic is used in a virtual network with a size of 4x4 nodes. The region is again located at the center of the architecture. In contrast to the previous scenario, the injection rate for global communication was set to 0.35 $Flits/Cycle/Node$. The injection



(a) Delay, Rand., Static

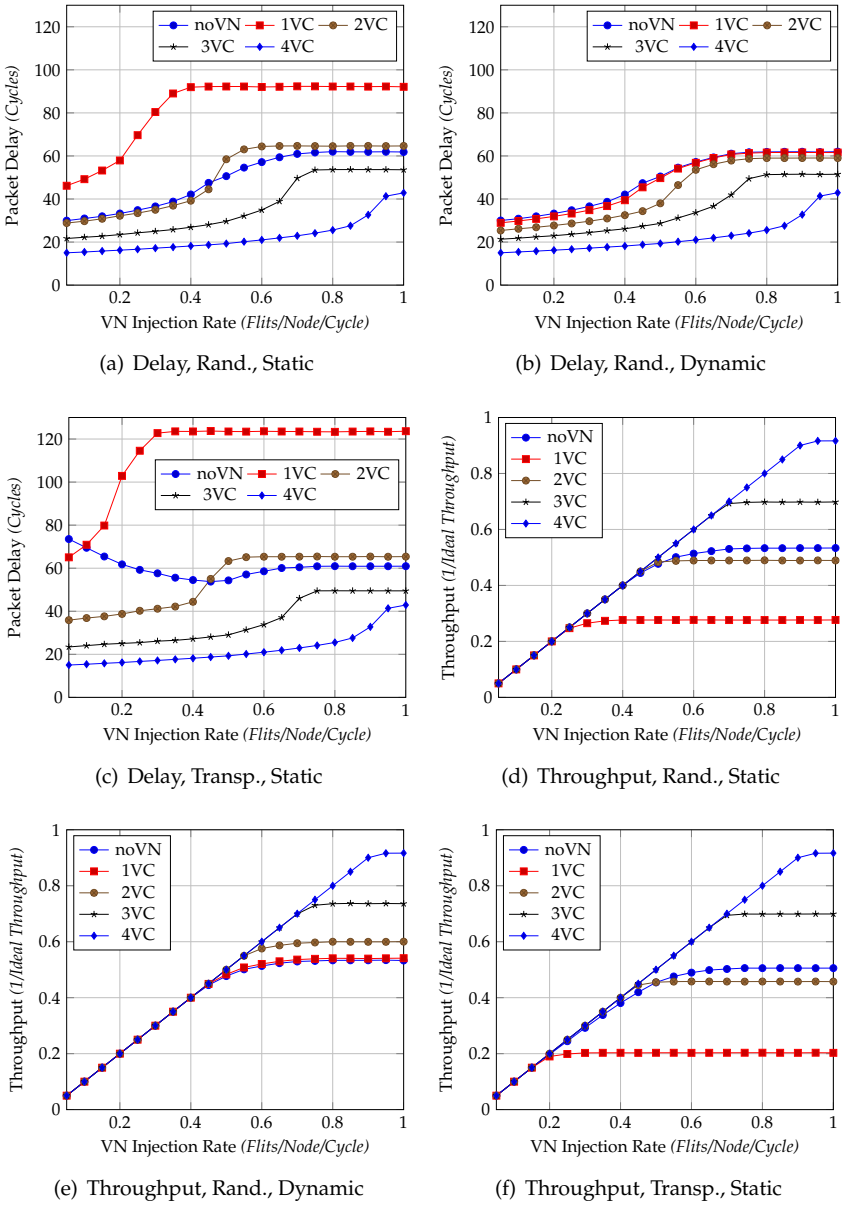
(b) Throughput, Rand., Static

(c) Throughput, Rand., Dynamic

Figure 6.21.: Delay and throughput of single data transmissions in VN regions with global communication.

rate of the nodes within the region is varied between 0 and 1 *Flits/Cycle/Node*. Using a static virtual network, at least 2 VCs must be assigned to the random VN communication to obtain a performance similar to the reference architecture, as presented in figure 6.22(a). A dynamic virtual network reduces the delay compared to the static VN, as shown in figure 6.22(b). Delay can be improved by more than 40 % for a traffic injection rate of 0.5 and a VN with 3 VCs, compared to the reference architecture. For global communication in transpose manner, shown in figure 6.22(c), at least 3 VCs must be allocated to the VN to improve the performance for high injection rates, compared to the reference architecture.

Figure 6.22(d) shows the throughput analyzes for a static VN under random traffic. 3 VCs must be assigned to the virtual network to improve throughput compared to the reference. Using a dynamic VN improves the performance, when 2 or more



(a) Delay, Rand., Static

(b) Delay, Rand., Dynamic

(c) Delay, Transp., Static

(d) Throughput, Rand., Static

(e) Throughput, Rand., Dynamic

(f) Throughput, Transp., Static

Figure 6.22.: Delay and throughput of uniform random traffic within the region for different VN configuration.

VCs are assigned to it, as shown in figure 6.22(e). A dynamic VN using 3 VCs can improve throughput of random communication by up to 72 %, compared to the reference architecture. Throughput measurements for transpose traffic used as global traffic are similar, as shown in figure 6.22(f).

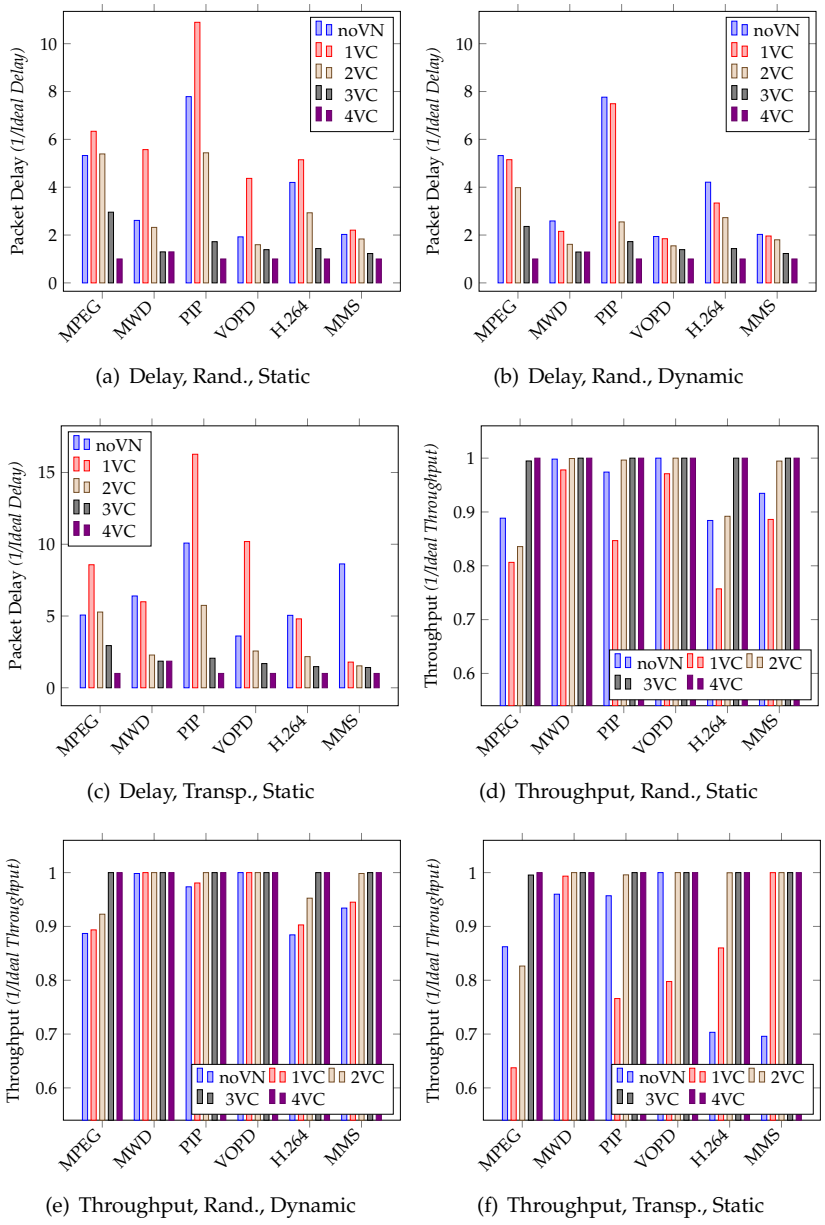
Multimedia Applications Communication task graphs are now used for latency and throughput investigations. Details about the six video processing applications (VOPD, MPEG4, PIP, MWD, H.264 and MMS), which have been used, are provided in section 5.2.3.3. Regions of 3x3 (PIP), 3x4 (VOPD), 4x4 (MPEG4, MWD and H.264) and 5x5 (MMS) are configured according to the calculated mapping and the number of nodes required by the respective application. Each application is mapped separately to its respective region, located in the center of the mesh. Transpose and uniform random traffic are used to simulate global communication through the region of the applications. All results shown in figure 6.23 are relative to an ideal scenario without global communication.

Figure 6.23(a) shows the latency for static virtual network with uniform random global communication. Assigning two or more VCs to the VN improves delay for all applications compared to the reference architecture without VN support. A VN using 3 VCs can reduce the communication latency of PIP application by up to 78 %. Once a dynamic virtual network is established, performance can also be improved for VNs using small numbers of VCs exclusively, as shown in figure 6.23(b). Transpose global communication has the greatest effect on the delay of the applications, as shown in figure 6.23(c).

The throughput of MWD and VOPD is rather low, thus they exhibit adequate performance for all VN configurations, as shown in figure 6.23(d). MPEG, H.264 and MMS require 3 VCs to reach the requested throughput under uniform random background traffic. For MPEG and H.264, the throughput can be increased by around 12 % relative to the reference, if using 3 VCs. Figure 6.23(f) summarizes the results for transpose global traffic, which strongly affects the communication within the region. If only 2 VCs are assigned to the virtual network, used by H.264 and the MMS, a significant increase in throughput of 42 % and 44 % respectively is obtained compared to the reference where no VN is used.

6.4.3.2. Synthesis

The NoC router, introduced in section 5.1, is used as a basis in order to implement virtual network support. A standalone router of an 8x8 mesh NoC is synthesized using the flow, introduced in section 5.3.2.1, with a 45 nm *TSMC* standard cell library. Two variants of the router were synthesized. The *small* router has the



(a) Delay, Rand., Static

(b) Delay, Rand., Dynamic

(c) Delay, Transp., Static

(d) Throughput, Rand., Static

(e) Throughput, Rand., Dynamic

(f) Throughput, Transp., Static

Figure 6.23.: Packet latency and throughput of video processing applications for different VN configuration.

same parameters[28] as the SystemC model used for simulation in the previous section. The parameters[29] of the NoC of Intel's Single-chip Cloud Computer [114] are used for the *large* router version.

Table 6.1 shows the synthesis results for area, clock frequency and power consumption[30]. Two versions of the small and the large router were synthesized. The *basic router* serves as a reference because it does not contain any special features, just as the router introduced in section 5.1. The second version includes one virtual network management unit with two VNCUs per router port, supporting two virtual networks per router. The presented results show the low implementation overhead of the VNMU. Adding a VNMU to each router port increases the area utilization by 5.2 % and 0.3 % for the small router and large router respectively. The total power consumption increases by 2.1 % for the small and 1.4 % for the large router once a VNMU is added to the design. The clock frequency is independent from the use of a VNMU, since the critical path of the router is in the transmission control unit. A detailed investigation showed that the size of the VNMU grows linearly with respect to the number of supported virtual networks.

Router Version	Area (μm^2)		Frequ. (MHz)		Power Dyn. (mW)		Power Stat. (μW)	
	small	large	small	large	small	large	small	large
Basic (BR)	30,558	479,753	1500	1000	4.29	18.28	373	4520
BR + VNMU	32,139	481,157	1500	1000	4.37	18.56	391	4553

Table 6.1.: ASIC synthesis results of a 5-port router without (*BR*) and with virtual network support (*BR + VNMU*).

The synthesis results show the small implementation overhead for virtual network support. Thus, it is reasonable to enable VN support in a NoC, although it may only be used in some situations.

6.5. Summary

In this chapter, three different QoS concepts were presented. The QoS schemes are carried out in the form of modular extensions of the best-effort packet switching NoC, introduced in chapter 5. At the beginning of the chapter, state of the art was discussed in general and by reference to existing QoS schemes.

[28] Parameters used for the *small* router design: 4 VCs, 4 buffer slots, 32 bit link size.
[29] Parameters used for the *large* router variant: 8 VCs, 4 buffer slots, 256 bit link size.
[30] Power consumption was obtained from a traffic scenario with heavy load, which can be seen as a worst case scenario.

In section 6.2, a concept enabling multiple hard guarantees for bandwidth and latency of end-to-end connections in packet switching NoCs was presented. It is the first scheme that enables to adapt throughput and latency guarantees at run-time for each guaranteed service connection. Unlike previous concepts, the proposed strategy has the advantage that higher service levels do not dominate lower ones; due to the use of a fair scheduling strategies. Best-effort and guaranteed service communication flows dynamically share uniform NoC resources. The concept comprises an extension of the basic router design as well as software support. A task mapping and GS connection management scheme is introduced, which takes the monitoring information into account that are provided by the monitoring infrastructure from section 5.5.3. The presented results show that GS connections clearly benefit from higher service levels regarding throughput and latency and that the given guarantees can be observed for all investigated scenarios. Communication performance of video applications could be improved by up to 40 % in delay and 31 % in throughput compared to state of the art GS connections using fair round-robin scheduling. An evaluation on a FPGA prototype with a matrix multiplication applications showed a reduction of the required bandwidth by up to 26 %. The power consumption for communication is reduced by up to 20 % when the proposed GS communication scheme is used, instead of best-effort packet switching.

In section 6.3, a region-based communication resource management strategy for the previously introduced guaranteed service communication concept was described. Hardware support allows run-time adaptation of resource allocation policies in different NoC regions with regards to a distributed OS management concept. The results obtained from the simulation framework, introduced in section 5.2, illustrate the gain in overall communication performance. The proposed decentralized management of communication resources leads to a delay reduction of up to 36 % and a throughput improvement of up to 17 % for GS traffic. Synthesis results demonstrate that the proposed hardware extensions have a small area footprint and a very low impact on the clock frequency.

Section 6.4 addressed a methodology that enables virtual networks in packet switched networks with virtual channels, as the NoC introduced in section 5.1. The virtual networks are defined according to the application requirements. The bandwidth and size of a virtual network can be configured at run-time. The proposed virtual network concept enables decoupling of the local communication of applications from global communication, which is crossing the region to which the application is mapped. Virtual networks can be used to isolate the communication of a specific application by offering an exclusive-used network. The presented results of various scenarios, including state of the art multimedia applications, show that virtual networks can reduce the delay for data transmission and improve the throughput of the applications. The delay may be reduced by up to 78 % and the throughput could be increased by 72 % at maximum compared to

a best-effort packet switching network on chip. ASIC synthesis results show that the additional hardware modules for resource management have a small impact on area utilization and power consumption.

7. Self-Optimization and Self-Organization

When increasing the size of the architecture, the overhead for hardware resource management grows [140]. Thus, it is necessary to limit the management demands of the hardware; ensuring scalability. Self-optimization and self-organization capabilities of the hardware can assist in reducing the management overhead for the software executed on the architecture.

In this chapter, novel self-optimization strategies are presented. The proposed mechanisms comprise strategies for improving guaranteed service connections, data collection and power management. All of the presented mechanisms address packet switching networks with virtual channels and are built as optional extensions of the router design, presented in section 5.1.

For each of the proposed mechanisms, related work is first discussed and limitations of existing work are analyzed. Subsequently, the proposed self-optimization features are presented in detail with the aim of overcoming said limitations.

7.1. Distributed Rerouting

Routing is an important aspect of a network on chip with respect to performance, power consumption and fault tolerance. Distributed routing schemes are necessary to ensure scalability. A detailed description is provided in section 2.4.5.2.

Adaptive routing schemes can offer multiple degrees of freedom when routing packets, enabling run-time optimization of communication flows. Thus, adaptive routing schemes have been proposed for fault tolerance [281], load balancing [94] and performance improvement [46] in NoC-based architectures. The selection of a particular route, which fulfills the rules of the adaptive routing scheme, is performed at run-time. A selection strategy [15] is used to determine the most suitable route[1]. When adaptive routing is used to enable fault tolerance, for example, the selection strategy could select a route according to the current fault

[1]The selection strategy can either be implemented in a decentralized manner [117] in case of distributed routing or as part of a source routing implementation [181].

map of the architecture [76] or dependent on CRC error rates. Once adaptive routing is used to improve communication performance, the selection strategy selects the route with the lowest utilization [15, 117]. Adaptive routing, in combination with a selection strategy, is very well suited for best-effort packet switched communication. As such, the routing decision is only made for a single packet with limited size and short transmission period.

Guaranteed service end-to-end connections, introduced in section 6.2, typically exist for long periods. Existing distributed routing schemes and selection strategies can only be used to find an optimal path for that point in time, at which the connection is established. However, the load or fault situation may change drastically during the lifetime of a GS connection. In a dynamic system, existing connections are released and new ones are established when applications finish execution and new ones are mapped and executed. This may lead to situations where a previous routing decision, based on a state-of-the-art selection strategy ([117, 15, 76]), which was optimal at the reservation time, is subpar at a later point in time. Consequently, unbalanced load situations and GS connection setup failures may result.

In the following sections, a distributed self-optimizing routing mechanism for guaranteed service end-to-end connections is introduced in order to address the QoS scheme; introduced in section 6.2. The proposed strategy enables rerouting of existing GS connections to react on changing load conditions during run-time. In contrast to existing routing schemes, the proposed method withdraws previously made routing decisions for established connections, transparently. The distributed implementation enables the optimization of the GS communication. The concept, implementation and evaluation of the scalable rerouting scheme, are detailed below.

Rerouting was implemented as part of a student work [Sin12] and published in [HSK+13][2].

7.1.1. Concept of Self-Adaptive Rerouting

For connection-oriented GS communication, the routing decision is made for a much longer period compared to best-effort transmissions. Typically, the lifetime of guaranteed service connections is not limited. Hence, they can exist for very long periods, e.g. for the complete run-time of an application. In a dynamic system, new applications are mapped and executed, while others incessantly finish execution. Thus, NoC load situations can change significantly over time. This can lead to unbalanced load situations with respect to GS connections.

[2] Extracts from [HSK+13], which are completely written by the author of the work in hand, are used verbatim in this section without further identification.

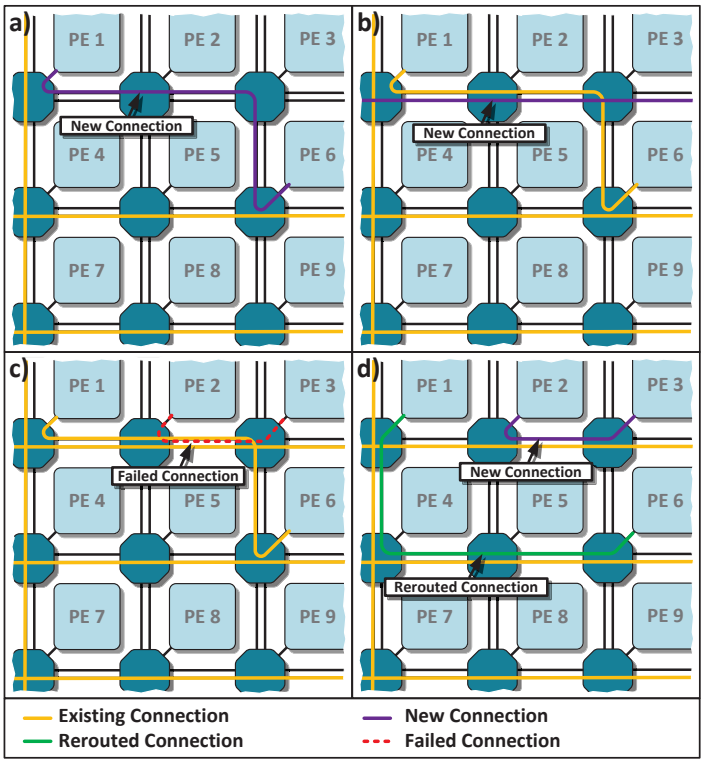


Figure 7.1.: Chronology of a situation where rerouting is required.

Figure 7.1 gives an example of such an unbalanced load situation for a 3x3 segment of a larger NoC. The network has two virtual channels and uses a fully adaptive routing scheme, as proposed in [74]. In figure 7.1(a), one new connection between two PEs, *PE 1* and *PE 6*, is established. The taken route is optimal at that point in time. The path with the lowest utilization of the links is taken. In figure 7.1(b), another connection is established. Assuming minimal routing, this connection does not have degrees of freedom while routing, because source and destination node are in the same row. Later on, a connection between *PE 2* and *PE 3* must be established. This is not possible since the required link and the VCs are already occupied by two other connections, as shown in figure 7.1(c). When rerouting is used, the connection between *PE 1* and *PE 6* can be rerouted to successfully establish the new connection between *PE 2* and *PE 3*, as presented in figure 7.1(d). The example provided shows how rerouting of connections can be used to increase the number of GS connections and to balance the load in the NoC. Communication bottlenecks are detected by the NoC. Subsequently, existing

guaranteed service connections are rerouted to balance the NoC load. The results, presented in section 7.1.3, will show that rerouting can increase the success rate for establishing new GS connections by intelligently reallocating existing connections. The selection of the best rerouting candidate and the search of an alternative route are described in section 7.1.2.

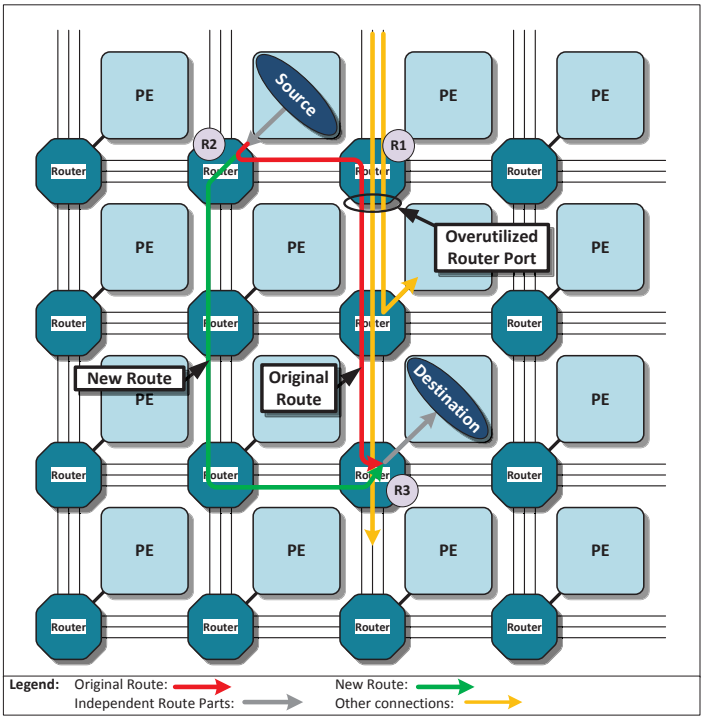


Figure 7.2.: Example of a rerouting procedure - one of the connections using the over-utilized port is rerouted.

Figure 7.2 gives a more detailed example of a rerouting process. A rerouting procedure involves three routers *R1*, *R2* and *R3* and is composed of three phases:

P1-Detection: Router *R1* detects a potential over-utilization. Such an overload situation can e.g. be the utilization of all VCs or a link utilization near 100 %. Afterwards, the router selects one of the connections using the congested output port by a selection algorithm, which will be described in detail in section 7.1.2.1. The selected route is the candidate for rerouting. A rerouting request is sent to the source router *R2* of the selected connection. All routers are able to issue rerouting request simultaneously.

P2-Search: The source router *R2* receives the rerouting request message. If multiple rerouting requests arrive at the same time, at one router, they are processed sequentially. Afterwards, router *R2* initializes to setup a new, alternative route to the connection destination. Initiating an alternative route from the source router of the selected connection has two main advantages compared to an implementation where an arbitrary router of the original connection is used as a source for an alternative connection: (1) implementation is simplified since the address of router *R2* is known to be the source address of the connection and (2) the length of the new alternative route is longest, if the source router is used, resulting in the most degrees of freedom for the new alternative route.

In order to set up an alternative route, *R2* generates a new head flit for connection reservation. The header is injected into an alternative output port to ensure a different path. Fully adaptive routing [74] or adaptive odd-even turn routing, described in section 5.1.4.2, can be used to ensure a minimal route through an alternative output of router *R2*. In the example provided, the new port is the south port, whereas the original port was the east port. The new head flit now performs an implicit search by using adaptive routing in combination with a selection strategy and is transmitted towards the destination and performs the required reservations at each node. Router *R3* at the destination node detects the incoming rerouting head flit. It stops the reservation process for the new alternative route, deletes the head flit and sends back an acknowledge message to router *R2*.

P3-Switching: Router *R2* is now responsible for switching between the old and the new route, once it receives an acknowledge message from *R2*. After receiving the control message, it redirects the data stream to the new route. A tail flit is injected into the virtual channel, associated with the old route, in order to release this connection. The old route is then released and data is forwarded through the new connection. Router *R3* must also perform a switching between the old and new connection. It detects the tail flit, coming through the old connection, and deletes it. Afterwards, it switches directly to the new route to avoid additional delay and to preserve QoS latency guarantees while rerouting. This switching strategy guarantees in-order arrival from the PE perspective, as detailed in section 7.1.1.1. After the switching process, the rerouting procedure is finished and the originally utilized resources are released. Subsequently, the resources are available for other connections or best-effort data transmissions.

In addition to the performance improvements later investigated, the properties of the proposed mechanism can be summarized as follows:

- It is transparent from the perspective of the processing element. Hence, no software support is required, as described in detail in section 7.1.1.1.

- QoS guarantees of GS connections are observed during rerouting, since switching between the original and new route is performed after the new route is successfully established.
- In-order arrival can be guaranteed by the proposed switching mechanism. Thus, reordering is not required at the destination node, as described in section 7.1.1.1.
- It is fully decentralized and distributed. Thus, it is independent from the NoC size and hence scalable.

7.1.1.1. In-Order Arrival & Transparency

The concept of in-order arrival is illustrated in figure 7.3. It is required to ensure that the communication is unaffected by the rerouting process. As illustrated in the picture, the flits reach the destination router in the original order and without interruption or additional delay. This is achieved by transmitting the flits through the old route until the rerouting process is completed. At this moment, the new route exists in parallel to the old route. Then, the communication is switched to the new route and the old route is released. The destination router (*R3*) ensures that the last flit, which has taken the old router, is forwarded before the first flit from the new route is processed. Thus, preserving the original order.

The rerouting process is fully transparent from the processing elements perspective. All QoS guarantees are preserved during rerouting. An existing QoS connection is only rerouted if an alternative route is found. Thus, performance and guarantees of existing QoS connections are never affected by rerouting.

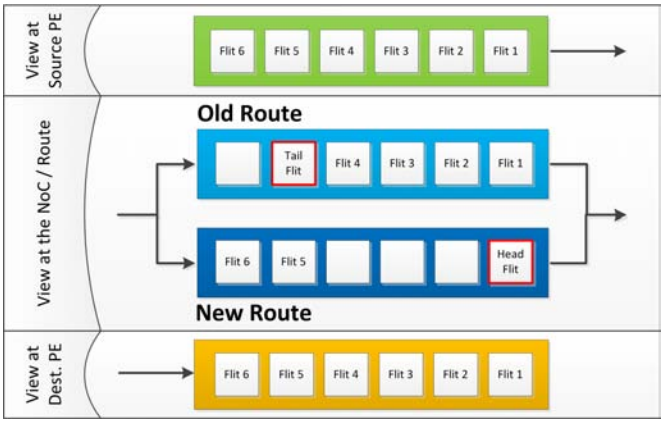


Figure 7.3.: Illustration of in-order arrival during rerouting procedure [Sin12].

7.1.1.2. Potential Miscues

In the previous paragraph, a successful rerouting procedure was explained. However, rerouting has the potential to fail for various reasons. *Phase 1* fails if a candidate for rerouting is not found. In such a case, rerouting is not triggered. If no alternative route is found, *phase 2* of the rerouting procedure fails. In this case, the transmitted head flit does not reach the destination router, due to unavailable communication resources. Failing of search phase (*P2*) is detected by the router, initializing the search via timer. When a timeout has lapsed, the resources that have been reserved for the new route are released. Therefore, a tail flit is generated and injected by router *R2*. *Phase 3* will not fail because switching is always successful once a new route is established.

In order to limit the communication overhead of *phase 1* and *phase 2*, the number of outstanding requests per phase is restricted to one request per router.

7.1.2. Hardware Implementation

Scalability is one of the main reasons for using NoCs for on-chip communication and is also a main objective of the communication infrastructure presented in this work. Consequently, rerouting functionality must be implemented in a distributed way within the routers of the NoC to ensure scalability.

As shown in figure 7.2, three routers are involved in a rerouting process. They fulfill three different tasks *T1-T3*: *Task 1* detects the congestion in router *R1*. *Task 2* triggers the reservation process for a new connection and later the switching between the old and new connection at the source router *R2*. *Task 3* detects and responds to successful rerouting events and later switching between the old and the new connection at the destination router *R3*. Finite state machine implementations are used to trigger the sequence of actions that are required for each task. Each router has equal functionality to fulfill each of the tasks *T1-T3*. The three tasks are not to be confused with the phases *P1-P3*, introduced in section 7.1.1 – *P1* uses only *T1*, *P2* uses the *T2* FSM in router *R2* and *P3* is carried out by an interaction between *T2* and *T3* FSM.

The implementation of the tasks *T1-T3* is explained briefly in the following paragraphs. A more detailed description is provided in [HSK+13].

7.1.2.1. Congestion Detection (T1)

In the proposed scheme, rerouting is triggered by a potential overload at a router port. Overload situations are detected per link, hence per router port. Basic metrics for overload detection are the link utilization, virtual channel utilization

and buffer utilization. Depending on the used metric, different hardware monitors are used to detect the utilization threshold, which is triggering a rerouting procedure. The infrastructure introduced in section 5.5.3 is used for monitoring. This work focuses on the VC utilization metric for overload detection, because this is most reasonable with respect to the reservation of virtual channels by GS connections. A potential overload is indicated once the number of VCs, occupied by GS connections, reaches the limit $VC_{GS,max}$ at any router port, according to equation 6.8. A detection of an overload triggers a rerouting procedure.

After a potential overload situation is detected by online hardware monitors, a connection using the congested output port is selected for rerouting. Ideally, the connection with the best chance to be rerouted successfully will be selected. In general, this is the connection with the highest number of minimal alternative paths through the NoC. Hence, the degrees of freedom while routing, i.e. the number of different shortest paths between source and destination in a meshed NoC, is used as metric. This number depends on the distance between source and destination in X and Y coordinates, defined as δx and δy. For a fully adaptive routing algorithm, where all possible minimal paths have the same Manhattan distance, the number of alternative paths ($\#path$) between source and destination can be calculated as follows:

$$\#path = \binom{\delta x + \delta y}{\delta x} = \frac{(\delta x + \delta y)!}{\delta x! \cdot \delta y!} \approx \delta x \cdot \delta y \tag{7.1}$$

Calculating a factorial is rather complex for higher values of δx and δy. Consequently, the approximation in equation 7.1 is used for an efficient hardware implementation of rerouting. It uses the area spanned by δx and δy as an approximation for connection selection. The metric is calculated per connection, then the maximum is extracted and finally, the respective connection is selected. A respective algorithm for rerouting candidate selection is presented in [HSK+13].

If a valid rerouting candidate is found, a rerouting request message is sent to the source router of the selected connection. Therefore, the source addresses of all connections are stored in each router. The rerouting request message and all other communication between the rerouting FSMs could either utilize normal BE packet switching or a dedicated control channel. However, the use of a dedicated control channel has several advantages, as described in section 5.5.1. With regard to rerouting, a dedicated control channel can isolate communication for self-optimization from normal communication. Consequently, the control network from section 5.5.1 is used.

In order to avoid a noticeable communication overhead by rerouting requests, the number of rerouting requests must be limited. Therefore, a timeout between two rerouting requests is used. After this delay, a new congestion can be detected

and rerouting can again be triggered. If the connection selection fails, a waiting period is used to avoid frequent triggering of rerouting and thus communication overhead.

7.1.2.2. Source Router (T2)

The source router of the connection, which was selected for rerouting, receives the rerouting request message sent by the overloaded router. In order to find an alternative route, the routing unit is requested to calculate an alternative output port for the connection to be rerouted[3]. A VC for the new connection is reserved at the alternative output port. If no VC is available at the selected output port, rerouting fails. After the VC reservation, a header is transmitted through the new output to search and establish the alternative route. The head flit is processed in the following routers, like any other header. Distributed adaptive routing, in combination with a utilization-aware port selection strategy, as introduced in section 5.1.4.2, is used in the current rerouting implementation. Consequently, the procedure of searching for an alternative route is equal to normal routing.

Once an alternative route is found and the new connection is established, the destination router sends an acknowledge message to the source router. If the acknowledge message does not arrive within a predetermined time frame, a timeout occurs. Then it is assumed that the establishment of the new connection was not successful in one of the other routers on the path to the destination node. Hence, the timeout triggers the release of the partially established new route by injecting a tail flit into the virtual channel, which was reserved for the new connection. If the acknowledge message arrives at the source router before the timeout occurs, the routing table is modified to forward incoming flits of the rerouted connection, via the new route. Next, the original route can be released by sending a tail flit through the appropriate output VC. This is the last step of a rerouting procedure, from the perspective of the source router.

7.1.2.3. Destination Router (T3)

The original and the new route could end up separately in the destination tile. However, without merging the old and the new connection, in-order arrival could not be guaranteed, as described in section 7.1.1.1. In order to ensure compliance of QoS guarantees and in-order arrival while rerouting, the FSM at the destination router is used. Once a rerouting head flit arrives at the destination router, the FSM blocks the head flit. An acknowledge message is sent to the source router to inform

[3]The use of fully adaptive routing [74] ensures that an alternative output port exists within the source router for the connections selected for rerouting, since they are only selected if $\delta x \cdot \delta y > 0$ is valid for them.

it of the successful rerouting. As described in section 7.1.2.2, the source router then injects a tail flit to release the old route. The FSM in the destination router waits for this tail flit. Upon its arrival, the routing table is modified to switch between the old and the new connection. The head flit of the new connections and the tail flit closing the original connection are erased at the destination router. This step completes the rerouting procedure. The remaining body flit from the old connection is forwarded by the router before the first flit from the new connection. Thus, the original flit-order is maintained.

7.1.3. Evaluation

The benefits of rerouting, with respect to performance improvement as well as the communication overhead, will be now investigated. Therefore, the simulation framework, introduced in 5.2, was extended to support rerouting. In addition, rerouting support was implemented for the HDL design in order to investigate the implementation overhead for an ASIC.

7.1.3.1. Simulation

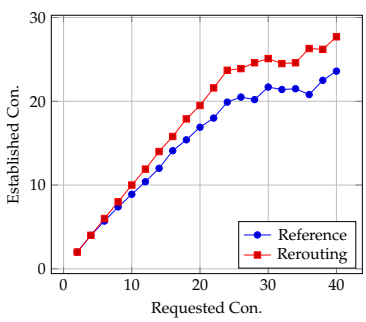
The simulation framework, developed as part of this work, was extended for the following evaluation in order to support DyAD smart routing [117]. DyAD is state of the art adaptive routing, combining adaptive odd-even turn routing with a load dependent selection strategy. DyAD is used as a *reference* for the following evaluation. The rerouting implementation uses odd-even turn routing, as described in section 5.1.4.2, for a better comparison to DyAD. Additionally, fully adaptive routing [74] was used to show that rerouting is suitable for different adaptive routing schemes. A 10x10 mesh topology is used for the following investigations. The routers are configured to have 4 virtual channels. According to equation 6.8, $VC_{GS,max}$ is set to 3 to allow three GS connections per link.

Synthetic traffic and video processing applications, introduced in section 5.2.3.3, are used for the following analysis. The synthetic traffic scenario generates transpose GS connections[4]. Transpose traffic results in hot spots at the center of the NoC and thus is suitable for investigating hot spot avoidance capabilities.

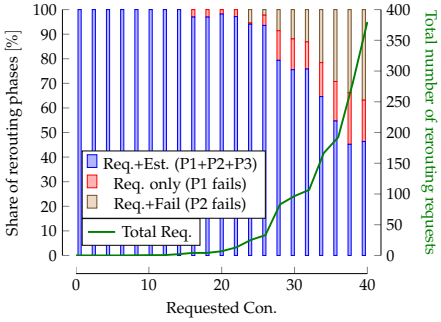
Number of Connections The main goal of the proposed rerouting concept is to increase the number of GS connections which can be successfully established. The number of connections represent the likelihood of successfully establishing requested QoS data transmissions from the application perspective. The number

[4]Transpose connections are established bi-directional in meander shape [81].

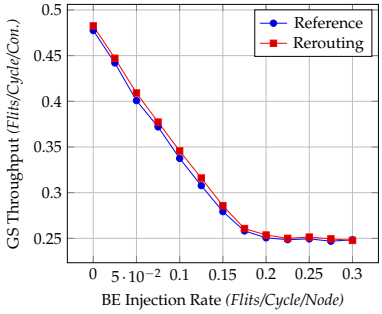
of possible GS connections is investigated by setting up connections in transposing manner. In figure 7.4(a), the number of established connections is given as a function of the requested connections. Enabling rerouting improves the number of established GS connections on average by 14 % for the given transpose scenario. At maximum, 26 % more connections could be established, compared to a NoC where rerouting is not supported.



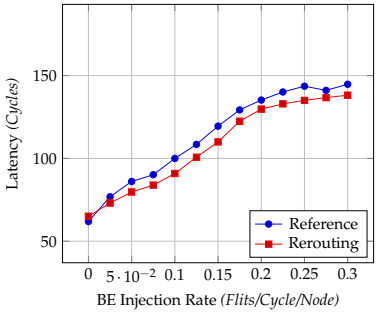
(a) Established Transp. GS Connections



(b) Overhead Transp. GS Connections



(c) Throughput of Transp. GS Connections



(d) Delay of Transp. GS Connections

Figure 7.4.: Simulation results for synthetic transpose GS connections.

Communication Overhead Analysis Rerouting induces communication overhead, due to additional communication between the three routers, involved in the rerouting procedure. Figure 7.4(b) analyzes the overhead, caused by rerouting, for different numbers of transpose GS connections. For a low number of connections, there is nearly no rerouting required, as shown in figure 7.4(b)(right y-axis). Between 14 and 26 bi-directional connections the success rate (*Req.+Est*) is near 100 %. This implies a very low overhead of only 4 flits per successful rerouting. If

the entire NoC is over-utilized[5], the number of failures during rerouting phases *P1* and *P2* (*Req.+Fail* and *Req.+Est.*) increases, as expected. In practice, this general over-utilization is irrelevant, because in that case the number of requested GS connections is higher than supported, as investigated in the previous paragraph. Taking the low overhead per rerouting procedure into account, the communication overhead is negligible during normal operation. This shows that rerouting does not lead to a self-accelerating process of congestion, as could be expected. Regardless, due to the use of the control network, rerouting communication is decoupled.

Latency and throughput Increasing the number of GS connections for a given scenario is the main objective of rerouting. In addition, rerouting's impact on throughput and latency is investigated. Therefore, a scenario with a high number of 24 bi-directional GS connections in transpose manner is used. BE traffic with random uniform distribution is used as background traffic and is varied. The additional BE traffic leads to higher load and shifts GS communication latency and throughput closer to the guaranteed limits[6].

Figure 7.4(d) shows the latency of established GS connections for variable BE injection rates. For very low injection rates, there is no delay reduction when enabling rerouting. For BE injection rates higher than 50 $Flits/Cycle/Node$, rerouting results in 6 % delay reduction on average and 9 % at maximum. The worst case latency of 180 $Cycles/Packet$ is never violated.

Figure 7.4(c) illustrates the average throughput for an established GS connection. Independent of the applied BE injection rate, rerouting slightly improves the throughput of the GS connections. On average, 2 % throughput improvement can be observed for the given scenario, with the maximum improvement being 3 %. The guaranteed throughput limit of 0.25 $Flits/Cycle/Node$ is not violated. This shows that rerouting does not harm the guarantees for throughput.

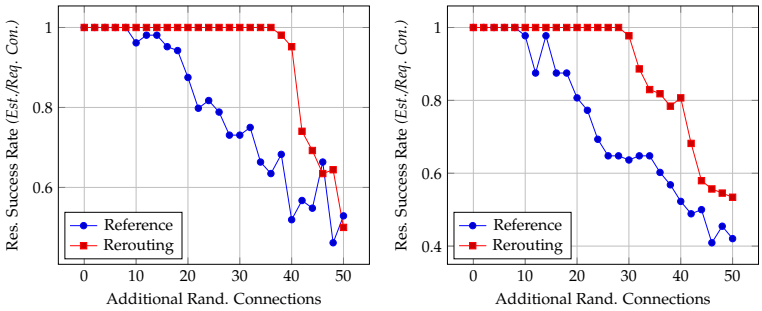
Better results for latency and throughput improvement are expected when a different, more sophisticated triggering mechanism for rerouting is used. Currently, only hot spots are detected, but rerouting can be adapted to balance the load before hot spots occur. However, this remains open for future work.

Application mapping In addition to the synthetic scenarios, the MWD, MPEG4, VOPD and PIP task graphs from section 5.2.3.3 are used for evaluation. The task graphs of the applications are mapped in a way that the distance between

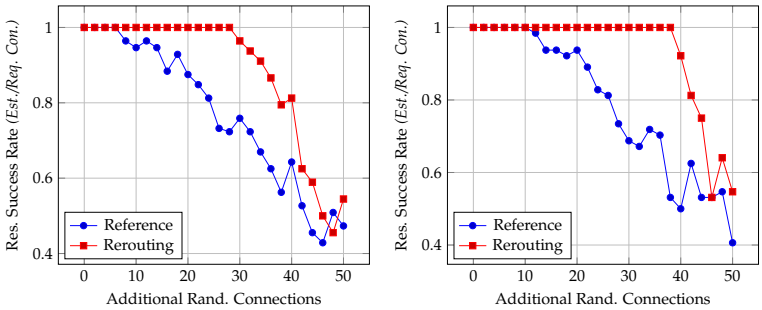
[5] An over-utilization takes place if more than 26 bidirectional transpose GS connections are established in the used 10x10 NoC. This was determined experimentally for the given NoC.

[6] For the used NoC and the investigated connections, the following guarantees can be given according to section 6.2.1.1: $Latency_{wc} = 180\,Cycles/Packet$, $Bandwidth_{wc} = 0.25\,Flits/Cycle/Connection$.

communicating cores is minimized. GS connections, in random manner, are initially established to generate a base load. Subsequently, the applications are mapped and initiate the setup of their GS connections. How rerouting can assist in successfully initiating the required connections, is investigated. Therefore, the *reservation success rate (RSR)* is defined as the ratio between requested and established connections.



(a) GS Connections - MWD App. (b) GS Connections - MPEG App.



(c) GS Connections - VOPD App. (d) GS Connections - PIP App.

Figure 7.5.: Rerouting evaluation with four video processing applications which use GS connections.

Figure 7.5(a) shows the amount of GS connections of the MWD application, which can be setup successfully, for different numbers of additional connections. In total, MWD requests for 13 GS connections. If rerouting is disabled (*reference*), only 10 additional connections are tolerated until the RSR drops. When utilizing rerouting, 35 additional GS connections can be established and it is still possible to embed the requested GS connections for MWD successfully. Figure 7.5(b) presents the RSR for the MPEG application. The number of existing GS connections can be three times higher, when establishing the new connections of the MPEG decoder with rerouting being activated. Figure 7.5(c) and figure 7.5(d) show similar results

for the mapping of VOPD and PIP application. In general, the results exhibit that rerouting can significantly improve the likelihood to establish requested connections successfully from an application perspective. The number of existing GS connections, which are tolerated when establishing the connection of the investigated applications, could be increased by a factor of 3.875 on average, by the use of rerouting.

7.1.3.2. Synthesis

In order to investigate the implementation overhead of rerouting, it was carried out for the HDL implementation of the router design, introduced in section 5.3. A standalone router of an 8x8 mesh NoC[7] is synthesized using the *TSMC* 45 nm standard cell library. The investigated NoC has a link size of 256 bit. Two designs, with 4 and 8 virtual channels respectively, were used to estimate the impact of this parameter in terms of power and area.

Synthesis results are presented in table 7.1 for a *reference* and *rerouting* variant. The rerouting version was built on the basis of the reference router by including the rerouting-specific components. A clock frequency of 800 MHz was targeted and achieved for all router configurations. Post-synthesis power and area consumption were analyzed. The rerouting implementation has an area overhead of 8.8 % in case of 4 VCs and 8.2 % in case of 8 VCs; compared to the reference. With a growing number of 8 VCs, the relative area overhead slightly decreases. The additional power consumption of the rerouting components is 14.7 % in case of 4 VCs and 13.6 % in case of 8 VCs. In addition to clock gating, power-gating may be used to significantly reduce the power overhead; due to the long idle periods of rerouting components. However, power-gating implementation is complex and left open for future work.

Router version	**VC**	**Frequency** (MHz)	**Area** (μm^2)	**Power** (mW)
Reference	4	800	245193.5	13.534
Rerouting	4	800	266696.2 *(+8.8 %)*	15.521 *(+14.7 %)*
Reference	8	800	476132.3	21.273
Rerouting	8	800	515194.5 *(+8.2 %)*	24.157 *(+13.6 %)*

Table 7.1.: ASIC synthesis results for a single router of an 8x8 mesh NoC in 45 nm.

In summary, it can be stated that rerouting increases the number of GS connections, which can be successfully established, by up to 29 % for synthetic hot spot traffic.

[7] In contrast to the simulation, the dimensions of the NoC are reduced from 10x10 to 8x8 because dimensions which are a power of two ensure an area-efficient hardware realization.

Delay and throughput of the established GS connections can also be improved. It was shown that the amount of tolerated GS connections could be increased by a factor of 3.875 on average, when mapping the four investigated video processing applications to a NoC with rerouting support. However, rerouting includes an area overhead of 8.2 - 8.8 %.

7.2. Auto-GS and Connection Replacement

Guaranteed service connections, introduced in section 6.2, enable QoS by hard guarantees for throughput and latency. Additionally, they typically improve latency and NoC utilization and consequently the performance and power consumption compared to best-effort communication. The reason for these positive side effects are (1) the reduced number of router pipeline stages used during GS data transmission[8] and (2) the reduced protocol overhead for GS communication[9]. Thus, it is desirable to increase the amount of GS communication in a network on chip independent from QoS requirements. Two different concepts for usage of GS communication, independent from QoS demands, have been investigated in [ZHW+13] and [Lu13]. These self-optimization strategies map BE communication flows with high bandwidth requirements to GS connections to improve their performance. The two concepts will be explained subsequently.

In the literature, there are existing approaches for hardware-controlled optimization of communication flows. Prediction-based flow control for network on chip traffic is presented in [193]. Models of traffic sources and router are used to predict possible congestion. This information is used to control the injection rates at the source nodes. However, global knowledge is required for this strategy, which limits scalability.

A run-time adaptation scheme for buffer allocation in NoC-based architectures is presented in [69] and [70]. Links are established at run-time between source and destination nodes. The presented results show that the proposed buffer allocation scheme leads to better buffer utilization in the NoC routers. However, the area overhead, introduced by this adaptive scheme, is significant and the scalability with respect to the number of QoS connections is limited.

Software-based approaches for dynamic management of NoC resource that take QoS demands into account, are presented in [197] and [159]. However, these

[8] Due to the previous resource allocation, enabled by GS end-to-end connections, data transmission is simplified because routing and virtual channel allocation are only performed once during connection setup, as described in detail in section 5.1.3.

[9] Head flits, containing network layer protocol information, are only required for connection setup. Subsequently, during data transmission there is no additional protocol overhead, as described in section 5.1.2.

approaches only address applications with QoS requirements. Improving the communication of other applications with the use of QoS schemes is not addressed.

In [5], a concept is proposed which accelerates network communication by exploiting communication temporal locality. Therefore, the arbitration history is captured within each router. For each flit, it is verified whether a pseudo-circuit is used instead of the normal router pipeline. These pseudo-circuits can reduce the processing latency for a limited number of virtual channels. However, a pseudo-circuit only accelerates the communication at a single hop, improvement in end-to-end communication flows is not considered. In addition, there is a significant implementation overhead experienced by the proposed scheme.

In contrast to existing schemes, the concepts introduced in the following have a low implementation overhead. They rely on the QoS scheme, introduced in section 6.2, which allows to improve the communication of complete end-to-end flows. The additional implementation costs only comprise monitoring and decision making components.

7.2.1. Concept and Implementation

GS connections enable hard guarantees for throughput and latency. However, idle resources, which are currently unoccupied by QoS communication flows may be allocated for other communication flows to improve their performance. Therefore, the communication hardware must take a decision, at run-time, regarding the establishment of additional GS connections. In order to do so, an optimization criteria is required, which can be used to decide where to establish an end-to-end connections. For the following implementation, this criteria is chosen to be the amount of communication between a pair of nodes. If two nodes communicate frequently, a GS connection can be used to reduce the communication latency, protocol overhead and thus the power consumption for communication.

However, guaranteed service connections that are established on behalf of software with QoS demand, must be treated differently, compared to those connections established just for optimization. Connections with QoS demands cannot be taken into account by the self-optimization strategies, presented now. A clear differentiation is required to ensure preservation of GS connections with QoS demands. Consequently, a protocol extension of a GS head flit is required for differentiation. An additional 1-bit flag is added to each GS header, defined in section 5.1.2, differentiating between GS connections which have QoS demands and others which can be taken into account for self-optimization. The flag is referred to as *replaceable*. It indicates that the respective connection has no QoS demands and can thus be taken into account for self-optimization by the *Auto-GS* or *connection replacement* hardware.

7.2.1.1. Connection Replacement

The *connection replacement* strategy is an optimistic self-optimization strategy. The network adapter initially tries to setup a GS connection for all communication flows. GS connections, with low utilization, are subsequently replaced by best-effort communication. The self-optimization strategy, which makes the decisions, is implemented within the NoC routers.

Each router periodically verifies the utilization of virtual channels by GS connections. When the limit for $VC_{GS,max}$ is reached, according to equation 6.8, and a new GS connection must be established, one existing GS connection is selected to be *replaced* by best-effort packet switching communication. In order to make the decision which connection should be replaced, the bandwidth utilization of each existing connection is monitored inside the routers. A separate monitor must be utilized for each virtual channel to capture the necessary monitoring data[10]. The GS connection with the lowest bandwidth requirements, which is marked to be replaceable (*replaceable* flag set to 1), is selected. Subsequently, a release process is triggered for this connection and best-effort communication must be used. On that point, the router, making the replacement decision, must communicate to the source network adapter to initiate the release process. The control network, introduced in section 5.5.1, can be used for communication. After the selected GS connection is released, the number of VCs occupied by GS communication flows, is again lower than $VC_{GS,max}$. This ensures a new connection, with QoS requirements, can be established successfully on demand.

A specific example for a replacement procedure in a NoC with two virtual channels is provided in figure 7.6. The example exhibits how a situation occurs when a replacement takes place. In figure 7.6(a), a new connection is established between *PE 1* and *PE 6* and marked as replaceable. The replaceable GS connection improves the performance of the communication flow between *PE 1* and *PE 6*. This is transparent to the software. Figure 7.6(b) shows another connection, which is requested to be established between *PE 2* and *PE 3*. This reservation request then triggers the replacement strategy inside the router connected to *PE 2*. The connection between *PE 1* and *PE 6* is selected to be replaced by BE packet switching. Subsequently, a new connection can be established, as presented in figure 7.6(c). Once the selected connection is released, as shown in figure 7.6(d), the new GS connection between *PE 2* and *PE 3* can be established.

Additional details about the concept and implementation of the proposed connection replacement strategy are provided in [Lu13].

[10] The monitoring required for replacement decision making differs from the monitors introduced in section 5.5.3.

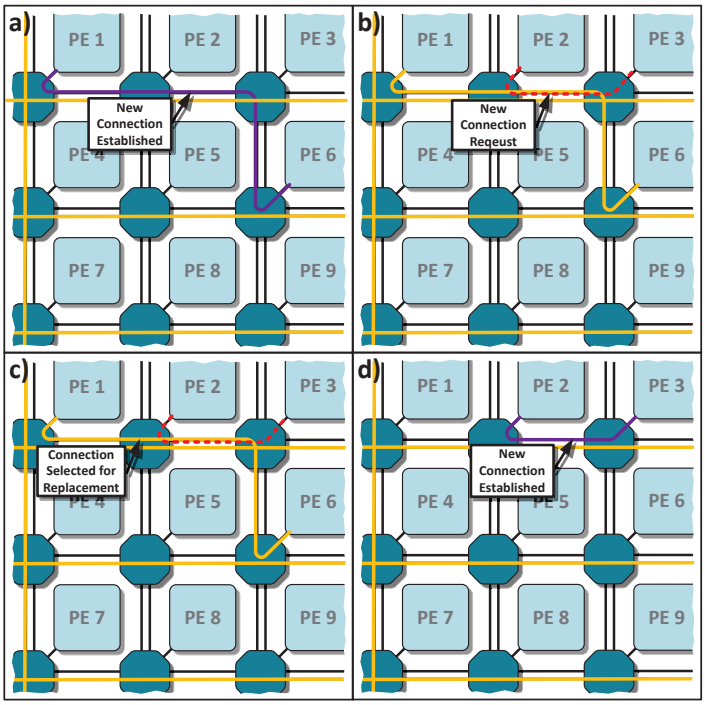


Figure 7.6.: Replacement of GS connections without QoS requirements – (a) New replaceable connection is established, (b) Another connection is requested to be established, (c) Existing connection is selected for replacement, due to low bandwidth utilization, (d) New connection can be established successfully due to replacement.

7.2.1.2. Auto-GS

Auto-GS is implemented in the network adapter, which was introduced in section 3.2.2.1. Thus, it is transparent from the NoC router perspective. Because the NA is not in the scope of this work, Auto-GS is briefly discussed. It is a collaborative work that arose from the InvasIC project and was published in [ZHW+13][11]. Auto-GS necessitates GS connections, introduced in section 6.2.

In contrast to connection replacement, Auto-GS is a pessimistic self-optimization strategy for best-effort communication. The network adapter monitors the com-

[11] In the work in hand, only those aspects of Auto-GS are discussed, which have been built and evaluated by the author of this book. Essentially, this is the general concept of Auto-GS as well as the evaluation by the use of the simulation framework, introduced in section 5.2.

munication behavior of the connected tile in order to obtain temporal locality of communication requests. If it detects frequent communication with a specific node, a GS connection is automatically set up to this tile. Such a frequently addressed tile is referred to as a hotspot from now on. BE communication to a hotspot node is subsequently converted transparently into GS communication by the NA at the source. Other communication partners, which are accessed less frequently, are still served with BE communication. These nodes are referred to as coldspots hereafter. A monitoring system is utilized within the NA to capture the amount of data transferred to the most frequently addressed destination nodes. These destination nodes are stored in a list termed *communication history*. After each monitoring period[12], the destination nodes with the lowest data volume are replaced in the communication history. This replacement strategy enables to use a limited number of history entries and monitors for observation of the most frequently accessed nodes. The communication history is processed after each monitoring cycle by the network adapter in order to update the *connection list*[13]. The connection list contains the communication flows which use GS communication. These are the hotspot destination nodes. If this list changes, GS connections to nodes, which are dropped from the *connection list*, are released. In addition, new GS connection to novel nodes, which are added to the list, are established. A parameter, named max_dist, can be used to limit the distance between nodes, contained in the connection list. This prevents the establishment of long distant GS connections, which would consume an abundance of communication resources.

A typical Auto-GS example is provided in figure 7.7. The network adapter attached to the source router at *PE 1* serves three communication flows to *PE 5*, *PE 6* and *PE 7*. Two of these communication flows have a high communication volume, thus they are treated as hotspot nodes and stored in the *connection list* as hotspot nodes. GS connections are established to these nodes to reduce the communication latency, protocol overhead and consequently the power consumption for communication. The third communication flow to *PE 7* has a low throughput, measured by the monitoring system within the source network adapter. Consequently, it is not contained in the connection list and treated as a coldspot using best-effort packet switching communication.

7.2.2. Evaluation

The connection replacement and Auto-GS concept have been evaluated using the simulation framework, introduced in section 5.2. Therefore, the framework was extended to support both self-optimization strategies. A 10x10 meshed NoC with XY routing and four virtual channels is used for evaluation. $VC_{GS,max}$ was

[12]The length of the monitoring period can be configured at run-time according to the requirements.
[13]The *connection list* is typically smaller than the *communication history*.

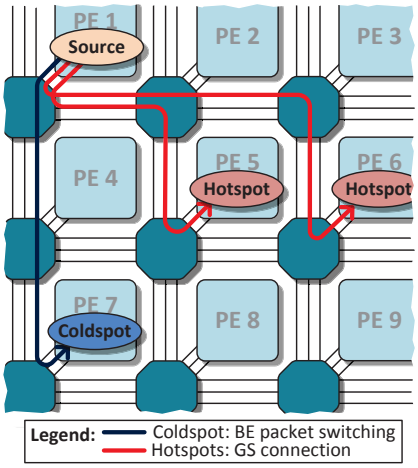


Figure 7.7.: Auto-GS is automatically setting up GS connections to frequently addressed nodes (hotspots) [ZHW+13].

set to 3, according to equation 6.8, in order to limit the resources occupied by GS connections. The most important results are summarized subsequently.

7.2.2.1. Connection Replacement

The connection replacement strategy shall replace GS connections with low bandwidth requirements and no QoS demands (*replaceable* flag set) by BE communication. This strategy was investigated by establishing GS connections with variable injection rates between arbitrary nodes. This is done to evaluate, whether the strategy is capable of replacing the connections with low injection rates when the available communication resources become scarce.

Figure 7.8 shows a comparison between a NoC with connection replacement enabled (*Repl.*) and a reference NoC (*Ref.*) without the self-optimization strategy. The number of established GS connections are measured for different numbers of requested connections. The connections with *high* and *low* bandwidth demands are measured separately[14].

For a low number of requested connections, all high and low bandwidth connections can be successfully established when adequate resources are available, as shown in figure 7.8. Once the number of requested GS connections increases,

[14]Half of the established connections have a low injection rate, which is lower than 0.15 *Flits/Cycle/Node*, the rest of the connections has *high* bandwidth demands with an injection rate greater than 0.15 *Flits/Cycle/Node*.

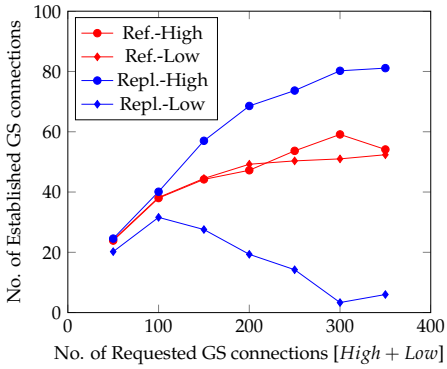


Figure 7.8.: Established GS connections with *high* and *low* bandwidth demands for different numbers of requested GS connections. A NoC with connection replacement (*Repl.*) and a reference (*Ref.*) are compared [Lu13].

the replacement strategy ensures that the connections with high bandwidth requirements can be established and remain established, whereas connections with low bandwidth requirements are replaced by BE communication. In contrast, the number of high and low bandwidth GS connections remains equal in the reference NoC (*Ref.*) for an increasing number of requested connections. Thus, in the reference NoC, more virtual channels are occupied by connections with low bandwidth requirements. The results show that connection replacement assists in improving the utilization of the VCs because they are allocated for frequently used GS connections. This increases the amount of GS communication, which has in turn a positive impact on communication latency and power consumption. This positive impact is evaluated in more detailed for the Auto-GS concept, which also aims to increase the amount of GS communication.

Additional performance evaluations for the replacement self-optimization strategy are presented in [Lu13]. The synthesis results, presented in [Lu13], investigate the area overhead and additional power consumption for the implementation of the replacement strategy. The results were obtained for the widely used *TSMC* 45 nm standard cell implementation. They show that the replacement self-optimization strategy raises the area requirements of a 5-port router by 10.7 % and the power consumption by 23.5 %, with clock frequency not being impacted. The monitoring system of the replacement implementation is the main reason for the increased area and power consumption. It must be optimized in order to improve the efficiency of the strategy.

7.2.2.2. Auto-GS

The evaluation of the Auto-GS concept was performed by the use of the simulation framework, introduced in section 5.2. The number of entries in the *communication history* table is set to 8 for the following evaluations. Three GS connections can be established in parallel according to $VC_{GS,max}$. Thus, the *connection list* has three entries for the following investigations. The investigated scenarios assume pure best-effort communication from the application perspective. All established GS connections are set up transparently by the Auto-GS hardware and thus marked as replaceable. A *reference* architecture without Auto-GS support is compared to the *Auto-GS* implementation later on. A protocol overhead of 1 flit per packet is assumed for BE packet switching communication[15]. GS connections have no protocol overhead once the connection is established, but necessitate one flit for connection setup and one for release.

Synthetic traffic is used in the first scenario. A predefined number of communication partners per node, with a maximum distance of 4 hops, is assumed for the synthetic scenario[16]. From the software perspective, all communication flows seem to use BE communication, with Auto-GS being transparent. Figure 7.9(a) shows the number of BE flits in the network with respect to the total number of flits. The number of communication partners per node varies. As predicted, the Auto-GS design reduces the share of BE communication significantly, compared to the reference, which only utilizes BE communication. With Auto-GS, the amount of BE flits varies between 20 % and 60 %. When the number of communication partners is increased, the amount of BE flits increases as well. This is due to the limited number of connections that can be established per node.

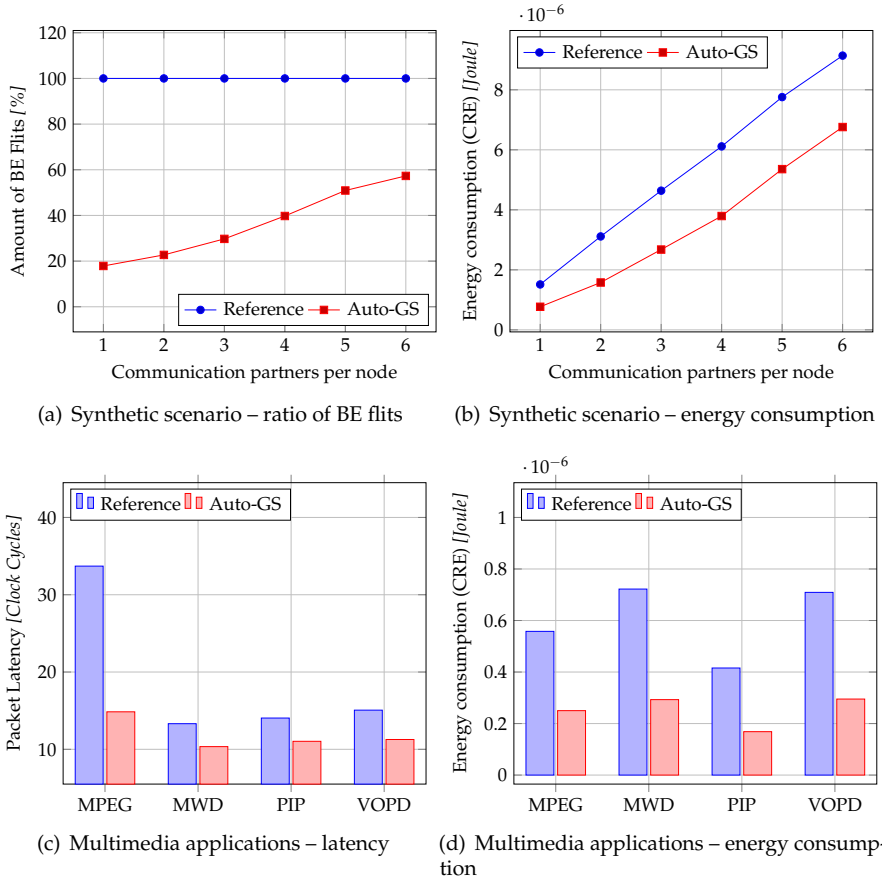
The *communication related energy (CRE)* consumption is introduced for power analysis. CRE is defined as the energy consumption that is directly related to data transmission. For this analysis, the energy consumption of a NoC router ASIC implementation is measured, as described in section 5.3.2.1. The CRE is calculated by subtracting the energy consumption of an idle router and a router under load[17]. Figure 7.9(b) shows the energy analysis results. Having only one communication partner per node, Auto-GS can save up to 50 % of CRE. In case of 6 communication partners per node, the Auto-GS implementation can reduce the CRE by 26 %. The energy saving of the Auto-GS technique relies on the fact that the amount of best-effort data transmissions is reduced. Best-effort communication consumes

[15] This is a valid assumption, since additional information, such as the destination network address contained in the head flit, must be included in each packet.

[16] This assumption of a maximum distance refers to the mapping algorithms, which minimize the distance between communicating nodes in order to reduce communication cost.

[17] Focusing on the CRE is reasonable because it represents the energy, which is affected by the Auto-GS concept. Static energy consumption of the router, which is 7.5mW for the used one, is independent from Auto-GS because no hardware modification is required in the NoC routers. In addition, static energy consumption can be reduced by techniques, such as power-gating or voltage scaling.

more power compared to GS data transmissions due to the protocol overhead and the more complex processing inside the routers[18].



(a) Synthetic scenario – ratio of BE flits

(b) Synthetic scenario – energy consumption

(c) Multimedia applications – latency

(d) Multimedia applications – energy consumption

Figure 7.9.: Simulation-based evaluation results for Auto-GS concept: (a) Amount of BE flits for a synthetic scenario, (b) Communication related energy consumption for synthetic scenario (10^5 cycles), (c) Average packet latency for video processing applications, (d) Communication related energy consumption for video processing applications (10^5 cycles) [ZHW+13].

[18]For each BE packet, a routing and reservation process is required in each router. These pipeline stages become obsolete for GS communication where only body flits are used after connection setup, as described in detail in section 5.1.3.

In addition to the synthetic traffic scenario, four video processing applications (MPEG, MWD, PIP, VOPD) from section 5.2.3.3 are used for investigation. Therefore, each application is mapped separately to the center of the mesh.

Figure 7.9(c) shows the average flit latency for the four applications. All applications profit from the Auto-GS self-optimization. The reason for the latency reduction is the use of GS connections and the accompanied lower processing latency within each router. VOPD reduces its latency by 25 % in Auto-GS configuration, as compared to the reference. The latency of MPEG application could be significantly reduced, because it operates near the saturation point of the NoC.

Figure 7.9(d) shows the energy perspective for the applications for a measurement period of 10^5 cycles. If Auto-GS is used, all applications reduce their communication related energy consumption in the NoC by at least 50 %. Again, this is because the reduced protocol overhead and the lower number of pipeline stages for GS communication are lowering the switching activities and power usage.

A more detailed evaluation of the Auto-GS concept is presented in [ZHW+13]. No modification of the NoC routers is required for Auto-GS support, thus there are no synthesis results presented now. However, the implementation within the network adapter, as well as the overhead, are discussed in [ZHW+13].

The *connection replacement* and *Auto-GS* self-optimization strategies, presented and evaluated in this section, show clear benefits from the communication perspective. Increasing the number of GS connections by the use of hardware-based self-optimization can assist in improving communication latency, NoC utilization and power consumption. Applications without QoS demands can profit from GS communication, if resources are available. Applications with QoS demands are not affected by the self-optimization strategies[19].

7.3. Adaptive Data Collection

Large scale many-core architectures, as the InvasIC architecture from section 3.2 or the Thousand Core Chips discussed in [32], must be managed in a decentralized way to ensure scalability. Distributed resource management schemes, as the one discussed in section 3.3.2.1, enable decentralized management of hardware resources. In such schemes, multiple operating system instances manage the hardware resources of different regions of the architecture. The regions are defined during run-time, depending on the resource requirements of the applications. In order to allow each OS instance to make appropriate decisions, information regarding the status of the system, within the respective region, is required [66].

[19] By the use of the *replaceable* flag it is ensured that GS connections of applications with QoS demands are not affected by the optimizations in order to preserve hard guarantees.

This status data is typically distributed among the region and must be collected on demand, or at fixed time-intervals. Typical examples for operating system mechanisms, which require status information of the system, are task mapping, resource allocation or load balancing.

In [7], a distributed application mapping scheme for NoC-based architectures is proposed. It collects communication and computation monitoring data, in order to make proper application mapping decisions. Their results show that 0.1-500 MByte of data must be collected, in total to map an application with 10 to 500 tasks to an architecture with 64x64 nodes. Another example for collection of status information is the game-theoretic approach for frequency adjusting by Puschini et al. [208]. It is an energy optimization technique that requires temperature-monitoring-data of all processing elements of a region, to reach an equilibrium.

However, data collection is not only necessary for management of parallel architectures, parallel applications [16, 14, 186] also must collect data, e.g. computation results. Consequently, efficient mechanisms are necessary in order to reduce the communication overhead for data collection in NoC-based architectures.

Data collection can be implemented efficiently by the use of 1-to-M and M-to-1 communication, where M stands for many. The efficient support of communication patterns with a single sender and multiple receivers and vice versa, are objects of ongoing research. Broadcasting and multicasting algorithms have been proposed and investigated. Such algorithms are currently well established in off-chip networks [45], [233] and have also become a research topic for on-chip networks. Essentially, two different approaches for multicast communication in networks exist – path-based and tree-based.

In path-based multicasting and broadcasting schemes, all destination nodes are visited after each other. In [161], Lin et. al describe a heuristic for path-based multicast algorithm named sorted multicast path. It is based on the Hamilton cycle[20], also known as Hamiltonian cycle. A Hamilton cycle is constructed for the entire network and used for routing. In [8], the network nodes are labeled with the position in the Hamilton path[21]. After the labeling, the network is divided into two disjoint acyclic networks, guaranteeing the absence of cyclic dependencies as well as deadlock-freedom. However, in both existing path-based algorithms every node must be addressed. The Hamilton path, or cycle, is defined at design-time. Run-time adaptation and multicast, as well as M-to-1 data collection, are not supported.

[20] A Hamilton cycle describes a path which visits every node in a network or region exactly once and ends up in the same node where it began.

[21] In contrast to a Hamilton cycle, a Hamilton path does not end in the source node and therefore does not form a cycle.

Other broadcast and multicast approaches have a tree-shape, where a message is duplicated in certain nodes. In [119], two tree-based multicast algorithms are presented, which attempt to minimize the number of occupied links or the latency. In [123], a virtual circuit tree is created which connects all members of the multicast. In these tree-based approaches, tables that identify multicast connections are stored in each router, significantly increasing its implementation costs. The ability to address a certain region for tree-based multicast includes high protocol overhead. The tree must be constructed in the source node and the address of every single node must be encoded in the packet header. In [143], an architecture for 1-to-M and M-to-1 traffic is presented. However, the presented concept focuses on the performance of cache coherence protocols.

Most of the existing work focuses on dedicated applications for broadcast and 1-to-M communication. M-to-1 communication and data collection is only addressed marginally. Existing mechanism are either adaptive at run-time or are not suitable for on-chip networks, due to their complexity. Region-based resource management, which is promising with respect to scalability, is not addressed. Hence, flexible and adaptive data collection mechanisms for regions that are defined during run-time, are introduced and investigated in [HWZ+13][22]. An overview on these mechanisms, as well as a discussion of their implementation and evaluation is presented in the following sections. The proposed mechanisms also enable run-time adaptive multicast communication, as described in section 7.3.2.1.

7.3.1. Concepts for Region-based Data Collection

A promising concept for management of large many-core architectures is to divide them into regions that are managed separately. Such a management scheme is also proposed for the InvasIC architecture, as described in section 3.3.2.1. The number and shape of the regions is defined during run-time, depending on the current requirements. The operating system instance, managing one region, must collect status information from the associated cores. Depending on the algorithm, the executed parallel applications may also be required to aggregate data. As such, a run-time adaptive data collection mechanism will now be introduce. It enables to define the shape[23] and position of the region at run-time.

An example for such a region, where data must be collected, is provided in figure 7.10. In this example, one core (*C*) must collect data from the other nodes of the marked region. Henceforth, it is assume that data must be collected from all the nodes of the region. Existing masking schemes, as [161], can be used when

[22] Extracts from [HWZ+13], which are my literary and intellectual property, are used verbatim in this section without identification.

[23] Currently only foursquare regions in mesh-based NoCs are supported. However, the mechanisms could be extended to support arbitrary shapes.

a subset of the nodes in the region must be addressed by the request. A square region is defined by four parameters, relative to the X and Y coordinate of the node that is initiating the request: *Size x* and *size y* define the size of the region. The size is equal for all cores of the same region[24]. *Offset x* and *offset y* provide the offset of the requesting node relative to the region's upper left border. The offset represents the position of the requesting node within the defined region. Thus, each core of the same region has a different offset.

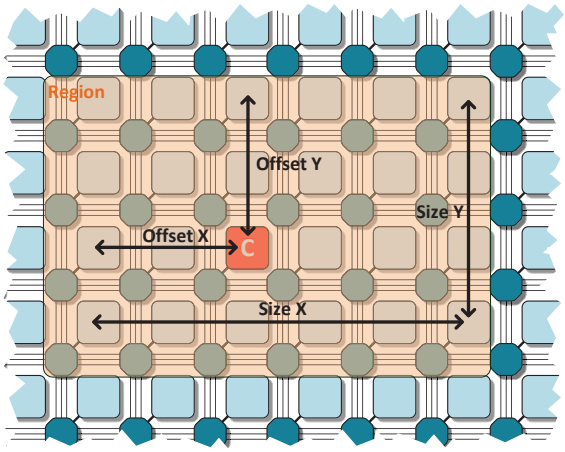


Figure 7.10.: Partial view of a NoC-based architecture: one core (red) requesting for data collection in a run-time defined region.

Three different mechanisms for data collection in such regions, which are defined at run-time, are proposed in [HWZ+13] and discussed now. A schematically representation of these schemes is provided in figure 7.11. Each example shows a region of a tile-based architecture.

Mechanism (a) in figure 7.11 represents a method to collect data using only point-to-point communication with a separate request being transmitted to each node. The nodes respond by sending the requested information. This mechanism is named *request-and-response (RaR)* from now on. The RaR mechanism can be used without additional hardware, in a meshed NoC, by implementing it in software. Load operations can be used in non-uniform memory access architectures to access the required data from the other nodes of the region. However, these load operations are typically implemented in a blocking way. Thus, the requests are processed one after another. The sequential processing of the requests results in

[24]For the mechanisms, using a Hamilton cycle (round-trip and mixed), at least one of the region's parameters *size x* (*size.x*) and *size y* (*size.y*) must be even and both must be greater than one in order to establish a Hamilton cycle successfully [276].

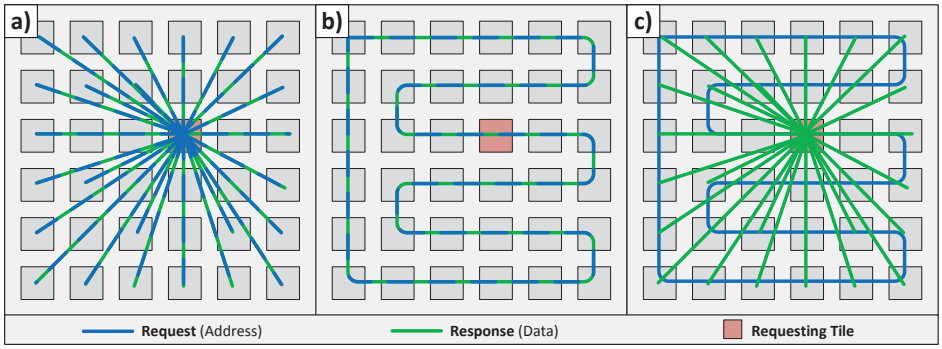


Figure 7.11.: Three data collection types: (a) request-and-response, (b) round-trip, (c) mixed.

a huge delay. Hence, a hardware implementation of the RaR mechanism is also investigated. The hardware implementation transmits the requests continuously, while greatly decreasing the delay. Existing routing algorithms, such as XY routing or odd-even turn, can be used for request-and-response data collection. The communication cost C for RaR and a region with dimensions $size.x$ and $size.y$ can generally be calculated as follows:

$$C_{RaR} = \sum_{i=0}^{size.x \cdot size.y-1} S_{request}(i) + \sum_{i=0}^{size.x \cdot size.y-1} S_{response}(i) \tag{7.2}$$

$S_{request}(i)$ and $S_{response}(i)$ are the packet sizes for the request and response messages respectively. However, assuming a quadratic region[25] where the requesting node is located in the center of the regions and where $S_{request}$ and $S_{response}$ are equal for all cores, equation 7.2 can be approximated as follows:

$$C_{RaR,quad} \approx N^2 \cdot \frac{N}{2} S_{request} + N^2 \cdot \frac{N}{2} S_{response} = \frac{N^3}{2}(S_{request} + S_{response}) \tag{7.3}$$

The mechanism in figure 7.11(b) shows a technique named *round-trip (RT)* as from now. A Hamilton cycle [276] is used to transfer the request message as well as the requested data of the addressed nodes in the respective region. Each node places the requested data at a predefined position in the *round-trip packet (RTP)*.

[25] A quadratic region with a diameter of N is defined as follows: $size.x = size.y = N$.

Figure 7.12 shows a typical RTP. The packet size S_{RTP} of a round-trip packet can be calculated as follows:

$$S_{RTP} = \lceil \frac{D_{size}(size.x \cdot size.y) + O_{packet}}{W_{link}} \rceil \tag{7.4}$$

D_{size} is the size of the data to be collected per node and O_{packet} is the protocol overhead per RTP, which is later described in detail. W_{link} is the width of a link, or flit, respectively and is defined at design-time of the NoC. $size.x$ and $size.y$ define the region's size. Using equation 7.4, the communication cost $C_{RT,Quad}$ for a round-trip data collection in a quadratic region with $N \times N$ cores can be approximated as follows:

$$C_{RT,quad} \approx N^2 \cdot S_{RTP} = N^2 \lceil \frac{D_{size} \cdot N^2 + O_{packet}}{W_{link}} \rceil \tag{7.5}$$

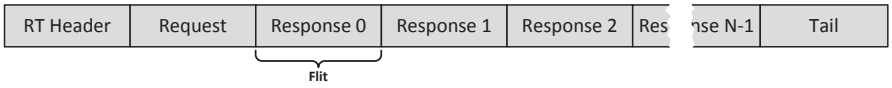


Figure 7.12.: Round-trip packet including a flit indicating the type of request and one flit[26]per responding node.

In existing work [8], Hamilton paths are defined statically at design-time. The proposed round-trip mechanism uses a novel routing algorithm, introduced in section 7.3.2, to adapt the Hamilton cycle at run-time to any square region.

Mechanism (c) in figure 7.11 represents a combination of the previous two strategies. Hence, it is referred to as *mixed* as from now. The request is transmitted on a Hamilton cycle[27] and the response is transferred directly from each node of the region to the requesting node using unicast or point-to-point communication. Compared to request-and-response, the mixed mechanism reduces the NoC utilization by using one Hamilton path for the requests instead of multiple request packets to each node. Compared to the round-trip mechanism, the response takes a direct path to its destination. The communication cost $C_{mixed,quad}$ for a quadratic region with $N \times N$ cores, using mixed data collection, can be approximated as follows:

$$C_{mixed,quad} \approx N^2 \cdot S_{request} + \frac{N^3}{2} S_{response} \tag{7.6}$$

[26]Depending on the data size D_{size} to be collected, multiple flits are required per node or data from multiple nodes fit into one flit. However, in figure 7.12 D_{size} is equal to the size of a flit.
[27]A round-trip packet is used by the mixed mechanism to transmit the request message.

7.3.1.1. Deadlocks

The proposed Hamilton path routing uses all possible turns in a meshed network. As described in section 2.4.5.5, this could lead to deadlocks in wormhole switched NoCs, if different Hamilton cycles overlap. Such a situation is shown in figure 7.13(a). The two overlapping Hamilton routing packets *A* and *B* block each other and can no longer be forwarded. This situation only occurs if opposed turns are used by the two packets. Again, this depends on the position of the requesting node, the size and the offset of the region. Packets using the same Hamilton cycle cannot lead to deadlocks as long as the following constraints are satisfied by the NoC design:

- The flits of one Hamilton packet fit completely into the buffer of one router.
- Each node using a Hamilton cycle restricts the number of outstanding packets to 1.

Neither synchronization nor arbitration is required when using Hamilton routing, if the previously stated rules are fulfilled.

In order to prevent deadlocks between Hamilton packets, overlapping of different Hamilton cycles must be avoided. This can be done by dividing the architecture into non-overlapping regions as previously proposed. In each region, a Hamilton path can be used by all cores in parallel as long as the previous rules are fulfilled. The regions can certainly be defined and changed at run-time. If overlapping cannot be prevented, different dedicated virtual channels can be used to prevent deadlocks. Thus, a pinning of packets to VCs would be required.

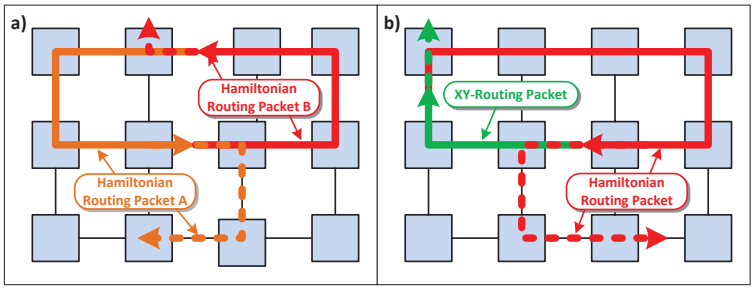


Figure 7.13.: Deadlock situations for Hamilton path routing: (a) deadlock between overlapping Hamilton routing packets and (b) deadlock between a XY and a Hamilton routing packet.

The mixed mechanism combines Hamilton path routing and point-to-point communication using XY routing. XY routing only uses 4 of the 8 possible turns in a meshed NoC to avoid deadlocks. Hence, it is deadlock-free by definition [185].

However, in combination with Hamilton packets, deadlocks can occur, as shown in figure 7.13(b).

In order to avoid deadlocks between Hamilton and XY routed packets, they must be separated from each other. One possibility would be to use an independent NoC for Hamilton routing. Because this leads to a huge implementation overhead, another strategy is preferable: One or several virtual channels can be designated for Hamilton packets, similar to the strategy used for deadlock prevention in fully adaptive routing schemes [59]; described in section 2.4.5. Because the VCs are independent of each other, Hamilton packets using a separate VC cannot interfere with XY routed packets. This strategy is used for the concept, presented in the following paragraphs. Therefore, the VC reservation process must be modified to take this type of packet (XY or Hamilton) into account.

7.3.1.2. Node-internal Data Processing

For the discussed data collection mechanisms three steps of node-internal data processing can be identified as part of a data collection phase:

1. Request message generation at the node initializing the request
2. Fetching the data inside the node receiving the request
3. Processing the incoming data inside the requesting node

These phases are nearly independent of the mechanism used for communication. Hence, they are briefly discussed, for clarity reasons.

Initializing the data collection request for the request-and-response software implementation is trivial. Load operations can be used to access the required data in a distributed shared memory architecture, as introduced in section 3.2. In contrast to the request-and-response software implementation, all other mechanisms use hardware support. Memory mapped registers trigger the network adapter to generate a round-trip packet or to trigger the generation of request packets in case of a RaR hardware implementation.

The generated requests arrive at a destination node, typically in the form of memory access requests. The read address of the memory access is part of the request message payload. According to the request, the data are fetched from their memory location, or from memory mapped register. Memory mapped registers are e.g. used when collecting the monitoring information of the NoC, described in section 5.5.3. Single cycle memory accesses are assumed for evaluation, but all mechanism are also applicable in case of multi-cycle access latencies.

For round-trip and the mixed mechanism the requested data are returned to the requesting node as separate packets. The RT mechanism inserts the data into the

round-trip packet, shown in figure 7.12. Therefore, a special data insertion unit is used, which will be introduced in section 7.3.2.

When the requested data arrives at the destination node, a message buffer or dedicated registers can be used to store the incoming data and make it accessible. As an alternative, pre-allocated memory could be used to store the incoming data. For the RaR software implementation using load operations, processor registers acquire the requested data.

7.3.2. Implementation

The request-and-response data collection mechanism relies on point-to-point connections which are typically supported by every NoC, such as the one introduced in chapter 5. Thus, no hardware extension is required in the routers to support RaR. However, round-trip and mixed mechanism necessitate to transmit packets on a Hamilton cycle. Therefore, the routing unit must be extended by an appropriate routing algorithm, as described in detail in section 7.3.2.1. The round-trip mechanism requires an additional component named *insertion unit*, to insert the requested data at their corresponding position into the round-trip packet. Figure 7.14 shows the router, introduced in section 5.1, with additional components required for *adaptive Hamilton routing (AHR)* and data insertion into round-trip packet. In order to select AHR for routing, the packet protocol must be extended by the *AHR enable* flag, as described in section 7.3.2.1. It defines which routing algorithm will be used within the routers. The same flag is used to enable the insertion unit at the router's output port. The reservation table must be modified slightly to assign RT packets to a dedicated VC in order to avoid deadlocks. The AHR enable flag is used to trigger reservation of the dedicated VC in the reservation table.

7.3.2.1. Adaptive Hamilton Routing

An *adaptive Hamilton routing (AHR)* algorithm is used to enable run-time adaptive multicast, which is required for round-trip and mixed data collection. As explained in section 7.3.1, the proposed concept assumes regions that are specified by their size and their offset, relative to the position of the requesting node. This information (size and offset) is transmitted in the head flit, shown in figure 7.15[28]. The *AHR enable* flag is used to activate AHR and the Insertion Unit. The AHR

[28] GS support is not taken into account for the head flit shown in figure 7.15. A *BE/GS* bit is inserted in front of the AHR enable flag in order to support QoS concept, introduced in section 6.2.

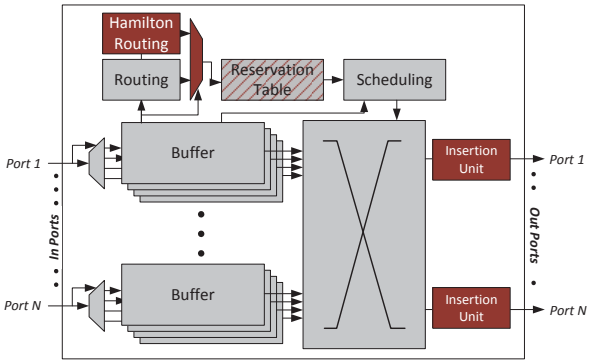


Figure 7.14.: Packet switching router from section 5.1 extended for round-trip data collection by an adaptive Hamilton routing module and an insertion unit (additional and modified components are highlighted).

algorithm then processes the fields, size and offset, which are only included if the AHR enable flag is set[29].

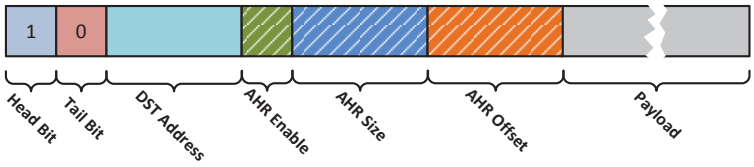


Figure 7.15.: Head flit with extended network layer protocol. Additional fields are used for adaptive Hamilton routing (shaded).

The pseudo code of the adaptive Hamilton routing is provided as algorithm 7.1. It is a distributed algorithm calculating the output port at each router, which will be used by the AHR packet. The algorithm uses the destination address, the size and the offset as input. These data are included in the AHR header. The given algorithm uses this information to calculate the output port. The adaptivity is enabled by the size and offset fields. They are used by the AHR algorithm to calculate the relative coordinates. This adaptivity results in slightly increased implementation costs, as investigated in section 7.3.3.2.

Algorithm 7.1 will only result in a Hamilton cycle if the following conditions are fulfilled:

1. Dimensions of the regions (*size.x* and *size.y*) must be greater than one.

[29] As an implementation alternative, the size and offset could be stored in each router within the respective region. This can reduce the protocol overhead in case of frequent AHR transmissions.

2. One dimension must to be even ($size.x$ for the given algorithm).
3. The requesting node must be within the region ($\Rightarrow offset < size$).

If these conditions are fulfilled, a Hamilton cycle, as illustrated in figure 7.11(b), is made by the packet. The adherence of the given conditions is carried out by the node generating the packet. The pathway, resulting from the proposed algorithm, depends on the position of the initiating node and the parameters $size$ and $offset$.

Algorithm 7.1 Pseudo code of the adaptive Hamilton routing algorithm.

```
 1: function adaptive_hamilton(req, size, offset)
 2: reg.n ← 0, reg.w ← 0 {Def. region}
 3: reg.e ← size.x − 1, reg.s ← size.y − 1
 4: rel.x ← this.x + offset.x − req.x {Def. relative coords}
 5: rel.y ← this.y + offset.y − req.y
 6: if (req.inp!=LOCAL)&&(this.id==req.id) then
 7:     return LOCAL
 8: end if
 9: if rel.x==reg.w then
10:     if rel.y==reg.n then
11:         return EAST
12:     else
13:         return NORTH
14:     end if
15: else if rel.x==reg.e then
16:     if (rel.y%2)==reg.n then
17:         return SOUTH
18:     else
19:         return WEST
20:     end if
21: else
22:     if (rel.y%2)==reg.n then
23:         return EAST
24:     else
25:         if (rel.x==(reg.w + 1))&&(rel.y!=reg.s) then
26:             return SOUTH
27:         else
28:             return WEST
29:         end if
30:     end if
31: end if
32: end function
```

Run-time adaptive multicast Multicast communication can be used for various purposes, such as cache-coherency or synchronization. A wide range of on-chip communication scenarios that benefit from hardware multicast support, is provided in [123]. The *adaptive Hamilton routing* algorithm, introduced previously, is used by the round-trip and mixed mechanisms to distribute the data collection requests efficiently in the defined region. Thus, it enables multicast or 1-M communication in the respective regions. It is the first adaptive multicast algorithm using Hamilton cycles, which is suitable for a hardware implementation. Compared to an existing tree-based multicast scheme for 1-M communication [143], it

reduces the implementation complexity of the routers[30]. The application of the AHR algorithm for multicast communication is straight forward. However, pure multicast usage has not yet been investigated in detail.

7.3.2.2. Insertion Unit

The *insertion unit* is used to insert the requested data in a round-trip packet. It is activated when an RTP header passes a router output port. Subsequently, it calculates the flit number for data insertion into the round-trip packet, shown in figure 7.12. First, the relative coordinates $rel.x$ and $rel.y$ are calculated by equation 7.7 and 7.8. Therefore, the offset, which is encoded in the round-trip packet header, is used. The parameter *this* represents the coordinates of the current router and is defined at design-time. The variable *req* represents the coordinates of the requesting node, which are provided in the form of a destination address in the RTP header.

$$rel.x = this.x + offset.x - req.x \tag{7.7}$$

$$rel.y = this.y + offset.y - req.y \tag{7.8}$$

The relative coordinates are used to calculate the flit number using equation 7.9 and the $size.x$, which is also part of the RTP header. D_{size} is the size of the data to be collected from each node and W_{link} is the size of a link or flit, known at design-time.

$$F_{number} = \lfloor (rel.y \cdot size.x + rel.x) \cdot \frac{D_{size}}{W_{link}} \rfloor \tag{7.9}$$

The insertion unit counts the past flits of the packet. Once the flit with the calculated number passes the router output port, the requested data are inserted. After the tail flit of the round-trip packet has passed the router output port, the insertion unit returns to an idle state and waits for the next RTP.

If the requested data cannot be accessed within one cycle, the transmission of the response flits of RTP may be delayed according to the access latency. The number of delay cycles in each node depends on the position for data insertion. If the flit number F_{number} is greater than the access latency, no additional waiting time is

[30] Packet duplication, required for tree-based multicast, comes along with a noticeable implementation overhead inside the NoC routers.

required[31]. For the results presented in the following section, single cycle access is assumed.

7.3.2.3. Request-and-Response Hardware Implementation

The request-and-response hardware mechanism requires a module that generates the request messages within the issuing node. In order to enable the same flexibility as for the other mechanisms, the used module must be located in each tile. It uses the same inputs (*size* and $offset$) as the adaptive Hamilton routing algorithm. The destination addresses of all nodes, which are part of the region, can be calculated using their own coordinates, as well as *size* and $offset$ of the region. After calculation of the destination addresses, the request messages can be generated and continuously transmitted. The RaR hardware implementation is not within the scope of this work, since the components are not located in the NoC itself. Thus, it is not discussed in detail here.

7

7.3.3. Evaluation

The three data collection mechanisms are evaluated with respect to communication overhead and implementation costs in the following paragraphs.

7.3.3.1. Performance

The simulation framework, introduced in section 5.2, is employed for performance and communication cost evaluation. A NoC with eight VCs and a link size of 128 bit was chosen, referring to the parameters of the NoC used in Intel's Single-chip Cloud Computer [113], described in detail in section 2.2.5.2.

For the following investigations, a NoC with 32x32 nodes was instantiated, having nearly a thousand cores, as assumed in [33]. Square regions, of different sizes, are defined for collecting data from all nodes within the region. The size of the data to be collected per node is assumed to be 32 bit first, but also changed later.

Depending on the application, where the data collection mechanism is used, the time until the data arrives at the requesting node may be an important factor. Hence, the latency for collecting data within square regions of sizes from 4 to 144 nodes was analyzed. Figure 7.16(a) shows the analysis results for a scenario

[31] A trivial implementation of the round-trip mechanism for multi-cycle access latencies would insert a corresponding number of empty flits between the *request flit* and the first *response flit* of the RTP shown in figure 7.12.

where the requesting node is located in the center of the region[32]. The presented results are always relative to the RaR hardware implementation, which serves as a baseline. For small region sizes, round-trip has the lowest delay. The reason is that the local router port at the requesting node is a bottleneck. RT has the lower link utilization, as investigated later and is thus least affected by said bottleneck. The mixed mechanism has a worse delay than RT, but performs better than RaR hardware implementation up to a region size of 16 nodes[33]. For region sizes of 36 or more cores, the request-and-response hardware implementation has the lowest delay. The delay of the RaR software mechanism is orders of magnitude greater than for other mechanisms, due to the sequential issuing of the requests via the software.

The same scenarios used previously for the latency analysis, is now employed to investigate the link utilization. The link utilization is defined as follows:

$$link\_utilization = packet\_size \cdot link\_cnt \tag{7.10}$$

The *packet_size* is given in flits. The number of traversed links resp. hops is denominated with *link_cnt*. The link utilization represents the accumulated bandwidth which is required for a transmission.

In figure 7.16(b), the relative link utilization is provided for different regions. The requesting node is located at the center of the region[34]. Round-trip has the lowest link utilization for regions of up to 16 nodes. For a region size of 16 nodes, round-trip reduces the link utilization by 54.8 %, compared to the request-and-response mechanism[35]. The mixed mechanism is best for regions, larger than 36 nodes. For a region of 12x12 nodes, request-and-response has a 74.8 % and round-trip a 132.8 % higher link utilization compared to the mixed mechanism.

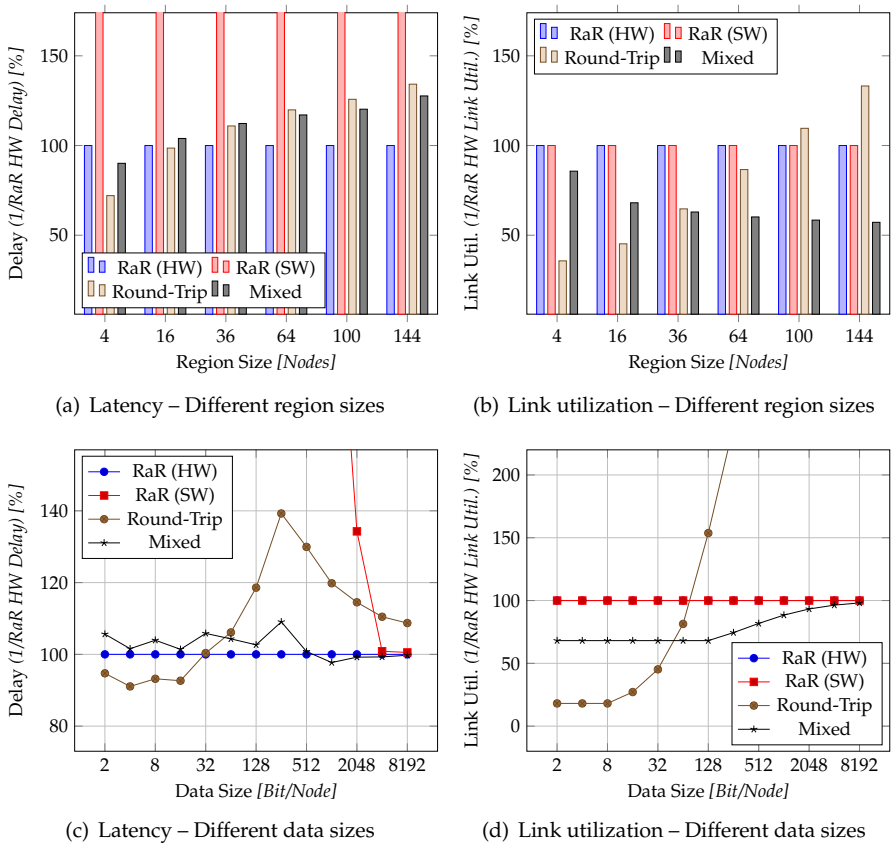
For the previous investigations, the collected data had a size of 32 bit per node. Depending on the application, the size of the data may be smaller (e.g. just a view status bits) or very large (e.g. complex data structures or huge sets of data). Hence, the impact of the data size is investigated for a region size of 4x4 nodes. The requests are again initiated by one core, located in the center of the region.

[32] In [HWZ+13] additional results are provided for latency and bandwidth utilization of the investigated mechanisms for a scenario where the requesting node is located at the upper left corner of the region. Except round-trip mechanism, all mechanisms show better results if the requesting node is located at the center of the region.

[33] This inflection point depends on the region size and the size of the collected data, as shown by equation 7.3, 7.5 and 7.6.

[34] Additional results are provided in [HWZ+13] for a scenario, where the requesting node is located in the upper left corner of the region. For request-and-response and mixed, the link utilization can be reduced by 40.0 % resp. 36.3 % (region size $10x10$), if the requesting node is located at the center of the region.

[35] The link utilization is equal for the RaR hardware and the software implementation.



(a) Latency – Different region sizes

(b) Link utilization – Different region sizes

(c) Latency – Different data sizes

(d) Link utilization – Different data sizes

Figure 7.16.: Latency and bandwidth utilization of the different data collection strategies for different regions sizes (a), (b) and variable data sizes per node (c), (d).

Figure 7.16(c) shows the latency for collecting data of different size with the investigated mechanisms. The link utilization, caused by data collection, is investigated in figure 7.16(d). For small data sizes of 32 bit or less, round-trip provides the best results because the collected data of several tiles can be transmitted in the same flit when the payload per node is small. The dependency between payload size and packet size of the RTP is provided by equation 7.4. In contrast, large data sets in round-trip packets lead to heavy link utilization. The position of the peak for round-trip data collection latency, shown in figure 7.16(c), depends on the region size and the size of the collected data sets. The latency of the mixed mechanism is similar to the request-and-response hardware implementation, as shown in

figure 7.16(c). For larger data sets, link utilization of the mixed mechanism and request-and-response[36] converge, as presented in figure 7.16(d).

Summary The previous analysis showed that the choice of the appropriate data collection mechanism strongly depends on the requirements. Table 7.2 gives an overview on the data collection strategies, which are best in terms of latency and link utilization under certain constraints. In the table, regions are denoted as small for sizes up to 16 nodes, larger regions are referred to as big. This meets the requirements of the applications, introduced in [22]. A data set is assumed to be small if 32 bit or less must be collected per node. This suits for status information, such as monitoring data, which are required by an OS for decision making.

Parameter	Size	Latency	Link Util.
Region	***small***	RT	RT
	large	RaR	mixed
Data Set	***small***	RT	RT
	large	mixed/RaR	mixed/RaR

Table 7.2.: Summarizing comparison of the investigated mechanisms.

Table 7.2 summarizes the evaluation. It shows that round-trip provides the best results with respect to latency and link utilization for small regions. For large regions, request-and-response provides the lowest latency. The mixed data collection achieves the lowest link utilization for large regions. If the data sets are much larger than 32 bit, the mixed mechanism and RT offer the best performance.

For nearly all the investigated cases, either the mixed mechanism or round-trip offer better performance compared to request-and-response hardware or software implementation. If the adaptive Hamilton routing algorithm is supported by the hardware, mixed and round-trip can be alternated. This enables a decision at run-time which mechanism to use, according to region size and data size, in order to combine the advantages of both strategies.

7.3.3.2. Synthesis

The router design, shown in figure 7.14, is based on the router, introduced in section 5.1. Therefore, the basic router design was extended by the adaptive Hamilton routing and the insertion unit. Subsequently, the implementation overhead was investigated by using the ASIC synthesis flow from section 5.3.2.1 and a *TSMC* 45 nm standard cell library.

[36] The link utilization of the RaR hardware and the software implementation is again equal.

The synthesis results of the ASIC implementation for a single 5-port router of a 10x10 mesh with different configurations according to the investigated data collection mechanisms are summarized in table 7.3. The router has 8 VCs per port, a buffer size of 8 flits and a link size of 128 bit. The router configuration is equal to the parameters of Intel's Single-chip Cloud Computer [112], introduced in section 2.2.5.2.

Router Configuration	**Area** $[\mu m^2]$	**Clk. Freq.** $[MHz]$	**Power** $[mW]$
Basic	246295	1200	14.3914
Simple Hamilton	246770 (+0.2 %)	1200	14.51 (+0.8 %)
Simple Hamilton+Ins.	249524 (+1.3 %)	1200	14.92 (+3.7 %)
Adapt. Hamilton	248997 (+1.1 %)	1150 (-4.2 %)	14.96 (+4.0 %)
Adapt. Hamilton+Ins.	252138 (+2.4 %)	1150 (-4.2 %)	15.15 (+5.2 %)

Table 7.3.: ASIC implementation overhead of non-adaptive (simple) and adaptive Hamilton routing for a single 5-port router.

The basic router in table 7.3 does not support Hamilton routing or data insertion and as such, can only be used for the request-and-response mechanism. The *simple Hamilton* routing unit does not support AHR and consequently no region-based data collection. It can only be used to collect data from all nodes of an architecture and only suffices for small architectures. The simple Hamilton implementation was carried out in [Spi12] and serves as a reference to investigate the overhead of the adaptivity, proposed by this work. Hamilton routing is required to support round-trip and mixed mechanism. Additionally, round-trip requires the insertion unit. In order to support the proposed region-based data collection, adaptive Hamilton routing was introduced. AHR has a more complex hardware implementation and affects the critical path of the router by 4.2 %[37]. Compared to the simple Hamilton routing, the area utilization of the router is increased by only 0.9 % for the adaptive version. The router with AHR support and an insertion unit has an area overhead of 2.4 % compared to the basic router and consumes 5.2 % additional power. It supports the mixed mechanism as well as the round-trip. This enables to select the used strategy at run-time depending on the requirements. In general, the implementation overhead of the proposed mechanisms is quite low for the ASIC implementation. Additional synthesis results for an FPGA implementation are provided in [HWZ+13].

In summary, the previous evaluations showed that the proposed mechanisms named round-trip and mixed outperform the straight forward implementation, named request-and-response, in most cases. The presented synthesis results

[37] The clock frequency reduction could be avoided by pipelining the adaptive Hamilton routing unit or by applying a route pre-computation technique [114].

exhibit the low implementation overhead. When adding support for round-trip and mixed data collection, the area utilization of a router is increased at maximum by 2.4 % and the power consumption by 5.2 % for an ASIC implementation.

7.4. Autonomous Power Management

Power efficiency and dark silicon are crucial in future many-core architectures that contain hundreds or even thousands of cores [32][38]. The network on chip of such a parallel architecture can consume a substantial amount of the total power budget [114]. Consequently, power saving techniques for NoCs have become an important research aspect [147], in order to manage a limited power budget.

A low-power NoC is proposed by Lee et al. in [153]. The authors applied various power saving techniques on the circuit level. Mainly, they focus on efficient coding techniques for data transmission, such as low-swing signaling and serial link coding. Energy-aware and performance-aware mapping and adaptive routing schemes for regular NoC topologies are discussed in [118]. The presented mapping algorithm targets design-time mapping of IP cores in order to minimize power consumption for communication. Centralized routing is applied to plan the communication flows of a distributed application, taking power consumption as well as performance into account. Another design-time power optimization flow for NoC-based architectures is presented in [228]. The authors propose to adapt the voltage level of each link at design-time to the communication requirements of the specific applications. The used algorithm takes task assignment, tile mapping, routing path allocation, task scheduling and link speed assignment into account. This work focuses on application-specific architectures where power-optimization can be planned at design-time. The run-time flexibility, with respect to changing communication requirements, is very limited in existing work. So-called *voltage-frequency islands (VFI)* for regular NoC topologies are introduced by Orgas et al. in [189, 190, 191]. The authors propose the division of a NoC at design-time into different static regions. Voltage and frequency of each region can be adapted dynamically, using dynamic voltage and frequency scaling. In [189], a design methodology for partitioning a NoC architecture into multiple VFIs and assigning supply and threshold voltage levels to each VFI is investigated. However, run-time adaptation is not taken into account. In contrast, run-time energy management is proposed in [191] and [190]. On-the-fly workload monitoring is used to control voltage in combination with frequency dynamically. In [173] a fine-grained power-gating scheme is proposed. This scheme utilizes 30 micro power domains per router to reduce the leakage power and energy overhead. In order to hide the weak-up latency, *look-ahead routing* is used. Each router deter-

[38] The invasive computing paradigm, introduced in section 3.1, addresses such systems explicitly.

mines the route of a packet, two hops in advance, enabling the early wake-up of required components. However, the use of look-ahead routing limits the choice of the routing algorithm. Thus, only static routing schemes can be used.

In the following sections, a scheme for fine-grained power gating of virtual channel buffers will be presented and investigated. In contrast to [173], it is not restricted to static routing schemes and also takes the requirements of QoS communication into account. Therefore, it allows to switch buffers on and off, via software, when they are allocated for GS connections. For BE communication, hardware monitors are used to determine the number of virtual channels, which is currently needed. The proposed scheme was implemented and evaluated as a student work [Wie14b] and will be briefly introduced in the following paragraphs.

7.4.1. Run-time Power Management Concept

In packet switching NoCs, the buffers have a large share of the total power consumption of the routers. When using virtual channels, even more buffer resources are required and their power share increases. Figure 7.17 shows the distribution of the average power consumption of the router design, introduced in section 5.1. The buffers dissipate almost 80 % of the total power of the router[39].

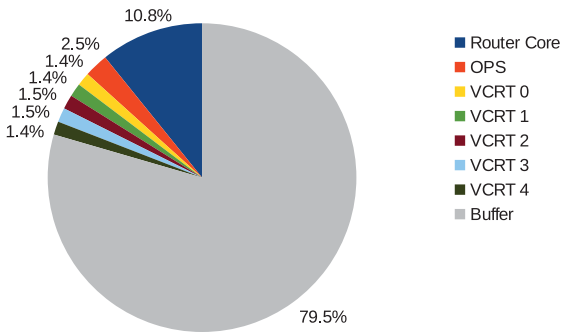


Figure 7.17.: Proportional share of different router components in power consumption for a router with 4 VCs and 8 buffer slots [Wie14b].

The main idea of the proposed power management scheme is to identify idle virtual channels and to apply fine-grained power-gating techniques to their buffers. As each buffer is dedicated to a single VC, it may be powered off without influencing the performance of any other channel. In order to determine the number of virtual channels, which can be turned off, the monitoring infrastructure from

[39] This share of 80 % is consumed by the 20 VC buffers (four per port) of the router. Thus, each buffer causes about 4 % of the router's total power consumption.

section 5.5.3 is used. In particular, the virtual channel utilization monitors are used to determine the VC utilization. These monitoring values are processed by the *power management unit (PMU)*, described in detail in section 7.4.2.

The power management scheme must identify and induce situations where a VC is a viable candidate for shutdown. In order to shut down a buffer safely, it must be ensured that the buffer is empty and that it will not be used during power-gating. In the router design, introduced in section 5.1, the buffer management and transmission control is located at the output port of each router, whereas the buffers are located at the input port of the neighboring router. The fact that the buffers and their management are spread over the output and the input of two neighboring router necessitates a distributed implementation of the buffer's power management. The power management unit at an output port must decide which VCs shall be disabled. Therefore, a selection policy is used, as detailed later. In order to disable a buffer safely, the buffer management at the output port must be modified in way that designated VCs can be excluded from the reservation and transmission process. Once the VC or buffer is in a safe mode[40], it can be disabled. Therefore, the PMU at the output port sends a power-gating request to the PMU at the input port of the neighboring router. This PMU then triggers the actual power-gating and turns of the router. Once the PMU at an output port decides to reactivate a virtual channel, it forwards a reactivation request to the neighboring router and waits until the power is switched on. Subsequently, the virtual channel can once again be used.

The power management unit at an output port decides which of its router's outbound channels should be power-gated. A *selection policy* is used to determine the virtual channels that should currently be available and those, who can be disabled. According to [Wie14b], a selection policy can be defined as follows:

Definition. *A selection policy optimizes the number of active VCs. It is optimal if at every point in time, only the minimal number of VCs is powered on, while ensuring that a connection does not experience any additional delay due to power management on one of its reserved channels.*

A *static policy* cannot satisfy the demands of dynamically changing load conditions: If the load is higher as assumed by the static policy, the active VCs are over-utilized, resulting in performance degradation. However, a load situation with little utilization would result in a waste of power due to idle VC buffers. In order to cope with changing traffic scenarios and load distributions, a *dynamic selection policy* is necessary. It should consider the current and the expected load situation for the near future. According to [Wie14b], the following parameters are worth consideration for this purpose:

[40] The buffer is in a safe mode for power-gating, if it is empty and shielded against new transmissions.

- Amount of time required to turn on and off a VC and its buffer
- Number of VCs reserved currently, their utilization and life-time
- Traffic characteristic: e.g. overall traffic load, traffic patterns, hot spots or life-time of connections

In [Wie14b], two dynamic policies are implemented. These policies consider the VCU monitoring values of the most recent cycle to adjust the selection. One of the two dynamic policies uses the *peak* and one uses the *average* monitoring values as a basis. Both policies consider the monitoring data of the most recent monitoring period as a prediction for future load. Pessimistically, it is assumed that the load may increase, compared to the last period. Thus, one additional VC is kept active in addition to the predicted values for *peak* and *average* utilization. This pessimistic assumption reduces the performance penalty of power management.

7.4.1.1. GS Connection Power Management

The previously presented power management scheme only provides the possibility to switch off free (non reserved) virtual channels. This is sufficient for best-effort packet switching, where a VC is only occupied for a single packet and thus for a very short time. For GS end-to-end connections, introduced in section 6.2, a reservation typically remains valid for an extended period of time. However, an established GS connection might be idling for a given interval of time. This idle time can also be used for power saving by switching off reserved VCs when they are not being used.

In order to switch off a reserved buffer, forwarding of flits to the power-gated buffer must be prevented. Consequently, the power-off state can only be entered once the buffer is empty. Therefore, all flits must be forwarded to the next router before power of the respective buffer is turned off. Subsequently, the PMU at the egress end of the link may power down the virtual channel buffer.

A scheme named *connection freezing* is carried out in [Wie14b] to enable power-gating of buffers, allocated for GS connections. By default, a GS connection is always active to ensure QoS guarantees. However, the source node may know that the connection is not needed in the near future. Consequently, it may choose to freeze an established connection to save power in the system. Freezing indicates that the source node allows every buffer and VC on the path of the connection to be offlined. Freezing and subsequent *unfreezing*, of a GS connection, is indicated by the source's network adapter to the first router on the path. The command is then propagated along the connection to each router's PMU, which adjusts the power state accordingly. Once an unfreeze command or freeze command has fully propagated along the connection's route, its virtual channels are either powered on or powered off, respectively.

An example for such freezing and unfreezing is provided in figure 7.18. In this example, the last routers on the path are not yet frozen, due to the propagation delay of the freeze command. However, a part of the routers is currently frozen, whereas the anterior routers on the path are already unfreezed again.

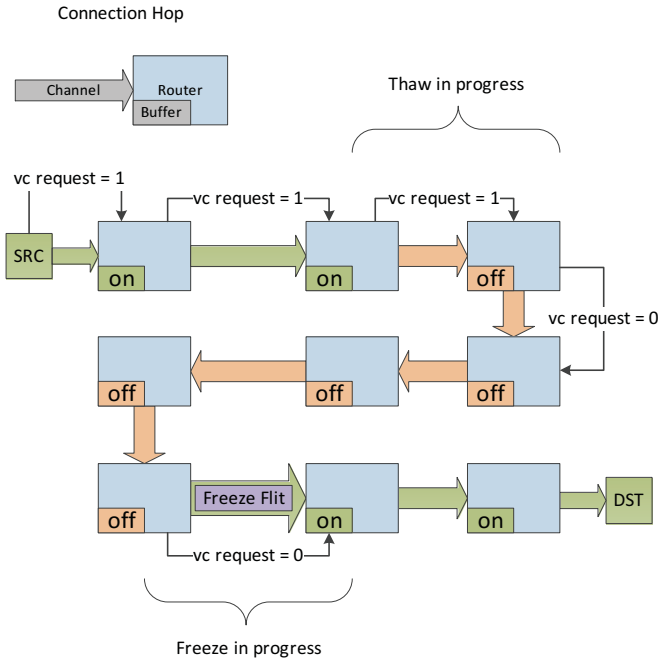


Figure 7.18.: Freezing and unfreezing (thawing) of a connection [Wie14b].

More details about the implementation of connection freezing are provided in the following section and in [Wie14b].

7.4.2. Implementation

One power management unit is added to each router to control the power state of the router's buffer resources. It tracks the resource utilization by using the VCU monitors. Then it determines the desired power state by using a dynamic selection policy and advertises available buffer resources to its consumers. The current PMU implementation, presented in [Wie14b], permits the management of virtual channels and drives the power domains that contain the VC buffers. Figure 7.19 shows the integration of the power management unit into the router design, introduced in section 5.1. The PMU is connected to the monitoring unit and to the

reservation manager of the router; known as ORT. The outputs of the PMU control the power domains of the VC buffers at the input of the neighboring router. PMUs of neighboring routers are connected by two types of signals named *advertise* and *request*. The request signal, which is driven by the upstream router, indicates the VCs which shall be enabled or disabled respectively. The advertise signal is controlled by the downstream router and provides the actual power-gating status of each virtual channel buffer[41].

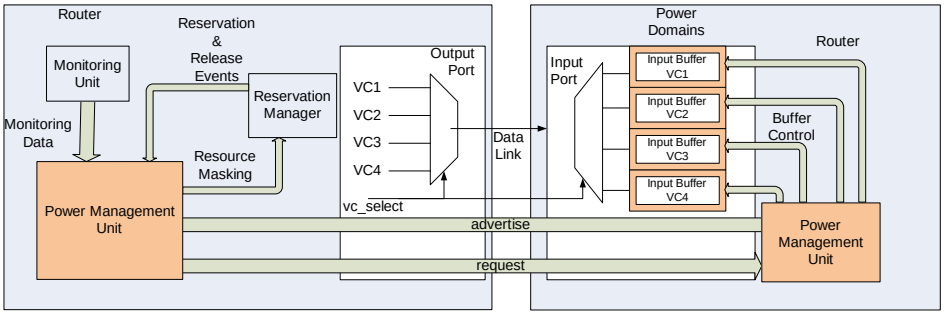


Figure 7.19.: Integration of the power management unit into the existing router design [Wie14b].

Controlling the power domain of a buffer requires a specific sequence of actions when changing its state. A separate FSM is used for each VC to track the power domain's progression through *power off* and *power on* sequence. The power off sequence ensures isolation of the power domain before its supply voltage is turned off. The power on sequence is used to ensure that the buffers reach a stable state. When reaching this state, the buffer is reset and the isolation cells are disabled.

The power management for GS connections requires additional logic within the PMU. A *connection table* is required to propagate freeze and unfreeze commands and to ensure readiness of the virtual channels on the path.

A freeze command is accomplished by injecting a special control flit into an idle GS connection. In this flit, neither the head nor the tail bit are set. The freeze flit is not processed by the router's control logic because neither the head nor the tail bit are set, as described in section 5.1.2. However, the PMU triggers power-gating of the associated VC after forwarding a freeze flit, as shown in figure 7.18.

A connection which is currently powered off cannot be used to transmit an unfreeze command. Thus, unfreezing requires an alternate approach. The request signal between the PMUs of neighboring routers is used to unfreeze a connection.

[41] The advertise signal is used by the upstream router to know when a reactivated VC is available again and can be used for transmission.

As soon as a PMU receives a power-on request for a frozen GS connection, it enables the respective buffers. The connection table is used to obtain the output port and VC ID of the connection. Subsequently, the procedure is returned and the power-on request is forwarded to the next hop of the connection. This specific implementation enables the resume transmission as soon as the VC at the source router is ready for transmission[42]. Hence, the proposed connection freezing method guarantees a constant amount of time for unfreezing from the perspective of the source node. This enables predictability and the fulfillment of quality of service requirements.

7.4.3. Evaluation

For an accurate evaluation of the self-optimizing power management scheme, the ASIC synthesis flow of the router was extended to support power-gating. The ASIC flow from section 5.3.2.1 is used as a basis for a *low power flow*[43]; described in detail in [Bis14].

In contrast to previous evaluations, where a 45 nm was typically used, a 65 nm library from *TSMC*[44] is used for the following investigations[45].

The HDL model and the synthesized netlist of a 5x5 mesh are used for evaluation. The routers are configured to have four virtual channels and a 32 bit wide data link.

In addition to the reference design and the dynamic policies, introduced earlier, three static policies are taken as a baseline for comparison. Thus, six different configurations are compared in the following:

- **Reference:** The basic router design from section 5.1 is used as a reference. It has no instrumentation for run-time power management.
- **All:** Power management with static selection policy. All VCs are enabled statically.
- **Half:** Half of the virtual channels on each link are permanently selected for power-gating.

[42] Since the unfreeze command is passed forward immediately when a router receives it, a data flit experiences no further delay during transmission. By the time that it arrives at any router along the connection, the required virtual channels are already powered on again [Wie14b].

[43] The low power flow uses the *unified power format (UPF)* to define the power domains of the design. The *Synopsys VCSMX* simulator is used for accurate power measurements based on realistic communication scenarios.

[44] The exact name of the *TSMC* 65 nm standard cell library is tcbn65lp.

[45] It was the only available library that provides the header, footer and isolation cells that are necessary to implement power-gating.

- **One:** Just on VC is available for data transmission, the other VCs are statically power-gated.
- **Peak:** The virtual channel utilization monitors from section 5.5.3 are used for dynamic selection. The monitoring peak value of the most recent monitoring period is used to determine the number of active VCs.
- **Average:** The number of active VCs is dynamically selected according to the average VC utilization during the most recent monitoring cycle.

The achievable throughput of the network on chip is one of the most basic measures for its performance. Table 7.4 shows the achievable throughput for the six investigated configurations. The numbers were empirically determined by consciously increasing the packet injection rate in the HDL simulation environment. Merely instrumenting the NoC for the proposed run-time power management scheme does not impact its throughput (cf. *all* configuration). However, if other policies are used, which disable a part of the virtual channels for power saving, the throughput decreases, as expected. The best results could be achieved when using the peak policy. Compared to the reference, the throughput is only reduced by 11.7 %. The highest throughput degradation of 63.3 % results from the use of the static policy with one active VC. However, this is also the configuration with the lowest power consumption, as described later. This shows that the proposed power management scheme can be used to trade-off between the power consumption and the throughput of a NoC.

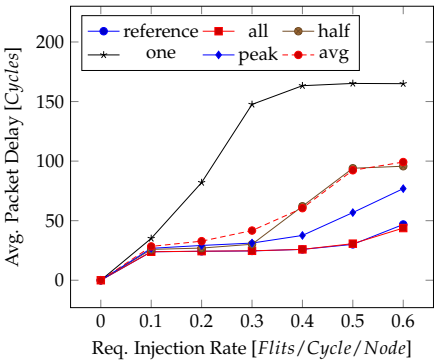
Configuration	**Max. Throughput** $[Flits/Cycle/Node]$ absolute	relative
Reference	0.60	±0 %
All	0.60	±0 %
Half	0.40	-33.33 %
One	0.22	-63.33 %
Peak	0.53	-11.67 %
Average	0.44	-26.67 %

Table 7.4.: Comparison of the power management configurations' maximum throughput [Wie14b].

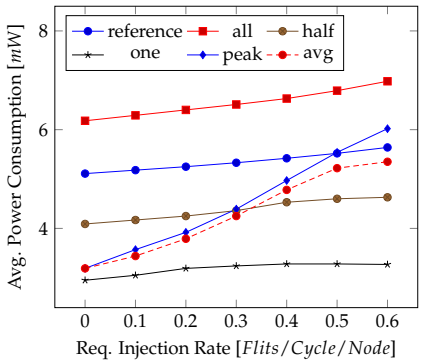
In addition to throughput, the latency is another important performance aspect of a network. For latency analysis, the injection rate for uniform random traffic is increased stepwise from 0.0 to 0.6 $Flits/Cycle/Node$[46]. Figure 7.20(a) shows the latency measurements of the six different NoC configurations. The reference and all configuration provide the lowest packet delay. The peak policy also enables

[46]Not all configurations reach an injection rate of 0.6 $Flits/Cycle/Node$. The saturation point of each configuration is provided in table 7.4.

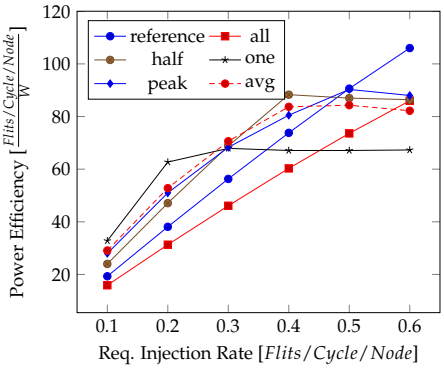
very low latency, especially for low injection rates and can compete with reference and *all* configuration. The average (avg) policy has a higher latency but compared to the static policy *one* it still offers significantly lower latency.



(a) Average packet latency



(b) Average power consumption



(c) Power efficiency

Figure 7.20.: (a) Average packet transmission latency, (b) power consumption and (c) power efficiency for a power managed router with different static and dynamic selection policies.

Aside from performance, of course, power consumption is an important aspect. Especially in the evaluation of a power saving mechanism, such as the one previously discussed. In order to obtain accurate power measurements, the router at the center of the NoC was replaced in the simulation environment by a synthesized netlist. The measurements results are illustrated in figure 7.20. The absolute values for the power consumption of a single router are provided in figure 7.20(b) for the same scenario as used for latency analysis. The reference design and the

configurations with static policies (all, half and one) exhibit only a slight impact on the power consumption when NoC load changes. The reason for this low impact is that only dynamic power consumption is affected by the load of the NoC. In contrast, average and peak policy adapt the static power consumption of the router to the NoC load. Unused virtual channel buffers are power-gated if the load is low, resulting in a very low power consumption. For an injection rate of 0.1 *Flits/Cycle/Node*, the total power consumption of a router could be reduced by 31 % and 34 % using peak and average policy[47].

In addition to the absolute power consumption, power efficiency, provides a suitable measure for evaluating the proposed power management scheme. Power efficiency is defined in section 4.1.2 as the ratio between the bandwidth and the power consumption. This definition is used in figure 7.20(c)[48]. For very low injection rates, the *one* configuration provides the best power efficiency. The reference design, without any power-gating support, is predominant for very high injection rates, which cannot be reached for other configurations. However, the dynamic power-gating configurations (average and peak) exhibit good power efficiency for the entire range of injection rates. They are the only schemes which can adapt the bandwidth and power budget of the router at run-time by disabling unused resources, dynamically.

In addition to performance and power consumption, the implementation overhead of the proposed scheme was evaluated in [Wie14b] for the 65 nm standard cell library[49]. A single router was synthesized to obtain the achievable clock frequency and power consumption. The results for the investigated configurations are summarized in table 7.5. By adding the power management unit and the power management instrumentation to the router, its required area increases by about 20 %. The achievable clock frequency of 350 MHz is not affected by the power management implementation. The power management support for GS connections, described in section 7.4.1.1, is responsible for approximately half of the additional area. This is due to the connection table, which requires a significant amount of area. Consequently, the area overhead of the proposed power management scheme could be noticeably reduced if GS support is not needed.

The previous studies of the proposed power management scheme showed that the strategy can significantly reduce the power consumption of the NoC. The peak policy achieves the best trade-off between power saving and performance

[47] The impact of the NoC utilization on power consumption is relatively small, due to the usage of a zero wire-load model. It was used because of the unavailability of an accurate floor-planning for the router.

[48] The achievable throughput of each configuration, provided in table 7.4, is taken into account for power efficiency analysis.

[49] The synthesis results for the 65 nm standard cell library differ heavily from other results presented in this work. The 45 nm which is typically used, is designed for performance, whereas the 65 nm library is tuned for low power consumption. This is the reason for the much lower clock frequency achieved with the 65 nm library.

Configuration	Clk. Freq. [MHz]	Area [μm^2]	
		Absolute	Relative
Reference (Basic)	350	105,550	±0 %
All	350	127,751	+21.03 %
Half	350	123,651	+17.05 %
One	350	120,378	+14.05 %
Peak	350	128,981	+22.20 %
Average	350	129,199	+22.41 %

Table 7.5.: Achievable clock frequency and area required for different power management configurations.

degradation. It can save more than 30 % of the total power compared to the reference, while reducing the maximum throughput by only 12 %. For moderate injection rates, the latency is only increased by 12 %. Power management, with peak policy, has an area overhead of 22 % compared to the reference.

A more detailed evaluation, which also investigates the impact of the buffer reset delay[50] is presented in [Wie14b]. Moreover, additional details about the implementation are also provided.

7.5. Summary

In this chapter, four different self-optimization and -organization strategies were introduced. Just as the QoS schemes, presented in the previous chapter, they are designed in the form of modular extensions for the basic NoC architecture from chapter 5. The proposed strategies shall on the one hand simplify the management of the network on chip and reduces the management overhead. On the other hand they shall improve the functional and non-functional properties of the network, such as throughput, latency and power-consumption.

In section 7.1, a distributed, scalable, self-optimization strategy, named *rerouting*, was introduced. The rerouting scheme detects communication hot spots of guaranteed service connections by the use of the monitoring infrastructure. Subsequently, GS connections are relocated in a distributed way. The evaluation of the concept showed that rerouting can increase the number of GS connections, which can be established successfully, by up to 29 % for synthetic traffic patterns. Delay and throughput of established GS connections could also be improved by up to 9 % and 3 % respectively. It was shown that the amount of tolerated existing

[50] The buffer reset delay is the time between the point where the buffer is switched on again and the point where the power of the buffer is stable and the buffer can be used again.

GS connections could be increased by a factor of 3.875 on average, when setting up new connections of video applications with rerouting support. The area overhead for rerouting support is 8.2-8.8 % for an ASIC implementation, depending on the NoC parameters.

The concepts which were presented in section 7.2 also rely on GS end-to-end connections, introduced in section 6.2. The two strategies, named *Auto-GS* and *connection replacement*, shall optimize best-effort communication flows by transforming them to guaranteed service connections. The monitoring infrastructure, introduced in section 5.5.3, is used to identify candidates for transformation. Applications not supporting GS communication from the software perspective can profit from their lower power consumption and reduced latency. Real world applications were used to illustrate the gain in overall communication performance. When using the Auto-GS self-optimizing strategy, the communication related power consumption could be reduced by up to 50 % and the latency could be reduced by 25 % at maximum.

In section 7.3, adaptive data collection mechanisms were proposed and investigated. The three mechanisms target region-based data collection in large, many-core architectures and can be used to gather status information as a basis for system state optimization. An adaptive Hamilton routing algorithm, which is required by two of the investigated mechanisms, was proposed. The mechanisms were investigated in terms of performance and implementation costs. The results show that the choice of the strategy depends on different parameters such as region size and data set size. The proposed mechanisms named *round-trip* and *mixed* outperform the straight forward state of the art implementation for most cases. Synthesis results exhibit that the implementation overhead is very low. Data collection hardware increases the area utilization of a router at maximum by 2.4 % and the power consumption by 5.2 % for an ASIC implementation.

A power-optimization and management strategy was introduced section 7.4. It uses communication monitoring to power gate idle virtual channels automatically. This strategy reduces the energy overhead that is introduced with virtual channels. The strategy supports software-controlled power management for GS connections. Different management policies are proposed to optimize performance or power consumption. The results show that the power management scheme can save more than 30 % of total NoC power compared to a reference design, while reducing the maximum throughput by only 12 %. The overhead for an ASIC implementation is 22 % at maximum.

8. Fault Tolerance and Reliability

Improvements in chip manufacturing technology have driven an astonishing growth of the number of cores or IP blocks integrated as one system on a chip. However, predicted limits which are mainly related to dependability issues must be overcome to continue the observance of Moore's law for future manufacturing technologies [105]. A paradigm shift must be applied by *"building dependable systems with non-dependable components"* [31] in order to vanquish the predicted dependability issues.

The integration of a high number of cores, which will be enabled by future technology nodes, necessitated the research and development of NoCs, as explained earlier. Consequently, network on chip architectures must address the paradigm shift towards dependable and reliable systems to enable their implementation with future chip manufacturing technology. Hence, methods for fault tolerance in network on chip-based architectures have become an important field of research [210].

In this chapter, a fault tolerance scheme for the proposed network on chip will be introduced. The fundamentals and definitions for fault tolerance haven been provided in section 2.5 as a basis. At the beginning of this chapter, existing fault tolerant NoC concepts will be discussed. From the limitations of existing fault tolerance mechanisms, the concepts for error detection and localization, as well as error handling are derived. These concepts are described and evaluated in section 8.2 and section 8.3.

8.1. State of the Art

Further technology shrinking will increase the risk of transient, intermittent and permanent faults. Networks on chip are intended to be manufactured using said advanced technology nodes. Consequently, fault tolerance became an important aspect for research and development of NoCs.

A good overview of existing fault tolerance methods for networks on chip is provided by Radetzki et al. [210]. However, the most relevant methods for the fault tolerance scheme, presented in this work, are discussed in the following paragraphs.

The most basic principle to achieve tolerance against transient and intermittent faults is the use of temporal redundancy. A corrupted message is detected by the receiver and requested to be transmitted again in the hope that it arrives undistorted after retransmission [184]. Another strategy, used to cope with transient and intermittent faults, is the use of spatial redundancy. In [196] and [198] fault tolerant routing is used to send a message over disjoint paths. In case of an error on a transmission path, the error-free message transferred through an alternative path can be used directly. Compared to a retransmission scheme, multi-path routing [182] can reduce the latency in case of errors and consequently improve the performance. Unfortunately, multi-path routing increases the utilization of the NoC due to spatial redundancy. However, retransmissions schemes and multi-path routing are well accepted means that manage transient and intermittent faults.

The concept presented in the following sections addresses permanent errors resulting from aging or manufacturing. Coping with transistor defects necessitates architectural changes. In [48], naive *triple modular redundancy (TMR)* is compared to domain-specific techniques such as end-to-end error detection, resource sparing, automatic circuit decomposition, iterative diagnosis and reconfiguration. The results show that designs are attainable that can tolerate a larger number of defects with less overhead, as opposed to simple TMR. One basic assumption of existing fault tolerance schemes is the defect of a complete router due to one, or multiple permanent faults. Such a defective router can be detected and localized either online [184, 227] or offline [52]. In [184], run-time error detection for end-to-end flow control is investigated. An interleaved error-locality-aware code for end-to-end error correction is presented in [227]. An online diagnosis of defective wires, presented in the same work, can be used for the diagnosis of defective routers and for locating intermittent faults. A diagnoses and localization scheme for faulty links is presented in [212]. An offline *built-in self-test (BIST)* method with a cost-effective test sequence for error detected and localized is introduced in [52]. The above-mentioned strategies may only be used to detect and localize link errors. In [126], a distributed approach for online testing is presented. The strategy enables to detect up to 85 % of errors in the control paths including routing logic, FIFO's control path and the arbiter. The methodology is applied to a basic packet switching NoCs without VCs and QoS support. In contrast, the test and localization method presented later in the work in hand is applied to the much more complex QoS supporting NoC, introduced in the previous chapters.

If a defective router is localized by one of the aforementioned schemes, it must be bypassed to mask the erroneous router. One strategy to bypass faulty routers is the modification of the routing strategy in the surrounding of the defective router in order to route the packets along its borderline [210]. The authors of [28] employ XY routing as the basic routing technique. An adaptive routing scheme is only used to surround nodes or regions. Adaptivity is reached by allowing

a packet to switch from Y back to the X direction upon reaching or leaving the region boundary. However, the strategy is not proven to be deadlock-free for the widely used wormhole switching NoCs. Schäfer et al. [220] use the adaptivity of the odd-even turn model to bypass rectangular blocks. The strategy presented in [270], allows multiple fault regions of complex shape. The odd-even turn model is again used as the basis for deadlock-free routing. A look-ahead to faults farther away, based on deflection routing, enables more complex, even concave fault regions [75]. A list and a comparison of existing fault tolerant routing schemes can be found in [210]. However, the existing methods all have limitations due to the used routing scheme. Either the shape of a region or their number is limited. Later-on a bypass method is proposed that can be applied independent of the routing scheme. Another common drawback of existing schemes is that the bandwidth-loss due to defective regions is not substituted. Consequently, the routers in the surrounding of the defective region have the burden of additional traffic. In contrast, the fault tolerance scheme presented later substitutes a part of the lost bandwidth and bypasses a faulty region transparently with respect to the surrounding routers.

8.2. Fault Detection and Localization

For error handling, it is necessary to distinguish between transient, or intermittent and permanent faults. Transient and intermittent faults can be counteracted by well-known techniques, such as retransmission, multi-path routing or error-correcting codes [184]. In contrast, handling permanent faults is more challenging. These permanent faults are expected to become more relevant with shrinking technologies due to aging and other physical failure mechanisms. Thus, a concept for detection and localization of permanent errors, in meshed NoCs, is presented subsequently. The implementation and evaluation of the concept was carried out in a student research work [Klö13].

8.2.1. Distributed Fault Localization Concept

The concept, introduced now, is mainly implemented in software. Minimal hardware support is required to enable test pattern generation and analysis. This hardware/software co-design approach enables minimization of the hardware overhead for error localization.

The concept is based on systematic flooding of the NoC, or a region of it, by the use of special test packets. These packets are analyzed after reception to determine the location of the fault or error. By using an intelligent synchronized test pattern generation, the pair of communicating nodes for error localization is known to

the test software. This simplifies error localization. All test packets that leave the NoC are logged and analyzed for potential errors. Due to the knowledge about the expected test data, erroneous sections of the NoC can be identified easily. Only the network adapter has to support the transmission and reception of test packets. A software interface is required to trigger test packet transmission and to access test packets for analysis after arrival. The NA is not required to depacketize received test packets. The raw test data are accessed by the diagnosis software to analyze them.

The basic diagnosis flow is presented in figure 8.1. In order to test the network for failures and to identify and localize the erroneous components, the entire NoC or a region[1] is flooded sequentially by test packets. Therefore, every router $R_{x,y}$ sends test packets to all other routers in the NoC or region. The results of each tested path are stored in a *status matrix*, as described in detail later. The status matrix contains information about the state of every transmission section.

Definition. *A transmission section is defined as the set of components that is required to transmit data in the network from hop to hop. For two neighboring routers of a NoC, this path contains all components between the two crossbars of the routers. The crossbar that defines the direction taken by the data flow, represents the start-point and the end-point of a transmission section.*

At the beginning of the diagnosis, the NoC is assumed to be faultless and the status matrix is initialized accordingly. By sequential testing of all possible paths of the NoC, the status matrix is successively updated with the test results. Consequently, the status matrix contains the state of every transmission section after testing. It shows whether the test data transmission between router $R_{x,y}$ and $R_{x+i,y+j}$ with $i, j \in \{-1, 0, 1\}$ was successful or not. The localization of a faulty transmission section consists of two phases, a *path analysis* and a *link analysis*. The initial path analysis is used for coarse localization. Subsequently, the link analysis limits the location of a defect to a designated transmission section. For link analysis, test packets are transferred between two directly neighboring routers. If an error occurs the respective transmission section is known to be faulty for the direction of analysis[2]. By systematically transferring test packets for path and link analysis through all links of the NoC, all transmission sections and directions are tested. This systematic analysis is used to create a status matrix for the entire NoC.

In [Klö13], two different test packet transmission schemes were built and studied. The more trivial approach is named *global flooding*. For global flooding, each

[1] In [Klö13], two strategies for flooding are compared. Global flooding is used to test the whole NoC sequentially. In contrast, linear flooding enables a parallel flooding of disjoint regions of the architecture. Both schemes will be described later.

[2] Typically independent components are used in NoC for the incoming and outgoing port of a link. Consequently, a fault is typically limited to one direction of a transmission path or link.

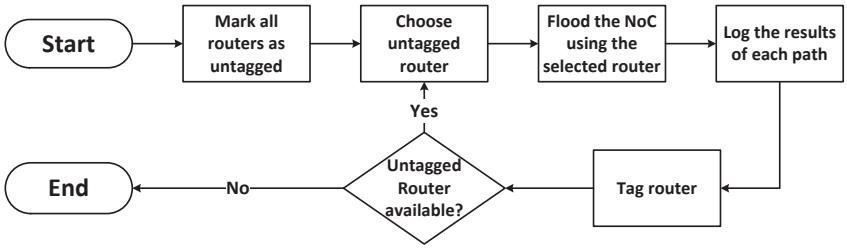


Figure 8.1.: Basic error detection and localization flow [Klö13].

router sends test packets to all other routers of the NoC. However, this strategy is very time-consuming and energy-dissipating. The any-to-any test communication results in redundant testing of links and routers. The time complexity for testing grows quadratic ($O(n^2)$) with the number of nodes n, because each node or tile must communicate to all others. This limits the scalability of global flooding. Minimizing redundant testing and reducing the time for testing is the goal of the second strategy, named *linear flooding*. Compared to global flooding, it enables to limit the number and the distance of test packets to be transmitted by each node. The basic idea of linear flooding is illustrated in figure 8.2. A fixed number of packets is transmitted by each node, reducing the time complexity to $O(n)$. Each router or node in the network assumes the role of the *transmitter* to cover testing of the entire NoC. In contrast to global flooding, the transmitting node does not send test packets to all other nodes of the architecture. It only sends packets to all nodes which are two hops away. By using two hops, this method not only validates parts of the routers, attached to the transmitter and *receiver* nodes, but also the intervening router, which is *forwarding* the packets. However, the test packet communication is spatially limited to a small region. This locality enables to run tests in disjoint regions in parallel and independent of each other. Parallel testing can be used to reduce the time for testing the entire architecture, when using linear flooding. Thus, the time complexity can be reduced to $O(1)$ with respect to the number of nodes, when using scalable *parallel linear flooding*. Moreover, linear flooding can also be used to limit testing to a specific region of a NoC. In order to test the corners with the linear flooding procedure completely, the scheme shown in figure 8.2 is slightly adapted. Further details can be found in [Klö13].

The proposed diagnosis scheme enables to detect different types of errors:

1. Loss of one or several flits (also covers control-flow errors in the router)
2. Data corruption in the flits (due to stuck-at faults)
3. Wiring errors at the data channels

4. Faults in the routing and reservation unit
5. Loss of the correct flit sequence
6. Wiring and logic faults in the crossbar

Once the faulty router is identified, different methods can be used to ensure operation of the architecture. Adaptation of the routing scheme can be used to bypass a faulty router, as described in detail in section 8.1. However, the presented diagnosis scheme is intended to be used in combination with the second layer network, introduced in section 8.3.

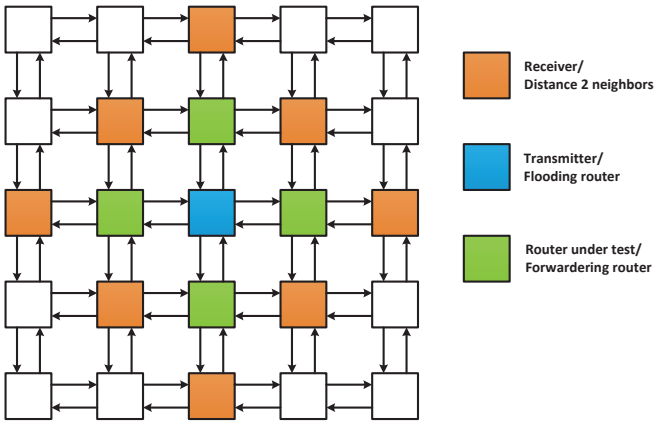


Figure 8.2.: Linear flooding limits the communication for diagnose to a small region of neighboring routers [Klö13].

8.2.2. Software Implementation

In order to fully analyze and test the network, each router $R_{x,y}$ must send multiple test packets. For *global flooding*, each router sends test packets to all other routers of the NoC. When *linear flooding* is used, the number of test packets that are sent per node, is independent of the NoC size according to figure 8.2. The test packet transmission is followed by an analysis phase at the receiving node. The source router $R_{x,y}$ waits for a feedback from the current destination node before transmitting test packets to the next destination node. The sequential delivery and the subsequent analysis simplify the packet classification and error analysis and consequently the software implementation[3]. The analysis flow, which is

[3]The conditions for a parallel analysis are discussed in [Klö13].

performed after each test packet, consists of the two phases named *path analysis* and *link analysis*.

Figure 8.3 illustrates the entire analysis flow for a complete network on chip in detail. A list L contains all the nodes that have not assumed the role as a transmitter so far. After this list is initiated, one node or router $r0$ is selected from the list L as source node. This is the beginning of the path analysis phase for the current *flooding router*, which is $r0$ in figure 8.3. Subsequently, the list T of target or destination nodes for $r0$ is set up. Next, one node or router rx is selected from the list T to be used as a *receiver*. Afterwards, several[4] test packets are generated and transmitted from node $r0$ to node rx. The reception of each flit is recorded in the node rx by writing into a shared data structure. More details about the packet transmission and analysis process are provided in section 8.2.2.1. If no error is detected, the path analysis is finished and the next path is tested. However, if an error (e.g. loss of flits, changes of the flit order or data corruption) is detected, the path analysis is interrupted and the detailed link analysis is started.

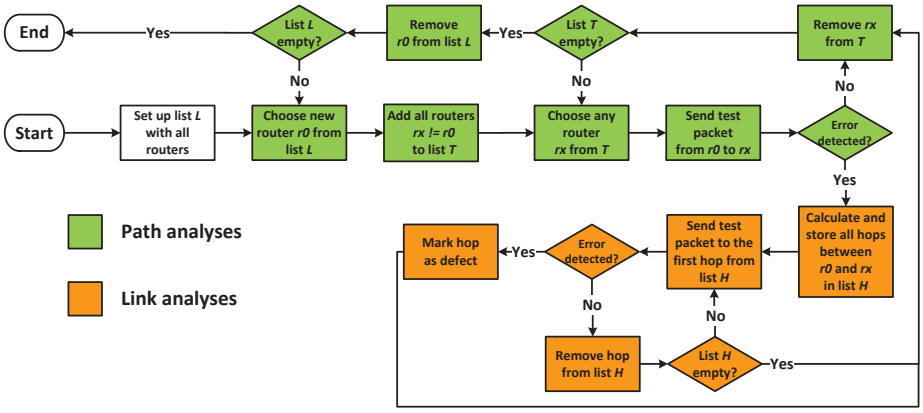


Figure 8.3.: Detailed flow of the path and link analysis and diagnosis [Klö13].

At the beginning of the link analysis, the routers on the path between node $r0$ and rx are ascertained. Deterministic routing is necessary to enable calculation of the route and the hops. All these nodes are stores in a list H. Then, the first node is selected from the list. The link of this node, which is also part of the path between $r0$ and rx, is then tested. Therefore, the test packets and analysis methods, described in detail in section 8.2.2.1, are used again. The link analysis verifies the same characteristics (completeness, order and correctness) as the path analysis. However, in contrast to the path analysis, the location of the fault can

[4]The exact number of test packets depends on the configuration of the NoC. Typically, one test packet is transmitted per virtual channel. For a good test coverage, all packets transmitted between a pair of nodes, are injected interleaved flit by flit.

be narrowed to a single transmission section. If no error is found at the actual transmission section, the next pair of hops on the path is selected by removing the first hop from the list H. This process is repeated until a defect is detected or the list H is empty. If an error is recognized, the respective transmission section is marked faulty in the status matrix.

8.2.2.1. Test Packet Transmission and Analysis

Figure 8.4 shows the time sequence for transmission of test packets between two routers. When QoS support is enabled in the NoC, the path and link analysis starts by setting up a maximum number of GS connections between the nodes $r0$ and rx. Once the connection is established, body flits with a one-hot coding payload are transferred. This coding scheme, which is further described in [Klö13], enables to detect multiple faults in the router components and links. After transmitting the final body flit, a tail flit is used to close the established connection.

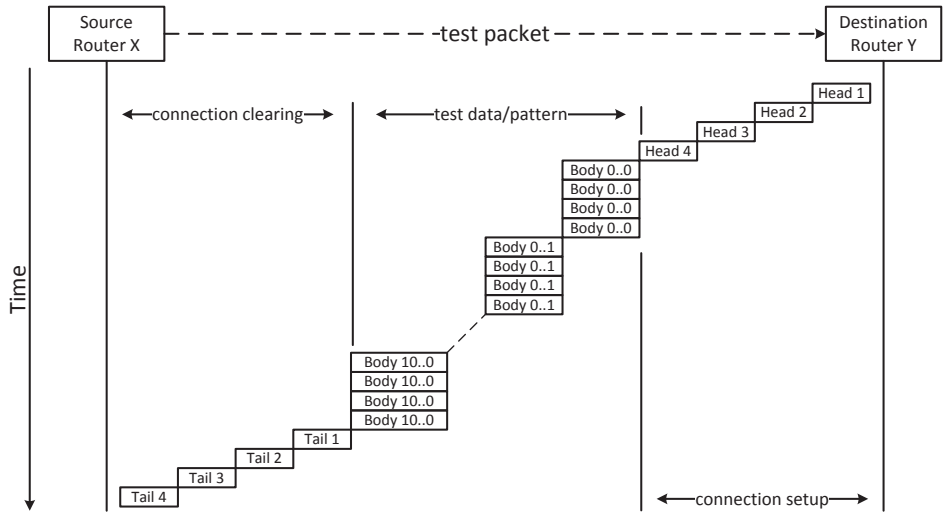


Figure 8.4.: Test packet transmission for path analysis [Klö13].

The analysis of the test data is performed by the destination node rx. The one-hot coding in each flit[5], as well as the connection setup phase enable to detect various faults. Loss of one or multiple flits is detected by the receiver. Therefore, it counts the number of received flits preceding the tail flit. The one-hot coding scheme

[5]The one-hot coding scheme, used for the body flits, is as follows: $Body_0$: 000...000, $Body_1$: 000...001, $Body_2$: 000...010, ...

can be used to determine the missing flit(s) exactly. Due to the known content of the expected flits, data corruption and changes in the flit order can be detected by logging the received flits. Improper wiring in the crossbar, buffers and links also affects the content of the body flits and can be recognized in the same way. Faults in the control path of the router (e.g. reservation or routing logic) can also be detected. Such errors typically result from forwarding packets to the wrong router or loss of flits. Both situations in turn lead to a discrepancy between the expected and the received number of flits at the destination node. By using this analysis strategy for path and link analysis, a defect can be narrowed down to one transmission section. More details about the packet format and the packet analysis can be found in [Klö13].

8.2.3. Evaluation

In order to evaluate the presented concept, defective routers must be present in the network on chip. Consequently, faults must be injected into the model of the NoC. Essentially, two models of the network could be used as a basis: (1) the SystemC model introduced in section 5.2 and (2) the HDL model from section 5.3. However, for precise fault modeling, the technology perspective must be taken into account. Permanent faults, addressed by this work, typically occur at the granularity of single transistors or wires. For accurate evaluation, the faults have to be modeled precisely. Consequently, the HDL model of the NoC is used because it is more accurate compared to the SystemC model. In order to further increase the accuracy, a synthesized netlist is generated from the HDL model. The netlist reflects the real hardware implementation for the chosen target technology[6].

For fault modeling, the synthesized netlist is manipulated after synthesis, according to the flow illustrated in figure 8.5. A parser, implemented in C++, is used for fault injection. It reads the netlist and analysis it to determine the type and distribution of the instantiated standard cells. According to the distribution, one type of standard cell is selected to be manipulated. Subsequently, one entity of the selected type is manipulated. This method enables a fair uniform random distribution of the faults in the design. The current set of supported manipulations comprises *stack-at-0*, *stack-at-1* and *inverting*.

The manipulated gate-level netlist is subsequently used in the HDL simulation environment, described in section 5.3.1. Therefore, the proposed concept for fault detection and localization was implemented in the HDL simulation environment, as described in detail in [Klö13]. A 4x4 and an 8x8 meshed NoC, with four virtual channels and GS support, introduced in section 6.2, is used for the following evaluation. In order to trade-off between accuracy and simulation speed [158]

[6] The ASIC synthesis flow, introduced in section 5.3.2.1, and the *TSMC* 45 nm standard cell library are used for netlist generation.

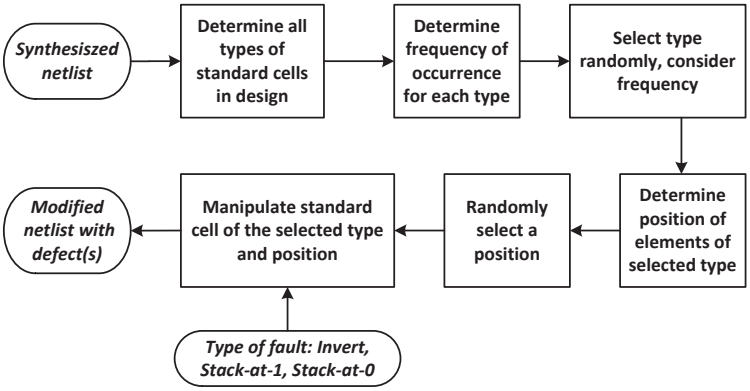


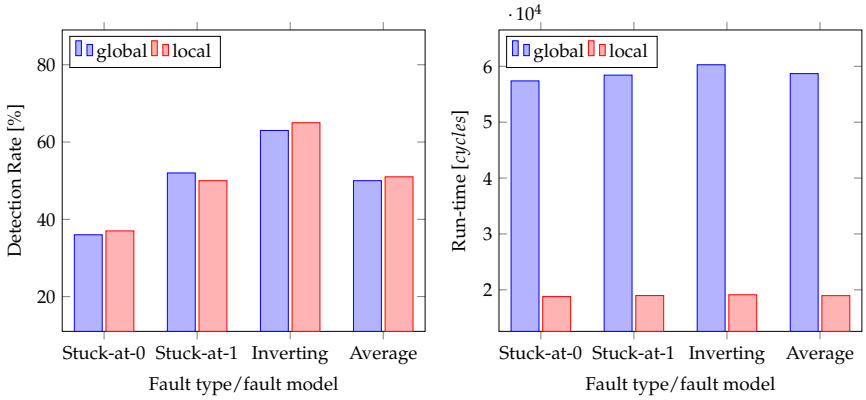
Figure 8.5.: Schematic flow for netlist parsing and fault insertion [Klö13].

a mixed granularity simulation is used. For mixed granularity simulation, one router in the NoC is selected to have a defect. The manipulated gate-level netlist is then used for the entity of the selected router. The other routers and components in the NoC are simulated by using the more abstract HDL representation. The mixed granularity simulation enables to model the fault accurately, while maintaining the simulation speed, by using more abstract HDL models elsewhere.

More than 1,000 iterations have been performed to obtain the fault recognition rate and the run-time[7] for error detection and localization. In each iteration, the NoC is completely tested. Both, linear and global flooding are used. When linear flooding is used, all nodes take on the role of the transmitter sequentially, as shown in figure 8.2. Figure 8.6 provides the measurement results for global and linear flooding. The three different types of faults are evaluated separately.

Figure 8.6(a) summarizes the fault recognition rates, which reflect the relative number of detected faults. The analysis shows that both approaches, global and linear flooding, provide comparable results. The highest recognition rate of 63-65 % is measured for inverting faults. Stuck-at-1 faults can be detected with a probability of 50-52 %. However, stuck-at-0 faults are detected in 36-37 % of the cases. The significantly lower recognition rate for stuck-at-0 faults is justified by transparency. Stuck-at-0 are very likely to be transparent faults, because 0 is the reset value of most registers in the design. Averaging the recognition rate, 50-51 % of all faults can be detected. Additional analysis could show that 100 % of the link errors can be detected and localized. Compared to the only existing testing scheme [126], which addresses not only link but also control errors in the

[7] In this section, the run-time refers to the duration of time necessary for the transmission of the test pattern. The computation overhead for analysis of the received test pattern could not be estimated due to the modeling of the software analysis in the SystemVerilog test environment.



(a) Fault recognition rate

(b) Run-time for localization

Figure 8.6.: Evaluation of the fault detection and localization scheme - (a) recognition rate and (b) run-time [Klö13].

routers, the error recognition rates presented here are lower. The authors claim to detect up to 85 % of all errors, however, the NoC used in [126] has a much lower complexity due to the omission of VCs and GS support.

The run-time for fault localization by the proposed scheme was also measured and is illustrated in figure 8.6(b). As expected, the results show that the run-time is almost independent of the fault type. However, the results also exhibit the huge advantage of the linear flooding compared to the more naive global flooding approach. The run-time for test pattern transfer can be reduced by a factor of 3 when linear flooding is used in a $4x4$ mesh. Additional evaluations, summarized in table 8.1, show that the run-time improvement of linear flooding increases with the NoC size. A factor of 15 is obtained when comparing the run-time of linear and global flooding for an $8x8$ mesh.

NoC dimensions	Variant	Run-time [*cycles*]	Relative to Linear
4x4	linear	19120	-
4x4	global	60277	$\sim$ 3x
8x8	linear	134639	-
8x8	global	2057644	$\sim$ 15x

Table 8.1.: Run-time for fault detection and localization in a 4x4 and 8x8 mesh for the linear and global flooding variant [Klö13].

8.3. Error Treatment

The previously presented method can be used to localize a faulty router. Following the detection and localization of a fault, the resulting error must be treated in order to ensure continuing operation of the network. In existing work, this is typically done by adapting the routing scheme to bypass the faulty router, as discussed in section 8.1. However, this strategy has several drawbacks:

- Non-minimal adaptive routing schemes are necessary, which could lead to deadlocks and communication overhead
- The routers surrounding the faulty router or region have to carry additional load

Motivated by these drawbacks, an alternative approach for error treatment in regular NoCs is presented in this work. It has been implemented in the context of two student works [Ste13] and [Dud14].

8.3.1. Second Layer Network Concept

A transparent bypass is used to circumvent faulty routers. This bypass shall be activated on demand to form a ring around a defective router. In order to establish a bypass at run-time, the necessary infrastructure must be provided at design-time. Generally speaking, such an infrastructure consists of a unit that injects and ejects data from, and to, the normal NoC. These units must be coupled flexibly to form arbitrary bypass networks.

A *second layer network (SLN)* infrastructure enables transparent bypassing of faulty routers. A 4x4 NoC with a second layer network is shown in figure 8.7. It consists of two modules, named *multiplexers* and *switches*. The switches take on the task of coupling the normal NoC and the second layer network. The multiplexers connect the switches to form the required bypass on demand. However, if there are no errors in the system, the second layer network can be disabled or used for other purposes. A short discussion of alternative usage of the second layer network is provided in section 8.3.1.2. Otherwise, power gating [133] can be used to power off the SLN components when no defects are present. This would ensure that no power is consumed by idle SLN components.

When a defective router has been identified, the second layer network must be configured accordingly. However, it must be taken into account that the faulty router should not be used during configuration. In order to enable a flexible and distributed configuration of the second layer network, it may be configured by columns and rows. The configuration scheme is described in section 8.3.2.3. In order to completely bypass a faulty router, all adjacent switches of the SLN must be enabled. The multiplexers in between are used to connect

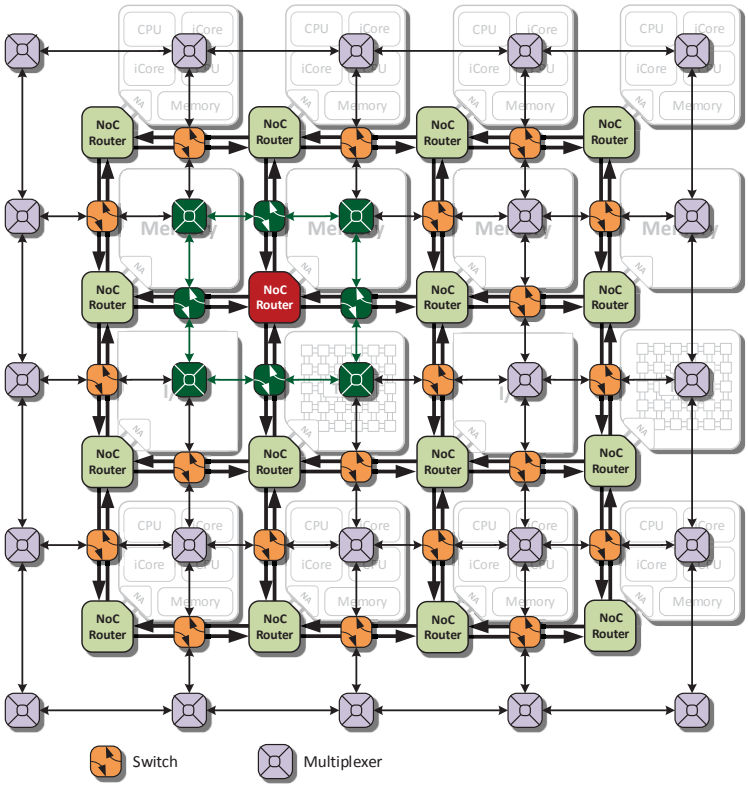


Figure 8.7.: Example of a 1x1 ring bus configuration in a 4x4 NoC architecture with a second layer network [Ste13].

the switches on demand. Such a configuration for bypassing a single router is shown in figure 8.7. The router, colored red, is assumed to be faulty and the green switches and multiplexers are activated and configured to bypass this router. Activated switches and multiplexers of the SLN form a ring. This ring operates as a *ring bus (RB)* [223] and is named accordingly hereafter. However, the size and shape of a ring is not limited to the 1x1 configuration, as shown in figure 8.7. Although it is most likely that a single router is faulty, the concept also supports larger regions, such as 2x1, if two neighboring routers are faulty. Bypassing of larger regions can also be used to save energy, as described in detail in section 8.3.1.2.

An established ring bus works with so-called *containers*. In order to ensure a fair bandwidth sharing, each router connected to the ring bus gets the same number of containers assigned. The containers are passed around the ring with half of

the containers moving clockwise and the other half counterclockwise. Each VC at an input port of the ring gets the same number of containers. Depending on various run-time and design-time parameters, the number of containers defines the bandwidth of the ring. The bandwidth per cycle at each input port of an established ring can be calculated as follows:

$$BW_{rb,port} = W_{cont,SLN} \cdot \frac{2 \cdot C}{N} \tag{8.1}$$

$W_{cont,SLN}$ is the size of a container of the second layer network and the factor of two results from the two directions of the ring. C defines the number of containers per router and direction, while N represents the number of switches forming the ring[8]. As can be seen from equation 8.1, the number of containers C can be used at design-time to adjust the bandwidth of the second layer network. If the number of containers per input port is chosen to be smaller than the number of VCs, a fair TDM arbitration is used to equalize the bandwidth of each VC. The control of the container assignment is carried out in a distributed way in the switches of the second layer network. During configuration of a ring, each switch is configured in order to adjust the arbitration scheme to the size of the ring. A flit within a container on the ring bus is forwarded during each cycle to the next switch. Once the container arrives at the position where the ring bus crosses a link of the original router of the flit, the flit is taken from the ring bus. This position is determined by the switch, by taking (1) the destination address of the packet, (2) the position of the switch in the NoC and (3) the used routing algorithm into account. The vacant container then returns empty to its associated switch, where it can be filled again with a new flit. The control signals of the router's physical interface, described in section 5.1.5.1, are carried on the ring in the same way as the flits. However, the acknowledge-signals that are used to build the credit-based flow control scheme must also be transported on the ring bus to enable flow control and to prevent buffer overflow. Figure 8.8 shows the flow of data on the ring between two adjacent routers. The credits transferred from the receiving router to the transmitting router take the opposite direction, as illustrated in figure 8.8.

Ensured by the credit-base flow control scheme, body and tail flits on the ring are always accepted by the router at the output of the ring. However, head flits on the ring might not be accepted by the router at the output switch because the used VC might be occupied by another packet. In such a case, the head flit remains in the ring and is returned to the input switch. Depending on the load of the ring and on the virtual channel multiplexing, the input switch either decides to leave the flit in the container or to buffer it for a retransmission. More details about the buffering scheme are provided in [Dud14].

[8] Four switches are required to build the smallest possible ring. Then one has $N \geq 4$.

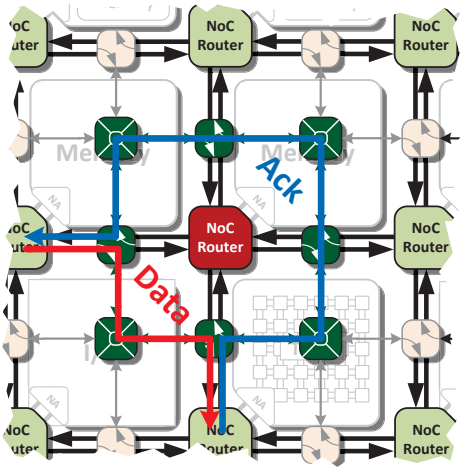


Figure 8.8.: Data flow of flits and acknowledge flow of credits on an established ring bus.

8.3.1.1. QoS Support

In section 6.2, a concept for QoS support is introduced. This concept provides hard guarantees for throughput and latency of end-to-end connections. The second layer network is designed to support this QoS transmission scheme[9]. Once a GS connection is established by transmitting a header from the source to the destination node, the bandwidth of the connection can be calculated according to equation 8.1, when a ring bus is on the path. Due to the known round-trip time of an established ring of size N, the worst case latency is also known.

It is obvious that the bandwidth of each connected component and each VC is dependent on the size N of the ring. The same applies for the latency. Consequently, the size of a ring bus is limited by the guarantees given for throughput and latency. As long as the bandwidth of a ring bus port $BW_{rb,port}$ is greater or equal to the bandwidth of a normal link BW_{link}, the guarantees of GS connections will not be violated when using a ring bus. According to equation 8.1, $(W_{cont,SLN} \cdot 2 \cdot C) \geq (N \cdot BW_{link})$ must be met in order to comply with the guarantees of the primary NoC.

[9]Currently, only service level 1 is supported for guaranteed service communication through the second layer network.

8.3.1.2. Alternative Usage

The original purpose of the second layer network is the replacement of faulty routers. Therefore, the redundant components of the SLN are configured on demand to bypass faulty routers. However, if no defects are present in the architecture, the SLN is not required. Power gating techniques [133, 4] may be used when components of the SLN are disabled. This would ensure that no additional power is consumed during normal operation by idle SLN components.

However, as an alternative to just disable SLN components that are currently not required for error treatment, they can also be used for other purposes. Two approaches for alternative usage of the SLN components exist.

The first approach addresses power saving by using the slim second layer network, instead of the primary NoC in regions where the processing elements are disabled. A single router or groups of routers and the attached tiles can be power gated in case of low workload or to optimize the heat distribution in the architecture. A ring bus can be established around the disabled region to redirect the traffic transparently, which would cross the region otherwise. An example of a disabled region of 3x3 nodes is provided in figure 8.9. The routers within the primary network do not need to be aware of disabled neighboring routers. The second layer network will bypass them automatically. The decision making for disabling routers of the primary network in addition to the cores attached to them is the duty of the operating system. It can take the load situation of the primary NoC into account for decision making by accessing the communication monitors, introduced in section 5.5.3. If the load in a region is lower than the bandwidth provided by a ring around the respective region, the primary routers can be disabled. Equation 8.1 can be used to verify this. The power saving potential of the second layer network is investigated in section 8.3.3.

The second approach for alternative usage of the second layer network addresses an operation in parallel to the primary network. It is inspired by the *express virtual channel (EVC)* concept, presented in [144]. EVCs enable to bypass the complex pipeline of packet switching routers for packets that must take multiple hops in the same direction. The presented results show significant improvement in power consumption, network delay and throughput. The second layer network could be extended and used to establish EVCs dynamically at run-time. Especially for data transmission between distant nodes, the SLN can be used to reduce latency and power consumption for data transmission. Moreover, express virtual channels, using SLN components could increase the bandwidth of the communication infrastructure on demand. Online monitoring can be used to dynamically decide at run-time the length and position of EVCs according to the current load of the primary network. This approach is currently examined in the context of a student work [Wie14a], but is not further described now.

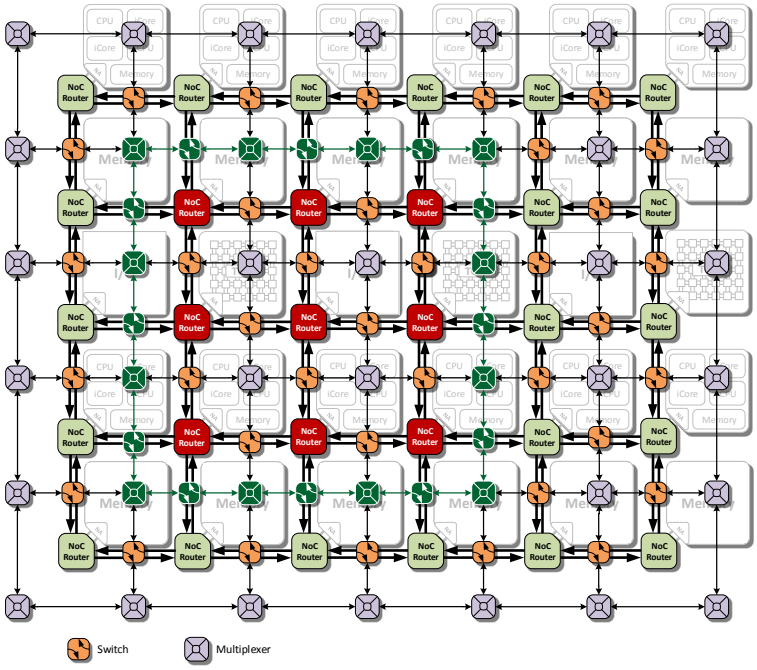


Figure 8.9.: A large ring bus of 3x3 can be established to save energy by disabling the routers enclosed by the ring [Ste13].

8.3.2. Hardware Implementation

The second layer network consists of two types of components, *switches* and *multiplexers*. These components must be flexible in order to establish any shape of ring on demand. Their implementation is described in section 8.3.2.1 and 8.3.2.2. The distributed configuration scheme for the SLN is described in section 8.3.2.3, providing a software interface for run-time configuration of the SLN.

8.3.2.1. Switch

The switch is responsible for injecting and ejecting the data into and out of an established ring bus. The general structure of a second layer network switch is shown in figure 8.10. New flits, which are forwarded by a router to an SLN switch, are stored in the *buffers* first. A separate buffer is used for each virtual channel. However, the size of the buffers can be kept very small to minimize the implementation cost of the second layer network. Buffers are required for

two reasons: they are (1) used to keep the ring busy in case of load fluctuations and (2) to gather head flits that are rejected by the router at the output of the ring[10]. In order to enqueue headers, that have been rejected, the buffer must support writing to both of its ports. Thus, the buffer operates as a FIFO for the port attached to the router output and as a LIFO for the port attached to the SLN or ring. The routers of the primary network must be extended by an enable signal for each VC. The signal is needed to indicate whether space is available in the buffers. If no space is available, the respective VC is not scheduled by the router.

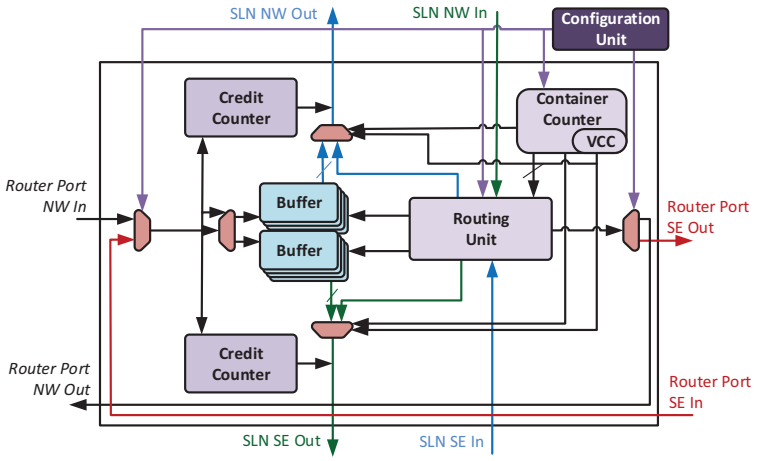


Figure 8.10.: Structure of the second layer network switch with input buffers, routing unit and counters for containers and credits [Dud14].

Head flits are analyzed by the switch, in order to extract the destination address and to store it in a look up table. The destination address is used by the routing unit of the SLN switch to decide where to eject the flits. Therefore, the destination address of the packet is taken from the LUT and added to the container, which is used to transport the flits of the packet on the ring. Counters are used, in order to manage the access to the containers of the SLN. These counters are configured when a ring bus is established. The maximum counter value is set according to the ring bus size. Separate counters are used for each virtual channel. These counters indicate the point in time, where data of a respective VC can enter the ring. Whenever the counter value is 0, a container is designated for the respective VC. Then, the flit is taken from the VC buffer and inserted into the container. The containers rotate cycle by cycle on the ring. At any point in time, where no

[10] Head flits might be rejected by the router at the output of the ring due to occupancy of the respective VC by an other packet. Consequently, the head flit remains in the ring and is returned to the input switch, if containers are shared between multiple VCs.

counter value is equal to 0, containers from other switches pass by at the SLN switches. A *request* flag in each container indicates whether the container is in use, or empty.

In each cycle where a full container from another switch comes by, the destination address of the container is processed by the *routing unit*, shown in figure 8.10. The routing unit determines whether a flit shall be ejected from the ring at this switch or left there to be forwarded to the next switch. The routing algorithm in the routing unit is implemented as an FSM. It must reflect the routing, used in the primary NoC. In the current implementation of the SLN, dimension ordered XY routing is used. In addition to the destination address of the packet, the routing algorithm takes the position of the switch in the current ring and the dimensions of the ring into account. This additional information is obtained from the *configuration entity*, which contains the current ring bus configuration data. More details about the routing algorithm and its implementation are provided in [Ste13].

In addition to the flits, the flow control credits must be transferred to the ring bus and vice versa. The credits that are returned by the receiving router are aggregated by the *credit counters* in the switch, which is attached to the router. Once the appropriate container passes by, the credit counter values are stored in the container and the credit counter is reset. A separate field is available for each virtual channel in each container. At the designated switch, the credits are extracted from the containers and are returned to the router attached to it. Therefore, the acknowledge signal to the router is set by the switch accordingly. A packet transfer is finished once the last credit is returned to the sending router. At this point, the LUT entry for the packet is erased so that it can be used for further transmissions.

8.3.2.2. Multiplexer

The multiplexers take over the task of connecting the switches flexibly to form the desired ring bus. Each multiplexer has four ports that can be connected arbitrarily[11]. This enables to establish any shape of a ring. The multiplexers only consist of combinational logic. As the name of the module suggests, it is internally built from a variety of multiplexer circuits that connect the inputs and outputs. These multiplexers are controlled by the configuration vector, as described in detail in section 8.3.2.3. Registers are only inserted between adjacent multiplexers at the border of the architecture in order to break up the combinational path and to avoid combinational loops.

[11] However, it is not intended that the data leave the multiplexer through the same port where they have entered it.

8.3.2.3. Configuration

The second layer network must be configured at run-time according to the fault scenario. The scheme from section 8.2 can be used to localize a faulty router. Once the faulty router is identified, the bypass must be configured without using the defective router for communication and configuration. Consequently, a decentralized and flexible configuration scheme is required. The scheme must tolerate faults in the primary network before they can be hidden by the bypass.

The second layer network is organized in so-called *configuration entities*. Such a configuration entity is highlighted in figure 8.11. It consists of one router of the primary network, two SLN switches and one multiplexer. All components that form a configuration entity have the same address, which is given in X and Y coordinates. The coordinates of the SLN components are used for configuration, as later described, in detail. Configuration data are first written to the memory mapped registers of the router, described in appendix A.4. The content of these register is forwarded to the switches of the same configuration entity. The SLN configuration registers are summarized in table A.7.

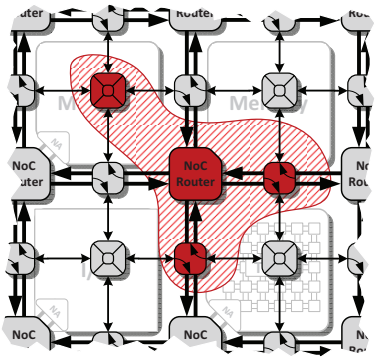


Figure 8.11.: Ring bus addressing scheme used for configuration. The components colored in red represent one configuration entity [Ste13].

Once the configuration data have been forwarded to the second layer network through a switch, they must be forwarded to the component that requires configuration. This is not necessarily one component belonging to the same configuration entity. To do so, the configuration data for the switches and multiplexers of the second layer network can be transferred in a line or column. Configuration data are forwarded cycle-wise from component to component. This configuration scheme is shown in figure 8.12. The router, highlighted in green, can configure each of the switches and multiplexers of the yellow colored row or the blue colored column. Consequently, including a faulty router in a configuration process

can be easily avoided because each node in the same row or column can be used for configuration. In order to establish a complete ring, at least two different rows and columns must be configured.

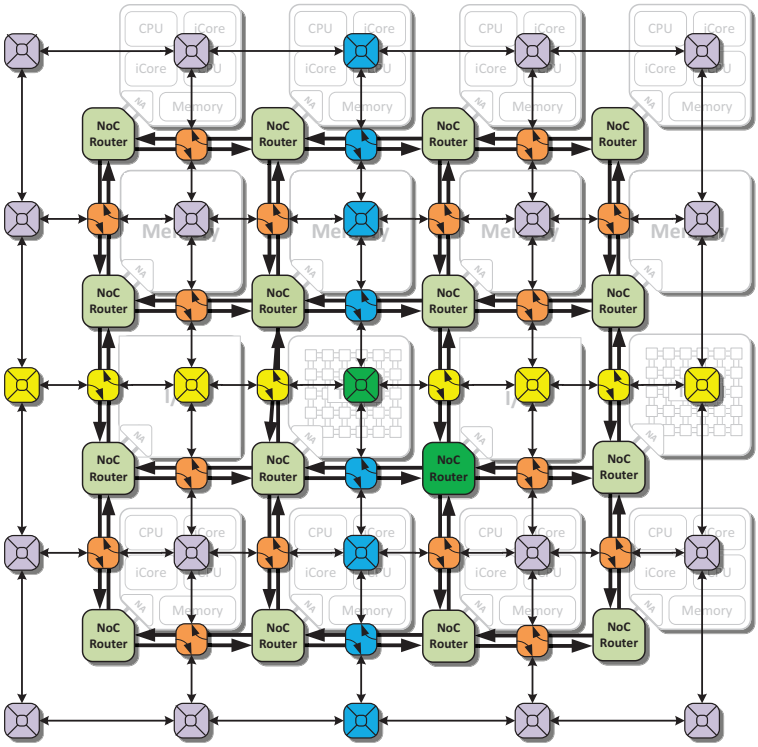


Figure 8.12.: Row and column wise configuration scheme of the second layer network. The router highlighted in green is able to configure the colored switches and multiplexers [Dud14].

The configuration data are broadcasted in a row or column. In order to identify the component to be addressed, a destination address field with X and Y coordinates is included in each configuration message. Each message has a size of 32 bit[12]. The configuration unit in each multiplexer or switch compares the destination address with its own address and processes the configuration data if both addresses match. Different types of configuration messages are used to transfer a complete configuration for a switch. The message type is defined by another field, which is included in each configuration message. Four messages are required to configure

[12]The size of the configuration message can be defined at design-time. The value of 32 bit reflects the current configuration that is also used for evaluation.

one switch. These messages include the ID of the corrupted router port, the size and dimensions of the ring bus and the *activation point*. Configuration of a multiplexer only necessitates a single configuration message. Additional details about the structure and protocol of the configuration messages are provided in [Dud14].

The individual components, forming a ring bus, are configured sequentially. However, the ring bus can only operate, if all components are properly configured. Consequently, the activation of a ring must be synchronized. To do so, *synchronized counters* are used. One of these counters is available in each configuration entity. Due to the synchronized reset of the entire architecture, all counters run synchronous during operation. During configuration of a ring, the current counter value $T_{current}$ is accessed through memory mapped registers, described in detail in table A.7. Based on the current count, the activation point T_{active} is determined as follows:

$$T_{active} = T_{current} + t_{conf,wc} \tag{8.2}$$

The configuration latency $t_{conf,wc}$, must reflect the worst case delay for configuration of all components of a ring in order to ensure that all components are configured successfully prior to activation. Once the activation point T_{active} is reached, all components are activated in the same cycle. Therefore, the activation point T_{active} is transferred via a configuration message for the SLN switches[13]. In the switch, T_{active} is compared with $T_{current}$ and the configuration is activated when both values are equal.

The entire configuration process is managed by software. It determines the size and position of the ring according to the detected fault, generates the configuration messages accordingly, determines T_{active} and writes the configuration registers subsequently. The software implementation is described in [Dud14].

8.3.3. Evaluation

The components of the second layer network were implemented using SystemVerilog. The HDL model of the NoC (see section 5.3) was extended by an SLN. The test environment for the HDL model of the NoC was extended to support the configuration of the second layer network[14].

[13] The multiplexers can be activated directly after configuration as long as the configuration of the switches is synchronized.

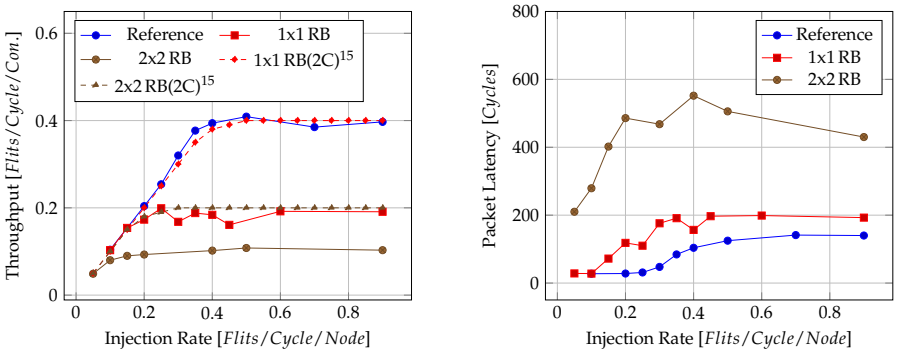
[14] In addition, an FPGA prototype of a 2x2 mesh with SLN support was implemented as a proof of concept. Therefore, the architecture, introduced in appendix A.1, was used.

An 8x8 mesh NoC with a second layer network is used to evaluate the impact on performance, when establishing a ring bus. The primary NoC and the second layer network components have a link size of 32 bit. One container is available per direction at each switch of the SLN. Consequently, each container is shared between two of the four VCs that are present in the network. Uniform random traffic is used for evaluation, because it is well suited to expose the heterogeneity that is introduced by an established ring bus. Three different configurations are compared. The reference (*Ref*) is a NoC where the SLN is completely disabled and is compared with two fault scenarios. In the first scenario (*1x1 RB*), a single router in the center of the architecture is assumed to be faulty. Consequently, a bypass is configured by establishing a ring bus around the faulty router. A second faulty scenario (*2x2 RB*) assumes four neighboring routers to be defect. These routers are also located in the center of the architecture and a ring is established accordingly. Figure 8.13 summarizes the results of the performance analysis of the three configurations for different injection rates.

The centering location of the faulty router(s) results in a worst case scenario in combination with uniform random traffic. Packets, which have to travel a longer distance in the network, must bypass the faulty router with a high probability. Consequently, the load on the ring bus is expected to be very high. The throughput measurements for the three aforementioned configurations are provided in figure 8.13(a). For low injection rates, a 1x1 ring bus does not impact the throughput. However, a centric ring bus with a single container per direction (*1x1 RB*) impacts the achievable throughput for random traffic significantly when the injection rate is high. This is due to the fact that an overloaded ring results in back pressure and impacts the overall throughput for uniform random traffic. For the *1x1 RB* configuration, the achievable throughput is approximately halved, compared to the reference, and fits the expectations. The achievable throughput can be calculated by the use of equation 8.1 for the current configuration with $C = 1$ and $N = 4$. If a 2x2 ring with one container per direction is established, the throughput is reduced to one quarter of the original throughput. These measurements fit to the calculated bandwidth, when using equation 8.1 with $C = 1$ and $N = 8$. However, further investigations have shown that the impact on throughput is much lower if the faulty router is located at the border of the architecture. A more detailed investigation can be found in [Ste13]. Another way to avoid performance reduction is to increase the number of containers at design-time, as previously described. In order to demonstrate this principle, the estimated throughput[15] for the *1x1 RB* and *2x2 RB* configuration, now with two containers per direction, is also provided in figure 8.13(a). The results for the *1x1 RB(2C)* configuration show that it reaches the same bandwidth as the reference design. The bandwidth of the *2x2 RB* configuration could also be doubled, when using two containers instead of one. Half of the throughput of the reference could be achieved for the *2x2 RB(2C)* configuration, which fits to equation 8.1 with $C = 2$ and $N = 8$. This

is a doubling of the bandwidth compared to the *2x2 RB* configuration, with only one container per direction.

However, it must be mentioned, that other traffic scenarios with a communication-aware mapping of tasks will reduce the amount of global communication through a faulty region. This in turn reduces the amount of communication through the ring and consequently the negative impact of its potentially lower bandwidth, depending on the configuration.



(a) Throughput of an 8x8 mesh

(b) Average latency of an 8x8 mesh

Figure 8.13.: Throughput (a) and latency (a) of an 8x8 mesh NoC under uniform random traffic. A *reference* is compared against a configuration where a 1x1 ring bus (*1x1 RB*) and one where a 2x2 ring bus exists (*2x2 RB*). Additional throughput estimations are provided for the same ring bus configurations but with two containers (*2C*) per direction instead of one.[15]

The latency was measured for the same scenario. The results are shown in figure 8.13(b). A small ring bus for one faulty router does not impact the latency for small injection rates. However, if the injection rate increases, the ring bus configuration has an increased average latency; compared to the reference. The higher latency mainly results from the bandwidth limitations of the ring. The results for a 2x2 ring bus show that a bigger ring has a much higher impact on latency, as opposed to the smaller one. However, it must again be mentioned that the traffic scenario considered here is a negative corner case. A traffic scenario with a higher amount of local communication is less impacted by a ring bus with

[15]The values for throughput of the *1x1 RB(2C)* and *2x2 RB(2C)* configurations with two containers per direction have been estimated using the measurements for single containers and the bandwidth calculations from equation 8.1. The reason for estimating these values instead of measuring them, are limitations of the SystemVerilog implementation. It does currently not support multiple containers per direction.

respect to latency. Increasing the number of containers in the SLN at design-time is one way to reduce the latency when a ring bus is established.

In addition to the performance evaluation, the implementation costs of the second layer network are evaluated. Therefore, one configuration entity, presented in figure 8.11, was synthesized using a 45 nm *TSMC* standard cell library. It is compared against a single router of the primary network. A 32 bit and a 64 bit variant of the router and SLN components are synthesized with different settings for the number of virtual channels. One container is used per direction in the SLN[16]. The synthesis results indicate that the critical path of the network is not impacted by the SLN[17]. However, the SLN components surely impact the area requirements, as shown in figure 8.14. 10 % additional area is required, if a NoC with four VCs and a link size of 32 bit is equipped with a second layer network. For a configuration with 8 VCs, the implementation overhead is around 14 %. Adding an SLN to a NoC with 64 bit links and 4 VCs results in 15 % area overhead. If eight virtual channels are used, 18 % additional area is consumed when adding SLN components. The synthesis results show that the area overhead of the SLN is very low compared to spatial redundancy techniques, introduced in section 2.5.3. Spatial redundancy, which would duplicate or triplicate the entire router, would increase the area overhead of each router by a factor of 2 or 3 respective. Consequently, the presented concept is an efficient way to hide permanent errors and tolerate defective routers.

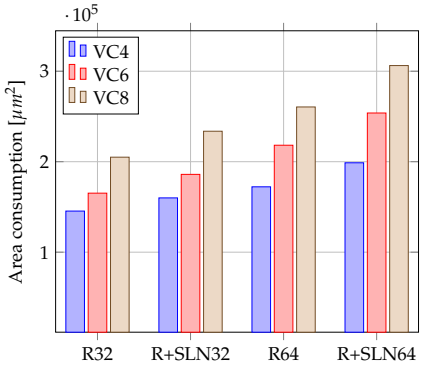


Figure 8.14.: Area consumption of a single router with 32 bit links (*R32*), a 64 bit router (*R64*), a 32 bit router with one SLN configuration entity (*R+SLN32*) and a 64 bit router with SLN components (*R+SLN64*).

[16] The number of containers might be increased in order to raise the bandwidth of the SLN. The area overhead of the SLN is approximately proportional to the number of containers.

[17] A clock frequency of 1.4 GHz was targeted and achieved for the 32 bit router with SLN.

The standard cell implementation of the router and SLN components is also used to evaluate the power consumption and power saving potential of a ring bus. Therefore, the power consumption of a single router of the primary network and an SLN entity are obtained from a synthesized netlist. Two power measurements are performed - the *idle* power consumption without any load and the *active* power consumption for a realistic traffic scenario with high utilization. The results are summarized in table 8.2. Compared to the primary network, the SLN components have a lower power consumption. An idle SLN entity consumes only 2.08 mW. This exhibits the low energy footprint of an idle second layer network. Regardless, power gating can be used to further reduce the power consumption of idle switches and multiplexers.

Component	**Power** (idle) [mW]	**Power** (active) [mW]	**Power Saving**[18] [mW]
Primary router	7.39	12.45	-
SLN entity[19]	2.97	5.18	-
1x1 ring	5.94[20]	10.37[20]	2.08 (16.7 %)
2x2 ring	11.88[20]	20.74[20]	29.06 (58.4 %)
3x3 ring	17.82[20]	31.10[20]	80.93 (72.2 %)

Table 8.2.: Power consumption of second layer network components and ring bus configurations. *Power saving* shows the total savings when disabling parts of the primary NoC and using the SLN instead.

The low power consumption of the second layer network justifies its usage for power saving, as introduced in section 8.3.1.2. If the utilization of the primary network in a region is low and the cores are idle or power gated, the SLN could be used to bypass the respective region. Thus, the primary network can be disabled to save power. The throughput thresholds for disabling parts of the primary network can be determined using equation 8.1.

In table 8.2, the power saving potential is investigated for three regions with different dimensions. Disabling a single router can save 16.7 % of the power that would have been consumed by the router. However, the power saving potential increases if the region becomes bigger. When a 3x3 region of the primary network is disabled and a ring bus is established to bypass the region, 72.2 % of the power can be saved that would have been consumed by the 3x3 routers of the primary network. These results justify the huge power saving potential of the second layer network.

[18] Power saving when disabling routers in a region and bypassing them with an activated ring bus.
[19] The power values refer to one configuration entity, consisting of two switches and one multiplexer.
[20] This value is an estimation, based on the measurements of a single configuration entity, consisting of two switches and one multiplexer.

8.4. Summary

This chapter addressed a fault tolerance concept for the proposed network on chip. Existing work on fault tolerance was discussed and summarized in section 8.1. Permanent errors were identified to be the most challenging. However, they will become relevant for future technology nodes. Consequently, permanent errors are in the scope of the concepts, presented in this chapter.

In section 8.2, the concept and implementation of a fault detection and localization scheme was described in detail. The scheme addresses the complex router design that was elaborated in the previous chapters. It uses test patterns that are generated in software in order to localize permanent errors in the network on the granularity of routers. The analysis of the received test patterns is also implemented in software to reduce the hardware overhead for localization. A parallel approach is taken to minimize the run-time for localization and to ensure scalability of the concept. A detailed evaluation of the proposed scheme was presented, which uses fault injection on the granularity of standard cells in ASIC netlists. The presented results showed that 51 % of all faults in a complex, QoS supporting router design could be detected and localized. Additional analysis exhibited that 100 % of the link errors could be detected and localized.

The localization of faulty routers is subsequently taken as a basis for the error treatment scheme, introduced in section 8.3. It allows to disable one or multiple routers completely that were identified to be faulty. A so-called second layer network was introduced to bypass faulty routers transparently and to substitute the lost bandwidth. A distributed configuration scheme was presented in order to set up the second layer network according to the detected errors. Depending on the design-time configuration of the second layer network, it can be used to bypass a faulty router with a negligible impact on performance. The investigated second layer network is accompanied by an area overhead of 10 % to 18 %. Thus, the implementation overhead is very low compared to spatial redundancy techniques, which duplicate or triplicate the functionality of the entire router.

The second layer network can alternatively be used to disable complete regions of the primary network in case of low utilization, as discussed in section 8.3.1.2. When using the second layer network to disable idle regions of the primary NoC for power saving, up to 72.2 % of the power of the communication infrastructure in a $3x3$ region could be saved.

9. Conclusion & Future Work

9.1. Conclusion

The continuous increase in the integration density of semiconductors enables ever-more-complex systems. This trend was previously used in order to improve the performance of an architecture, by increasing the complexity of the micro-architecture. Moreover, the increase of the integration density was accompanied by an increase of the clock frequency of an architecture. The increased clock frequency and architectural complexity were considered to be the state of the art techniques for performance improvement. Interestingly enough, architectural improvements could no longer be used to significantly raise the performance. In addition, physical limitations prevent designers from further increasing the clock frequency. Consequently, the performance of a single core can no longer be significantly increased.

In order to increase the performance of a system, regardless of these limitations, the number of components was increased. Systems on a chip that integrate multiple components on a single chip resulted from this development. As a subgroup of these systems, multi-core architectures can be considered. Such architectures combine multiple processing cores on a single piece of silicon. Multi-core and even many-core processors are considered today as the prevailing architecture in all areas where a high computation power is required.

However, an ever increasing number of cores in an architecture includes various challenges. From the software perspective, new concepts for programming and management of such parallel architectures are required. Architecturally speaking, the communication requirements increase with the number of components. As such, shared buses can no longer fulfill these requirements. Consequently, the parallelism with respect to the number of cores must be reflected by the communication infrastructure.

Networks on chip have emerged as the method of choice to cope with the communication demands of large many-core architectures. In contrast to bus systems, they enable parallel and scalable communication. However, this new paradigm for on-chip communication includes a variety of challenges and requirements:

- The communication subsystem must be scalable in order to enable efficient communication for a large number of cores or components.
- Quality of service comes to the focus when the communication infrastructure is used by a large number of components that may have different communication demands.
- Power consumption and thermal issues must be addressed to cope with the limitations of future VLSI technologies.
- Management of the communication system becomes more complex due to the increasing complexity of networks on chip.
- Dependability aspects must be considered for future technology nodes, addressed by the NoC paradigm.

All of these challenges must be considered when designing a NoC for future many-core architectures.

In this context, the present work is an attempt to provide a holistic and versatile approach for NoC-based on-chip communication. As a starting point, a NoC was designed, which combines concepts and techniques that are considered as state of the art for scalable on-chip communication. This NoC uses wormhole packet switching with virtual channels, credit-based flow control and distributed routing. This base architecture was implemented as a cycle-accurate SystemC simulation model and as a synthesizable HDL model. Both models are designed to be easily customizable and extendable. Numerous parameters enable an adaptation of the developed network on chip at design-time. A semi-automated evaluation and design flow is proposed in order to select an appropriate configuration of the design template, with respect to a short time to market. The combination of the SystemC and the HDL model enables fast and accurate evaluation of performance and technology related aspects; such as throughput, latency, power consumption, area requirements and clock frequency. The base architecture with its wide-ranging evaluation capabilities is used as a starting point. It is extended towards the goal of a holistic concept for scalable on-chip communication.

An essential contribution of this work is the development of a quality of service scheme, which enables adjustable hard guarantees for point-to-point communication. The bandwidth and latency of an end-to-end connection between two nodes can be specified at run-time according to the requirements of the application. Different service levels and the sharing of communication resources between best-effort traffic and guaranteed service connections assist in optimizing the bandwidth utilization in the NoC. A QoS management scheme enables to define policies, which regulate the amount of resources to be used by GS and BE traffic in different regions of the architecture. A rerouting scheme for GS connections is proposed and evaluated in order to fulfill the routing requirements of such durable connections. The scheme detects overload situations by the use of online moni-

toring and balances the load by rerouting existing GS connections. However, the QoS scheme for GS connections is limited by the number of connections per node. In order to overcome these limitations, a second concept for QoS communication is proposed. This concept can be used to establish so-called virtual networks in a region of the architecture. A VN can then be used exclusively by an application in such a way that the communication is not affected by other communication crossing the respective region. This enables to execute a distributed application in an environment with a quasi-exclusive network on chip.

In order to address the power limitations and the closely related thermal issues of future VLSI technologies, different strategies for power optimization are introduced and evaluated in the scope of this work. Two self-optimization schemes are investigated, establishing end-to-end connections transparently, in order to optimize performance and power consumption. A fine grained self-managed power gating scheme for the buffers of the NoC routers is built and evaluated. It uses run-time monitoring to decide about the number of buffers to be disabled. In addition, a concept is proposed that enables to power off complete regions of the network on chip. A deactivated region is bypassed transparently by the use of an additional network layer.

The additional network layer is not only used for power saving, but also for the dependability concept, introduced in this work. This concept focuses on detection, localization and treatment of permanent faults. Such faults are expected to play an increasing role for future technology nodes, due to the increased impact of manufacturing process variations and aging of the device. A software-based scheme was implemented, which uses special communication patterns for error localization. A main challenge is the complexity of the network, making it difficult to achieve a high fault coverage. However, the localization method allows to identify a faulty router in the network for an extensive range of errors. The localization is used as a basis for error treatment. A light-weight configurable network layer is built to bypass a single or multiple faulty router transparently. The scheme enables to tolerate multiple permanent errors in the architecture without restrictions on their distribution.

The proposed concepts for quality of service, fault tolerance, power-optimization and self-optimization are all founded on the same base architecture. In conjunction with one another, they meet the goal of a solution for future many-core architectures. Using the proposed semi-automated evaluation and design flow, an architecture-specific network on chip with a subset or a complete range of the proposed functionality can be generated and analyzed efficiently. This enables the use of the proposed architecture for fast building-block-based system design.

9.2. Future Work

The proposed network on chip is a versatile and comprehensive basis for the design of future many-core architectures. The semi-automated design and evaluation flow enables fast requirement analysis and easy generation of specific NoC-incarnations. Thus, the presented communication infrastructure provides a variety of opportunities for further work.

A discussion of possible extensions and improvements with respect to individual aspects and mechanisms was previously provided in the respective chapters and sections. In the following, more general and elementary extensions of the proposed concept are discussed.

In order to meet the general requirement of a short time to market, the proposed semi-automated design flow is a good basis. However, in order to further accelerate the design process of NoC-based architectures, a full automation of the design flow is desired. In order to achieve this goal, a fully automatic derivation of a valid configuration of the NoC design template from abstract requirements is needed. In addition, an automatic analysis of the fulfillment of defined requirements by a specific configuration is necessary. With the aid of these extensions, a fully automated search of the design space for the most suitable configuration of the design parameters of the network could be implemented. Such a design flow would enable to automatically create a suitable simulation model and a netlist by defining the communication requirements of the desired system.

The architecture proposed in this work claims to be a holistic approach for on-chip communication. However, the trend is not explicitly taken into account, in regards to 3D integration. In fact, the proposed concepts for quality of service, power saving, self-optimization and dependability can be easily extended to support a 3D integration and a 3D NoC topology. Nevertheless, support for 3D topologies is left open for future work because it has minimal research aspects and essentially can be considered as implementation work.

In addition to the packet switching communication, which is in the scope of this work, circuit switching is promising for energy efficient communication. Some of the concepts presented in this work, such as rerouting, Auto-GS or the second layer network, can also be applied to a circuit switching network. The circuit switching extension, introduced in section 5.5.2, can be used to adapt and evaluate the aforementioned concepts in the context of circuit switching. Moreover, the hybrid approach for a combination of packet and circuit switching opens extensive research opportunities. The run-time mapping of communication flows to packet and circuit switching communication is an interesting topic for future research, especially with regards to their QoS requirements and energy footprint. The architecture presented in this work provides an ideal starting point for these studies.

This work mainly focuses on the hardware perspective of networks on chip. However, the software-based management of the presented concepts and features by an operating system or application is providing adequate space for further investigations. The proposed fault detection and localization scheme could be embedded into a distributed operating system. As a reaction to localized errors, the OS should support the configuration of the second layer network in order to ensure continuous operation of the network. The proposed fault injection method could be used to evaluate the dependability concept on an FPGA prototype with a complete software stack comprising an OS and applications. Additionally, the proposed second layer network concept enables the shutdown of routers, in the primary network, to save energy. This mechanism provides great potential for power optimization and thermal management in the context of dark silicon. In combination with the proposed monitoring infrastructure, a power management scheme can be carried out in the OS of the architecture.

An application driven power management of GS connection is enabled by the connection freezing scheme, introduced in section 7.4.1.1. Power gating decisions could be generated at compile-time in order to manage the power gating of buffers at run-time. The compiler could generate power gating hints during the serialization phase of the data, which occurs prior to a communication period. This allows to shut down GS connections and reactivate them for data transmission without violating the provided guarantees.

The communication monitoring infrastructure, proposed in this work, is currently used for hardware-based self-optimization. However, it also enables software-based optimization. The load of the communication infrastructure could be taken into account by the operating system during allocation of computation resources and mapping of tasks. If the communication behavior of the application is known in advance (e.g. by the use of profiling), an optimized allocation and mapping could be done. Balancing of the NoC load, reduction of the NoC power consumption or latency reduction for a certain application may be possible optimization criteria for task mapping when taking the current load situation of the NoC into account by the use of monitors.

The proposed network on chip will be continuously used in the Transregional Collaborative Research Center 89 “Invasive Computing” [244]. Most of the future work, discussed in this section, will be addressed in the second phase of this research project [80]. Thus, the presented NoC architecture forms the basis for further research.

A. Appendix

A.1. FPGA-based Many-Core Architecture Prototype

An FPGA prototype is on the one hand desired for verification, on the other hand it enables investigation of real applications. Thus, an FPGA-based prototyping of an MPSoC was carried out on the *Xilinx ML605* development board. This board contains a *Xilinx Virtex-6 LX240T* FPGA.

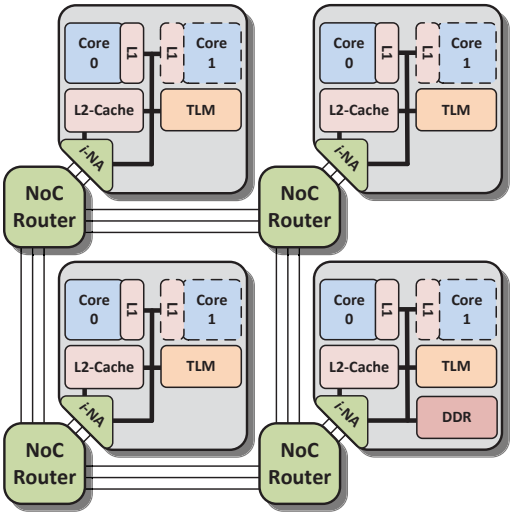


Figure A.1.: FPGA-based architecture prototype carried out on the *Xilinx ML605* development board.

The general structure of the MPSoC prototype is given in figure A.1. It consists of an incarnation of the proposed NoC template design. Table A.1 summarizes the most important parameter setting of the instantiated NoC. A description of the parameters is provided in appendix A.6.

Each of the NoC routers connects one of the RISC core tiles, developed for the InvasIC architecture (see section 3.2.1.1). One i-NA instance is used in each tile,

Setting	Value
G_FLIT_SIZE	32
G_VCS	4
G_TS	8
G_VC_BUFFER_DEPTH	4
G_CTRL_ENABLE	1
G_CTRL_FLIT_SIZE	16
G_CTRL_BUFFER_DEPTH	2

Table A.1.: Parameter settings of the NoC instance used for the FPGA prototype.

as described in detail in section 3.2.2.1. The tile with the highest ID[1] is typically carried out as a combination of RISC core and memory tile (see section 3.2.1.4)[2]. The prototype uses the memory map, defined in appendix A.3, and contains the NoC router registers, defined in appendix A.4.

Due to the complexity of the architecture, the realizable size is quite small due to the limited resources of the FPGA. The number of LEON3 cores per tile is either set to 1 or 2, depending on the current requirements on the prototype. The number of tiles is varied between two and four. The main limiting factor, preventing further extension, are the available block RAMs of the used FPGA. Thus, the size of the caches and memories is very limited. For example, if the number of tiles is increased from two to four, the size of the TLMs has to be halved. However, a large TLMs is required to store the executable and to have some additional space left for frequently accessed data. Thus, the sizes of the L1 and L2 caches had to be strictly limited. This can be seen in table A.2. It gives an overview on the important settings of the memories and caches of the prototype.

Setting	Value
L1D Cache (Sets, Linesize, Lines)	2, 4, 4
L1I Cache (Sets, Linesize, Lines)	2, 8, 8
L2 Cache (Sets, Linesize, Lines)	2, 32, 8
Tile Local Memory	256 / 512 kB
DDR3 Memory Size	1024 MB

Table A.2.: Settings of the tile-internal components of the FPGA prototype.

[1] The tile IDs are assigned from left to write, row by row, downwards: $TILE\_ID = X_{coord} + Y_{coord} \cdot X_{size}$.

[2] This tile is employed as a processing tile during evaluation due to the limited number of tiles available on the prototype.

The clock frequency of the prototype is set to 50 MHz. This is not the maximum achievable clock frequency, but the performance is not the main objective of the prototype. A two-tile design with two cores per tile and 512 kB of TLM consumes only 17 % of the slice registers available on the *LX240T* FPGA. 51 % of the FPGA LUTs are required for the design. The limiting factor is the block RAM utilization, which is 85 % for the two-tile prototype.

A.1.1. Scalable FPGA Prototype

Using a single FPGA board for prototyping has strong limitations with respect to the size of the architecture. The prototyping system, introduced in section 3.4.2, can be employed to build a larger architecture. However, it is also limited to a fixed number of FPGAs and cannot be extended. A scalable prototyping concept is required to enable prototyping and evaluation of large many-core architectures, which are addressed by this work.

Such a concept was implemented and evaluated in [Hau14]. It uses multiple cost-effective off-the-shelf FPGA boards. Each FPGA board is used to implement one or multiple tiles, as shown in figure A.2. The NoC that is used to connect all tiles, must be distributed among all the FPGAs. Consequently the FPGA boards have to be connected to each other to spread the NoC. Unfortunately, standard FPGA boards do not have many free general-purpose I/O pins, which can be employed for inter-FPGA communication. However, modern FPGAs have high speed serial transceivers [64] that are available directly on an FPGA board or by the use of extension boards. These transceivers enable high data rates of up to 16.3 Gb/s per transceivers [64]. Large Virtex 7 FPGAs from Xilinx have up to 120 of these transceivers.

In [Hau14] the GTX transceivers of the *Xilinx ML605* development board are used to build the NoC links between different boards. The number of GTX transceivers of the Xilinx ML605 development board is high enough to establish a mesh topology between the FPGA boards. This enables a scalable prototyping which is only limited by the number of available FPGA development boards. The general principle of this prototyping approach is shown in figure A.2. The approach is a cost-effective alternative to the use of expensive prototyping systems, such as the CHPIit system introduced in section 3.4.2. Especially in the university environment, where FPGA development boards are widely used, the concept can be applied to build large prototypes.

To implement the proposed concept, all signals of the physical interface of the NoC routers must be transmitted through the GTX connection. Therefore, a wrapper was implemented in [Hau14]. In addition to the serialization and deserialization of data, the wrapper enables fault detection and correction. This is necessary because data might be corrupted while transmission. Consequently, a transparent

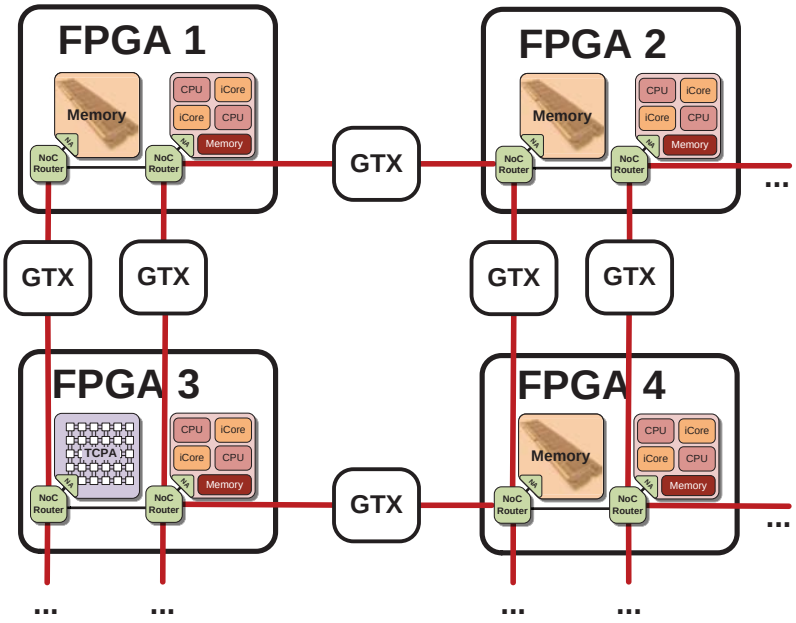


Figure A.2.: Structure of a scalable multi-FPGA prototype of a NoC-based many-core architecture.

fault detection and retransmission scheme was implemented to ensure an error-free data transmission. The wrapper and its retransmission scheme as well as the physical interface of the GTX transceivers come along with additional delay. The implementation overhead and the additional latency for inter-FPGA communication through the NoC is evaluated in [Hau14].

A

A.2. Service Level Assignment Algorithm

In section 6.2.1.5, a QoS-aware task mapping flow was introduced. This flow contains steps for service level assignment and GS connection reservation. The algorithm A.1 is utilized to calculate the optimal service levels for all GS connections using the communication requirements, defined for a distributed application.

In the first phase of the algorithm, shown in algorithm A.1, initializations take place. The connections of all nodes are added to a list. Subsequently, the connection list is sorted by bandwidth requirements. In the next step, the sorted list is employed to decide which communication flows are mapped to GS and

Algorithm A.1 Pseudo code of the service level assignment algorithm.

```
1:  task_graph ← mapping_result
2:  for each node in task_graph do
3:      node.gs_vc.in = gs_vc_max
4:      node.gs_vc.out = gs_vc_max
5:      connection_list.add(node.connection)
6:  end for{Initialization}
7:  connection_list ← sort_by_bandwidth(connection_list) {Sorting}
8:  for each connection in connection_list do
9:      if (connection.src_node.gs_vc_in > 0)&
        (connection.dst_node.gs_vc_out > 0) then
10:         connection.set_type(gs)
11:         connection.dst_node.gs_vc_out ← connection.dst_node.gs_vc_out − 1
12:         connection.src_node.gs_vc_in ← connection.src_node.gs_vc_in − 1
13:     else
14:         connection.set_type(be)
15:     end if
16: end for{GS connection mapping}
17: task_graph.update(connection_list) {Task graph update}
18: connection_list.erase_be_connections() {Erase BE connection}
19: for each connection in connection_list do
20:     opt_sl ← rnd_up(connection.bandwidth/(bw_total/ts_total))
21:     if opt_sl ≥ ts_max then
22:         connection.set_sl(ts_max)
23:     else
24:         connection.sel_sl(opt_sl)
25:     end if
26: end for{Service level calculation and assignment}
27: task_graph.update(connection_list) {Task graph update}
28: for each node in task_graph do
29:     sum_in ← 0
30:     sum_out ← 0
31:     for each connection in node.connection do
32:         sum_in ← sum_in + connection.get_sl() − 1
33:     end for
34:     if sum_in > ts_total − vc_total then
35:         for each dst_node in node.dst_list do
36:             for each connection in dst_node.connection do
37:                 sum_out ← sum_out + connection.get_sl() − 1
38:             end for
39:             if sum_out > ts_total − vc_total then
40:                 reduce_sl(connection)
41:                 sum_in − −
42:                 if sum_in <= ts_total − vc_total then
43:                     break
44:                 end if
45:             end if
46:         end for
47:         if sum_in > ts_total − vc_total then
48:             connection ← search_connection(get_sl() > 1&
                min(get_sl() · (bw_total/ts_total) − get_bw()))
49:             reduce_sl(connection)
50:         end if
51:     end if
52: end for{Time slot policy verified and adjusted}
53: return task_graph
```

BE traffic. The communication flows with the highest bandwidth requirements use GS connections. For nodes where the number of connections exceeds the limit gs_vc_max, the connections with the lowest bandwidth requirements must use BE communication. If hard real-time guarantees are required, BE communication cannot be used. Consequently, the task graph has to be designed by the application developer in a way that gs_vc_max is not exceeded for any node.

After determining the use of GS and BE communication, the task graph representing the application is updated. The communication flows mapped to BE traffic are erased from the connection list. They must not be taken into account during the following service level assignment phase. In the next step, the optimal service level (opt_sl) is calculated for each connection. It is assigned to the connection if ts_max, the limit of time slots per connection, is not exceeded. The last phase of the algorithm verifies that the service levels of all connections do not exceed the number of available time slots. If violations are encountered for a node, the service level of the connection with the lowest penalty[3] is reduced. This process is repeated until ts_max is met for all nodes.

The final task graph includes the type (BE or GS) for each communication flow as well as the service level assignments for all GS flows. This is the outcome of the proposed service level assignment algorithm. It can be employed by the operating system to setup the GS connections accordingly.

A.3. Memory Map

The address space of the InvasIC architecture is divided into a tile local (cache-coherent) range and a global (non-coherent) address range. It is a 32 bit address space due to the use of the 32bit LEON3 SPARC V8 cores [83, 84]. The local address space (0x80000000 - 0xFFFFFFFF) is used to address the tile local components such as the tile local memory or configuration registers. The global address space (0x00000000 - 0x7FFFFFFF) allows to access the shared main memory of the architecture and the distributed memory of the architecture, so the TLMs of all tiles.

The current memory map of the InvasIC architecture is given in table A.3. The range for the global shared memory contains the address space for main memory access. It is followed by the global distributed memory address range that is utilized to address the lower 16 MB of the tile local address range (0x80000000 - 0x80FFFFFF) of each tile. Consequently it is possible to address not only the TLM of each tile from remote, but also monitoring, APB, i-NA and i-NoC registers. Access to the global address range is carried out by the i-NA and the i-NoC (see

[3] The penalty is defined as the difference between the required and the guaranteed bandwidth.

Memory Access Domain	Unit	Start Address	End Address
Global shared memory	DDR0	0x00000000	0x1FFFFFFF
	DDR1	0x20000000	0x3FFFFFFF
Global distributed mem.	Tile 0	0x40000000	0x40FFFFFF
	Tile 1	0x41000000	0x41FFFFFF
	Tile 2	0x42000000	0x42FFFFFF
	Tile...	...	...
Tile local *(remote access)*	TLM	0x80000000	0x807FFFFF
	reserved	0x80800000	0x809FFFFF
	APB	0x80A00000	0x80BFFFFF
	Monitoring	0x80C00000	0x80CFFFFF
	reserved	0x80D00000	0x80DFFFFF
	i-NA	0x80E00000	0x80EFFFFF
	i-NoC	0x80F00000	0x80FFFFFF
Tile local *(no remote access)*	I/O	0x81000000	0x81FFFFFF
	reserved	0x82000000	0x9FFFFFFF
	CiC	0xA0000000	0xA00FFFFF
	reserved	0xA0100000	0xEFFFFFFF
	DSU	0xF0000000	0xF7FFFFFF
	reserved	0xF8001000	0xFFFFEFFF
	AHB P&P	0xFFFFF000	0xFFFFFFFF

Table A.3.: Memory map of the InvasIC architecture.

section 3.2.2.1). In contrast, access to the tile local address range is enabled by the tile local AMBA bus. All registers of the components of a tile are mapped to the tile local address range. As mentioned, the first 16 MB can also be addressed from remote. The rest is employed by components that must not be access from outside the tile, such as the debug support unit, CiC or AHB plug&play registers. Table A.3 shows that parts of the address range are left free (reserved) for future extensions.

A.4. Control Registers of the NoC

The invasive network on chip, introduced in section 3.2.2, provides different registers that are mapped to the local address range of each tile (see appendix A.3). Access to these registers is provided by the invasive network adapter; described in section 3.2.2.1. A dedicated connection between the i-NA and the connected i-NoC router is employed to access these registers. They are used to access status information and for configuration purposes.

Table A.4 provides the definitions for the monitoring registers of the i-NoC. They are utilized to access the information generated by the monitoring units, as detailed in section 5.5.3. Table A.5 shows the mapping of the debug registers of the routers. They can be used to read the latest flits transmitted or received at any port of the router. In table A.6 the registers for configuration of the virtual channel policies are provided. Policy configuration is described in detail in section 6.3. Other registers of the i-NoC provide counters for execution time measurements, a router revision number and ID. These registers are summarized in table A.7.

The following definitions and settings are applied in the tables:
Monitoring interval length (MON_INT): = 2^{24}
Number of VCs (VC_CNT): = 4
Size of the VC input buffers (BUF_SIZE): = 8
Ports: north = 0, east = 1, south = 2, west = 3, local = 4
Abbreviations: port (P), virtual channel (V)

Address Range	Name	Comment	R/W
0x80F00000-0x80F00013	MON_LINK_AVG	Link Utilization - AVG, Completed Sampling Period, Value Range: 0 - MON_PERIOD-1, (P0:0x80F00000, P1:0x80F00004 ...)	R
0x80F00020-0x80F00033	MON_VC_AVG	Virtual Channel Utilization - AVG, Value Range: 0 - MON_PERIOD*VC_CNT-1, (P0:0x80F00020, P1:0x80F00024 ...)	R
0x80F00040-0x80F00053	MON_VC_MAX	Virtual Channel Utilization - MAX, Value Range: 0 - VC_CNT, (P0:0x80F00040, P1:0x80F00044 ...)	R
0x80F00060-0x80F0007F	MON_BFL_AVG	Buffer Fill Level - AVG - P0, Value Range: 0 - MON_PERIOD*BUF_SIZE-1, (V0:0x80F00060, V1:0x80F00064 ...)	R
0x80F00080-0x80F0009F	MON_BFL_AVG	Buffer Fill Level - AVG - P1, ...	R
0x80F000A0-0x80F000BF	MON_BFL_AVG	Buffer Fill Level - AVG - P2, ...	R
0x80F000C0-0x80F000DF	MON_BFL_AVG	Buffer Fill Level - AVG - P3, ...	R
0x80F000E0-0x80F000FF	MON_BFL_AVG	Buffer Fill Level - AVG - P4, ...	R
0x80F00100-0x80F0011F	MON_BFL_MAX	Buffer Fill Level - MAX - P0, Value Range: 0 - BUF_SIZE, (V0:0x80F00100, V1:0x80F00104 ...)	R
0x80F00120-0x80F0013F	MON_BFL_MAX	Buffer Fill Level - MAX - P1, ...	R
0x80F00140-0x80F0015F	MON_BFL_MAX	Buffer Fill Level - MAX - P2, ...	R
0x80F00160-0x80F0017F	MON_BFL_MAX	Buffer Fill Level - MAX - P3, ...	R
0x80F00180-0x80F0019F	MON_BFL_MAX	Buffer Fill Level - MAX - P4, ...	R
0x80F00200-0x80F0021F	MON_LINK_CURR	Link Utilization - AVG, Current Sampling Period, Value Range: 0 - MON_PERIOD-1, (P0:0x80F00200, P1:0x80F00204 ...)	R
0x80F00220-0x80F0023F	MON_LINK_CNT	Link Utilization - Current Length of Current Sampling Period, Value Range: 0 - MON_-PERIOD-1, (P0:0x80F00220, P1:0x80F00224 ...)	R
0x80F00240-0x80F0025F	MON_VC_CURR	Virtual Channel - AVG, Current Sampling Period, Value Range: 0 - MON_PERIOD*VC_-CNT-1, (P0:0x80F00240, P1:0x80F00244 ...)	R
0x80F00260-0x80F0027F	MON_VC_CNT	Virtual Channel - Current Length of Current Sampling Period, Value Range: 0 - MON_-PERIOD-1, (P0:0x80F00260, P1:0x80F00264 ...)	R
0x80F00280-0x80F003FF	MON_BFL_CURR	Buffer Fill Level - AVG, Current Sampling Period, Value Range: 0 - MON_PERIOD*BUF_-SIZE-1, (P0-V0:0x80F00280, P0-V1:0x80F00284 ... P1-V0:0x80F00300, P1-V1:0x80F00304)...	R
0x80F00400-0x80F0057F	MON_BFL_CNT	Buffer Fill Level - Current Length of Current Sampling Period, Value Range: 0 - MON_PERIOD-1, (P0-V0:0x80F00400, P0-V1:0x80F00404 ... P1-V0:0x80F00420, P1-V1:0x80F00424)...)	R
0x80F00580-0x80F00583	MON_PERIOD	Number of cycles per monitoring period (MON_PERIOD), Value Range: 0 - *MON_INT*	R+W
0x80F00584-0x80F00587	MON_RST	A write access to this register (any value) resets all monitors of the router	W

Table A.4.: Monitoring registers of the i-NoC.

Address Range	Name	Comment	R/W
0x80F01000-0x80F010FF	DBG_INP_HIST_DATA	Input Port History Data - P0, [31:0]:Flit Payload [31:0], (Jungest: 0x80F01000, Older: 0x80F01004)	R
0x80F01100-0x80F011FF	DBG_INP_HIST_DATA	Input Port History Data - P1, ...	R
0x80F01200-0x80F012FF	DBG_INP_HIST_DATA	Input Port History Data - P2, ...	R
0x80F01300-0x80F013FF	DBG_INP_HIST_DATA	Input Port History Data - P3, ...	R
0x80F01400-0x80F014FF	DBG_INP_HIST_DATA	Input Port History Data - P4, ...	R
0x80F01500-0x80F015FF	DBG_OP_HIST_DATA	Output Port History Data - P4, ...	R
0x80F01600-0x80F016FF	DBG_INP_HIST_CTRL	Input Port History Control - P0, [0]:Ctrl-Bit, [2:4]:VC , (Jungest: 0x80F01600, Older: 0x80F01604)	R
0x80F01700-0x80F017FF	DBG_INP_HIST_CTRL	Input Port History Control - P1, ...	R
0x80F01800-0x80F018FF	DBG_INP_HIST_CTRL	Input Port History Control - P2, ...	R
0x80F01900-0x80F019FF	DBG_INP_HIST_CTRL	Input Port History Control - P3, ...	R
0x80F01A00-0x80F01AFF	DBG_INP_HIST_CTRL	Input Port History Control - P4, ...	R
0x80F01B00-0x80F01BFF	DBG_OP_HIST_CTRL	Output Port History Control - P4, ...	R

Table A.5.: Debug registers of the i-NoC.

Address Range	Name	Comment	R/W
0x80F03000-0x80F03003	POL_CFG_P0	Policy Configuration - Sets new VALUE+1 as BE/GS policy for P0, [31]0=BE/1=GS, [30:0]VALUE	W
0x80F03004-0x80F03007	POL_CFG_P1	Policy Configuration - Sets new VALUE+1 as BE/GS policy for P1, [31]0=BE/1=GS, [30:0]VALUE	W
0x80F03008-0x80F0300B	POL_CFG_P2	Policy Configuration - Sets new VALUE+1 as BE/GS policy for P2, [31]0=BE/1=GS, [30:0]VALUE	W
0x80F0300C-0x80F0300F	POL_CFG_P3	Policy Configuration - Sets new VALUE+1 as BE/GS policy for P3, [31]0=BE/1=GS, [30:0]VALUE	W
0x80F03020-0x80F03023	POL_STAT_BE_P0	Policy Status - BE - P0	R
0x80F03024-0x80F03027	POL_STAT_BE_P1	Policy Status - BE - P1	R
0x80F03028-0x80F0302B	POL_STAT_BE_P2	Policy Status - BE - P2	R
0x80F0302C-0x80F0302F	POL_STAT_BE_P3	Policy Status - BE - P3	R
0x80F03040-0x80F03043	POL_STAT_GS_P0	Policy Status - GS - P0	R
0x80F03044-0x80F03047	POL_STAT_GS_P1	Policy Status - GS - P1	R
0x80F03048-0x80F0304B	POL_STAT_GS_P2	Policy Status - GS - P2	R
0x80F0304C-0x80F0304F	POL_STAT_GS_P3	Policy Status - GS - P3	R

Table A.6.: Resource management policy registers of the i-NoC.

Address Range	Name	Comment	R/W
0x80F02000-0x80F02003	CYC_CNT_MSB	Cycle Counter MSBs [63:32] for performance measurements	R
0x80F02004-0x80F02007	CYC_CNT_LSB	Cycle Counter LSBs [31:0] for performance measurements	R
0x80F02008-0x80F0200B	ROUTER_ID	32Bit Integer Router ID	R
0x80F0200C-0x80F0200F	ROUTER_REV	32Bit Integer i-NoC-Router Revision Number (a.b.c.d) - a[31:24], b[23:16], c[15:8], d[7:0]	R
0x80F02100-0x80F02103	SLN_CONFIG_EN	Writing a “1” to this register activates the SLN configuration	W
0x80F02104-0x80F02107	SLN_CONFIG_DATA	Configuration data for the SLN configuration (see [Dud14])	W

Table A.7.: Additional registers of the i-NoC.

A.5. Parameters of the Simulation Framework

The network on chip simulation framework, which was described in detail in section 5.2, supports a large number of configuration options. These options or parameters are defined in the *parameter file* of the simulator. They are processed at design-time to improve the simulation speed. A selection of the most important parameters of the simulation framework is provided by the following tables. The parameters for the *network adapter* and the *NoC model* are summarized in table A.8 and A.9. Table A.10 and A.11 provide the parameters for the *traffic generation* and the *packetization layer*. Table A.12 summarizes the parameters for the *statistics module* of the simulation framework.

Name	Explanation	Def. Value
G_MESH_DIM_X	X dimension of the network	10
G_MESH_DIM_Y	Y dimension of the network	10
G_MAX_PORTS	Number of ports per router including the local port (other numbers might be applied for other topologies, such as 3D).	5
G_ROUTER_DELAY	Per hop/router delay for a general flit (header flits maight have additional delay). Min. value = 1.	2
G_RESERVATION_DELAY	Additional latency for VC reservation (processing of header flits).	1
G_FLIT_SIZE	Flit size or link width in bits (corresponds to the data width of the physical connection between the routers).	32
G_VCS	Number of virtual channels per router port.	4
G_MAX_GS_VCS	Number of VCs which may be used simultaneously for GS communication.	3
G_MAX_GS_VCS	Number of VCs which may be used simultaneously for BE communication.	4
G_ONE_BE_VC_FROM_RE	Use only a single VC for packet injection at the local port (only for BE traffic).	false
G_RESERVE_CTRL_C	Reserve a virtual channel for NoC internal control purpose.	false
G_SIMPLE_VN_EN	This parameter enables a simple virtual network, where BE traffic is used in one VN and GS traffic is used in the other virtual network (G_MAX_GS_VCS and G_MAX_BE_VCS define the resources assigned to each VN).	false
G_SIMPLE_VN_VCS	Number of virtual channels reserved for the virtual network.	2
G_SIMPLE_VN_DYN	The private network (VN) is allowed to use the VCs of the public network.	false

Table A.8.: List of the parameters for the *network adapter* and the *NoC model* of the simulation framework - Part I.

Name	Explanation	Def. Value
G_TS	Number of time slots (must be greater or equal to G_VCS).	4
G_TS_MAX	Maximum number of time slots per GS connection (Connections with higher SL are automatic reduced).	1
G_SIMPLE_BUFFER	Simple distributed separate input buffer for each VC (If disabled, centralized buffers are employed).	true
G_SEPERATE_PORTS	0: All ports have totally separate central buffer (but in contrast to the simple buffer not the VCs), 1: Central buffer is shared between the ports, but each port has its own address-queue, 2: Buffer and address-queue is shared between all ports.	0
G_RESERVE_ADDR	Reserve buffer resources in the centralized buffer for one flit per port and VC (to ensure independence of VCs).	true
G_MULT_PKG	Allow multiple packets per buffer at the same time.	true
G_BUFFER_SIZE	Total size of the centralized buffer (in flits).	40
G_VC_QUEUE_DEPTH	Size of the simple buffer separate per VC input queue.	4
G_EN_BUFF_RD_PORT...	Emulates a single read port per input port buffer (shared by all VCs), false: each VC has its own read port, no arbitration is required.	false
G_USE_GS_IF_EXIST	BE packets transparently employ GS connection if one exists (feature implemented within the network adapter).	false
G_AUTO_GS_EN	*AutoGS* enables automatic setup of GS connections for BE packets to the most frequent utilized nodes (feature implemented within the network adapter).	false
G_AUTO_GS_CYCLE	Number of cycles between two *AutoGS* decisions.	1000
G_AUTO_GS_CON_MON	Number of connections to be monitored for *AutoGS* decisions.	4
G_AUTO_GS_CON_GS	GS connections per node established by *AutoGS*.	3
G_FC_ENABLE_GS_FC	Enable GS end-to-end flow control.	true
G_FC_DEL_GS_HT	Delete GS head & tail flits in router, if header is directly followed by tail flit (required G_FC_ENABLE_GS_FC = true).	true
G_FC_TIMEOUT_GS_FC	Timeout in cycles for GS connection setup.	500
G_FC_ENABLE_DISCON	Enable GS setup failed after timeout (disconnect after timeout) (required G_FC_ENABLE_GS_FC = true).	true
G_REROUTING	Enable rerouting self-optimization of GS connections.	false
G_CYCLES_NEXT_REROU...	Cycles between two rerouting requests.	100
G_ENABLE_ROUND_TRIP	Enable round trip routing.	false
G_ROUTING_TYPE	0: XY routing (selection impossible), 1: West First routing, 2: Fully Adaptive routing (can lead to deadlocks), 3: Odd Even routing, 4: Fully Adaptive routing (test), 5: special XY routing for memory routers.	0
G_SELECTION_TYPE	0: Random VC selection, 1: VC selection according to the free no. of VCs.	0
G_ADAPTIVE_ROUTE_VC...	Restricted VC selection for fully adaptive routing - 0: disabled, 1: xy, 2: only x, 3: only y (enabling VC selection depending on the routing decision).	0
G_MAX_TURN	Max. number of turns for fully adaptive routing (test).	10

Table A.9.: List of the parameters for the *network adapter* and the *NoC model* of the simulation framework - Part II.

Name	Explanation	Def. Value
G_ENABLE_TG	Selection of the used traffic generator - 0: No traffic generation, 1: Simple synthetic traffic generation, 2: Trace code execution, 3: Application model (Traffic cluster) (random traffic generation also possible), 5: User defined traffic.	1
G_BE_MIN_PKG_SIZE	Minimum size of the generated BE packets.	4
G_BE_MAX_PKG_SIZE	Maximum size of the generated BE packets.	8
G_BE_OVERHEAD_FLIT	A protocol overhead of 1 flit is added to each BE packet (Not supported with G_ENABLE_TG = 0)	false
G_BE_INJECTION_RATE	Average package injection rate per 1000 cycles for BE communication (Is also used as initial traffic cluster setting).	100
G_SIMPLE_TG_TYPE	Traffic type generated by the synthetic traffic generator - 0: Random, 1: Transpose1, 2: Transpose2, 3: BitReversal, 4: Shuffle, 5: Butterfly.	0
G_GS_MIN_PKG_SIZE	Minimum size of the generated GS packets.	4
G_GS_MIN_PKG_SIZE	Maximum size of the generated GS packets.	4
G_GS_MAX_PARALLEL...	The number of GS connections can be limited by lowering this number.	1000000
G_GS_INJECTION_RATE	Average package injection rate per 1000 cycles for GS communication	0
G_GS_WAIT_FOR_SETUP	Traffic generator waits until setup (header) flit was received, before transmitting other flits (except from tail flits).	true
G_TRACEFOLDER	Location of the trace file.	"/pip.tgff."
G_TX_CON_TYPE	Define whether packets are converted between BE and GS by the *Packetization Layer* - 0: Convert to BE, 1: Convert to GS, 2: mixed (no conversion).	2
G_TRAFFIC_GEN_TYPE	Define the type of generated packets (not supported by all traffic generators) - 0: BE-only, 1: GS-only, 2: mixed.	2
G_CONV_GS	Convert all GS traffic to BE.	false
G_RANDOM_CLUSTER	Application model (Traffic cluster) configuration - false: Neighboring cluster, true: Random cluster.	false
G_CLUSTER_CON_TYPE	Communication type selection for Application model - 0: Random connection between PEs, all connected to a memory node, 1: All PEs connected (Clique), all connected to a memory node — 2: All PEs connected to seed (master), all connected to a memory node.	1
G_PE_MEM_COM_RATE	Amount of memory communication for application model proportional to the total injection rate.	50
G_CL_MIN_SIZE	Min. number of applications generated by the application model.	4
G_CL_MAX_SIZE	Max. number of applications generated by the application model.	8
G_INF_RET_RATE_CL	Average cycles between invade/retreat phases for a complete application in the application model - 0: disabled.	0
G_INF_RET_RATE_ND	Average cycles between invade/retreat phases for a single node/PE in the application model - 0: disabled.	0
G_INF_RET_RATE_CL	Min. utilization (in %) of the REs or nodes.	10
G_INF_RET_RATE_ND	Max. utilization (in %) of the REs or nodes.	90
G_TC_RET_NON_EST...	Enable to release connections (and retreat nodes & cluster), which are not yet established.	true
G_TC_ENABLE_PER_C...	Injection (injection rate, packet size, service level, ...) is controlled per connection (not per core).	true

Table A.10.: List of the parameters for the *traffic generation* and the *packetization layer* of the simulation framework - Part I.

Name	Explanation	Def. Value
G_BANDWIDTH	Bandwidth in MByte (Might be employed as a scaling factor for the task graph based traffic generation).	2048
G_GRAPH_USE_BE	Use BE communication for all connections of the mapped task graphs.	false
G_GRAPH_AUTO_SL	Assign service levels automatically for GS connections.	false
G_GRAPH_SEL_MPEG	Map MPEG4 video decoding (12 cores), Offset to be centric (for 10x10 nodes): x = 3, y = 3, G_BANDWIDTH = 2048.	false
G_GRAPH_SEL_VOPD	Map Video Object Plan Decoding (VOPD, 12 cores), Offset to be centric (for 10x10 nodes): x = 3, y = -3, G_BANDWIDTH = 1024.	false
G_GRAPH_SEL_PIP	Map Picture-In-Picture (PIP, 8 cores), Offset to be centric (for 10x10 nodes): x = 3, y = -1, G_BANDWIDTH = 256.	false
G_GRAPH_SEL_MWD	Map Multi-Windows Display (MWD, 12 cores), Offset to be centric (for 10x10 nodes): x = 0, y = 0, G_BANDWIDTH = 256.	false
G_GRAPH_SEL_H264	Map H.264 CAVLC encoder (H.264, 16 cores), Offset to be centric (for 10x10 nodes): x = 3, y = 3, G_BANDWIDTH = 2048.	false
G_GRAPH_SEL_MMS	Map Multimedia System (MMS, 25 cores), Offset to be centric (for 10x10 nodes): x = 2, y = 2, G_BANDWIDTH = 196000.	false
G_GRAPH_MAPP_OFFS_X	Task graph mapping offset X.	3
G_GRAPH_MAPP_OFFS_Y	Task graph mapping offset Y.	3
G_ROUND_TRIP_TYPE	Use special communication patterns for data collection/multicast - 0: Request Response (parallel) / disabled, 1: Hamiltonian Path, 2: Mixed Mode (Req: Hamiltonian Path, Resp: Normal), 3: Request Response (serial).	0
G_ROUND_TRIP_TILE...	ID of the tile requesting for the round trip communication.	44
G_ROUND_TRIP_PL...	Payload size per core (in Bit) for round trip packets.	32

Table A.11.: List of the parameters for the *traffic generation* and the *packetization layer* of the simulation framework - Part II.

Name	Explanation	Def. Value
G_VERBOSE	Print detailed debug information.	false
G_TX_MSG	Print message if a packet is transmitted (header and tail).	false
G_RX_MSG	Print message in case of packet arrival (header and tail).	false
G_FLIT_RX_MSG	Print massage if a flit is received by a router.	false
G_HEAD_RX_MSG	Print massage if a header flit is received by a router.	false
G_VC_RES_FAILED_MSG	Print massage if reservation of a virtual channel failed.	false
G_GS_CON_MSGS	Print information about GS connect and release.	false
G_TC_MSG	Print application model (traffic cluster) messages.	false
G_SIM_SPEEDUP	If enabled, the packets within the NoC aren't logged for debugging.	true
G_PRINT_BUF_STATS	Print detailed buffer related statistics.	false
G_WARMUP_TIME	Exclude warmup time (in cycles) from statistics.	1000
G_EVENT_CYCLE_LENGTH	Number of cycles between two short status messages (Printing of accumulated injection rate).	1000

Table A.12.: List of the parameters for the *statistics module* and debugging of the simulation framework.

A.6. Parameters of the HDL Template

The HDL template of the NoC and the router has a lot of configuration options, which are defined at design-time. These options or parameters are defined in the *parameter file* of the HDL template. They enable to adapt the router design according to the requirements of the architecture, where the NoC is generated for. These parameters can either be generated automatically, e.g. when using the script-based synthesis flow. Automated parameter setting is also employed in the semi-automated design flow, introduced in section 4.4.

Table A.13 provides an overview on the parameters of the basic router design, described in section 5.1. The parameters for the monitoring infrastructure of the NoC, discussed in section 5.5.3, are provided in table A.14. Further parameters for various other features presented in this work, are summarized in table A.15.

Name	Explanation	Type
G_DIM_X	X dimension of the network.	uint
G_DIM_Y	Y dimension of the network.	uint
G_FLIT_SIZE	Flit size or link width in bits (corresponds to the data width of the physical connection between the routers).	uint
G_PORTS	Number of ports per router including the local port (other numbers might be applied for other topologies, such as 3D mesh or torus).	uint
G_VCS	Number of virtual channels per router port.	uint
G_VC_POL_ENABLE	Enable VC policy management. The resource management unit is instantiated to controls of VC reservation for GS and BE traffic.	logic
G_TS	Number of time slots (must be greater or equal to G_VCS).	uint
G_MAX_TS	Limit for the number of time slot to be used by a single GS connections.	uint
G_VC_BUFFER_DEPTH	Depth of the FIFOs, which is used as input buffer for each VC.	uint
G_FAST_RESERVATION	New Header flits are forwarded from the input of the VC buffer to the reservation unit (might reduce the clock frequency).	
G_MULTI_PKG_PER_BUFFER	Allow to store parts of different packets in one buffer at the same time (not tested).	logic
G_OUTPUT_REG	Enable output registers.	logic
G_PIPELINE_RES	Enable pipelining of the VC reservation process.	logic
G_PIPELINE_SEL	Enable pipelining of the flit selection process (part of the reservation process).	logic
G_USE_DW	Enable instantiate of *DesignWare* building blocks.	logic

Table A.13.: List of the configuration options of the HDL template of the basic router and NoC design.

Name	Explanation	Type
G_ENABLE_MON	Enable monitoring data access via control channel.	logic
G_REGISTER_SIZE	Size of the memory mapped registers of the router (see appendix A.4).	logic
G_ENABLE_RVC_DET	Enable monitoring of the virtual channel reservation.	logic
G_ENABLE_BFL_DET	Enable buffer fill level monitoring.	logic
G_ENABLE_OPR_DET	Enable link monitoring.	logic
G_NO_OF_VALUES_RVC	Number of values added to cache of virtual channel reservation monitors (per monitoring cycle) => Represents the maximum monitoring cycle length.	uint
G_NO_OF_VALUES_BFL	Number of values added to cache of buffer fill level monitors (per monitoring cycle) => Represents the maximum monitoring cycle length.	uint
G_NO_OF_VALUES_OPR	Number of values added to cache of link monitors (per monitoring cycle) => Represents the maximum monitoring cycle length.	uint
G_HISTORY_PRECISION_RVC	Width/precision (in bit) of the registers of the virtual channel reservation monitoring history table.	uint
G_HISTORY_PRECISION_BFL	Width/precision (in bit) of the registers of the buffer fill level monitoring history table.	uint
G_HISTORY_PRECISION_OPR	Width/precision (in bit) of the registers of the link monitoring history table.	uint
G_HISTORY_LENGTH_RVC	Number of old monitoring values stores in the virtual channel reservation monitoring unit.	uint
G_HISTORY_LENGTH_BFL	Number of old monitoring values stores in the buffer fill level monitoring unit.	uint
G_HISTORY_LENGTH_RVC	Number of old monitoring values stores in the link monitoring unit.	uint
G_ENABLE_RVC_TX	Enable automatic transmission of the virtual channel reservation monitoring data (necessitates G_ENABLE_MON = 1).	logic
G_ENABLE_BFL_TX	Enable automatic transmission of the fill level monitoring data (necessitates G_ENABLE_MON = 1).	logic
G_ENABLE_OPR_TX	Enable automatic transmission of the link monitoring data (necessitates G_ENABLE_MON = 1).	logic
G_TRIGGER_PERIOD	Duration of a monitoring cycle (time between two automatic monitoring data transmissions) in *ns*.	int
G_CLK_PERIOD	Router clock cycle length in *ns*.	int
G_TX_DATA_PRECISION	Precision (int bit) of the monitoring data transmitted via the control channel.	int
G_NO_OF_VALUES_OPR	number of values added to cache	
G_TX_TO_LOCAL_ROUTER	if set to "1", the cyclic monitoring data are sent to the local router. If set to "0", they are sent to the coordinates defined by G_TX_TO_DST_X and G_TX_TO_DST_Y.	logic
G_TX_TO_DST_X	Destination X address for cyclic monitoring data transmission.	uint
G_TX_TO_DST_Y	Destination Y address for cyclic monitoring data transmission.	uint
G_ENABLE_DEBUGGING	Enable the debugging unit, which captures the latest flits received at each router port.	logic

Table A.14.: List of the monitoring and debugging configuration options of the HDL template design.

Name	Explanation	Type
G_CTRL_ENABLE	Enable control NoC/channel.	logic
G_CTRL_FLIT_SIZE	Flit size or link width of the control NoC in Bits (corresponds to the data width of the physical connection between the routers).	uint
G_CTRL_BUFFER_DEPTH	Depth of the FIFOs, which is used as input buffer for the control NoC.	uint
G_SIMPLE_CTRL_ENABLE	Simplistic implementation of a control output.	logic
G_CTRL_ARB_SIZE	Number of entities which use the control channel within the router.	uint
G_REROUTING_ENABLE	Enable rerouting self-optimization of GS connections.	logic
G_OPS_COUNTERS	Use fill level counters within *Transmission Control* unit to improve timing - 0: disabled, 1: version with less stall cycles, 2: version with higher clock frequency, but higher delay	uint
G_IN_ORDER	Ensure processing of the reservation requests in the order of arrival (necessary to ensure in-order arrival).	logic
G_VC_IN_LINE	Restrict to reserve multiple VCs in parallel for transmissions to the same destination node (necessary to ensure in-order arrival).	logic
G_USE_ERROR_DETECT	Enable error detect at the router ports - 0: Off; 1: Parity; 2: CRC8(ISDN Header); 3: CRC10.	uint
G_CTRL_ARB_ADDR	Router internal address of the error detection unit at the router internal port of the control channel.	uint
G_PARITY_EVEN_ODD	Use odd or even parity bit - 0: even; 1: odd parity.	logic
G_RING_CIRCLE_BUS	Enable ring bus instantiation.	logic
G_RING_CONFIGCH..._W...	Width of the configuration channel of the ring bus.	uint
G_RING_CONFIG_MAX_ADR	Number of bits used for address encoding in the configuration channel of the ring bus.	uint
G_RING_CONFIG_CNT_SIZE	Size of the configuration counter (employed to synchronize the activation of a ring bus configuration).	uint
G_RING_DIRECTIONS	Directions (ports) of a ring bus multiplexer (is normally set to G_PORTS-1).	uint
G_RING_BUFFER_DEPTH	Buffer depth of the ring buffer switch input port.	uint
G_LOCAL_SWITCH	Instantiate a ring bus switch at the local router port.	logic

Table A.15.: List of configuration options of the HDL template design for advanced features.

A

List of Figures

List of Tables

Abbreviations

AHB advanced high-performance bus
AHR adaptive Hamilton routing
AMBA advanced microcontroller bus architecture
APB advanced peripheral bus
ARM advanced RISC machines
ASB advanced system bus
ASIC application-specific integrated circuit
ASP advance peripheral bus
ATD asynchronous time division

BE best-effort
BIST built-in self-test
BU buffer utilization

CiC core ilet controllers
CISC complex instruction set computer
CMOS complementary metal oxide semiconductor
CMP chip multiprocessor
CPU central processing unit
CRE communication related energy
CS circuit switching
CSV comma-separated values

DDR double data rate
DEMUX demultiplexer
DFT discrete Fourier transform
DMA direct memory access

DSE design space exploration
DSM distributed shared memory
DSU debug support unit
DVFS dynamic voltage and frequency scaling
DVI digital visual interface

ECC error-correcting code
EPIC explicitly parallel instruction computing
EVC express virtual channel

FEC forward error correction
FI frequency island
FIFO first in - first out
flit flow control digit
FPGA field programmable gate array
FPU floating-point unit
FSM finite state machine

GPU graphics processing unit
GS guaranteed service
GT guaranteed throughput

HDL hardware description language
HLS high-level synthesis
HPC high-performance computing
HRE heterogeneous reconfigurable engine

IET independent execution time
i-NA invasive network adapter
i-NoC invasive network on chip
I/O input/output
IDN input/output dynamic network
IP intellectual property
IRA input reservation arbitration

iRTSS invasive run-time support system
ISA instruction set architecture
ITRS International Technology Roadmap for Semiconductors

JTAG joint test action group

LAN local area network
LU link utilization
LUT look up table

MC memory controller
MDN memory dynamic network
MPB message passing buffer
MPI message passing interface
MPPA multi-purpose processor architecture
MPSoC multiprocessor system on a chip
MPU message passing unit

NA network adapter
NI network interface
NoC network on chip
NUMA non-uniform memory access

OPRA output port reservation arbitration
ORT output reservation table
OS operating system

PAR place and route
PCB printed circuit board
PCI peripheral component interconnect
PE processing element
PGAS partitioned global address space
PIO programmed input/output
PMU power management unit

PS packet switching

QoS quality of service

RAM random-access memory
RaR request-and-response
RB ring bus
RE resource element
RGMII reduced gigabit media independent interface
RISC reduced instruction set computer
RMP resource management policy
RMU resource management unit
RR round-robin
RSR reservation success rate
RT round-trip
RTP round-trip packet

SAF store and forward
SCC Single-chip Cloud Computer
SDM spatial division multiplexing
SER soft error rate
SET single event transient
SEU single event upsets
SL service level
SLN second layer network
SMU shared memory unit
SoC system on a chip
SPMD single program, multiple data
SRAM static random-access memory
SSRAM synchronous static random access memory
STN static network

TC transmission control

TCPA Tightly-Coupled Processor Array
TDM time division multiplexing
TDMA time division multiple access
TDN tile dynamic network
TGFF task graphs for free
TLM tile local memory
TMR triple modular redundancy
TS time slot
TSV through-silicon via

UART universal asynchronous receiver transmitter
UDN user dynamic network
UMR Universal Multi-Resource
UPF unified power format
USB universal serial bus

VC virtual channel
VCD value change dump
VCI virtual component interface
VCT virtual cut through
VCU virtual channel utilization
VFI voltage-frequency islands
VI voltage island
VLIW very long instruction word
VLSI very-large-scale integration
VN virtual network
VNCU virtual network control unit
VNMU virtual network management unit

WC worst case
WRR weighted round-robin

XAUI 10 gigabit media independent interface

I

Bibliography

[1] ADRIAHANTENAINA, A., H. CHARLERY, A. GREINER, L. MORTIEZ and C. ZEFERINO: *SPIN: a scalable, packet switched, on-chip micro-network*. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 70–73 suppl., 2003.

[2] AGARWAL, A., S. AMARASINGHE, R. BARUA, M. FRANK, W. LEE, V. SARKAR, D. SRIKRISHNA and M. TAYLOR: *The RAW compiler project*. In *Proceedings of the Second SUIF Compiler Workshop*, pp. 21–23, 1997.

[3] AGARWAL, A., C. ISKANDER and R. SHANKAR: *Survey of network on chip (NoC) architectures & contributions*. Journal of Engineering, Computing and Architecture, 3(1):21–27, 2009.

[4] AGARWAL, K., K. NOWKA, H. DEOGUN and D. SYLVESTER: *Power Gating with Multiple Sleep Modes*. In *Proceedings of the 7th International Symposium on Quality Electronic Design*, ISQED '06, pp. 633–637, Washington, DC, USA, 2006. IEEE Computer Society.

[5] AHN, M. and E. J. KIM: *Pseudo-Circuit: Accelerating Communication for On-Chip Interconnection Networks*. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 399–408, 2010.

[6] AL-BABTAIN, B. M., F. J. AL-KANDERI, M. F. AL-FAHAD and I. AHMAD: *A SURVEY ON AMDAHL'S LAW EXTENSION IN MULTICORE ARCHITECTURES*. International Journal of New Computer Architectures and their Applications (IJNCAA), 3(3):30–46, 2013.

[7] AL FARUQUE, M., R. KRIST and J. HENKEL: *ADAM: Run-time agent-based distributed application mapping for on-chip communication*. In *45th ACM/IEEE Design Automation Conference, 2008. DAC 2008*, pp. 760–765, June 2008.

[8] ALCEU CARARA, E. and F. MORAES: *Deadlock-Free Multicast Routing Algorithm for Wormhole-Switched Mesh Networks-on-Chip*. In *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pp. 341–346, Apr. 2008.

[9] ALI, M., M. WELZL and S. HESSLER: *A Fault tolerant mechanism for handling Permanent and Transient Failures in a Network on Chip*. In *Fourth International Conference on Information Technology, 2007. ITNG '07*, pp. 1027–1032, Apr. 2007.

[10] AMDAHL, G. M.: *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities*. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), pp. 483–485, New York, NY, USA, 1967. ACM.

[11] ANDERS, M., H. KAUL, M. HANSSON, R. KRISHNAMURTHY and S. BORKAR: *A 2.9Tb/s 8W 64-core circuit-switched network-on-chip in 45nm CMOS*. In *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pp. 182–185, 2008.

[12] ANJAN, K. V. and T. M. PINKSTON: *An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA*. In *Proceedings of the 22Nd Annual International Symposium on Computer Architecture*, ISCA '95, pp. 201–210, New York, NY, USA, 1995. ACM.

[13] ARCHIBALD, J. and J.-L. BAER: *Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model*. ACM Trans. Comput. Syst., 4(4):273–298, Sept. 1986.

[14] ASANOVIC, K., R. BODIK, B. C. CATANZARO, J. J. GEBIS, P. HUSBANDS, K. KEUTZER, D. A. PATTERSON, W. L. PLISHKER, J. SHALF, S. W. WILLIAMS et al.: *The landscape of parallel computing research: A view from Berkeley*. Techn. Rep., Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.

[15] ASCIA, G., V. CATANIA, M. PALESI and D. PATTI: *Implementation and Analysis of a New Selection Strategy for Adaptive Routing in Networks-on-Chip*. IEEE Transactions on Computers, 57(6):809–820, June 2008.

[16] BAILEY, D. H., E. BARSZCZ, J. T. BARTON, D. S. BROWNING, R. L. CARTER, L. DAGUM, R. A. FATOOHI, P. O. FREDERICKSON, T. A. LASINSKI, R. S. SCHREIBER, H. D. SIMON, V. VENKATAKRISHNAN and S. K. WEERATUNGA: *The Nas Parallel Benchmarks*. International Journal of High Performance Computing Applications, 5(3):63–73, Sept. 1991.

[17] BARNEY, B.: *Introduction to parallel computing*. https://computing.llnl.gov/tutorials/parallel_comp/, 2014. Accessed: 2014-02-06.

[18] BELL, S., B. EDWARDS, J. AMANN, R. CONLIN, K. JOYCE, V. LEUNG, J. MACKAY, M. REIF, L. BAO, J. BROWN, M. MATTINA, C.-C. MIAO, C. RAMEY, D. WENTZLAFF, W. ANDERSON, E. BERGER, N. FAIRBANKS, D. KHAN, F. MONTENEGRO, J. STICKNEY and J. ZOOK: *TILE64 - Processor: A 64-Core SoC with Mesh Interconnect*. In *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, pp. 88–598, Feb. 2008.

[19] BENINI, L. and G. DE MICHELI: *Networks on chips: a new SoC paradigm*. Computer, 35(1):70–78, Jan. 2002.

[20] BENINI, L. and G. DE MICHELI: *Networks on chips: a new SoC paradigm*. Computer, 35(1):70–78, 2002.

[21] BERTOZZI, D. and L. BENINI: *Xpipes: a network-on-chip architecture for gigascale systems-on-chip*. IEEE Circuits and Systems Magazine, 4(2):18–31, 2004.

[22] BERTOZZI, D., A. JALABERT, S. MURALI, R. TAMHANKAR, S. STERGIOU, L. BENINI and G. DE MICHELI: *NoC synthesis flow for customized domain specific multiprocessor systems-on-chip*. IEEE Transactions on Parallel and Distributed Systems, 16(2):113–129, Feb. 2005.

[23] BJERREGAARD, T. and S. MAHADEVAN: *A Survey of Research and Practices of Network-on-chip*. ACM Comput. Surv., 38(1), June 2006.

[24] BJERREGAARD, T. and J. SPARSO: *Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip*. In *Norchip Conference, 2004. Proceedings*, pp. 269–272, Nov. 2004.

[25] BJERREGAARD, T. and J. SPARSO: *A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip*. In *Design, Automation and Test in Europe, 2005. Proceedings*, pp. 1226–1231 Vol. 2, Mar. 2005.

[26] BJERREGAARD, T. and J. SPARSO: *Implementation of guaranteed services in the MANGO clockless network-on-chip*. Computers and Digital Techniques, IEE Proceedings -, 153(4):217–229, July 2006.

[27] BJERREGAARD, T. and J. SPARSOE: *Scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip*. In *11th IEEE International Symposium on Asynchronous Circuits and Systems, 2005. ASYNC 2005. Proceedings*, pp. 34–43, Mar. 2005.

[28] BOBDA, C., A. AHMADINIA, M. MAJER, J. TEICH, S. FEKETE and J. VAN DER VEEN: *DyNoC: A dynamic infrastructure for communication in dynamically reconfugurable devices*. In *International Conference on Field Programmable Logic and Applications, 2005*, pp. 153–158, Aug. 2005.

[29] BOLOTIN, E., I. CIDON, R. GINOSAR and A. KOLODNY: *QNoC: QoS architecture and design process for network on chip*. Journal of Systems Architecture, 50(2-3):105–128, Feb. 2004.

[30] BORKAR, S.: *Design challenges of technology scaling*. IEEE Micro, 19(4):23–29, July 1999.

[31] BORKAR, S.: *Designing reliable systems from unreliable components: the challenges of transistor variability and degradation*. IEEE Micro, 25(6):10–16, Nov. 2005.

[32] BORKAR, S.: *Thousand Core Chips: A Technology Perspective*. In *Proceedings of*

the 44th Annual Design Automation Conference, DAC '07, pp. 746–749, New York, NY, USA, 2007. ACM.

[33] BORKAR, S.: *Thousand Core Chips: A Technology Perspective*. In *Proceedings of the 44th Annual Design Automation Conference*, DAC '07, pp. 746–749, New York, NY, USA, 2007. ACM.

[34] BOURA, Y. M. and C. DAS: *Efficient fully adaptive wormhole routing in n-dimensional meshes*. In , *Proceedings of the 14th International Conference on Distributed Computing Systems, 1994*, pp. 589–596, June 1994.

[35] BOURGADE, R., C. ROCHANGE, M. DE MICHIEL and P. SAINRAT: *MBBA: A Multi-Bandwidth Bus Arbiter for Hard Real-Time*. In *Embedded and Multimedia Computing (EMC), 2010 5th International Conference on*, pp. 1–7, 2010.

[36] BRAUN, M., S. BUCHWALD, M. MOHR and A. ZWINKAU: *An X10 Compiler for Invasive Architectures*. Techn. Rep. 9, Karlsruhe Institute of Technology, 2012.

[37] BULL, C.: *Erweiterung der Kahrisma Architektur um Network-on-Chip Kommunikation mit architektur-spezifischen Optimierungen*. Diploma thesis ID-1802, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[38] BUNGARTZ, H.-J., C. RIESINGER, M. SCHREIBER, G. SNELTING and A. ZWINKAU: *Invasive Computing in HPC with X10*. In *Proceedings of the Third ACM SIGPLAN X10 Workshop*, X10 '13, pp. 12–19, New York, NY, USA, 2013. ACM.

[39] CARARA, E., R. DE OLIVEIRA, N. L. V. CALAZANS and F. MORAES: *HeMPS - a framework for NoC-based MPSoC generation*. In *IEEE International Symposium on Circuits and Systems, 2009. ISCAS 2009*, pp. 1345–1348, May 2009.

[40] CARROLL, A. and G. HEISER: *An Analysis of Power Consumption in a Smartphone*. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pp. 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[41] CARVALHO, E. and F. MORAES: *Congestion-aware task mapping in heterogeneous MPSoCs*. In *International Symposium on System-on-Chip, 2008. SOC 2008*, pp. 1–4, Nov. 2008.

[42] CHANDRAKASAN, A., S. SHENG and R. BRODERSEN: *Low-power CMOS digital design*. IEEE Journal of Solid-State Circuits, 27(4):473–484, Apr. 1992.

[43] CHANG, X.: *Network Simulations with OPNET*. In *Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future - Volume 1*, WSC '99, pp. 307–314, New York, NY, USA, 1999. ACM.

[44] CHARLES, P., C. GROTHOFF, V. SARASWAT, C. DONAWA, A. KIELSTRA,

K. EBCIOGLU, C. VON PRAUN and V. SARKAR: *X10: An Object-oriented Approach to Non-uniform Cluster Computing*. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pp. 519–538, New York, NY, USA, 2005. ACM.

[45] CHIANG, C.-M. and L. M. NI: *Multi-address Encoding for Multicast*. In *Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, PCRCW '94, pp. 146–160, London, UK, UK, 1994. Springer-Verlag.

[46] CHIU, G.-M.: *The odd-even turn model for adaptive routing*. Parallel and Distributed Systems, IEEE Transactions on, 11(7):729–738, 2000.

[47] CONSTANTINESCU, C.: *Trends and challenges in VLSI circuit reliability*. IEEE Micro, 23(4):14–19, July 2003.

[48] CONSTANTINIDES, K., S. PLAZA, J. BLOME, B. ZHANG, V. BERTACCO, S. MAHLKE, T. AUSTIN and M. ORSHANSKY: *BulletProof: a defect-tolerant CMP switch architecture*. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*, pp. 5–16, Feb. 2006.

[49] COPPOLA, M., S. CURABA, M. D. GRAMMATIKAKIS, G. MARUCCIA and F. PAPARIELLO: *OCCN: A Network-On-Chip Modeling and Simulation Framework*. In *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 3*, DATE '04, pp. 174–179 Vol.3, Washington, DC, USA, 2004. IEEE Computer Society.

[50] COPPOLA, M., R. LOCATELLI, G. MARUCCIA, L. PIERALISI and A. SCANDURRA: *Spidergon: a novel on-chip communication network*. In *2004 International Symposium on System-on-Chip, 2004. Proceedings*, Nov. 2004.

[51] CORPORATION, S. P. E.: *SPEC CPU2000*.

[52] COTA, E., F. KASTENSMIDT, M. CASSEL, M. HERVE, P. ALMEIDA, P. MEIRELLES, A. AMORY and M. LUBASZEWSKI: *A High-Fault-Coverage Approach for the Test of Data, Control and Handshake Interconnects in Mesh Networks-on-Chip*. IEEE Transactions on Computers, 57(9):1202–1215, Sept. 2008.

[53] CRAMER, T., D. SCHMIDL, M. KLEMM and D. AN MEY: *OpenMP Programming on Intel R Xeon Phi TM Coprocessors: An Early Performance Comparison*, 2012.

[54] CUVIELLO, M., S. DEY, X. BAI and Y. ZHAO: *Fault Modeling and Simulation for Crosstalk in System-on-chip Interconnects*. In *Proceedings of the 1999 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '99, pp. 297–303, Piscataway, NJ, USA, 1999. IEEE Press.

[55] CYR, G., G. BOIS and M. ABOULHAMID: *Generation of processor interface*

for SoC using standard communication protocol. Computers and Digital Techniques, IEE Proceedings -, 151(5):367–376, Sept. 2004.

[56] DAGUM, L. and R. MENON: *OpenMP: an industry standard API for shared-memory programming*. Computational Science Engineering, IEEE, 5(1):46–55, Jan 1998.

[57] DALL'OSSO, M., G. BICCARI, L. GIOVANNINI, D. BERTOZZI and L. BENINI: *Xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs*. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*, pp. 45–48, Sept. 2012.

[58] DALLY, W.: *Virtual-channel flow control*. IEEE Transactions on Parallel and Distributed Systems, 3(2):194–205, Mar. 1992.

[59] DALLY, W. and H. AOKI: *Deadlock-free adaptive routing in multicomputer networks using virtual channels*. IEEE Transactions on Parallel and Distributed Systems, 4(4):466–475, Apr. 1993.

[60] DALLY, W. and C. SEITZ: *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks*. IEEE Transactions on Computers, C-36(5):547–553, May 1987.

[61] DALLY, W. and B. TOWLES: *Route packets, not wires: on-chip interconnection networks*. In *Design Automation Conference, 2001. Proceedings*, pp. 684–689, 2001.

[62] DICK, R. P., D. L. RHODES and W. WOLF: *TGFF: Task Graphs for Free*. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, CODES/CASHE '98, pp. 97–101, Washington, DC, USA, 1998. IEEE Computer Society.

[63] DINECHIN, B. D. D., P. G. D. MASSAS, G. LAGER, C. LEGER, B. ORGOGOZO, J. REYBERT and T. STRUDEL: *A Distributed Run-Time Environment for the Kalray MPPAÂ®-256 Integrated Manycore Processor*. Procedia Computer Science, 18:1654–1663, 2013.

[64] DORSEY, P.: *Xilinx stacked silicon interconnect technology delivers breakthrough fpga capacity, bandwidth, and power efficiency*. Xilinx White Paper: Virtex-7 FPGAs, pp. 1–10, 2010.

[65] DUATO, J.: *A new theory of deadlock-free adaptive routing in wormhole networks*. IEEE Transactions on Parallel and Distributed Systems, 4(12):1320–1331, Dec. 1993.

[66] EBI, T., M. FARUQUE and J. HENKEL: *TAPE: Thermal-aware agent-based power econom multi/many-core architectures*. In *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009*, pp. 302–309, Nov. 2009.

[67] EBRAHIMI, M., M. DANESHTALAB, P. LILJEBERG, J. PLOSILA and H. TENHUNEN: *A High-Performance Network Interface Architecture for NoCs Using Reorder Buffer Sharing*. In *2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 546–550, Feb. 2010.

[68] EL-GHAZAWI, T., E. EL-ARABY, M. HUANG, K. GAJ, V. KINDRATENKO and D. BUELL: *The promise of high-performance reconfigurable computing*. IEEE Computer, 41(2):69–76, 2008.

[69] FARUQUE, M. A. A., T. EBI and J. HENKEL: *Run-time adaptive on-chip communication scheme*. In *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pp. 26–31, San Jose, California, 2007. IEEE Press.

[70] FARUQUE, M. A. A., T. EBI and J. HENKEL: *ROAdNoC: runtime observability for an adaptive network on chip architecture*. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pp. 543–548, San Jose, California, 2008. IEEE Press.

[71] FAZZINO, F., M. PALESI and D. PATTI: *Noxim: Network-on-chip simulator*. URL: http://sourceforge. net/projects/noxim, 2008.

[72] FEERO, B. and P. PANDE: *Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation*. IEEE Transactions on Computers, 58(1):32–45, Jan. 2009.

[73] FEIGE, U. and P. RAGHAVAN: *Exact analysis of hot-potato routing*. In *Foundations of Computer Science, 1992. Proceedings., 33rd Annual Symposium on*, pp. 553–562, Oct 1992.

[74] FELPERIN, S. A., L. GRAVANO, G. D. PIFARRÉ and J. L. C. SANZ: *Fully-adaptive Routing: Packet Switching Performance and Wormhole Algorithms*. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, Supercomputing '91, pp. 654–663, New York, NY, USA, 1991. ACM.

[75] FENG, C., Z. LU, A. JANTSCH, J. LI and M. 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, pp. 11–16, New York, NY, USA, 2010. ACM.

[76] FICK, D., A. DEORIO, G. CHEN, V. BERTACCO, D. SYLVESTER and D. BLAAUW: *A Highly Resilient Routing Algorithm for Fault-tolerant NoCs*. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pp. 21–26, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[77] FISHER, J. A.: *Very Long Instruction Word Architectures and the ELI-512*. In *Pro-*

ceedings of the 10th Annual International Symposium on Computer Architecture, ISCA '83, pp. 140–150, New York, NY, USA, 1983. ACM.

[78] FLEURY, E. and P. FRAIGNIAUD: *A general theory for deadlock avoidance in wormhole-routed networks*. IEEE Transactions on Parallel and Distributed Systems, 9(7):626–638, July 1998.

[79] FLYNN, D.: *AMBA: enabling reusable on-chip designs*. IEEE Micro, 17(4):20–27, July 1997.

[80] FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG, KARLSRUHE INSTITUTE OF TECHNOLOGY and TECHNICAL UNIVERSITY MUNICH: *Transregional Collaborative Research Centre 89 - Funding Proposal 2014/2-2018/1*, 2014.

[81] FULGHAM, M. L. and L. SNYDER: *Performance of Chaos and Oblivious Routers Under Non-uniform Traffic*. Techn. Rep., University of Washington, 1993.

[82] FURBER, S. B.: *ARM System-on-chip Architecture*. Pearson Education, 2000.

[83] GAISLER, J.: *The LEON Processor User's Manual*, July 2001.

[84] GAISLER, J.: *A portable and fault-tolerant microprocessor based on the SPARC v8 architecture*. In *International Conference on Dependable Systems and Networks, 2002. DSN 2002. Proceedings*, pp. 409–415, 2002.

[85] GAISLER, J., E. CATOVIC, M. ISOMAKI, K. GLEMBO and S. HABINC: *GRLIB IP core user's manual*. Gaisler research, 2007.

[86] GANGULY, A., K. CHANG, S. DEB, P. PANDE, B. BELZER and C. TEUSCHER: *Scalable Hybrid Wireless Network-on-Chip Architectures for Multicore Systems*. IEEE Transactions on Computers, 60(10):1485–1502, Oct. 2011.

[87] GANGWAL, O. P., A. RADULESCU, K. GOOSSENS, S. G. PESTANA and E. RIJPKEMA: *Building Predictable Systems on Chip: An Analysis of Guaranteed Communication in the Aethereal Network on Chip*. In STOK, P. V. D. (ed.): *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, no. 3 in *Philips Research*, pp. 1–36. Springer Netherlands, Jan. 2005.

[88] GAO, X., J. J. LIOU, W. WONG and S. VISHWANATHAN: *An improved electrostatic discharge protection structure for reducing triggering voltage and parasitic capacitance*. Solid-State Electronics, 47(6):1105–1110, June 2003.

[89] GERLA, M. and L. KLEINROCK: *Flow Control: A Comparative Survey*. IEEE Transactions on Communications, 28(4):553–574, Apr. 1980.

[90] GLASS, C. J. and L. M. NI: *The Turn Model for Adaptive Routing*. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, ISCA '92, pp. 278–287, New York, NY, USA, 1992. ACM.

[91] GOCHMAN, S., A. MENDELSON, A. NAVEH and E. ROTEM: *Introduction to Intel Core Duo Processor Architecture*. Intel Technology Journal, 2006.

[92] GOOSSENS, K., J. DIELISSEN and A. RADULESCU: *AEthereal network on chip: concepts, architectures, and implementations*. IEEE Design Test of Computers, 22(5):414–421, Sept. 2005.

[93] GOOSSENS, K. and A. HANSSON: *The aethereal network on chip after ten years: Goals, evolution, lessons, and future*. In *2010 47th ACM/IEEE Design Automation Conference (DAC)*, pp. 306–311, June 2010.

[94] GRATZ, P., B. GROT and S. KECKLER: *Regional congestion awareness for load balance in networks-on-chip*. In *IEEE 14th International Symposium on High Performance Computer Architecture, 2008. HPCA 2008*, pp. 203–214, Feb. 2008.

[95] GROT, B., J. HESTNESS, S. KECKLER and O. MUTLU: *Express Cube Topologies for on-Chip Interconnects*. In *IEEE 15th International Symposium on High Performance Computer Architecture, 2009. HPCA 2009*, pp. 163–174, Feb. 2009.

[96] GROT, B., J. HESTNESS, S. KECKLER and O. MUTLU: *A QoS-Enabled On-Die Interconnect Fabric for Kilo-Node Chips*. IEEE Micro, 32(3):17–25, May 2012.

[97] GROT, B., J. HESTNESS, S. W. KECKLER and O. MUTLU: *Kilo-NOC: A Heterogeneous Network-on-chip Architecture for Scalability and Service Guarantees*. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pp. 401–412, New York, NY, USA, 2011. ACM.

[98] GROT, B., J. HESTNESS, S. W. KECKLER and O. MUTLU: *Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees*. SIGARCH Comput. Archit. News, 39(3):401–412, June 2011.

[99] GROT, B., S. W. KECKLER and O. MUTLU: *Preemptive virtual clock: a flexible, efficient, and cost-effective QOS scheme for networks-on-chip*. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pp. 268–279, New York, NY, USA, 2009. ACM.

[100] GROVE, D., O. TARDIEU, D. CUNNINGHAM, B. HERTA, I. PESHANSKY and V. SARASWAT: *A Performance Model for X10 Applications: What's Going on Under the Hood?*. In *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*, X10 '11, pp. 1:1–1:8, New York, NY, USA, 2011. ACM.

[101] GUERRIER, P. and A. GREINER: *A Generic Architecture for On-chip Packet-switched Interconnections*. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '00, pp. 250–256, New York, NY, USA, 2000. ACM.

[102] HACK, S., D. GRUND and G. GOOS: *Register Allocation for Programs in SSA-Form*. In MYCROFT, A. and A. ZELLER (eds.): *Compiler Construction*, no. 3923 in *Lecture Notes in Computer Science*, pp. 247–262. Springer Berlin Heidelberg, Jan. 2006.

[103] HANDY, J.: *The Cache Memory Book*. Morgan Kaufmann, Jan. 1998.

[104] HANNIG, F., S. ROLOFF, G. SNELTING, J. TEICH and A. ZWINKAU: *Resource-aware Programming and Simulation of MPSoC Architectures Through Extension of X10*. In *Proceedings of the 14th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '11, pp. 48–55, New York, NY, USA, 2011. ACM.

[105] HENKEL, J., L. BAUER, J. BECKER, O. BRINGMANN, U. BRINKSCHULTE, S. CHAKRABORTY, M. ENGEL, R. ERNST, H. HARTIG, L. HEDRICH, A. HERKERSDORF, R. KAPITZA, D. LOHMANN, P. MARWEDEL, M. PLATZNER, W. ROSENSTIEL, U. SCHLICHTMANN, O. SPINCZYK, M. TAHOORI, J. TEICH, N. WHEN and H. WUNDERLICH: *Design and architectures for dependable embedded systems*. In *2011 Proceedings of the 9th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 69–78, 2011.

[106] HENKEL, J., L. BAUER, M. HÜBNER and A. GRUDNITSKY: *i-Core: A run-time adaptive processor for embedded multi-core systems*. Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), 2011.

[107] HENKEL, J., W. WOLF and S. CHAKRADHAR: *On-chip networks: a scalable, communication-centric embedded system design paradigm*. In *17th International Conference on VLSI Design, 2004. Proceedings*, pp. 845–851, 2004.

[108] HENNESSY, J. L., D. A. PATTERSON and K. ASANOVIC: *Computer architecture: a quantitative approach*. Elsevier ; Morgan Kaufmann, Amsterdam [etc.]; Waltham (MA), Fourth edition ed., 2007.

[109] HILL, M. and M. MARTY: *Amdahl's Law in the Multicore Era*. Computer, 41(7):33–38, July 2008.

[110] HO, A., S. SMITH and S. HAND: *On deadlock, livelock, and forward progress*. Technical Report, University of Cambridge, Computer Laboratory (May 2005), 2005.

[111] HO, R., K. W. MAI, S. MEMBER and M. A. HOROWITZ: *The Future of Wires*. In *Proceedings of the IEEE*, pp. 490–504, 2001.

[112] HOWARD, J., S. DIGHE, Y. HOSKOTE, S. VANGAL, D. FINAN, G. RUHL, D. JENKINS, H. WILSON, N. BORKAR, G. SCHROM, F. PAILET, S. JAIN, T. JACOB, S. YADA, S. MARELLA, P. SALIHUNDAM, V. ERRAGUNTLA, M. KONOW, M. RIEPEN, G. DROEGE, J. LINDEMANN, M. GRIES, T. APEL, K. HENRISS, T. LUND-LARSEN, S. STEIBL, S. BORKAR, V. DE, R. VAN DER WIJNGAART and T. MATTSON: *A 48-Core IA-32 message-passing processor with DVFS in 45nm CMOS*. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pp. 108–109, 2010.

[113] HOWARD, J., S. DIGHE, S. VANGAL, G. RUHL, N. BORKAR, S. JAIN, V. ER-

RAGUNTLA, M. KONOW, M. RIEPEN, M. GRIES, G. DROEGE, T. LUND-LARSEN, S. STEIBL, S. BORKAR, V. DE and R. VAN DER WIJNGAART: *A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling*. IEEE Journal of Solid-State Circuits, 46(1):173–183, Jan. 2011.

[114] HOWARD, J., S. DIGHE, S. VANGAL, G. RUHL, N. BORKAR, S. JAIN, V. ERRAGUNTLA, M. KONOW, M. RIEPEN, M. GRIES, G. DROEGE, T. LUND-LARSEN, S. STEIBL, S. BORKAR, V. DE and R. VAN DER WIJNGAART: *A 48-Core IA-32 Processor in 45 nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling*. IEEE Journal of Solid-State Circuits, 46(1):173–183, 2011.

[115] HU, J. and R. MARCULESCU: *Energy-aware mapping for tile-based NoC architectures under performance constraints*. In *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pp. 233–239, Jan. 2003.

[116] HU, J. and R. MARCULESCU: *Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures*. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pp. 688–693, 2003.

[117] HU, J. and R. MARCULESCU: *DyAD: Smart Routing for Networks-on-chip*. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pp. 260–263, New York, NY, USA, 2004. ACM.

[118] HU, J. and R. MARCULESCU: *Energy- and performance-aware mapping for regular NoC architectures*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 24(4):551–562, Apr. 2005.

[119] HU, W., Z. LU, A. JANTSCH and H. LIU: *Power-efficient tree-based multicast support for Networks-on-Chip*. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pp. 363–368, Jan. 2011.

[120] ITRS: *International Technology Roadmap for Semiconductors 2012 Update Overview*. http://www.itrs.net/Links/2012ITRS/2012Chapters/2012Overview.pdf, 2012. Accessed: 2014-01-16.

[121] JAIN, L.: *NIRGAM: A Simulator for NoC Interconnect Routing and Application Modeling Version 1.1*, 2007.

[122] JANTSCH, A. and H. TENHUNEN: *Networks on chip*. Springer, 2003.

[123] JERGER, N., L.-S. PEH and M. LIPASTI: *Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support*. In *35th International Symposium on Computer Architecture, 2008. ISCA '08*, pp. 229–240, June 2008.

[124] JIAO, J., Y. FU, T. LIU, H. WANG, X. HAN and J. WANG: *Performance analysis and optimization for homogenous multi-core system based on 3D Torus Network on Chip*. In *NEWCAS Conference (NEWCAS), 8th IEEE International*, 2010.

[125] KAHNG, A. B., B. LI, L.-S. PEH and K. 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*, DATE '09, pp. 423–428, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[126] KAKOEE, M. R., V. BERTACCO and L. BENINI: *A Distributed and Topology-agnostic Approach for On-line NoC Testing*. In *Proceedings of the Fifth ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '11, pp. 113–120, New York, NY, USA, 2011. ACM.

[127] KALRAY: *MPPA MANYCORE: a multicore processors family - Many-core processors - KALRAY - Agile Performance*, 2014.

[128] KATEVENIS, M., S. SIDIROPOULOS and C. COURCOUBETIS: *Weighted round-robin cell multiplexing in a general-purpose ATM switch chip*. Selected Areas in Communications, IEEE Journal on, 9(8):1265–1279, 1991.

[129] KAVALDJIEV, N., G. SMIT, P. JANSEN and P. WOLKOTTE: *A virtual channel network-on-chip for GT and BE traffic*. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, p. 6, March 2006.

[130] KAVALDJIEV, N., G. J. M. SMIT and P. G. JANSEN: *A virtual channel router for on-chip networks*. In *Proceedings of IEEE International SOC Conference*, pp. 289–293. Society Press, 2004.

[131] KEANE, J. and C. KIM: *An odomoeter for CPUs*. IEEE Spectrum, 48(5):28–33, 2011.

[132] KEATING, M. and P. BRICAUD: *Reuse Methodology Manual for System-on-a-Chip Designs: For System-on-a-chip Designs*. Springer, Jan. 2002.

[133] KEATING, M., D. FLYNN, R. AITKEN, A. GIBBONS and K. SHI: *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[134] KIM, J.: *System on chip processor for multimedia devices*, 2007. US Patent 7,171,050.

[135] KIM, J., C. NICOPOULOS, D. PARK, R. DAS, Y. XIE, V. NARAYANAN, M. S. YOUSIF and C. R. DAS: *A Novel Dimensionally-decomposed Router for On-chip Communication in 3D Architectures*. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pp. 138–149, New York, NY, USA, 2007. ACM.

[136] KIM, J. H., Z. LIU and A. A. CHIEN: *Compressionless Routing: A Framework for Adaptive and Fault-tolerant Routing*. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, ISCA '94, pp. 289–300, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[137] KIM, Y. B. 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, 2007. DFT '07*, pp. 12–20, Sept. 2007.

[138] KISSLER, D., F. HANNIG, A. KUPRIYANOV and J. TEICH: *A highly parameterizable parallel processor array architecture*. In *IEEE International Conference on Field Programmable Technology, 2006. FPT 2006*, pp. 105–112, Dec. 2006.

[139] KNICKERBOCKER, J., P. ANDRY, B. DANG, R. R. HORTON, M. J. INTERRANTE, C. S. PATEL, R. POLASTRE, K. SAKUMA, R. SIRDESHMUKH, E. SPROGIS, S. SRI-JAYANTHA, A. M. STEPHENS, A. TOPOL, C. K. TSANG, B. WEBB and S. WRIGHT: *Three-dimensional silicon integration*. IBM Journal of Research and Development, 52(6):553–569, Nov. 2008.

[140] KOBBE, S., L. BAUER, D. LOHMANN, W. SCHRÖDER-PREIKSCHAT and J. HENKEL: *DistRM: Distributed Resource Management for On-chip Many-core Systems*. In *Proceedings of the Seventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '11, pp. 119–128, New York, NY, USA, 2011. ACM.

[141] KOENIG, R., L. BAUER, T. STRIPF, M. SHAFIQUE, W. AHMED, J. BECKER and J. HENKEL: *KAHRISMA: A Novel Hypermorphic Reconfigurable-Instruction-Set Multi-grained-Array Architecture*. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pp. 819–824, Mar. 2010.

[142] KONGETIRA, P., K. AINGARAN and K. OLUKOTUN: *Niagara: a 32-way multithreaded Sparc processor*. IEEE Micro, 25(2):21–29, Mar. 2005.

[143] KRISHNA, T., L.-S. PEH, B. M. BECKMANN and S. K. REINHARDT: *Towards the Ideal On-chip Fabric for 1-to-many and Many-to-1 Communication*. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pp. 71–82, New York, NY, USA, 2011. ACM.

[144] KUMAR, A., L.-S. PEH, P. KUNDU and N. K. JHA: *Express Virtual Channels: Towards the Ideal Interconnection Fabric*. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pp. 150–161, New York, NY, USA, 2007. ACM.

[145] KUMAR, S., A. JANTSCH, J.-P. SOININEN, M. FORSELL, M. MILLBERG, J. OBERG, K. TIENSYRJA and A. HEMANI: *A network on chip architecture and design methodology*. In *IEEE Computer Society Annual Symposium on VLSI, 2002. Proceedings*, pp. 105–112, 2002.

[146] KWON, W.-C., S. YOO, J. UM and S.-W. JEONG: *In-network reorder buffer to improve overall NoC performance while resolving the in-order requirement problem*. In *Design, Automation Test in Europe Conference Exhibition, DATE '09.*, 2009.

[147] LAJOLO, M., G. PALERMO and C. SILVANO: *Low Power Networks-on-Chip*. Springer, 2011.

[148] LAN, Y.-C., H.-A. LIN, S.-H. LO, Y. H. HU and S.-J. CHEN: *A Bidirectional NoC (BiNoC) Architecture With Dynamic Self-Reconfigurable Channel*. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 30(3):427–440, 2011.

[149] LANKES, A., T. WILD, S. WALLENTOWITZ and A. HERKERSDORF: *Benefits of Selective Packet Discard in Networks-on-chip*. ACM Trans. Archit. Code Optim., 9(2):12:1–12:21, June 2012.

[150] LARI, V., S. MUDDASANI, S. BOPPU, F. HANNIG, M. SCHMID and J. TEICH: *Hierarchical Power Management for Adaptive Tightly-coupled Processor Arrays*. ACM Trans. Des. Autom. Electron. Syst., 18(1):2:1–2:25, Jan. 2013.

[151] LARI, V., A. NAROVLYANSKYY, F. HANNIG and J. TEICH: *Decentralized dynamic resource management support for massively parallel processor arrays*. In *2011 IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 87–94, Sept. 2011.

[152] LEE, J. W., M. C. NG and K. ASANOVIC: *Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks*. SIGARCH Comput. Archit. News, 36:89–100, June 2008.

[153] LEE, K., S.-J. LEE and H.-J. YOO: *Low-power network-on-chip for high-performance SoC design*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 14(2):148–160, Feb. 2006.

[154] LEE, W. R. and R. A. BERGAMASCHI: *Designing Systems-on-Chip using Cores*. In *Design Automation Conference*, vol. 0, pp. 420–425, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[155] LEI, T. and S. KUMAR: *A two-step genetic algorithm for mapping task graphs to a network on chip architecture*. In *Euromicro Symposium on Digital System Design, 2003. Proceedings*, pp. 180–187, Sept. 2003.

[156] LEISERSON, C.: *Fat-trees: Universal networks for hardware-efficient supercomputing*. IEEE Transactions on Computers, C-34(10):892–901, Oct. 1985.

[157] LEROY, A., P. MARCHAL, A. SHICKOVA, F. CATTHOOR, F. ROBERT and D. VERKEST: *Spatial Division Multiplexing: A Novel Approach for Guaranteed Throughput on NoCs*. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '05, pp. 81–86, New York, NY, USA, 2005. ACM.

[158] LEVEUGLE, R. and K. HADJIAT: *Multi-Level Fault Injections in VHDL Descriptions: Alternative Approaches and Experiments*. J. Electron. Test., 19(5):559–575, Oct. 2003.

[159] LI, B., L. ZHAO, R. IYER, L.-S. PEH, M. LEDDIGE, M. ESPIG, S. E. LEE and D. NEWELL: *CoQoS: Coordinating QoS-aware Shared Resources in NoC-based SoCs*. J. Parallel Distrib. Comput., 71(5):700–713, 2011.

[160] LIAO, T. G. S., G. MARTIN, S. SWAN and T. GRÖTKER: *System design with SystemC*. Springer, 2002.

[161] LIN, X. and L. NI: *Multicast communication in multicomputer networks*. IEEE Transactions on Parallel and Distributed Systems, 4(10):1105–1117, 1993.

[162] LINDHOLM, E., J. NICKOLLS, S. OBERMAN and J. MONTRYM: *NVIDIA Tesla: A unified graphics and computing architecture*. IEEE micro, 28(2):39–55, 2008.

[163] LIS, M., K. S. SHIM, M. H. CHO, P. REN, O. KHAN and S. DEVADAS: *Darsim: A Parallel Cycle-Level NoC Simulator*. MIT web domain, 2010.

[164] LIU, C., L. ZHANG, Y. HAN and X. LI: *Vertical Interconnects Squeezing in Symmetric 3D Mesh Network-on-chip*. In *Proceedings of the 16th Asia and South Pacific Design Automation Conference*, ASPDAC '11, pp. 357–362, Piscataway, NJ, USA, 2011. IEEE Press.

[165] LOI, I., S. MITRA, T. H. LEE, S. FUJITA and L. BENINI: *A Low-overhead Fault Tolerance Scheme for TSV-based 3D Network on Chip Links*. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '08, pp. 598–602, Piscataway, NJ, USA, 2008. IEEE Press.

[166] LU, Z. and A. JANTSCH: *TDM Virtual-Circuit Configuration for Network-on-Chip*. VLSI, 2008.

[167] LU, Z., R. THID, M. MILLBERG and A. JANTSCH: *Nnse: Nostrum network-on-chip simulation environment*. In *In Proc. of SSoCC*, 2005.

[168] LU, Z., B. YIN and A. JANTSCH: *Connection-oriented multicasting in wormhole-switched networks on chip*. In *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006*, vol. 00, pp. 6 pp.–, Mar. 2006.

[169] MANFERDELLI, J., N. GOVINDARAJU and C. CRALL: *Challenges and Opportunities in Many-Core Computing*. Proceedings of the IEEE, 96(5):808–815, 2008.

[170] MARCULESCU, R., U. OGRAS, L.-S. PEH, N. JERGER and Y. HOSKOTE: *Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28(1):3–21, Jan. 2009.

[171] MARESCAUX, T., B. BRICKE, P. DEBACKER, V. NOLLET and H. CORPORAAL: *Dynamic time-slot allocation for QoS enabled networks on chip*. In *Embedded Systems for Real-Time Multimedia, 2005. 3rd Workshop on*, pp. 47–52, 2005.

[172] MARTINEZ, J.-M., P. LOPEZ, J. DUATO and T. PINKSTON: *Software-based deadlock recovery technique for true fully adaptive routing in wormhole networks*. In , *Proceedings of the 1997 International Conference on Parallel Processing*, 1997.

[173] MATSUTANI, H., M. KOIBUCHI, D. IKEBUCHI, K. USAMI, H. NAKAMURA and H. AMANO: *Performance, Area, and Power Evaluations of Ultrafine-Grained Run-Time Power-Gating Routers for CMPs*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30(4):520–533, Apr. 2011.

[174] MELLO, A., L. TEDESCO, N. CALAZANS and F. MORAES: *Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC*. In *Proceedings of the 18th Annual Symposium on Integrated Circuits and System Design*, SBCCI '05, pp. 178–183, New York, NY, USA, 2005. ACM.

[175] MICHELI, G. D. and L. BENINI: *Networks on Chips: Technology and Tools*. Academic Press, Aug. 2006.

[176] MICHELOGIANNAKIS, G., J. BALFOUR and W. DALLY: *Elastic-buffer flow control for on-chip networks*. In *IEEE 15th International Symposium on High Performance Computer Architecture, 2009. HPCA 2009*, pp. 151–162, Feb. 2009.

[177] MILLBERG, M., E. NILSSON, R. THID and A. JANTSCH: *Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip*. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, vol. 2, pp. 890–895 Vol.2, 2004.

[178] MILLBERG, M., E. NILSSON, R. THID, S. KUMAR and A. JANTSCH: *The Nostrum backbone-a communication protocol stack for Networks on Chip*. In *17th International Conference on VLSI Design, 2004. Proceedings*, pp. 693–696, 2004.

[179] MOORE, G. E. et al.: *Cramming more components onto integrated circuits*. Electronics Magazine, 38(8), 1965.

[180] MORAES, F., N. CALAZANS, A. MELLO, L. MÖLLER and L. OST: *HERMES: an infrastructure for low area overhead packet-switching networks on chip*. Integration, the VLSI Journal, 38(1):69–93, Oct. 2004.

[181] MUBEEN, S. and S. KUMAR: *Designing Efficient Source Routing for Mesh Topology Network on Chip Platforms*. In *Proceedings of the 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools*, DSD '10, pp. 181–188, Washington, DC, USA, 2010. IEEE Computer Society.

[182] MURALI, S., D. ATIENZA, L. BENINI and G. DE MICHEL: *A multi-path routing strategy with guaranteed in-order packet delivery and fault-tolerance for networks on chip*. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pp. 845–848, New York, NY, USA, 2006. ACM.

[183] MURALI, S., P. MELONI, F. ANGIOLINI, D. ATIENZA, S. CARTA, L. BENINI, G. DE MICHELI and L. RAFFO: *Designing Application-specific Networks on Chips with Floorplan Information*. In *Proceedings of the 2006 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '06, pp. 355–362, New York, NY, USA, 2006. ACM.

[184] MURALI, S., T. THEOCHARIDES, N. VIJAYKRISHNAN, M. IRWIN, L. BENINI

and G. DE MICHELI: *Analysis of error recovery schemes for networks on chips*. IEEE Design Test of Computers, 22(5):434–442, Sept. 2005.

[185] NI, L. and P. MCKINLEY: *A survey of wormhole routing techniques in direct networks*. Computer, 26(2):62–76, Feb. 1993.

[186] NICKOLLS, J., I. BUCK, M. GARLAND and K. SKADRON: *Scalable Parallel Programming with CUDA*. Queue, 6(2):40–53, 2008.

[187] NUMRICH, R. W. and J. REID: *Co-array Fortran for Parallel Programming*. SIGPLAN Fortran Forum, 17(2):1–31, Aug. 1998.

[188] OECHSLEIN, B., J. SCHEDEL, J. KLEINÖDER, L. BAUER, J. HENKEL, D. LOHMANN and W. SCHRÖDER-PREIKSCHAT: *OctoPOS: A parallel operating system for invasive computing*. Sventek, J.(Hrsg.), 2011.

[189] OGRAS, U., R. MARCULESCU, P. CHOUDHARY and D. MARCULESCU: *Voltage-Frequency Island Partitioning for GALS-based Networks-on-Chip*. In *44th ACM/IEEE Design Automation Conference, DAC '07*, pp. 110–115, June 2007.

[190] OGRAS, U., R. MARCULESCU and D. MARCULESCU: *Variation-adaptive feedback control for networks-on-chip with multiple clock domains*. In *45th ACM/IEEE Design Automation Conference, 2008. DAC 2008*, pp. 614–619, June 2008.

[191] OGRAS, U., R. MARCULESCU, D. MARCULESCU and E.-G. JUNG: *Design and Management of Voltage-Frequency Island Partitioned Networks-on-Chip*. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 17(3):330–341, Mar. 2009.

[192] OGRAS, U. Y., J. HU and R. 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, pp. 69–74, New York, NY, USA, 2005. ACM.

[193] OGRAS, U. Y. and R. MARCULESCU: *Prediction-based flow control for network-on-chip traffic*. In *Proceedings of the 43rd annual Design Automation Conference*, pp. 839–844, San Francisco, CA, USA, 2006. ACM.

[194] PALMA, J. C. S., L. S. INDRUSIAK, F. G. MORAES, A. G. ORTIZ, M. GLESNER and R. A. L. REIS: *Adaptive Coding in Networks-on-Chip: Transition Activity Reduction Versus Power Overhead of the Codec Circuitry*. In VOUNCKX, J., N. AZEMARD and P. MAURINE (eds.): *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, no. 4148 in *Lecture Notes in Computer Science*, pp. 603–613. Springer Berlin Heidelberg, Jan. 2006.

[195] PASRICHA, S., N. DUTT and M. BEN-ROMDHANE: *Constraint-driven Bus Matrix Synthesis for MPSoC*. In *Proceedings of the Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pp. 30–35. IEEE Press, 2006.

[196] PASRICHA, S., Y. ZOU, D. CONNORS and H. SIEGEL: *OE+IOE: A novel turn model based fault tolerant routing scheme for networks-on-chip*. In *2010 IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 85–93, 2010.

[197] PASTRNAK, M., P. DE WITH and J. VAN MEERBERGEN: *Realization of QoS management using negotiation algorithms for multiprocessor NoC*. In *2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. Proceedings*, pp. 4 pp.–, 2006.

[198] PATOOGHY, A. and S. MIREMADI: *XYX: A Power & Performance Efficient Fault-Tolerant Routing Algorithm for Network on Chip*. In *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pp. 245–251, Feb. 2009.

[199] PATTERSON, D. A.: *Reduced Instruction Set Computers*. Commun. ACM, 28(1):8–21, Jan. 1985.

[200] PATTERSON, D. A. and D. R. DITZEL: *The Case for the Reduced Instruction Set Computer*. SIGARCH Comput. Archit. News, 8(6):25–33, 1980.

[201] PHAM, D., S. ASANO, M. BOLLIGER, M. DAY, H. HOFSTEE, C. JOHNS, J. KAHLE, A. KAMEYAMA, J. KEATY, Y. MASUBUCHI, M. RILEY, D. SHIPPY, D. STASIAK, M. SUZUOKI, M. WANG, J. WARNOCK, S. WEITZEL, D. WENDEL, T. YAMAZAKI and K. YAZAWA: *The design and implementation of a first-generation CELL processor*. In *Solid-State Circuits Conference, Digest of Technical Papers. ISSCC. 2005 IEEE International*, pp. 184–592 Vol. 1, Feb. 2005.

[202] PINKSTON, T.: *Flexible and efficient routing based on progressive deadlock recovery*. IEEE Transactions on Computers, 48(7):649–669, July 1999.

[203] PIRRETTI, M., G. LINK, R. BROOKS, N. VIJAYKRISHNAN, M. KANDEMIR and M. IRWIN: *Fault tolerant algorithms for network-on-chip interconnect*. In *IEEE Computer society Annual Symposium on VLSI, 2004. Proceedings*, pp. 46–51, Feb. 2004.

[204] PLAS, G. VAN DER, P. LIMAYE, I. LOI, A. MERCHA, H. OPRINS, C. TORREGIANI, S. THIJS, D. LINTEN, M. STUCCHI, G. KATTI, D. VELENIS, V. CHERMAN, B. VANDEVELDE, V. SIMONS, I. DE WOLF, R. LABIE, D. PERRY, S. BRONCKERS, N. MINAS, M. CUPAC, W. RUYTHOOREN, J. VAN OLMEN, A. PHOMMAHAXAY, M. DE POTTER DE TEN BROECK, A. OPDEBEECK, M. RAKOWSKI, B. DE WACHTER, M. DEHAN, M. NELIS, R. AGARWAL, A. PULLINI, F. ANGIOLINI, L. BENINI, W. DEHAENE, Y. TRAVALY, E. BEYNE and P. MARCHAL: *Design Issues and Considerations for Low-Cost 3-D TSV IC Technology*. IEEE Journal of Solid-State Circuits, 46(1):293–307, Jan. 2011.

[205] POP, E.: *Energy dissipation and transport in nanoscale devices*. Nano Research, 3(3):147–169, Mar. 2010.

[206] PROTIC, J., M. TOMASEVIC and V. MILUTINOVIC: *Distributed Shared Memory: Concepts and Systems*. John Wiley & Sons, 1998.

[207] PUJARI, R., T. WILD, A. HERKERSDORF, B. VOGEL and J. HENKEL: *Hardware assisted thread assignment for RISC based MPSoCs in invasive computing*. In *2011 13th International Symposium on Integrated Circuits (ISIC)*, pp. 106–109, Dec. 2011.

[208] PUSCHINI, D., F. CLERMIDY, P. BENOIT, G. SASSATELLI and L. TORRES: *Temperature-Aware Distributed Run-Time Optimization on MP-SoC Using Game Theory*. In *Symposium on VLSI, 2008. ISVLSI '08. IEEE Computer Society Annual*, pp. 375–380, Apr. 2008.

[209] QIAN, Y., Z. LU and Q. DOU: *QoS scheduling for NoCs: Strict Priority Queueing versus Weighted Round Robin*. In *Computer Design (ICCD), 2010 IEEE International Conference on*, pp. 52–59, 2010.

[210] RADETZKI, M., C. FENG, X. ZHAO and A. JANTSCH: *Methods for Fault Tolerance in Networks-on-chip*. ACM Comput. Surv., 46(1):8:1–8:38, July 2013.

[211] RAGHUNATHAN, V., M. B. SRIVASTAVA and R. K. GUPTA: *A Survey of Techniques for Energy Efficient On-chip Communication*. In *Proceedings of the 40th Annual Design Automation Conference*, DAC '03, pp. 900–905, New York, NY, USA, 2003. ACM.

[212] RAIK, J., R. UBAR and V. GOVIND: *Test Configurations for Diagnosing Faulty Links in NoC Switches*. In *Test Symposium, 2007. ETS '07. 12th IEEE European*, pp. 29–34, 2007.

[213] RAMEY, C.: *Tile-gx100 manycore processor: Acceleration interfaces and architecture*. Hot Chips 23, 2011.

[214] RIJPKEMA, E., K. GOOSSENS, A. RĂDULESCU, J. DIELISSEN, J. VAN MEERBERGEN, P. WIELAGE and E. WATERLANDER: *Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip*. IEE Proceedings-Computers and Digital Techniques, 150(5):294–302, 2003.

[215] RONEN, R., A. MENDELSON, K. LAI, S.-L. LU, F. POLLACK and J. P. SHEN: *Coming challenges in microarchitecture and architecture*. In *Proceedings of the IEEE*, pp. 325–340, 2001.

[216] ROUSE, M.: *Multi-core processor*. http://searchdatacenter.techtarget.com/definition/multi-core-processor, 2014. Accessed: 2014-02-05.

[217] SAHA, S.: *Modeling Process Variability in Scaled CMOS Technology*. IEEE Design Test of Computers, 27(2):8–16, Mar. 2010.

[218] SAHU, P. K. and S. CHATTOPADHYAY: *A survey on application mapping strategies for Network-on-Chip design*. Journal of Systems Architecture, 59(1):60–76, Jan. 2013.

[219] SAMMAN, F. A., T. HOLLSTEIN and M. GLESNER: *Multicast Parallel Pipeline Router Architecture for Network-on-chip*. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08, pp. 1396–1401, New York, NY, USA, 2008. ACM.

[220] SCHAFER, M., T. HOLLSTEIN, H. ZIMMER and M. GLESNER: *Deadlock-free routing and component placement for irregular mesh-based networks-on-chip*. In *IEEE/ACM International Conference on Computer-Aided Design, 2005. ICCAD-2005*, pp. 238–245, Nov. 2005.

[221] SCHLANSKER, M. and B. RAU: *EPIC: Explicitly Parallel Instruction Computing*. Computer, 33(2):37–45, Feb. 2000.

[222] SCHONWALD, T., J. ZIMMERMANN, O. BRINGMANN and W. ROSENSTIEL: *Fully Adaptive Fault-Tolerant Routing Algorithm for Network-on-Chip Architectures*. In *10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2007. DSD 2007*, pp. 527–534, Aug. 2007.

[223] SCHWARZ, S.: *Ring network for communication between one chip processors*, July 26 1988. US Patent 4,760,571.

[224] SEILER, L., D. CARMEAN, E. SPRANGLE, T. FORSYTH, M. ABRASH, P. DUBEY, S. JUNKINS, A. LAKE, J. SUGERMAN, R. CAVIN, R. ESPASA, E. GROCHOWSKI, T. JUAN and P. HANRAHAN: *Larrabee: A Many-core x86 Architecture for Visual Computing*. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pp. 18:1–18:15, New York, NY, USA, 2008. ACM.

[225] SETHURAMAN, B., P. BHATTACHARYA, J. KHAN and R. VEMURI: *LiPaR: A Light-weight Parallel Router for FPGA-based Networks-on-chip*. In *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, GLSVLSI '05, pp. 452–457, New York, NY, USA, 2005. ACM.

[226] SHACHAM, A., K. BERGMAN and L. P. CARLONI: *On the Design of a Photonic Network-on-Chip*. In *Proceedings of the First International Symposium on Networks-on-Chip*, NOCS '07, pp. 53–64, Washington, DC, USA, 2007. IEEE Computer Society.

[227] SHAMSHIRI, S., A.-A. GHOFRANI and K.-T. CHENG: *End-to-end error correction and online diagnosis for on-chip networks*. In *Test Conference (ITC), 2011 IEEE International*, pp. 1–10, Sept. 2011.

[228] SHIN, D. and J. KIM: *Power-aware Communication Optimization for Networks-on-chips with Voltage Scalable Links*. In *Proceedings of the 2Nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '04, pp. 170–175, New York, NY, USA, 2004. ACM.

[229] SHIVAKUMAR, P., M. KISTLER, S. KECKLER, D. BURGER and L. ALVISI: *Modeling the effect of technology trends on the soft error rate of combinational logic*. In *International Conference on Dependable Systems and Networks, 2002.*

DSN 2002. Proceedings, pp. 389–398, 2002.

[230] SIGUENZA-TORTOSA, D. and J. NURMI: *Proteo: a new approach to network-on-chip*. In *Proceedings of IASTED International Conference on Communication Systems and Network, Malaga, Spain*, 2002.

[231] SINGH, A., W. J. DALLY, A. K. GUPTA and B. TOWLES: *GOAL: A Load-balanced Adaptive Routing Algorithm for Torus Networks*. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pp. 194–205, New York, NY, USA, 2003. ACM.

[232] SINGH, A. K., T. SRIKANTHAN, A. KUMAR and W. JIGANG: *Communication-aware Heuristics for Run-time Task Mapping on NoC-based MPSoC Platforms*. J. Syst. Archit., 56(7):242–255, July 2010.

[233] SIVARAM, R., D. PANDA and C. STUNKEL: *Efficient broadcast and multicast on multistage interconnection networks using multiport encoding*. IEEE Transactions on Parallel and Distributed Systems, 9(10):1004–1028, 1998.

[234] SNIR, M., S. W. OTTO, D. W. WALKER, J. DONGARRA and S. HUSS-LEDERMAN: *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995.

[235] SNIR, M., S. W. OTTO, D. W. WALKER, J. DONGARRA and S. HUSS-LEDERMAN: *MPI: The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995.

[236] SPEAR, CHRIS: *SystemVerilog for Verification*. Springer, 2006.

[237] STEFAN, R. and K. GOOSSENS: *A TDM slot allocation flow based on multipath routing in NoCs*. Microprocessors and Microsystems, 35(2):130 – 138, 2011. Special issue on Network-on-Chip Architectures and Design Methodologies.

[238] SUTHERLAND, S., P. MOORBY, S. DAVIDMANN and P. FLAKE: *SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer, Sept. 2006.

[239] TAKEDA, E., C. Y.-W. YANG and A. MIURA-HAMADA: *Hot-carrier Effects in MOS Devices*. Academic Press, 1995.

[240] TAKEUCHI, M., Y. MAKINO, K. KAWACHIYA, H. HORII, T. SUZUMURA, T. SUGANUMA and T. ONODERA: *Compiling X10 to Java*. In *Proceedings of the 2011 ACM SIGPLAN X10 Workshop*, X10 '11, pp. 3:1–3:10, New York, NY, USA, 2011. ACM.

[241] TAMHANKAR, R., S. MURALI, S. STERGIOU, A. PULLINI, F. ANGIOLINI, L. BENINI and G. DE MICHELI: *Timing-Error-Tolerant Network-on-Chip Design Methodology*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(7):1297–1310, July 2007.

[242] TAYLOR, M., J. KIM, J. MILLER, D. WENTZLAFF, F. GHODRAT, B. GREENWALD, H. HOFFMAN, P. JOHNSON, J.-W. LEE, W. LEE, A. MA, A. SARAF, M. SENESKI, N. SHNIDMAN, V. STRUMPEN, M. FRANK, S. AMARASINGHE and A. AGARWAL: *The Raw microprocessor: a computational fabric for software circuits and general-purpose programs*. IEEE Micro, 22(2):25–35, 2002.

[243] TEICH, J.: *Invasive Algorithms and Architectures*. it - Information Technology, 2008.

[244] TEICH, J.: *Transregional Collaborative Research Centre 89 - Invasive Computing - Homepage*, 2014.

[245] TEICH, J., J. HENKEL, A. HERKERSDORF, D. SCHMITT-LANDSIEDEL, W. SCHRÖDER-PREIKSCHAT and G. SNELTING: *Invasive Computing: An Overview*. In HÜBNER, M. and J. BECKER (eds.): *Multiprocessor System-on-Chip*, pp. 241–268. Springer New York, Jan. 2011.

[246] TEICH, J., A. TANASE and F. HANNIG: *Symbolic parallelization of loop programs for massively parallel processor arrays*. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pp. 1–9, June 2013.

[247] THOMA, F., M. KUHNLE, P. BONNOT, E. PANAINTE, K. BERTELS, S. GOLLER, A. SCHNEIDER, S. GUYETANT, E. SCHULER, K. M"ULLER-GLASER and J. BECKER: *MORPHEUS: Heterogeneous Reconfigurable Computing*. In *International Conference on Field Programmable Logic and Applications, 2007. FPL 2007*, pp. 409–414, Aug. 2007.

[248] THOMPSON, C. D.: *Area-time Complexity for VLSI*. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pp. 81–88, New York, NY, USA, 1979. ACM.

[249] THOMPSON, C. D. and H. T. KUNG: *Sorting on a Mesh-connected Parallel Computer*. Commun. ACM, 20(4):263–271, Apr. 1977.

[250] TILERA: *TILE64 Processor Product Brief*. http://www.tilera.com/sites/default/files/productbriefs/PB010_TILE64_Processor_A_v4.pdf, 2008. Accessed: 2014-02-06.

[251] TINDELL, K., H. HANSSON and A. J. WELLINGS: *Analysing real-time communications: controller area network (CAN)*. In *Real-Time Systems Symposium, 1994., Proceedings.*, pp. 259–263, Dec 1994.

[252] TOL, E. B. V. D. and E. G. T. JASPERS: *Mapping of MPEG-4 decoding on a flexible architecture platform*. In *Media Processors 2002*, pp. 1–13, 2002.

[253] TORRES, L., P. BENOIT, G. SASSATELLI, M. ROBERT, F. CLERMIDY and D. PUSCHINI: *An Introduction to Multi-Core System on Chip - Trends and Challenges*. In HÜBNER, M. and J. BECKER (eds.): *Multiprocessor System-on-Chip*, pp. 1–21. Springer New York, Jan. 2011.

[254] TRADOWSKY, C., F. THOMA, M. HUBNER and J. BECKER: *On Dynamic Runtime Processor Pipeline Reconfiguration*. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pp. 419–424, May 2012.

[255] TRAN, A. T. and B. BAAS: *NoCTweak: a Highly Parameterizable Simulator for Early Exploration of Performance and Energy of Networks On-Chip*. Techn. Rep., VCL, University of California, 2012.

[256] UNION, I. T.: *ITU-T Recommendation database - Information technology - Open Systems Interconnection - Basic Reference Model: The basic model*.

[257] VANGAL, S., J. HOWARD, G. RUHL, S. DIGHE, H. WILSON, J. TSCHANZ, D. FINAN, P. IYER, A. SINGH, T. JACOB, S. JAIN, S. VENKATARAMAN, Y. HOSKOTE and N. BORKAR: *An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS*. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pp. 98–589, Feb. 2007.

[258] VASSILIADIS, S., S. WONG, G. GAYDADJIEV, K. BERTELS, G. KUZMANOV and E. PANAINTE: *The MOLEN polymorphic processor*. IEEE Transactions on Computers, 53(11):1363–1375, Nov. 2004.

[259] VOROS, N. S., M. H"UBNER, J. BECKER, M. K"UHNLE, F. THOMAITIV, A. GRASSET, P. BRELET, P. BONNOT, F. CAMPI, E. SCH"ULER, H. SAHLBACH, S. WHITTY, R. ERNST, E. BILLICH, C. TISCHENDORF, U. HEINKEL, F. IEROMNIMON, D. KRITHARIDIS, A. SCHNEIDER, J. KNAEBLEIN and W. PUTZKE-R"OMING: *MORPHEUS: A Heterogeneous Dynamically Reconfigurable Platform for Designing Highly Complex Embedded Systems*. ACM Trans. Embed. Comput. Syst., 12(3):70:1–70:33, Apr. 2013.

[260] WANG, J. and Y. LEVY: *Managing performance using weighted round-robin*. In *Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on*, pp. 785–792, 2000.

[261] WANG, X., M. YANG, Y. JIANG and P. LIU: *Power-Aware Mapping for Network-on-Chip Architectures under Bandwidth and Latency Constraints*. In *4th International Conference on Embedded and Multimedia Computing, 2009. EM-Com 2009*, pp. 1–6, 2009.

[262] WANG, Y., K. ZHOU, Z. LU and H. YANG: *Dynamic TDM virtual circuit implementation for NoC*. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pp. 1533–1536, 2008.

[263] WENTZLAFF, D., P. GRIFFIN, H. HOFFMANN, L. BAO, B. EDWARDS, C. RAMEY, M. MATTINA, C.-C. MIAO, J. F. BROWN III and A. AGARWAL: *On-Chip Interconnection Architecture of the Tile Processor*. IEEE Micro, 27(5):15–31, Sept. 2007.

[264] WGSIMON: *Transistor Count and Moore's Law - 2011*. http://en.wikipedia.org/

wiki/File:Transistor_Count_and_Moore%27s_Law_-_2011.svg, 2014. Accessed: 2014-01-16.

[265] WIJNGAART, R. F. VAN DER, T. G. MATTSON and W. HAAS: *Light-weight Communications on Intel's Single-chip Cloud Computer Processor*. SIGOPS Oper. Syst. Rev., 45(1):73–83, Feb. 2011.

[266] WIKLUND, D. and D. LIU: *Switched interconnect for system-on-a-chip design*. In *Proceedings of the IP2000 Europe Conference*, pp. 185–192, 2000.

[267] WIKLUND, D. and D. LIU: *SoCBUS: Switched Network on Chip for Hard Real Time Embedded Systems*. In *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, IPDPS '03, p. 8 pp., Washington, DC, USA, 2003. IEEE Computer Society.

[268] WOLF, W., A. JERRAYA and G. MARTIN: *Multiprocessor System-on-Chip (MPSoC) Technology*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 27(10):1701–1713, Oct. 2008.

[269] WOLKOTTE, P., G. J. M. SMIT, G. RAUWERDA and L. SMIT: *An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip*. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 155a–155a, Apr. 2005.

[270] WU, J. and D. WANG: *Fault-tolerant and deadlock-free routing in 2-D meshes using rectilinear-monotone polygonal fault blocks*. In *International Conference on Parallel Processing, 2002. Proceedings*, pp. 247–254, 2002.

[271] WU, Q., M. PEDRAM and X. WU: *Clock-gating and its application to low power design of sequential circuits*. IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 47(3):415–420, Mar. 2000.

[272] XILINX: *Xilinx UG534 ML605 Hardware User Guide (v1.8)*, 2012.

[273] XU, T. C., A. W. YIN, P. LILJEBERG and H. TENHUNEN: *A study of 3D Network-on-Chip design for data parallel H.264 coding*. Microprocessors and Microsystems, 35(7):603–612, 2011.

[274] YANG, X., Z. QING-LI, F. FANG-FA, Y. MING-YAN and L. CHENG: *NISAR: An AXI compliant on-chip NI architecture offering transaction reordering processing*. In *7th International Conference on ASIC, 2007. ASICON '07*, pp. 890–893, Oct. 2007.

[275] YEAP, G.: *Practical Low Power Digital VLSI Design*. Springer US, Boston, MA, 1998.

[276] ZAMFIRESCU, C. and T. ZAMFIRESCU: *Hamiltonian Properties of Grid Graphs*. SIAM Journal on Discrete Mathematics, 5(4):564–570, Nov. 1992.

[277] ZEFERINO, C. and A. SUSIN: *SoCIN: a parametric and scalable network-on-chip*. In *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI*

2003. Proceedings, pp. 169–174, Sept. 2003.

[278] ZHANG, B. and M. ORSHANSKY: *Modeling of NBTI-Induced PMOS Degradation under Arbitrary Dynamic Temperature Variation*. In *9th International Symposium on Quality Electronic Design, 2008. ISQED 2008*, pp. 774–779, Mar. 2008.

[279] ZHANG, L., E. E. REGENTOVA and X. TAN: *Packet switching optical network-on-chip architectures*. Computers & Electrical Engineering, 39(2):697–714, 2013.

[280] ZHANG, L., M. YANG, Y. JIANG and E. REGENTOVA: *Architectures and routing schemes for optical network-on-chips*. Computers & Electrical Engineering, 35(6):856–877, Nov. 2009.

[281] ZHANG, Z., A. GREINER and S. TAKTAK: *A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh Network-on-Chip*. In *45th ACM/IEEE Design Automation Conference, 2008. DAC 2008*, pp. 441–446, June 2008.

[282] ZWINKAU, A., S. BUCHWALD and G. SNELTING: *InvadeX10 Documentation v0.5*. Techn. Rep. 7, Karlsruhe Institute of Technology, 2013.

I

Supervised Student Research

[Bis12] BISCHOF, SIMON: *Realisierung eines NoC-Routers für Knoten mit hohem Bandbreitenbedarf*. Bachelor thesis ID-1645, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Bis14] BISCHOF, SIMON: *Anleitung für den Synopsis Low power Flow*, 2014.

[Bou11] BOUHZAM, YOUNES: *Entwurf und Implementierung eines Network-on-Chip Routers mit dynamischer Buffer-Allokation*. Studienarbeit IL-936, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2011.

[Bul12] BULL, CHRISTIAN: *Evaluation und Implementierung von Adaptivem Routing für das invasive Network-on-Chip*. Studienarbeit IL-1007, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Dei12] DEININGER, TOBIAS: *Prototyping und Evaluation einer iNoC-basierten Multicore-Architektur*. Diploma thesis ID-1571, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Dud13] DUDEN, MARCO: *Fehlertoleranz bei Networks on Chips*. Seminararbeit, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Dud14] DUDEN, MARCO: *Behandlung permanenter Fehler im invasiven Network on Chip (Engl.: Facing Permanent Errors in the Invasive Network on Chip)*. Bachelor thesis ID-1834, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[Ese13] ESER, YILMAZ: *Lastabhängige Adaption der Paketgröße für On-Chip Netzwerke*. Diploma thesis ID-1720, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Hau14] HAUSSECKER, MANUEL: *Multi-FPGA NoC - Design & Prototyping*. Bachelor thesis ID-1809, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[Imr12] IMRANI, MOHAMMED ALI ZERBOUH: *Evaluation und Realisierung einer neuartigen Network-on-Chip-Topologie*. Studienarbeit IL-962, Karlsruhe

Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Kar12] KARLE, MARTIN: *Monitoring Funktionalitäten für Networks-on-Chip (Engl.: Monitoring Functionalities for Networks-on-Chip)*. Bachelor thesis ID-1650, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Ker13] KERN, MATTHIAS: *Entwicklung eines Many-Core Prozessormodells auf Basis eines Network-on-Chip mit QoS Unterstützung (Engl.: Development of a many-core processor model based on a network-on-chip with QoS support)*. Bachelor thesis ID-1691, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Klö13] KLÖPFER, ROMAN: *Erkennung und Lokalisation fehlerhafter NoC-Router*. Diploma thesis ID-1734, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Kön13] KÖNIG, STEFAN: *Entwicklung von Methoden zur haptischen Unterstützung des Chirurgen bei Verwendung telemanipulierter Leichtbauroboter (Engl.: Development of Methods for haptic assistance during telemanipulated surgical tasks using a light-weight robot)*. Bachelor thesis ID-1765, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV) & Institut für Prozessrechentechnik, Automation und Robotik (IPR), 2013.

[Kri14] KRIMM, MICHAEL: *Parallele Anwendungen für die InvasIC Architektur (Engl.: Parallel Applications for the InvasIC Architecture)*. Bachelor thesis ID-1811, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[Kro13] KROLACSEK, EDUARD: *iNoC Firmware Entwicklung (Engl.: iNoC Firmware Development)*. Bachelor thesis ID-1750, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Lu13] LU, QIAN: *Optimierung des Datendurchsatzes reservierter End-zu-End-Verbindungen in Networks-on-Chip*. Diploma thesis ID-1636, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Mat12] MATEEV, LAZAR: *Prototyping und Test einer InvasIC Many-Core-Architektur (Engl.: Prototyping and Test of an InvasIC Many-Core-Architecture)*. Bachelor thesis ID-1646, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Pas14] PASHKOVSKIY, ILYA: *Hardwarebeschleunigung paralleler Anwendungen auf der InvasIC Architektur (Engl.: Hardware acceleration of parallel applications*

on the InvasIC architecture). Diploma thesis ID-1827, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[Ric13] RICKELHOFF, TOBIAS: *Entwicklung und Test einer Debug-Schnittstelle für die InvasIC-Architektur*. Diploma thesis ID-1717, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Sch13] SCHATZ, STEFAN: *Entwurf und Implementierung eines hybriden Network-on-Chip*. Diploma thesis ID-1649, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Sin12] SINGH, MAXIMILIAN: *Implementierung einer verteilten Selbstoptimierungsstrategie für Network-on-Chip (Engl.: Implementation of a distributed self-optimization strategy for Networks-on-Chip)*. Bachelor thesis ID-1623, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Spi12] SPIRIC, MARCO: *Konzeption und Implementierung von Monitoring Funktionalitäten für Networks-on-Chip*. Bachelor thesis ID-1571, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2012.

[Ste13] STEIN, CARSTEN: *Fehlererkennung und -behandlung für reguläre Networks-on-Chip*. Diploma thesis ID-1718, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2013.

[Wie14a] WIEDMANN, JULIAN: *Dynamische Expressverbindungen für Networks on Chip (Engl.: Dynamic express channels for networks on chip)*. Diploma thesis ID-1894, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

[Wie14b] WIEDMANN, JULIAN: *Intelligentes energieoptimierendes Virtual Channel Management*. Studienarbeit IL-1787, Karlsruhe Institute for Technology, Institut für Technik der Informationsverarbeitung (ITIV), 2014.

The supervised student research work can be inspected at the *Karlsruhe Institute of Technology, Institute for Information Processing Technologies (Institut für Technik der Informationsverarbeitung – ITIV), Engesserstr. 5, 76131 Karlsruhe, Germany* or are available there in electronic form.

I

Own publications

Journals

[HKKB13] HEISSWOLF, JAN, RALF KÖNIG, MARTIN KUPPER and JÜRGEN BECKER: *Providing multiple hard latency and throughput guarantees for packet switching networks on chip*. Computers & Electrical Engineering, 39(8):2603–2622, November 2013.

[HZW+13] HEISSWOLF, JAN, AURANG ZAIB, ANDREAS WEICHSLGARTNER, RALF KÖNIG, THOMAS WILD, JÜRGEN TEICH, ANDREAS HERKERSDORF and JÜRGEN BECKER: *Virtual Networks – Distributed Communication Resource Management*. ACM Trans. Reconfigurable Technol. Syst., 6(2):8:1–8:14, August 2013.

Conferences / Symposia

[BFH+12] BECKER, J., S. FRIEDERICH, J. HEISSWOLF, R. KOENIG and D. MAY: *Hardware prototyping of novel invasive multicore architectures*. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 201–206, January 2012.

[FHB14] FRIEDERICH, STEPHANIE, JAN HEISSWOLF and JÜRGEN BECKER: *Hardware/Software Debugging of Large Scale Many-core Architectures*. In *Proceedings of the 27th Symposium on Integrated Circuits and Systems Design*, SBCCI '14, pages 45:1–45:7, New York, NY, USA, 2014. ACM.

[FHMB14] FRIEDERICH, S., J. HEISSWOLF, D. MAY and J. BECKER: *Hardware prototyping and software debugging of multi-core architectures*. In *Proceedings of the Synopsys Users Group Conference (SNUG)*, May 2014.

[HBRB13] HEISSWOLF, J., S. BISCHOF, M. RUCKAUER and J. BECKER: *Efficient memory access in 2D Mesh NoC architectures using high bandwidth routers*. In *2013 26th Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–6, September 2013.

[HHB+12] HENKEL, J., A. HERKERSDORF, L. BAUER, T. WILD, M. HUBNER, R.K. PUJARI, A. GRUDNITSKY, J. HEISSWOLF, A. ZAIB, B. VOGEL, V. LARI and S. KOBBE: *Invasive manycore architectures*. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 193–200, January 2012.

[HKB12] HEISSWOLF, J., R. KÖNIG and J. BECKER: *A Scalable NoC Router Design Providing QoS Support Using Weighted Round Robin Scheduling*. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 625–632, July 2012.

[HSK+13] HEISSWOLF, J., M. SINGH, M. KUPPER, R. KONIG and J. BECKER: *Rerouting: Scalable NoC self-optimization by distributed hardware-based connection reallocation*. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–8, 2013.

[HZZ+14] HEISSWOLF, JAN, AURANG ZAIB, ANDREAS ZWINKAU, SEBASTIAN KOBBE, ANDREAS WEICHSLGARTNER, JÜRGEN TEICH, JÖRG HENKEL, GREGOR SNELTING, ANDREAS HERKERSDORF and JÜRGEN BECKER: *CAP: Communication Aware Programming*. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference (DAC)*, DAC '14, pages 105:1–105:6, New York, NY, USA, 2014. ACM. **HiPEAC Paper Award**.

[KSHB11] KOENIG, R., T. STRIPF, J. HEISSWOLF and J. BECKER: *Architecture design space exploration of run-time scalable issue-width processors*. In *2011 International Conference on Embedded Computer Systems (SAMOS)*, pages 77–84, July 2011.

[PQHW+13] PHAM-QUOC, CUONG, JAN HEISSWOLF, STEPHAN WERNER, ZAID AL-ARS, JURGEN BECKER and KOEN BERTELS: *Hybrid interconnect design for heterogeneous hardware accelerators*. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 843–846, March 2013.

[ZHW+13] ZAIB, A., J. HEISSWOLF, A. WEICHSLGARTNER, T. WILD, J. TEICH, J. BECKER and A. HERKERSDORF: *AUTO-GS: Self-Optimization of NoC Traffic through Hardware Managed Virtual Connections*. In *2013 Euromicro Conference on Digital System Design (DSD)*, pages 761–768, September 2013.

[ZHW+15] ZAIB, AURANG, JAN HEISSWOLF, ANDREAS WEICHSLGARTNER, THOMAS WILD, JÜRGEN TEICH, JÜRGEN BECKER and ANDREAS

HERKERSDORF: *Network Interface with Task Spawning Support for NoC-based DSM Architectures*. In *28th GI/ITG International Conference on Architecture of Computing Systems (ARCS)*. Springer Lecture Notes on Computer Science (LNCS), 2015. To appear.

Workshops

[HWZ+13] HEISSWOLF, J., A. WEICHSLGARTNER, A. ZAIB, R. KONIG, T. WILD, A. HERKERSDORF, J. TEICH and J. BECKER: *Hardware Supported Adaptive Data Collection for Networks on Chip*. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, pages 153–162, 2013.

[HZW+12] HEISSWOLF, J., A. ZAIB, A. WEICHSLGARTNER, R. KONIG, T. WILD, J. TEICH, A. HERKERSDORF and J. BECKER: *Hardware-assisted Decentralized Resource Management for Networks on Chip with QoS*. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 234–241, 2012.

[HZW+14] HEISSWOLF, JAN, AURANG ZAIB, ANDREAS WEICHSLGARTNER, MARTIN KARLE, MAXIMILIAN SINGH, THOMAS WILD, JUERGEN TEICH, ANDREAS HERKERSDORF and JUERGEN BECKER: *The Invasive Network on Chip - A Multi-Objective Many-Core Communication Infrastructure*. In *2014 27th International Conference on Architecture of Computing Systems (ARCS), Workshop Proceedings*, pages 1–8, February 2014.

[KSHB11] KOENIG, R., T. STRIPF, J. HEISSWOLF and J. BECKER: *A Scalable Microarchitecture Design that Enables Dynamic Code Execution for Variable-Issue Clustered Processors*. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW)*, pages 150–157, 2011.

[RWH+13] ROLOFF, SASCHA, ANDREAS WEICHSLGARTNER, JAN HEISSWOLF, FRANK HANNIG and JÜRGEN TEICH: *NoC Simulation in Heterogeneous Architectures for PGAS Programming Model*. In *Proceedings of the 16th International Workshop on Software and Compilers for Embedded Systems (SCOPES)*, M-SCOPES '13, pages 77–85, New York, NY, USA, 2013. ACM.

[WHZ+15] WEICHSLGARTNER, ANDREAS, JAN HEISSWOLF, AURANG ZAIB, THOMAS WILD, ANDREAS HERKERSDORF, JÜRGEN BECKER and JÜRGEN TEICH: *Position Paper: Towards Hardware-Assisted Decentralized Mapping of Applications for Heterogeneous NoC Architectures*. In *Proceed-*

ings of the second International Workshop on Multi-Objective Many-Core Design (MOMAC) in conjunction with International Conference on Architecture of Computing Systems (ARCS), pages 1–4. IEEE, 2015. To appear.