A run-time reconfigurable Network-on-Chip for streaming DSP applications

Nikolay K. Kavaldjiev

Graduation Committee:

Prof. Dr.	P. H. Hartel	University of Twente (promotor)
Dr. Ir.	G. J. M. Smit	University of Twente (assistant-promotor)
Prof. Dr. Ir.	Th. Krol,	University of Twente
Ir.	P. G. Jansen	University of Twente
Prof. Dr.	H. Corporaal	Technical University Eindhoven
Dr. Ir.	K. Goossens	NXP Semiconductors, Eindhoven
Prof. Dr.	J. Nurmi	Tampere University of Technology, Finland

Distributed and Embedded Systems Group
P.O. Box 217, 7500 AE Enschede,
the Netherlands

This research is conducted within the Gecko project funded by the Dutch organisation of Scientific Research NWO under project number 612.064.103.

This thesis is published in the
CTIT Ph.D.-thesis Series No. 06-91
ISSN 1381-3617

This thesis is published in the IPA Dissertation Series under number 2007-02. The current list of titles in the series can be found in the end of this book.

Keywords: network-on-chip, system-on-chip

ISBN: 90-365-2410-5

Printed by Wohrmann Print Service, Zutphen, the Netherlands
www.wps.nl

A RUN-TIME RECONFIGURABLE NETWORK-ON-CHIP FOR STREAMING DSP APPLICATIONS

DISSERTATION

to obtain
the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof.dr. W.H.M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Wednesday, 31 January 2007 at 15.00

by

Nikolay Krasimirov Kavaldjiev

born on 3 December 1973

in Sofia, Bulgaria

This dissertation is approved by:

Prof. Dr. P. H. Hartel	(promotor)
Dr. Ir. G. J. M. Smit	(assistant-promotor)

Abstract

With the advance of semiconductor technology, global on-chip wiring is becoming a limiting factor for the overall performance of large System-on-Chip (SoC) designs. In this thesis we propose a global communication architecture that avoids this limitation by structuring and shortening of the global wires. The communication architecture is used in a multiprocessor SoC for streaming DSP applications. The SoC is intended as a platform for wireless multimedia devices, such as PDAs, mobile phones, mobile medical systems, car infotainment systems, etc.

To improve the performance of the communication in our SoC we use a Network-on-Chip (NoC) architecture. A NoC provides the chip with a high-performance global communication infrastructure, at the same time structures the global on-chip wires and makes their electrical parameters predictable and controllable. By contrast, the bus solutions and the ad-hoc communications solutions used till now in SoC designs result in long wires with unpredictable electrical parameters and require costly design iterations for improving the communication performance.

Our specific NoC uses virtual channel flow control and source routing to provide guaranteed communication services, as well as best effort services. Our NoC is the first on-chip network designed for a run-time reconfigurable system. It offers fast reconfiguration and requires low configuration overhead. Configuring a network path takes less than a millisecond and only costs a few bytes of data overhead. Such time and data overhead is affordable by the run-time reconfigurable SoC for the class of streaming applications we consider.

Our NoC is particularly suitable for the specific traffic conditions created by streaming DSP applications. These applications have a simple structure and create simple traffic patterns but need a high data throughput. The main part of the traffic consists of data streams that require guaranteed services. However, our NoC also supports the small part of the traffic with fine granularity and irregular behaviour that requires only best effort services.

The implementation area of our network router in 0.13 μm technology can be as small as 0.05 mm^2 depending on the network design parameters. A network channel throughput of several Gbit/s can be achieved, which is enough to satisfy the system applications demands.

The specific contributions of this thesis are:

1. We propose a NoC architecture for a run-time reconfigurable multiprocessor SoC that supports streaming DSP applications. To the best of our knowledge, this is the first NoC targeted at a run-time reconfigurable SoC.

2. We propose an architecture of a virtual channel router, which in contrast to conventional architectures is able to provide predictable communication services and has a lower implementation area cost than conventional architectures.

3. The predictable performance of our network simplifies the mapping of streaming DSP applications to our multiprocessor system. System reconfiguration can be done in linear time avoiding the NP-complete solutions common for statically scheduled real-time systems. Thanks to this linearity, system reconfiguration can be done at run-time.

Acknowledgements

At the end of these four years as a PhD student I have to admit that it was not easy. Indeed it was a period in my life when many things have changed, it was the first time for me to live far away from home – an experience that is not always easy to handle. Here I would like to thank to all these people with whom I have worked, from whom I have learned and who have supported me during this long period.

I would like to thank my family and especially my mother Maria and my aunts Rosica and Ilka for their constant support and encouragement. Their love makes me feel near home despite the distance and helps me pass the difficult moments.

First of all, I would like to express my deepest gratitude and respect to my supervisor Gerard Smit. Supporting me in my everyday work, Gerard is always ready to help me, to give me an advice or just talk with me without any reservation, no matter how busy he is. He has been able to sense the slightest signs of doubts and difficulties in my daily work and always was ready to offer his guidance, but in such delicate way that I never lost the feeling of complete freedom in my work.

Although my promoter prof. Pieter Hartel did not work with me on a daily basis I have learned a lot from him. Pieter is highly organised. He always amazes me with his thoroughness and efficiency. Pieter has taught me to be more precise and critical to my work, to question the obvious, and always look for unexpected points of view. He has inexhaustible optimism and enthusiasm. This positive attitude motivates and encourages me to improve.

I am much obliged to Pierre Jansen who has also helped me in my daily work. Pierre has always listened to my problems in patience. He is always willing to discuss and share with me his rich experience. I would like to thank Andre Kokkeler who has reviewed the first draft of the thesis. Andre is a thorough reviewer but with a positive attitude that helped me keeping my motivation. I like to thank Jenna Wells who has also reviewed the thesis and provided me with a lot of helpful suggestions for improvement.

Thanks to our always smiling and cheerful secretaries Marlous Weghorst and Nicole Baveld – always helping and never complaining. No doubt they contribute to the positive working atmosphere.

During these years I have shared the office with a number of different roommates, but the good and friendly atmosphere is always there. Thanks to Gerard Rauwerda, Pascal Wolkotte, Yanqing Guo, Qiwei Zhang for being affable mates and handy source of information. Pascal and I were working on the same subject. Pascal has always been ready to share his view and give me his opinion. Sharing ideas with him helped me to clear my own view.

Thanks also to Bert Molenkamp, Paul Heysters, Omar Mansour, Maarten Wiggers, Marcel van de Burgwal, Philip Hölzenspies and the other group members for the nice working atmosphere and for the chats. It was always nice to chat about the everyday life with Lodewijk Smit and Michel Rosien. It was kind of a pressure release and a way to know the Dutch habits better.

This thesis is based on the ideas developed in the Chameleon project and would not be possible without the contributions of the members of this project. The ideas presented here are directly related to the results obtained by Paul Heysters, Lodewijk Smit, Gerard Rauwerda and Pascal Wolkotte.

Working at University of Twente gave me the opportunity to meet many interesting people all over the world: Ricardo Corin and Laura Brandán Briones, Vasughi Sundramoorthy, Cheun Ngen Chong (aka Jordan), Supriyo Chatterjea, Gabriele Lenzini, Damiano, Kavitha, Ozlem, Anka, Marchin, Tim, Stefan, Ileana, Raluca, Mihai, Roland, Mohammed, Sinan, etc., etc…

Vasughi and Ricardo were the first people I knew in Enschede. They have introduced me to the Dutch culture and gave me some important tips and tricks for survival. I remember my first day in Enschede, Vasughi took advantage on my innocence and prepared a welcome dinner for me – a typical Malaysian meal. During the dinner, proudly looking at me and Ricardo crying over the extremely spicy dish, she started giving us advice how to extinguish the “”fire” in our throats: “It is useless with beer or water, it does not work; better try sweet juice, it is most effective …”. As far as I know there are many people who have the same experience with Vasu’s spicy meal.

Jordan has been my sport mate for a long time and this gave me the opportunity to know him better. To him I owe my introduction to the Chinese culture and kitchen. Jordan is a friend that cares and often has given me his moral support.

For a short time after his arrival, Supriyo and I were living in the same house. Supriyo has shared with me that his impression then about Bulgarians was formed only by watching strugglers and heavy weight lifters on sport TV programs. I was the first real-life Bulgarian he met and more importantly he had to live with and understandably this thought caused him some worries. In my opinion everything went well and I have nice memories from that period. Supriyo was the first Singaporean I met too. He is a very intelligent person addicted to fish.

An important part of my life in Enschede is the Bulgarian community there: Vania, Stanislav, Chris and Aleks, Zlatko Zlatev, Christian Tzolov, Ivan Kurtev, Nikolay Diakov, Julia Bachvarova, Natasha Jovanovich, Georg Koprinkov, Samuil Angelov, Boris Shishkov, Boriana Rukanova, Danail Rosnev, Robina, Snoopy. It is true than not all of them are Bulgarian but all of them helped me feel like at home and manage with the unavoidable homesickness and the consequence of the sunshine shortage. Here I would like to thank to two special friends of mine without whom my life in Enschede would have been completely different – Vania and Stanislav Pokraevi. This family has supported me in the difficult moments of my stay in Enschede. Their inexhaustible positive attitude has assured me that there is nothing to worry about. They kept reminding me about the nice things in life that are worth dreaming of, e.g. boza and milinki. They have inspired and encouraged my cooking attempts and other undertakings in my life. Recently Vania and Stanislav have been introduced to a completely new experience – being parents of the twins Alex and Chris (aka The Boys). As a witness of the progress of the parents and the boys since the very beginning, I can firmly say that Alex and Chris are the most beloved boys I have ever seen. Doubtless, these are boys with a bright future, children that will bring a lot of happiness in their parents’ life.

Finally I want to thank my best friends Kristian, Tania and Kristin, Sasho and Stefka, Strahil, Rostislav and Krasi, Rusi, Nadia, Milena, Koko, etc. for being with me all this time despite the long distance between us. They are among the things that make my country and my city of birth the best place to be, the place I will always want to return to. I also have to thank Skype and Internet in general for make the earth smaller and bringing the people I love close to me.

Contents

Chapter 1

Introduction

1.1. Introduction

In 1965, Intel co-founder Gordon Moore made a prediction, popularly known as "Moore's Law", which states that the transistor density on integrated circuits (IC) doubles about every two years [57]. For four decades silicon technology has been following this law and the number of transistors on a chip has been increasing exponentially. Today, it is commonly believed that from a purely technological perspective there are no obstacles to invalidate Moore's Law in the next decade [93].

The higher integration level achieved following Moore's law allows more and more functionality to be accommodated on a chip. It is now possible to integrate a complete electronic system, including its peripherals and all interfaces, on a single die. Such a system is known as a System-on-Chip (SoC).

Although there are no obstacles for the semiconductor manufacturing technology to continue reducing the IC feature size and increasing the IC integration level, there are several emerging IC design problems that prevent the full utilization of the technology potential. These problems are caused mainly by the smaller feature size and the high integration level. To continue exploiting the technology efficiently they must be overcome. This thesis addresses the main two of these emerging design problems:

- the lower performance of the global on-chip wires, which make the global communications in large SoC designs a performance limiting factor.
- the high design complexity resulting from the higher integration density, which makes SoC design an inefficient and time-consuming task.

We address these problems in the context of a specific class of large SoC architectures – a multiprocessor SoC for streaming Digital Signal Processing (DSP) applications. The solution we propose for such systems is a Network-on-Chip (NoC) communication architecture. A NoC replaces the slow ad-hoc global on-chip wiring with a high performance communication infrastructure which facilitates structured modular system design and thus helps reducing the system design complexity.

1.2. Network-on-Chip concept

A NoC [17, 25, 45, 60] is a lightweight communication network that interconnects the system modules replacing the traditional on-chip bus. An example SoC employing a NoC is shown in Figure 1.1. The chip area is divided into square *tiles*. Each tile contains a system module (e.g., a processor, DSP, peripheral controller, memory subsystem, etc.). Such a system is referred to as a *tiled system* [25]. The NoC is built of *routers*

interconnected by network *channels*. Each tile is connected to a network router through a *standard interface*. Tiles communicate only by sending messages over the network through their interfaces.

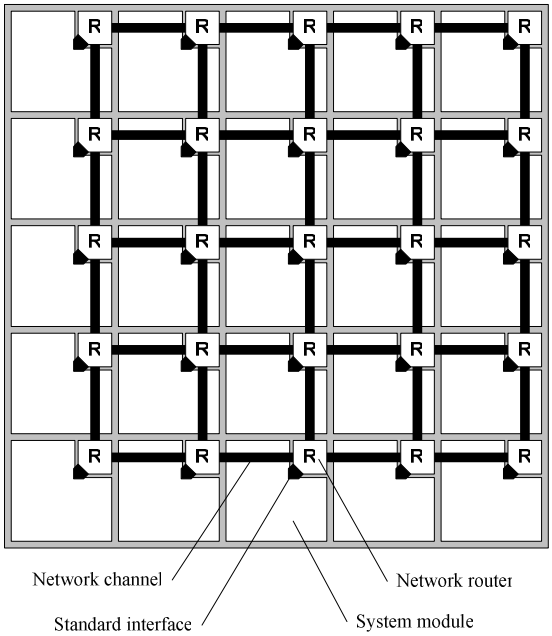


Figure 1.1: An example SoC architecture employing an on-chip network

The NoC serves as a global communication infrastructure. It provides shared global interconnects that can be highly optimised since its development cost can be amortised across many designs. The NoC can provide short and structured global wires with well controlled electrical parameters. This eliminates time consuming design iterations for improving the signalling performance and enables the use of high performance circuits to reduce the communication latency and increase the bandwidth [41, 88]. The network supports parallel communication, so a high aggregate bandwidth can be obtained. Increasing the number of modules in the system also adds routers and channels; hence, the aggregate bandwidth scales with the size of the system. By offering a standard interface, the network facilitates the reusability and interoperability of modules.

1.3. Application domain

The NoC proposed in this thesis is used in the Chameleon project [39]. The Chameleon project aims to design a dynamically reconfigurable multiprocessor SoC for wireless multimedia systems. Potential application areas for such a platform are mobile multimedia devices (e.g., PDAs, mobile phones), mobile medical systems, on-board multimedia systems, smart sensors (e.g., remote surveillance cameras), etc. These systems have to meet challenging requirements such as: high performance, low power consumption, support for Quality-of-Service (QoS) and small size. As part of the system, the NoC must also contribute to these requirements.

Below we summarise the architecture of the SoC defined by the Chameleon project. We focus on the aspects relevant to the NoC design. We start with typical system applications and then present the system architecture. Later we briefly discuss the organization and the operation of the system.

1.3.1. Streaming DSP applications

The majority of applications in our application domain are streaming DSP applications. Examples of such applications are wireless baseband processing applications (e.g., HiperLAN/2, WiMax, DAB, DRM, DVB) and audio/video processing applications (e.g., MPEG codecs). Streaming DSP applications operate on streams of continuously arriving data items which are processed one by one in the order of their arrival and the results are released as an output stream.

Typically, streaming DSP applications are structured as shown in Figure 1.2 [23, 67, 87]. Two parts can be recognized in this structure – a *processing* part and a *control* part. The processing part consists of a number of processing blocks, P_i, arranged in a pipeline. The streamed data items pass through the pipeline and each processing block there applies some transformation on them. Typically, the transformations are mathematical algorithms, such as Fast Fourier Transforms (FFTs) or Discrete Cosine Transforms (DCTs), demanding intensive computation. Therefore, the processing part has high computational demands. Since data items pass through the pipeline periodically, the processing blocks show repetitive timing behaviour. Because many applications process streams in real-time, their processing part requires performance guarantees and the pipeline throughput has to be guaranteed. Hence, the processing part demands Quality-of-Service (QoS) guarantees.

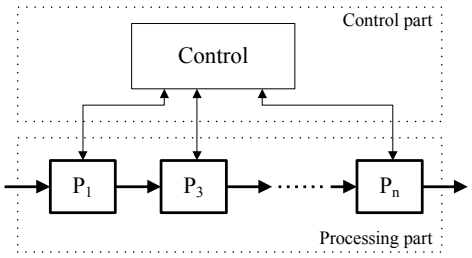


Figure 1.2: Typical structure of a streaming DSP application

The control part of the application implements the control functions associated with adaptation and efficient operation. For example, in a wireless baseband processing application, the control part could monitor the error rate of a communication channel and change the modulation scheme to increase the throughput or to reduce the required computation power. The control part shows more reactive and irregular behaviour and requires little computation. As long as the control part only improves the application efficiency by adding adaptability, its performance is not critical to the real-time operation of the entire application [71]. Hence, the computation and communication in the control part do not require strict performance guarantees.

Streaming DSP applications are computationally intensive, but they have a relatively simple structure. The aim of the Chameleon project is to provide a multiprocessor platform that exploits this simple parallelism naturally present in the pipeline structure of these applications. The potential of the simple pipeline structure to

simplify the design of predictable multiprocessor systems has also been recognized by others [49, 69]. The simple application structure simplifies a number of organisation issues in the Chameleon multiprocessor system. Because parallelism is naturally present in the application structure, partitioning the application into parallel tasks is straightforward. The simple data dependencies between the tasks ease the scheduling of applications on multiple processors. The running applications also generate simple communication patterns, which, as we shall see, help to achieve predictable communication behaviour. In general, the regular parallelism in streaming applications facilitates achieving a predictable and guaranteed operation of multiprocessor systems. Since the majority of applications running on our multiprocessor system are streaming DSP applications, this is the type of parallelism we consider.

Running a streaming DSP application on multiple processors entails mostly streaming communication between the processors. The communication will last for the duration of the application, which for our application domain is estimated from seconds to hours, e.g., watching a film, making a phone call, using a wireless communication channel, etc. Therefore, the traffic in the system consists of semi-static data streams.

The throughput of the data stream between the tasks in the processing part is application dependent. For more demanding applications the throughput is hundreds of Mbit/s. For example, a HiperLAN/2 receiver processes a stream demanding 512 Mbit/s [67]. The size of the data items is also application dependent, for example, it can be a 14-bit audio sample from an analogue-to-digital converter (ADC) or 1024×1024×24-bit video frame. Since often streams are processed in real-time, this traffic requires performance guarantees.

In contrast to the stream processing part, the communication in the application control part consists mainly of short control messages – several bytes of control or state information. To estimate the control traffic, we make the following assumption – each task in the processing part of a HiperLAN/2 receiver generates and receives a 10 Byte control message for every processed data item. This is an overestimation since most tasks do not communicate control messages for every data item. The control traffic generated by the application is then estimated at 10% of the total traffic, while the remaining 90% is streaming traffic. This is a rough estimation that gives the maximal amount of control traffic in the system. The estimation for other baseband processing applications and also for video applications gives similar results.

Thus, we assume the following model for the system traffic generated by streaming DSP applications: 90% of the traffic consists of high throughput semi-static streams that require communication guarantees; 10% of the traffic consists of fine granular control messages that require no strict service guarantees.

1.3.2. Heterogeneous tiled SoC architecture

The SoC proposed in the Chameleon project has a tiled architecture (Figure 1.1). The tiles are heterogeneous reconfigurable processing elements (PE). A tiled architecture has a number of advantages. It can achieve high performance because it supports massive parallelism. It is a future-proof architecture because the tiles do not grow in complexity with technology; instead, the number of tiles on the chip grows. The energy efficiency is improved by switching off tiles that are not used. Defective tiles can be switched off and isolated, which makes the architecture fault-tolerant.

In a heterogeneous system algorithms run on the type of PE which performs the required computation most efficiently. For example, some algorithms run more

efficiently on bit-level reconfigurable PEs (e.g. pseudo random generators), some on word-level reconfigurable PEs (e.g. FIR filter, FFT). Hence, the type of the PEs building the system is chosen according to the needs of the application domain.

Most of the tiles in our system are domain specific PEs, designed to perform fast and efficiently the DSP algorithms in the processing part of streaming applications. Because the DSP algorithms are mostly compute intensive and run periodically, multi-tasking is inappropriate. Hence, the domain specific PEs are single task processors. One or a few tiles in the system are multi-tasking general purpose processors (GPP), to run the control part of the applications and also system control tasks.

Each PE has its own local code and data memory. This reduces the need to access the shared global memories that can easily become a bottleneck in a streaming multiprocessor architecture. Since the communications between the PE and the shared global memory are reduced, the traffic and communication energy are reduced as well.

The number of tiles that will fit on a chip is estimated by comparing the maximal chip size with the tile size. Assuming the maximal chip size for the current and the next generation technologies is $26 \times 22 = 572$ mm^2 [93]. For a tile size estimation we use the area of the Montium tile processor [39, 40] – a domain specific processor for baseband processing applications. The area of the Montium tile processor together with its local memories (data and code) in 0.13µm technology is 2 mm^2. Hence, more than 200 such tiles will fit on a single chip. This number will increase exponentially with the coming generations of semiconductor technologies. Therefore, it is not unrealistic to consider arrays of hundreds of tiles [14].

1.3.3. Dynamic reconfiguration

In a tiled architecture, each tile is reconfigured independently; the tile is the natural unit of partial reconfiguration. Unlike other state of the art systems, in our system the reconfiguration is done at run-time. While some tiles are performing tasks, unused tiles can be configured for new tasks. Therefore the system is dynamically reconfigurable.

Dynamic reconfiguration is essential to support the dynamically changing demands of the application domain: the system operates in a constantly changing environment. The user demands change (e.g., starting/terminating applications), the environmental conditions change (e.g., available networks, wireless channel conditions) and the available power budget also changes (decreasing battery budget or connected to the mains). The set of running applications and tasks in the system adapts dynamically to these changes.

The run-time reconfiguration modifies the system communication demands. For example, a new data stream may be needed or some of the old streams may be redirected or replaced. The NoC must be able to handle such dynamically changing traffic conditions. Run-time changes in part of the traffic must be possible without disturbing the rest of the traffic. The network reconfiguration time must be short enough to enable adequate system reaction time and reconfiguration must be transparent to the user.

1.3.4. Centralised control

Tiles are configured by configuration messages. Generally, configuration messages may come from any tile or from outside the system. However, during normal system operation, configuration messages are generated only by the one tile responsible for

system run-time configuration, control and management. This tile acts as a central authority that manages the other tiles by configuring them. Therefore our system operates with centralised control. Because the central tile performs mainly control oriented tasks, it is appropriate for this tile to be a GPP.

The centralised control has the following advantages for our system. As a consequence of the centralized control, most of the tiles can be simple since they are not required to perform distributed control functions; all control functions are performed by the central tile. The central tile has a global view of the system and can distribute the system resources more efficiently. The central view facilitates also the QoS support and the system performance optimisation.

Drawbacks attributed to centralised control are its poor scalability and unreliability. However, these disadvantages can be avoided to some extent by adding more GPP tiles to the system. As the system size grows, the central GPP tile can delegate some tasks to subordinate GPPs and avoid scalability problems. In case of a malfunctioning central GPP, its functions can be taken over by another GPP with similar capabilities. However, making the centralized control in our system reliable is beyond the scope of this thesis.

The central tile starts and stops applications at run-time. To start an application, the central authority allocates and configures tiles for the application tasks. The procedure of tile allocation is referred to as *application mapping*. For the purpose of mapping, the applications are partitioned into tasks with appropriate granularity to run on tiles; this happens at compile time. At run-time, the mapping algorithm chooses the exact tiles where the task will run. By mapping communicating tasks on neighbouring tiles, the communication distances are reduced (or the communication locality is improved). As a result, the traffic and the communication energy are reduced.

A mapped application task communicates only with some of the other tasks of the same application and eventually with the central authority. Thus, it is not expected that a tile addresses other tiles randomly. Therefore, during its operation a tile needs to know only a small, fixed set of addresses of the other tiles.

The configuration messages also contribute to the system traffic. However, they form a small part of it. The configuration message size depends on the configuration space of the tile being configured. For domain specific tiles this space is usually several KBytes. For example, the total configuration space of the Montium tile is 2.6 KByte [39]. Tile configuration is required when an application is started. For applications such as wireless channels, video/audio players, etc., this may happen every several seconds or several hours. To estimate the amount of configuration traffic in the system, we use the HiperLAN/2 receiver again [67]. Assume that a new receiver is instantiated as an application every second (this is a strong overestimation, since it is not realistic that a new wireless channel is required every second). A HiperLAN/2 receiver is mapped on three Montium tiles [67]. To configure the new tiles, configuration messages of size at most 2.6 KByte are generated every second and these messages generate traffic of 20.8 Kbit/s per tile. Compared to the per-tile throughput of the main data stream which is 512 Mbit/s, the configuration traffic is estimated to be less than 0.005% of the total traffic. Thus, even with a strong overestimation, the fraction of configuration traffic in the system is negligibly small. Whether configuration messages require communication guarantees depends on whether the start-up time or adaptation time of an application is critical.

1.4. Semiconductor technology trends

The design of a large SoC like the one we consider in this thesis faces problems related to the global on-chip wiring. These problems are a result of the reduced dimensions in the new generation of semiconductor technology. The reduced dimensions change the electrical parameters of wires and cause two problems referred to as the *signal integrity problem* and the *clock distribution problem*. Another problem, referred to as the *productivity gap*, relates to the need for a more productive design methodology in order to cope with the increasing design complexity. We discuss each of these problems in the subsequent sections.

1.4.1. Signal integrity problem

The basic components of a digital CMOS IC are gates and wires. The gates do signal switching while the wires transport signals. Every silicon technology generation reduces the dimensions of gates and wires and so changes the physical and thus their the electrical properties. While in the previous technology generations these changes did not lead to serious complications, now they are recognised as a problem that requires urgent attention [42, 55, 76].

As the base fabrication technology shrinks to smaller dimensions, the gates become smaller and the wires become thinner and, as a result, the signal delay of gates and wires changes. Under scaling, the delay through a fixed-length wire (which is inversely proportional to the signal propagation velocity) increases, while the gate delay decreases. Thus, an increasing disparity between wire and gate delay is observed, assuming constant wire length.

Typically, IC designs consist of a number of modules. As designs scale to the newer technologies, modules get smaller, the wires in the modules get shorter and the relative change in the delay of wires to the delay of gates in a module is modest. However, a chip can accommodate more and more modules, which are also interconnected by wires. These wires communicate signals across the entire chip and in contrast with the local module wires their length does not scale with technology. They stay long and their delay scales upwards relative to the gate delay. Thus, we must distinguish two types of wires, the signal delay of which is influenced differently by the scaling. We refer to these two types as *local* and *global* wires.

If no special measures are taken, it might be expected that future ICs will consist of fast high-performance modules, interconnected by slow global wires. Thus the global wires will become a system bottleneck and will degrade the overall IC performance. Researchers agree that a solution to the problem can be provided by a new chip design methodology [42]. In the current methodology the chip wiring is automatically generated by the design tools and the designers cannot control the wires in the early design stages. The automatically generated wires are not structured and their electrical parameters, such as parasitic capacitance and crosstalk to adjacent wires, are difficult to predict early in the design process. This does not allow for optimisation of the global wires in early design stages and leads to many time consuming iterations in the late design stages.

Instead of assuming the wiring as something hidden and automatically generated by tools, researchers agree that *explicit structures that handle the inter-module communications* must be included in the system architecture. Such an approach will make global wires structured and controllable; it will make the global communication

latencies explicit and predictable in an earlier design stage and will allow particular measures to be taken for their improvement.

1.4.2. Clock distribution problem

The changes in the electrical properties of wires also affect the on-chip global clock distribution. It is getting more and more expensive in terms of energy to distribute a precise clock signal to all modules on the chip. For example, in complex high performance chips, clock distribution may cost near 50% of the total energy consumption [79]. Hence, chip-wide synchronous operation is becoming expensive. The envisioned solution is the *Globally Asynchronous Locally Synchronous* (GALS) systems design framework [19, 58]. A GALS system is a system consisting of many synchronous modules, which, however, operate at their own local clock frequencies. No global clock distribution is required and the system should be considered globally asynchronous. The synchronous modules are often referred to as *clock islands*.

Compared to a fully synchronous design, GALS can reduce the clock distribution power by 70% [38]. Another advantage of GALS is that the system modules are still synchronous and can be designed using standard tools and methodology. To complete the framework, asynchronous communication techniques for transporting data between the islands are required.

1.4.3. Productivity gap

As the integration level increases, the chip complexity grows. However, the chip complexity growth rate is about two times higher than the design productivity growth rate [46]. This means that the system design time will increase exponentially if the current design methodology and tools are not replaced by more productive ones. A complex design now can easily include 20-million logic gates. If such a design is started from scratch, it could easily take 200 engineers three to five years to architect, design, verify and build [82]. At the same time a common wisdom is that the product design cycle needs to be approximately one year to be competitive in the market.

The disparity between the complexity and the productivity growth rates is usually referred to as *productivity gap*. To narrow this gap, more productive methodologies are needed. We can already observe that driven by this need the design methodology changes. Instead of designing systems from scratch, currently more and more systems are built from existing modules (re-use), e.g. CPUs from ARM and MIPS, common I/O blocks such as Ethernet MAC, USB, PCI, etc. It is expected that future systems will consist mainly of pre-designed standard IP (Intellectual Property) modules, adding only a few proprietary modules. Hence, future system design will consist mainly of *integration* of pre-existing IP modules. These systems will need an integration technology that facilitates modularity and IP interoperability. Adding, removing or changing an IP module should be possible without major disturbances of the rest of the design. This can be achieved by using standard global on-chip communication architecture offering a standard interface to the IPs. By reusing the communication architecture over many designs, design time and cost are saved. Since the communication architecture is optimised and with a fixed layout, time consuming iterations for optimising the global communications are avoided when a new SoC is designed or when an old is modified by replacing or adding modules.

1.4.4. Directions

For continued benefit of the advances in silicon technology, all three design problems addressed above must be solved. We believe that the three problems can be solved in principle by an explicitly defined global communication infrastructure that structures and shortens the global wires and facilitates modular design.

Current complex on-chip systems are also modular, but most often the modules are interconnected by an on-chip bus. The bus is a communication solution inherited from the design of large board- or rack-systems in the 1990's. It has been adapted to the SoC specifics and currently several widely adopted on-chip bus specifications are available [89-91, 95].

While the bus facilitates modularity by defining a standard interface, it has major disadvantages. Firstly, a bus does not structure the global wires and does not keep them short. Bus wires may span the entire chip area and to meet constraints like area and speed the bus layout has to be customised [78]. Long wires also make buses inefficient from an energy point of view [9]. Secondly, a bus offers poor scalability. Increasing the number of modules on-chip only increases the communication demands, but the bus bandwidth stays the same. Therefore, as the systems grow in size with the technology, the bus will become a system bottleneck because of its limited bandwidth.

The current solution for the bus performance and scalability problems is bus partitioning. A bus is partitioned into several busses (most often two), connected trough bridges. A hierarchy may be introduced between the busses, e.g. a high-performance system bus and a low-performance peripheral bus. While the partitioned bus solution is satisfactory for the current system sizes (up to tenths of IPs) it does not help for structuring the chip layout.

Although the bus is the common communication solution in the current SoC, its future application is questionable because a bus is unable to solve the design problems foreseen for the future semiconductor technology. Therefore a communication paradigm shift is required. A new SoC communication solution that addresses the design problems is needed.

We believe that an appropriate solution can be found in the communication concept used in the 90's for the interconnection of processor arrays in multi-computers. In multi-computers many processors are interconnected by a communication network. It is proposed to use such a network to interconnect the modules in a SoC [25]. This concept has become popular as a Network-on-Chip (NoC). The cost of applying an interconnection network on-chip is the area overhead due to new system components (routers) needed to support the network. The system organisation must also take into account the network and may incur additional network exploitation costs in terms of configuration and time overhead.

1.5. Objectives

The arguments presented suggest that NoC structures have the potential to solve the key design problems in the future semiconductor technologies and motivate us to use a NoC as an on-chip communication infrastructure in our SoC design. However, there are still many open questions which have to be answered in order to show that the NoC concept is feasible. For example, to design a NoC, one has to decide which particular network techniques to employ. It is not known what performance can be achieved by a NoC and whether it can satisfy the system demands. It is not known whether the costs

of employing the NoC are acceptable. It is still not clear whether the network can support the overall system operation requirements.

These are the questions we address in this thesis. Our objective is to define an NoC architecture, evaluate its performance and estimate its implementation cost. Therefore we define the following research questions:

1. *What network techniques are appropriate to minimize the network overhead while maintaining satisfactory performance?*

This question addresses the design choices that have to be made. The design objective is to achieve a network performance that satisfies the system demands. We consider the typical network performance metrics throughput and latency, but also the services the NoC can provide e.g., guaranteed services or best effort services. The design constraint is given by the maximum acceptable network overhead. We consider two types of overhead – *implementation* and *exploitation* overhead. The implementation overhead comprises all the costs due to the physical implementation of the network; these are the area cost and the static energy cost. The exploitation overhead comprises all the costs for network support and exploitation, e.g. network configuration costs, costs for sending data over the network (dynamic energy cost and data overhead), etc.

The design choices are made on the basis of comparison between the relative cost and performance of the considered techniques and not the actual implementation performance and cost. For example, latencies are compared in terms of clock cycles but it is not taken into account what the maximal achievable operating frequencies are. Therefore, to evaluate the network we have to find its actual cost and performance and that leads to our second question:

2. *What is the overhead and the performance of a NoC architecture?*

This question addresses the evaluation of the implemented network and its answer requires estimation of all the costs the NoC employment entails. To estimate the exploitation overhead it is necessary to know exactly how the network is used by the system. Hence our third question is:

3. *What is the optimal use of the NoC?*

This question addresses the overall system operation and all the interactions between the NoC and the system. It is a difficult question to be answered in detail since it involves many aspects of the system operation which are beyond the scope of the thesis. However, we propose a system organization scenario that supports the NoC communication concept as much as possible.

In general, this thesis presents a proof of concept of the NoC idea. We define and evaluate an instance of a NoC architecture for our tiled Chameleon SoC. The NoC is not a general purpose one but aimed to support the streaming DSP applications in our system.

1.6. Contributions

The scope of our work is given by the Chameleon system [39, 71]. In this scope we provide a feasible NoC solution. The specific contributions of this thesis are:

1. We propose a NoC architecture for a run-time reconfigurable multiprocessor SoC that supports streaming DSP applications. To the best of our knowledge, this is the first NoC targeted at a run-time reconfigurable SoC.

2. We propose an architecture of a virtual channel router, which in contrast to conventional architectures is able to provide predictable communication services and has a lower implementation area cost than conventional architectures.

3. The predictable performance of our network simplifies the mapping of streaming DSP applications to our multiprocessor system. System reconfiguration can be done in linear time avoiding the NP-complete solutions common for statically scheduled real-time systems. Thanks to this linearity, system reconfiguration can be done at run-time.

1.7. Structure of the thesis and related publications

The rest of the thesis is organized as follows. In Chapter 2, we review communication network techniques and select those suitable for our NoC implementation. We choose to use a virtual channel network. In this chapter we also discuss some recent work on NoC design.

In Chapter 3, we discuss the possible architectural solutions for a virtual channel router. We study the influence of the architecture on the router performance and identify those architectures that can provide predictable communication services. Finally, we propose an approach for providing guaranteed communication services at network level. Major parts of this chapter have been presented at the IEEE International Symposium on VLSI 2006 [6].

Implementation details about the selected router architecture are presented in Chapter 4, in which we propose an implementation that simplifies the design and reduces the overall router area. Implementation results are also presented there. Major parts of this chapter have been presented at the IEEE International System-on-Chip Conference 2004 [5] and at the EUROMICRO Symposium on Digital System Design 2004 [1]

In Chapter 5, the proposed approach for providing guaranteed services is evaluated with a model of the expected traffic in the system. Also the overhead for applying the approach is estimated. Major parts of this chapter have been presented at the International Workshop on Applied and Reconfigurable Computing 2006 [7].

Chapter 6 shows how performance guarantees are given to streaming DSP applications. Major parts of this chapter have been presented at the EUROMICRO conference on Digital System Design 2005 [4] and at the Communicating Process Architectures Conference 2005 [8].

Chapter 7 gives conclusions and recommendations.

Chapter 2

Background and related work*

Multiprocessor networks have been studied for more than two decades and a solid foundation of design techniques is available. In this chapter we review the main techniques for interconnection networks. We also present some recent network-on-chip solutions and discuss the techniques they employ.

2.1. Introduction

The objective of this thesis is to define a Network-on-Chip (NoC) architecture for the tiled multiprocessor System-on-Chip (SoC), described in Chapter 1. The task of the NoC is to interconnect a set of processing tiles, allowing them to exchange data and to operate as an integral system. Building networks of processors is not a new research topic. Such networks have been investigated for more than two decades in the domain of parallel computing and are known as Multiprocessor networks (MP networks). As a result, a solid foundation of design techniques for MP networks is available in the literature. Since by nature on-chip networks are MP networks, MP network techniques can be adopted for building NoC architectures. However, on-chip networks have their specifics, which are result of the different realization technology and the different requirements of the multiprocessor system and the applications running on the system. These specifics must be taken into account when adopting MP network techniques.

This chapter consists of three parts. In the first part we discuss the NoC specifics in comparison with MP networks. In the second part we give an overview of the general interconnection network techniques, focusing on the techniques used for building MP networks. We discuss the techniques in the context of our tiled multiprocessor system. Our objective is to identify those of them which are most suitable for use, directly or after modification, in our NoC and also to identify potential gaps which require development of new specific techniques. We also introduce terminology and notation used in the other chapters. We adopt the terminology and notation used by Dally and Towles [27].

The third part of this chapter is devoted to existing NoC solutions. We discuss only the most mature solutions and techniques that are relevant to this thesis. The solutions are compared mostly qualitatively; the quantitative comparisons are restricted to chip area and clock frequency (when this information is available). We restrict our quantitative comparisons, because often NoC solutions are targeted at different systems, address different traffic types and pursue different goals, or detailed information about

* Major parts of this chapter have been presented at the PROGRESS Embedded Systems Symposium [2].

the system is not available. Therefore, detailed quantitative comparison cannot be made fairly.

2.2. NoC characteristics

To establish criteria on which we can assess the available MP network techniques, we first define the specifics of the NoC. The first specific criterion that distinguishes a NoC from MP networks is the implementation technology. While MP networks are used for inter-chip or inter-board communications, the NoC is entirely built on the chip. Inter-chip communication requires signals to go off chip on pins. Since the number of pins available on the chip is limited to less than 1000, the number of inter-chip signals to be used for communication is also limited. In contrast, on-chip networks do not have this limitation because they are built entirely on the chip and use only on-chip wiring resources. The number of the on-chip wires available for communication signals can go far beyond the pin limitation of the inter-chip networks. For example, in 130 nm technology [92], launched in 2002, the minimum global wiring pitch is 565 nm, so there can be up to 1770 wires crossing an edge of length 1 mm on each metal layer. In 70 nm technology [93], projected in 2006, the minimum global wiring pitch is 250 nm, hence there can be up to 4000 wires crossing an edge of length 1 mm on each metal layer. Hence, on-chip networks have extensive wiring resources at their disposal compared to the traditional MP networks.

The main limitation to which the on-chip network has to conform is the chip area. While in the MP networks each router (the building block of a network) is placed on a separate chip [26, 74] and utilizes the entire chip area, all routers of an on-chip network are placed on a single chip. The NoC is just part of the implemented SoC and utilizes only part of the available chip area. The area utilized by the NoC routers should be reasonably small compared to the area used by the computational resources. The computational resources in our SoC are the processing tiles. Each tile is accompanied by a network router. As an estimate, the area of the processing tile proposed by Heysters [39] is 2 mm^2 in 130 nm technology. If for a maximal acceptable size for a router we assume 1/10 of the tile area, then the router area should be less than 0.2 mm^2.

Another NoC specific is the requirement for a simple and regular layout. The wires used for network signalling form a large part of the global on-chip wiring. To cope with the signal integrity problem, described in the introduction chapter, the global wires must be short and structured. By employing a network topology that results in a simple and regular layout, a NoC has the potential to provide wiring with well controlled parameters, predictable at an early design stage and easy to optimize. Thus, the regular layout helps in coping with the signal integrity problem.

Since the integration level provided by new semiconductor technologies increases exponentially following Moore's law, more and more tiles will fit on a single chip. Thus, the network size is also expected to grow. To provide for an easy transition between technology generations, the network must be scalable, such that it can be extended with a minimal cost and redesign efforts.

At a functional level the major difference between MP networks and the NoC is the demand for quality-of-service (QoS). In traditional multiprocessor systems, like supercomputers, the focus has been mostly on high performance, while QoS has not been an active research topic. For that reason there is a lack of MP network techniques for providing QoS. However, recently multiprocessor systems have appeared in

consumer products, such as mobile phones, TV/video sets, etc. [94]. Many program applications in these devices require QoS, and that raises the demand for QoS support.

In summary, the characteristics that distinguish the on-chip networks from the traditional MP networks are:

- large amount of available wiring resources
- area limitation for the router size
- need for regularity of the network layout
- need for scalability
- demand for QoS

2.3. Interconnection networks

In this section we give an overview of general techniques for the design of interconnection networks where we focus mainly on MP network techniques. The techniques are discussed within the perspective of the NoC context, in order to assess how appropriate they are for a NoC implementation.

According to the definition given Dally and Towles [27], an interconnection network is a programmable system that transports data between terminals. Here *terminal* refers to a general source/sink of data that requires communication services. Such a system is shown in Figure 2.1. The figure shows six terminals, T1 through T6, connected to the network. When a terminal wishes to communicate data to another terminal, it sends a *message* containing the data over the network. The network delivers the message to the destination terminal. The network is programmable in the sense that it can make different connections at different points in time. The network in the figure may deliver a message from T3 to T5 in one cycle and use the same resources to deliver a message from T3 to T1 in the next cycle. The network is a system because it is composed of many components: buffers, channels, switches and control that work together to deliver data.

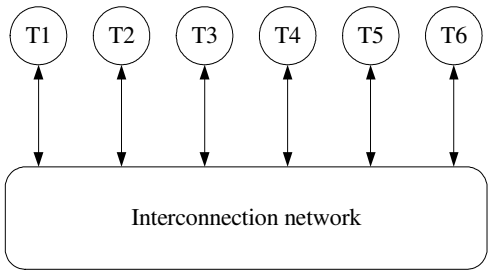


Figure 2.1: Functional view of an interconnection network

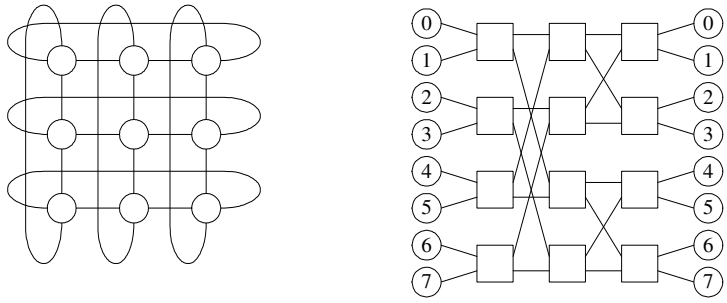
Networks meeting this broad definition may occur on many scales. However, here we restrict our attention only to small scale networks and MP networks, relevant to our SoC architecture. These networks have tens to hundreds of terminals positioned close to each other (on a board or on a chip). The terminals are processors, memories or other system modules.

A network is built out of *switching elements* interconnected by *physical channels*, also called *links*. A switching element has a number of input and output ports. Its main function is to forward data by establishing non-conflicting connections between input and output ports. Depending on the type of network the switching elements are referred to either as *switches* or *routers*.

The physical channels are sets of wires interconnecting the ports of neighbouring routers and transporting signals between them. The physical channels form the medium that transports information in the network. The switching elements allow physical channels to be time-shared between data from different source and destination pairs. In some networks sharing may cause data blocking. To prevent loss of blocked data, the switching elements may provide storage space for temporal data buffering. The buffers may also be shared, since at different times they may store data from different sources. Besides the physical channels, buffers are the other important network resource.

2.3.1. Direct and indirect networks

A network where every switching element is directly connected to a terminal is called a *direct network*. An example of a direct network is given in Figure 2.2.a. The circles there represent pairs of terminals and switching elements, often called *nodes*. In contrast, a network where not every switching element is connected to a terminal is called an *indirect network*. An example of an indirect network is given in Figure 2.2.b. The circles represent terminal nodes and the squares represent switching elements. In indirect networks there is a natural separation between the terminals and the switching elements, while in direct networks the separation is a matter of preference.



a) A 3-ary 2-cube b) A 2-ary 3-fly

Figure 2.2: An 8-node mesh network and an 8-node butterfly network as example of direct and indirect networks

In the topology of the tiled multiprocessor SoC architecture considered in this thesis (see Chapter 1), the tiles that construct the system are arranged into a two-dimensional array on the plane of the chip. Each tile has to be connected to the on-chip network and will play the role of a terminal. Furthermore, the network should provide simple and regular global on-chip wiring. The simplest and most natural way to satisfy these requirements is to add a switching element to each tile and to interconnect the neighbouring switching elements in a grid. This will result in a direct network topology. Therefore, we focus only on direct network topologies.

2.3.2. Performance of interconnection networks

The basic metrics of the network performance are *throughput* and *latency*. Throughput is the rate at which data is delivered by the network, in [bit/s]. Throughput, also referred to as *accepted traffic*, should be clearly distinguished from the *offered traffic*. The network cannot always accept all the traffic generated by the data sources.

The *ideal throughput*, θ_{ideal}, is the theoretical bound on the network throughput assuming that the traffic is perfectly balanced over the physical channels. However, this bound is rarely, if ever achieved because the network techniques used in practice cannot provide full network utilization, e.g. the routing cannot perfectly balance the traffic over the channels, the flow control results in idle channels because of resource dependencies, etc. Hence, the network saturates at throughput θ_s, $\theta_s < \theta_{ideal}$, referred to as *saturation throughput*.

Latency is the time required for a data item to traverse the network, from the time the first bit of data arrives at the input port of the network to the time the last bit is received at the output port of the network. Often latency is estimated under a zero-load assumption; that is, data never contends for network resources. Thus *zero-load latency*, T_0, gives a lower bound on the data latency in the network. Figure 2.3 shows an example graph depicting a typical dependency between the traffic load offered to the network and the data latency.

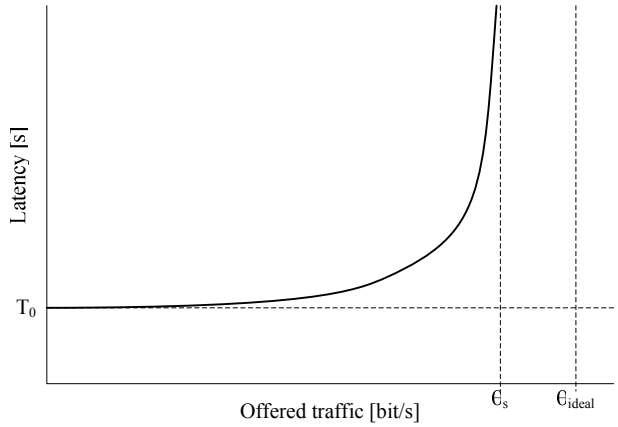


Figure 2.3: Typical dependency between offered traffic and data latency in a network

2.3.3. Network topologies

The physical structure of a network can be represented as a graph, called a *network graph*. The vertices in the network graph represent switching elements and the edges represent physical channels. The arrangement of the switching elements and channels is represented by the topology of the network graph, called the *network topology*.

Definitions

Since a network topology is represented through graphs, graph terminology is adopted when reasoning about networks. An interconnection network is formally defined as a directed graph $I=(N,C)$, where N and C are the set of nodes and the set of channels in the graph. The *degree* of a network node is the number of channels connected to the node. When all the nodes in the network have the same degree, the network is called *degree regular*.

A *path* in the network is a sequence of nodes and channels. More formally, a path is a sequence $<n_0, n_1, \ldots n_l>$ of nodes, such that $n_i \in N$ for $i \in [0..l]$, and edges $(n_i, n_{i+1}) \in C$ for $i \in [0..l-1]$. Sometimes it is more convenient to express the path in terms of channels.

The path is then a sequence $<c_0, c_1, \ldots c_{l-1}>$ of channels, such that $c_i \in C$ for $i \in [0..l-1]$ and $destination(c_{i-1})=source(c_i)$ for $i \in [1..l-1]$. The functions $source(c_i)$ and $destination(c_i)$ give the source node and the destination node of the channel c_i.

The *length* of a path equals the number of channels (n_i, n_{i+1}) traversed by the path. The number of traversed channels is also referred to as the *hop count*; a hop is the unit in which the network distances are usually given. Paths are also referred to as *routes*. A path between two nodes s and d is a network path $<n_0, n_1, \ldots n_l>$ such that $n_0=s$ and $n_l=d$. The *distance* between two nodes s and d is the length of the shortest path between s and d. The maximal distance D over all pairs of nodes in the network is called the *diameter* of the network – a characteristic often used for assessment and comparison of network topologies.

A *cut* of a network, $C(N_1, N_2)$, is a set of channels that partitions the set of all nodes N into two disjoint sets, N_1 and N_2. Each element of $C(N_1, N_2)$ is a channel with a source in N_1 and destination in N_2 or vice versa. The number of the channels in the cut is $|C(N_1, N_2)|$ and the total bandwidth of the cut is:

$$B(N_1, N_2) = \sum_{c \in C(N_1, N_2)} b_c , \tag{2.1}$$

where b_c is the bandwidth of channel c.

A *bisection* of a network is a cut that partitions the network nearly in half, such that $|N_2| \leq |N_1| \leq |N_2|+1$. The *channel bisection*, B_C, of a network is the minimum channel count over all bisections of the network:

$$B_C = \min_{bisections} \left| C(N_1, N_2) \right| . \tag{2.2}$$

The *bisection bandwidth*, B_B, of a network is the minimum bandwidth over all bisections of the network:

$$B_B = \min_{bisections} B(N_1, N_2) . \tag{2.3}$$

For networks with uniform channel bandwidth $b_c=b$ for every $c \in C$, the bisection bandwidth is $B_B=bB_C$. For simplicity, in the following sections we refer to the channel bisection only as bisection unless explicitly stated otherwise.

A topology characteristic related to the bisection is the network *connectivity*. A network is called k-connected when between any pair of nodes there exist at least k paths that do not share other nodes than the source and the destination (internally vertex disjoint paths). The maximal k for the network is called the connectivity of the network. Since the connectivity corresponds to the path diversity between the nodes, it is used as a measure of the fault-tolerance of the network. On the other hand, it can also be used as a network performance measure related to the bisection.

Requirements

The selection of the network topology will be driven by the following criteria:

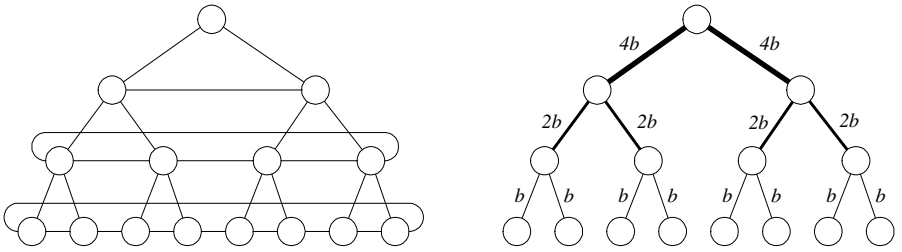
- Small and fixed degree. The degree of a node determines the number of ports of the corresponding switching element. Hence, node degree influences the switching element complexity and area cost. A small degree reduces the cost. Degree regularity allows for uniform design of the switching elements.

- Simple and uniform layout. Whatever the topology, the network is embedded in the plane of the chip. The layout of the embedded network defines the global on-chip wiring, which has to be as simple and uniform as possible in order to lessen the signal integrity problem.
- Small diameter. Networks with a smaller diameter have shorter distances and therefore lower communication latency.
- Simple routing. The network topology influences the algorithm used for searching routes in the network. Regularity and simplicity of the network topology can simplify the routing algorithm
- Scalability. Following Moore's law, the number of system components on-chip is expected to increase exponentially with the technology generations in the next 10 years [93]. Thus, the number of nodes in the on-chip network will increase. To provide easy transition between technology generations, the topology should support extension (an increase in the number of nodes) at minimal cost and redesign efforts.
- Fault tolerance. Defects in the manufacturing process may cause malfunctioning or failure of network components (channels or routers) in newly produced chips. To reduce the impact of such defects on the production yield, it is desirable that in the presence of a reasonably small number of faulty components, the network is still operable. During its operation, the network should be able to avoid the faulty components by using alternative paths. The presence of alternative paths is indicated by the connectivity of the network; hence, topologies with higher connectivity are preferable.

Having motivated our requirements for a topology, we continue by giving an overview of direct network topologies most commonly used in MP networks.

Tree topologies

As the name suggests, the tree topology interconnects the network nodes in the form of a tree. Each node (except the root) has one ancestor and k descendants. A tree in which every node, except the leaves, has exactly k descendants is a k-ary tree. Tree networks have a small diameter, $O(\log_k N)$, N being the number of nodes in the network, but their bisection and connectivity are small, only 1; there is only one path available between any pair of nodes. Therefore, the network is not fault-tolerant. To improve the fault tolerance, *ringed trees* have been proposed [29]. As shown in Figure 2.4.a, a ringed tree connects the nodes of each stage of the tree in a ring.



a) A Ringed tree b) A Fat tree

Figure 2.4: Tree topologies

The small bisection of the trees also causes performance problems. The nearer to the root a channel is, the more paths use it. Therefore, the channels nearer to the root have a higher load and become a bottleneck. To overcome this problem, *fat-trees* have been proposed [51]. In fat-trees, channels that are nearer to the root have higher bandwidth. For example, in the fat-tree shown in Figure 2.4.b, the channels connecting the leaves of the tree have bandwidth *b*, the channels connecting the next stage have bandwidth 2*b*, and so on. While fat-trees overcome the tree performance problem, they introduce irregularity in the physical design of the switching elements and do not improve the fault-tolerance.

Star topologies

A star graph [11] has *n*! nodes labelled by permutations of *n* different symbols. Each node in the graph is connected to *n*-1 other nodes, with labels that are obtained from the current node label by interchanging the first symbol and one of the other symbols. An example of a four-star graph (*n*=4) is given in Figure 2.5. The nodes are labelled by permutations of the symbols {1, 2, 3, 4}. Labels are shown only for one node and its three neighbours. The border edges labelled by the same letter are connected, but for clarity of representation the connecting edges are not shown.

The degree of an *n*-star graph is *n*-1 and the diameter is $\lfloor 3(n-1)/2 \rfloor$. Although, the degree and the diameter of star graph are lower than the degree and the diameter of mesh and torus network (discussed below) of similar sizes, the routing is more complicated and the node degree depends on the network size.

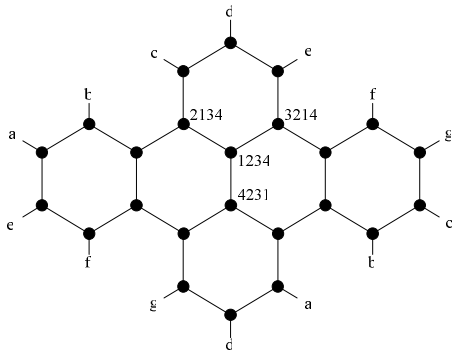


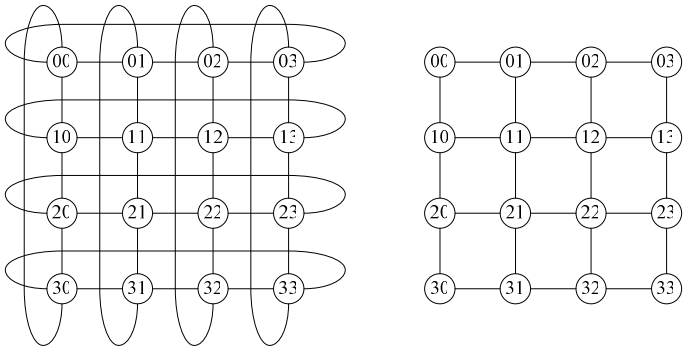
Figure 2.5: An example of star graph

Torus and mesh topologies

Among the direct network topologies, torus and mesh networks are the most popular, well studied and often used for practical implementations. Torus networks are characterised by their radix *k* and the number of dimensions *n*. An *n*-dimensional radix-*k* torus, also referred to as a *k*-ary *n*-cube, arranges $N=k^n$ nodes in a *n*-dimensional cube with *k* nodes in each dimension. Each node is assigned an *n*-digit radix-*k* address $\{a_{n-1}, \ldots a_0\}$. The *i*th digit in the address, a_i, represents the node position in the *i*th dimension. A node is connected by a pair of channels (one in each direction) to all nodes with

addresses that differ by ±1(mod k) in exactly one address digit. This requires 2 channels in each dimension per node or $2n$ channels. Tori are degree regular, their diameter is $n\lfloor k/2 \rfloor$ hops and the bisection is $4N/k=4k^{n-1}$. An example of a 4-ary 2-cube is given in Figure 2.6.a. The long channels connecting the nodes on the opposite edges of the node array are usually referred to as wraparound channels.

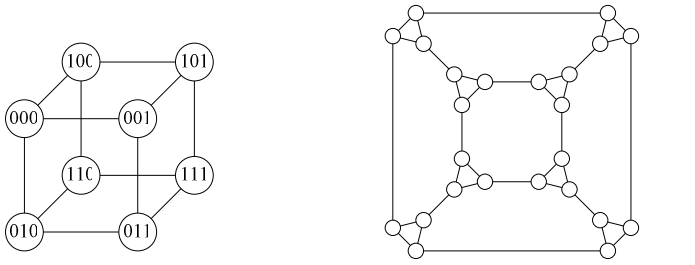
A mesh network is a torus network with wraparound channels removed. Each node in a mesh connects to all nodes that differ by ±1 in exactly one address digit. Figure 2.6.b gives an example of a 4-ary 2-mesh. The mesh network has the same node degree, but half the bisection channels of a torus with the same radix and dimension. The bisection is $2N/k=2k^{n-1}$, and the diameter is $n(k-1)$ hops. Removing the wraparound channels destroys the symmetry of the torus. This can cause load imbalance, as the demand for the central channels can be significantly higher than for the edge channels.



a) A 4-ary 2-cube b) A 4-ary 2-mesh

Figure 2.6: Torus and mesh networks

A k-ary 1-cube is simply a ring of k nodes. The 2-ary n-cube networks form a subclass of tori called hypercubes or binary cubes. An example of a 3-dimensional hypercube is given in Figure 2.7.a. In a hypercube network every node is connected to n other nodes. A hypercube keeps the hop count small with an increasing number of nodes. However, increasing the number of nodes increases the number of dimensions n and the node degree. For MP networks the higher degree may cause a packaging problem. Figure 2.7.b presents a hypercube modification called cube-connected cycles [65] for which the node degree remains 3, independently of the number of dimensions. It is derived by replacing the n-degree nodes in the hypercube by a ring of n nodes.

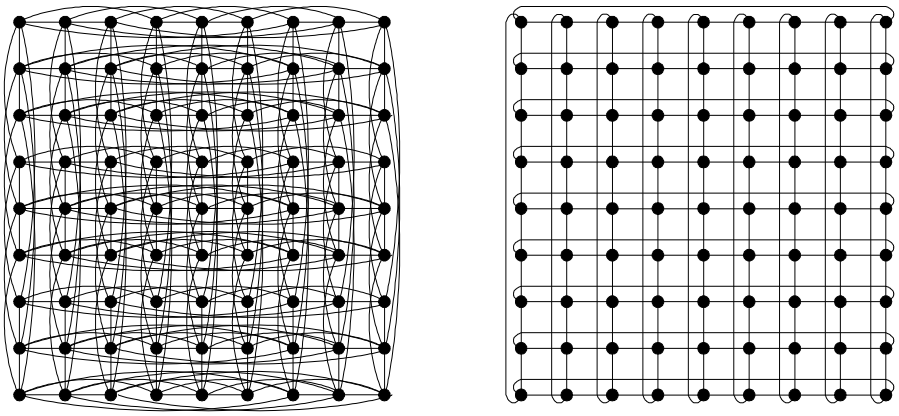


a) A binary 3-cube b) 3rd order cube-connected cycles (n=3)

Figure 2.7: Binary cube and cube-connected cycles

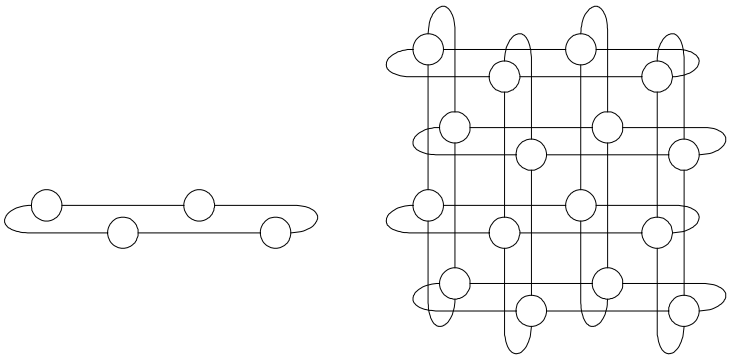
Torus and mesh networks of low dimension are attractive for several reasons. The regular topological arrangement is well matched with the two-dimensional arrangement of the tiles on the chip. At low dimensions tori have uniformly short wires allowing high speed operation without repeaters (the wraparound channels are an exception which can be avoided by folding, discussed later). Logically minimal paths in tori are almost always physically minimal as well. The wiring complexity and performance of torus networks is studied by Dally [22]. The results show that low-dimensional networks are advantageous compared to high-dimensional networks. In particular, with up to 1024 nodes a 2-dimensional topology provides lower latency and higher throughput than networks of higher dimensions with the same bisection. Networks of many dimensions require more and longer wires than low-dimensional networks.

Besides performance, wire regularity is another major concern when choosing a network topology. The network topology embedded in the plane of the chip determines the structure of the global on-chip wiring. To solve the signal integrity problem, discussed in the introduction chapter, the global on-chip wires must be as uniform and simply structured as possible. Figure 2.8 compares the wiring of two cubes containing the same number of nodes, but of different dimensions. Figure 2.8.a shows a 3-ary 4-cube embedded in a plane. Although the required wiring is regular, it is complicated, wires of different length are required and it would be practically difficult to keep it well structured and optimised. On the other hand, the two dimensional cube shown in Figure 2.8.b has a simple wiring structure which is intuitive and easy to handle. Similar observations can be made for meshes of low and high dimension. Thus, the performance and wiring issues suggest that low-dimensional networks are advantageous for on-chip implementation.



a) A 3-ary 4-cube embedded in the plane b) A 9-ary 2-cube

Figure 2.8: Wiring of cubes of different dimensions



a) A folded ring b) A folded torus

Figure 2.9: Folding cubes in order to avoid wraparound channels

In two-dimensional cubes the wraparound channels may cause practical problems, because they are long, require repeaters and are slower than the other channels. The wraparound channels can be avoided by *folding* the network as shown in Figure 2.9.b [22]. The folding keeps the graph intact but reshuffles its nodes in the plane such that the wraparound channels are avoided at the expense of doubling the length of the other channels. The idea of folding is illustrated in Figure 2.9.a by folding a ring.

2.3.4. Flow control

The terminals (the processing tiles in our system) exchange data in the form of *messages*. The size of the messages is entirely determined by the applications and the storage space available in the terminals. However, the unit of information that networks work with is the *packet*. Packets encapsulate the transported data adding to it some control information that is used by the network. The packet length may be fixed or variable, which is determined by the network buffers capacity and the employed flow control mechanism. When there is a limitation on the packet length, it may be necessary to split the messages when injected in the network and later reassemble them on the receiving side.

When transmitted over the network, packets are divided into smaller fixed size data units called flow control digits, or *flits*. A flit is the smallest unit of information recognized by the flow control. Finally, the unit of information that can be transferred across a physical channel in a single cycle is called physical digit, or *phit*.

Flow control determines how network resources, such as channel bandwidth and buffer capacity are allocated to packets traversing the network. A good flow control method allocates these resources in an efficient manner so the network achieves a high fraction of its ideal throughput and delivers packets with low latency. The flow control can be classified as a *buffered* or *bufferless* flow control. Bufferless flow control is the simplest form of flow control that uses no data buffering and simply acts to allocate a channel bandwidth to competing packets. Buffered flow control is a more complicated form of flow control that relies on buffering space in the routers, but it also more efficient in distributing the network resources between packets. We describe each form of flow control in the subsequent sections.

Bufferless flow control

This type of flow control does not use buffer space for storing packets in the routers. Therefore, bufferless flow control cannot hold the packets at a place, but has to ensure they advance every cycle. When a packet cannot advance because the next channel is occupied, the packet is either dropped or misrouted. Thus, we distinguish two types of bufferless flow control: *dropping flow control* and *misrouting flow control*.

Dropping flow control drops from the network the packets that cannot advance because of blocking. The dropped packets have to be retransmitted, which requires that: i) a copy of the packet is stored at the sending terminal; ii) the flow control provides a mechanism for notifying the sending terminal about the packet dropping. Two methods are used for notifying the senders of the dropped packets. The first one relies on explicit negative acknowledgement (*explicit NACK*), while the second simply uses a *timeout*.

A time-space diagram of dropping flow control with explicit NACK is given in Figure 2.10. As before, the figure shows a 5-flit packet being sent along a 4-hop route. The packet consists of a header (H), body (B) and a tail (T). The vertical axis shows the forward (F) and the return (R) directions of the four channels (0-3) traversed by the route. In the example, the first transmission of the packet is unable to allocate channel 3 and the packet is dropped. A NACK signal (N) is sent back to the sender to initiate a retransmission. The NACK signal follows backwards the path reserved for the dropped packet. The retransmission succeeds and the receiver sends an acknowledgement (A) to the sender to notify that the packet has been received. With explicit NACK, flow control channels are allocated to a packet by the packet header flit (H) and are released by the ACK (A) or NACK (N) signals, which follow the return route reserved by the header flit. The role of the tail (T) is to notify the receiver that the whole packet is successfully received.

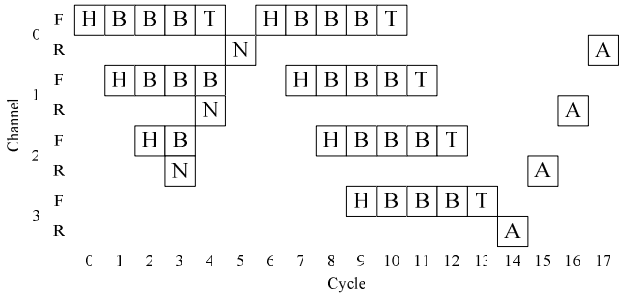


Figure 2.10: Time-space diagram of dropping flow control with explicit NACK

The second method for notifying a sender about packet dropping is by using a *timeout*. A time-space diagram of a dropping flow control with timeout is given in Figure 2.11. The figure shows a 5-flit packet being sent along a 4-hop route. The packet fails to acquire channel 3 on the first transmission. In this case, however, a NACK is not sent. Instead, the packet transmission continues across channels 0, 1 and 2. On each of these channels the tail flit (T) deallocates the resources held by the packet as it leaves the node. Thus channels 0, 1, and 2 become free during cycles 4, 5 and 6, respectively. After a timeout elapses without the source having received an acknowledgement, the source terminal retransmits the packet starting at cycle 12. This time the packet is successfully received. The receiving terminal sends an acknowledgement which arrives at the sender at cycle 23. Since no resources are reserved for the packet after the tail flit

passes, the acknowledgement must compete for reverse channels and may it self be dropped. In this case the packet will be retransmitted even though it was correctly received the first time. The timeout length equals the time needed for the packet to be received and the ACK to propagate back to the sender.

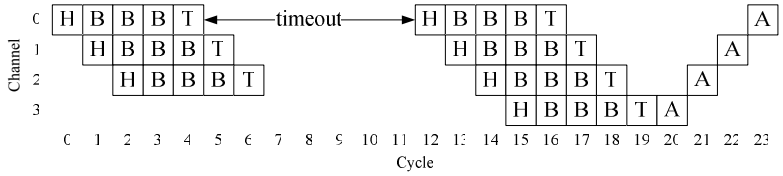


Figure 2.11: Time-space diagram of dropping flow control with timeout

Although simple, dropping flow control is inefficient because it uses bandwidth for transmitting packets that might later be dropped.

Misrouting flow control sends blocked packets in alternative directions instead of dropping them. In this case, there must be sufficient path diversity and an appropriate routing mechanism to route the packet to its destination from this point. While misrouting does not drop packets, it wastes bandwidth by sending packets in wrong directions. In some cases, this leads to instability; the throughput of the network drops after the offered traffic exceeds certain level. When misrouting is used, livelock is an issue – if a packet is misrouted too often, it may never come close to its destination.

Circuit switching is a form of bufferless flow control which operates by first allocating channels to form a *circuit* from source to destination and then sending one or more packets along the circuit. When no further packets need to be sent, the circuit is deallocated. The process involves four phases illustrated by the time-space diagram in Figure 2.12. During the first phase a request (R) propagates from the source to the destination and allocates channels. In a case of contention the request waits in the switch until the requested channel is freed. In this example no contention is encountered. After the circuit is allocated, an acknowledgement (A) is returned to the source during the second phase. Once the acknowledgement is received, the circuit is established and can handle an arbitrary number and size of data packets with no further control. In the example, two 4-flit data packets are sent and each is followed by three idle cycles. When no further data needs to be sent, a tail flit (T) is sent to deallocate the channels used by the circuit.

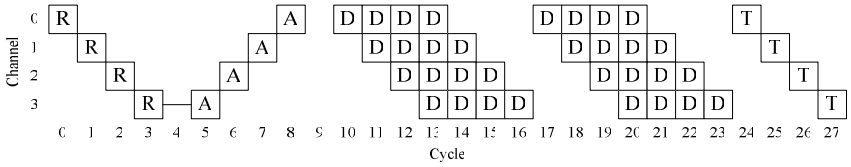


Figure 2.12: Circuit switching

Circuit switching differs from dropping flow control in that if the request flit is blocked, it is not dropped but held in place. The switches can store a request but not data.

Circuit switching is simple to implement, but it also has weaknesses: high latency and low throughput. At first, time is needed for establishing a circuit before sending a packet. The period of time the circuit is reserved is longer than the time it is used.

Time division multiplexing (TDM) is probably the simplest form of bufferless flow control. It is suitable for small scale networks. All routers in the network must have a common notion of time. In a TDM network all communications are statically scheduled. Each router has a local timetable which contains a cyclic schedule for forwarding the data from input ports to output ports. The schedules in all routers can be made conflict free. They are computed centrally for the network and then loaded in the routers. Such a network provides static communication channels with fixed throughput between source and destination pairs. However, to change the current state of communication channels, new schedules have to be computed and loaded in the routers.

Buffered flow control

Adding buffers to the network results in more efficient flow control, since the buffer decouples the allocation of adjacent channels. Buffers provide a place to store the packets while waiting for the allocation of the next channel, allowing the allocation to be delayed.

To explain and compare different buffered flow control mechanisms, we use the example situation illustrated in Figure 2.13. A packet traverses the network on a 4-hop path. The packet starts from router 0, passes through routers 1 to 3 and ends in router 4. Each router provides buffering space where the packet (or a part of the packet) can be stored. The figure shows the situation where the packet (shown in gray) is traversing from buffer 1 to buffer 2 on channel 1. All other details about the network and the routers have been ommited; the figure shows only the channels and the buffers traversed by the packet. We use this example in the following discussion on buffered flow control mechanisms.

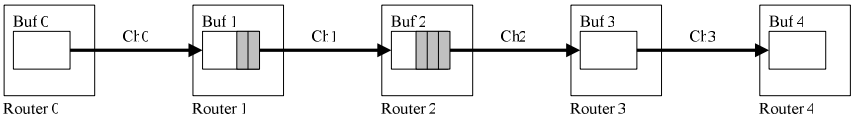


Figure 2.13: A packet traversing a network channel on its way through the network

Store-and-forward [27] is historically the first flow control mechanism used in the first computer network – ARPANET [48]. With store-and-forward flow control, each router along the path waits until a packet has been completely received (stored) and then forwards the packet to the next router. Each router should provide enough storage space to buffer at least one packet. The maximal packet length is limited by the provided buffer space. Before a packet is forwarded, it must be allocated two resources: buffer space in the next router and a physical channel.

Figure 2.14.a shows a time-space diagram of a store-and-forward flow control. The diagram shows a 5-flit packet being forwarded over a 4-hop route with no contention. At each step the entire packet is forwarded over one channel before proceeding to the next channel, which increases the packet latency. In each node the packet spends time t_r until the resources are allocated for its forwarding, assuming these resources are free and the packet does not wait. The time for forwarding the packet to the next node is L/b, where L is the length of the packet and b is the bandwidth of the allocated channel. Therefore, the zero-load latency of a packet travelling H hops route is:

$$T_0 = H\left(t_r + \frac{L}{b}\right) \tag{2.4}$$

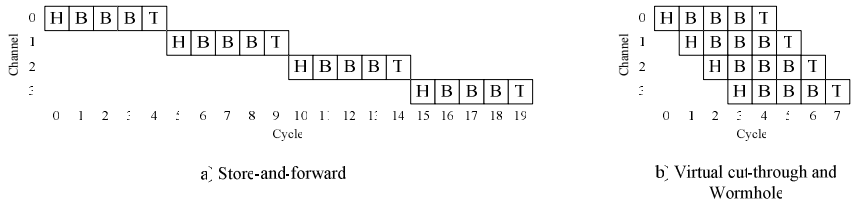


Figure 2.14: Time-space diagram showing a 5-flit packet sent over 4-hop route with no congestions using different flow control mechanisms

Virtual cut-through flow control [47] overcomes the latency penalty of the store-and-forward flow control by forwarding the packet as soon as the header is received, without waiting for the entire packet to be received. Nevertheless, buffer space is reserved for the entire packet, so that in a case of blocking the whole packet can be buffered.

A time-space diagram of a virtual cut-through flow control is shown in Figure 2.14.b. By transmitting the packet as soon as possible, virtual cut-through flow control reduces the packet latency to

$$T_0 = Ht_r + \frac{L}{b} \tag{2.5}$$

Both, store-and-forward and virtual cut-through flow control require large buffers. The buffer must be large enough to store at least one packet and the size of the packets is limited by the buffer space.

Wormhole flow control [68] minimizes the required buffer space by allocating buffers in units of flits instead of in units of packets. Like virtual cut-through it starts forwarding the packet as soon as its header is received, but buffer space is allocated only for several flits instead of for the entire packet. In the absence of congestion, wormhole and virtual cut-through perform in the same way. The time-space diagram for wormhole is the same as for virtual cut-through given in Figure 2.14.b and the zero-load latency of a wormhole packet is given by (2.5). The difference in the performance of both flow control mechanisms is observed when congestions occur. While in a virtual cut-through network the whole blocked packet is buffered in a single router and blocks only one input channel, in a wormhole network a router can buffer only part of the packet. The body of the blocked packet spreads over multiple routers along the path occupying one channel per router. Thus, in a wormhole network a blocked packet occupies multiple channels along its path, which results in a lower saturation throughput than virtual cut-through.

To reduce the effect of blocking, Po-Chi et al. [43] propose a scheme in which the blocked packets are dropped after a certain timeout expires and later retransmitted. The timeout period is calculated as a function of the packet retransmission cost and the cost of the performance penalties due to the blocking. The costs are functions of the current state of the network and their calculation may lead to an expensive implementation.

The advantage of wormhole flow control is that the required buffer space is reduced from the size of the packet to the size of only a few flits. This is of importance for the area constrained network-on-chip implementation, because buffers are the major area consuming components in the routers. Furthermore, wormhole flow control decouples the packet length from the buffer size. In a wormhole network we can have packets of

virtually any length. Thus, the procedure of message splitting and reassembling is avoided.

Virtual-channel flow control [21] improves network saturation throughput compared to wormhole flow control, while still keeping the required buffer space small and the packet length independent of the buffer size. *Virtual channels* (VCs) are logically independent channels that share the same physical channel. The packets are forwarded in the network over the virtual channels. When a packet is blocked, it blocks only the virtual channel it uses over a certain physical channel, but the other virtual channels can still use the physical channel. A physical channel is blocked only when all its virtual channels are blocked, the probability of which is lower than the packet blocking probability of wormhole flow control. Thus, the virtual channels keep the physical channels well utilised and the network throughput high. In a virtual channel network, the zero-load latency can be expressed by (2.5), but the term accounting for the serialization latency, L/b, must be adapted to take into account the sharing policy of the physical channels. The VCs do not use the full bandwidth b of the physical channel, but only a fraction of it, depending on the channel sharing policy and the occupation of the other VCs on the same physical channel.

The benefits of the virtual-channel flow control come at the expense of a more complicated control. Virtual channels introduce an additional stage of arbitration and allocation in the routers. While the previous flow control methods allocate to packets only physical channels, the virtual-channel flow control first allocates a virtual channel and then allocates bandwidth for the virtual channel.

The *Flit-reservation flow control* [61] addresses performance penalties in wormhole networks which are caused by specifics in the hardware implementation as follows. While wormhole flow control reduces the communication latency, the idealized router model can differ significantly from a hardware implementation. Typically, a router implementation is pipelined – in the router a packet has to pass through several pipeline stages till resources are allocated to it. Thus, the pipelining unnecessarily increases the packet latency. The idea of the flit-reservation flow control is to hide the latency for resource allocation by sending in advance control information to the routers about the arriving packets. A router then can allocate resources for a packet before the packet has arrived. The control information is separated from the data and sent on a separate faster control network, where the control flits race ahead of the data flits to reserve network resources. As the data flits arrive, they have already been allocated resources and can proceed with less latency overhead.

Table 2.1: Buffered flow control techniques
'+' = advantage, '-' = disadvantage;

Flow control	Buffer size	Zero-load latency	Saturation throughput	Independent packet/buf size	Control complexity
Store-and -forward	-	-	+	No	+
Virtual cut-through	-	+	+	No	+
Wormhole	+	+	-	Yes	+
Virtual channels	+	+	+	Yes	-

The small implementation area is one of the constraints that on-chip networks have to meet. As pointed out in [34] and as we shall see in Chapter 4, where router implementation issues are discussed, buffers are one of the main area consuming components in the on-chip routers. Therefore, minimization of the buffer size is crucial.

This makes bufferless flow control an attractive solution because of its low area cost. However, bufferless flow control makes inefficient use of the network channels.

Table 2.1 compares the advantages and disadvantages of the buffered flow control techniques. We choose virtual-channel flow control, which among the buffered flow control mechanisms requires minimum buffer space while offering high throughput and low latency. Moreover, with virtual-channel flow control packet length is not restricted by the buffer size and message splitting and reassembling can be avoided.

2.3.5. Routing

The final point in the network which a packet has to reach is given by the packet destination address. However, the destination address normally does not contain complete information how these point is reached. In most network topologies there is more than one possible path between any pair of nodes. Therefore, to deliver the packet from source to destination, first a path has to be selected. The procedure of selecting a path is called *routing*.

Classifications

Depending on *where* the routing decisions are taken, the routing is classified as source (or centralized) or distributed routing. With *source routing*, the exact path taken by a packet is known before the packet is injected in the network. The routing decision is taken either by the source node or by a routing function that is central for the network. The packets sent by the source node have a packet header containing not only a destination address, but also a description of the path. Each router on the path reads the packet header in order to determine in which direction to forward the packet. The routers do not take routing decisions, but simply follow the instructions given in the packets header.

With *distributed routing*, the path a packet takes is not known in advance. When the packet is injected in the network, only the address of its destination node is known. Each router the packet enters decides in which direction to forward the packet. Thus, the routing decision is distributed among the routers in the network. The source node does not have control over the paths taken by the packets it sends.

Depending on how the routing algorithm selects a path from the set of possible paths R_{xy} from source node x to destination node y, the routing is classified as deterministic, oblivious or adaptive. *Deterministic routing* always chooses the same path between x and y even if there are multiple possible paths ($|R_{xy}|>1$). These algorithms ignore the path diversity of the underlying topology and typically do a poor job on balancing the load. Despite this, they are common practice because they are easy to implement and easy to make deadlock-free. *Oblivious routing* algorithms, which include deterministic routing algorithms as a subset, choose a route without considering any information about the present network state. *Adaptive routing* algorithms choose a route taking the current network state into consideration. They adapt their decision to the state of the network as the usual goal is to balance the network load, to increase the network throughput and to reduce the packet latency. The state information used by adaptive routing to take its decision may include the state of a node or channel, length of a queue and historical information about the channel load. Adaptive routing algorithms differ in whether the algorithm uses local or global state information and whether current or history state information is used. The classification of the routing algorithms is represented in Figure 2.15.

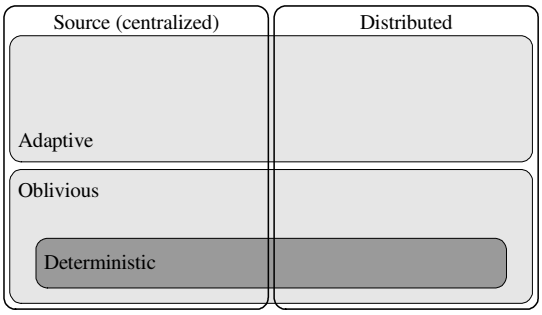


Figure 2.15: Classification of routing algorithms

When a routing function always returns a minimal path, it is referred to as *minimal*. Otherwise it is referred to as *non-minimal*.

Examples of routing algorithms

Dimension-ordered routing [59, 75] is a simple deterministic routing algorithm for *k*-ary *n*-cubes (tori and meshes). A packet injected in the network is first routed along the highest order dimension until it reaches its final position in this dimension. Then the routing continues in the next dimension and so on until the lowest dimension when the packet reaches its destination. If the address of the current packet position is $\{c_{n-1}, c_{n-2}, \ldots c_0\}$ and the address of the destination node is $\{d_{n-1}, d_{n-2}, \ldots d_0\}$, the packet is routed in the ith dimension until $|c_i - d_i|=0$, for $i = n-1, n-2, \ldots 0$. For example, in a 2D mesh, packets are routed first in *x*-direction and then in *y*-direction. For that reason the algorithm is also known as *xy-routing*.

Valiant's randomized routing algorithm [80] is an example of an oblivious routing algorithm. It balances the load for any traffic pattern and almost any topology by randomizing it. A packet sent from *s* to *d* is first sent to a randomly chosen intermediate node *x* and then to *d*. An arbitrary routing algorithm can be used for routing from *s* to *x* and from *x* to *d*.

The *Minimal adaptive routing* [53] algorithm is an example of a distributed adaptive routing algorithm that uses local state information. Each node can forward a packet only on channels that will bring the packet closer to the destination. Network state, typically a queue length, is used to select one of the possible channels.

Definition of a routing algorithm

A routing function is defined formally in the following way. Let an interconnection network *I* be defined as a strongly connected directed graph $I=(N,C)$. The vertices of the graph *N*, represent the set of network nodes. The edges of the graph *C*, represent the set of channels. Depending on whether the routing algorithm is source or distributed and whether it is node-based or channel-based, the routing function can be defined in three different ways:

(2.6) $$R : N \times N \mapsto P$$

(2.7) $$R : N \times N \mapsto C$$

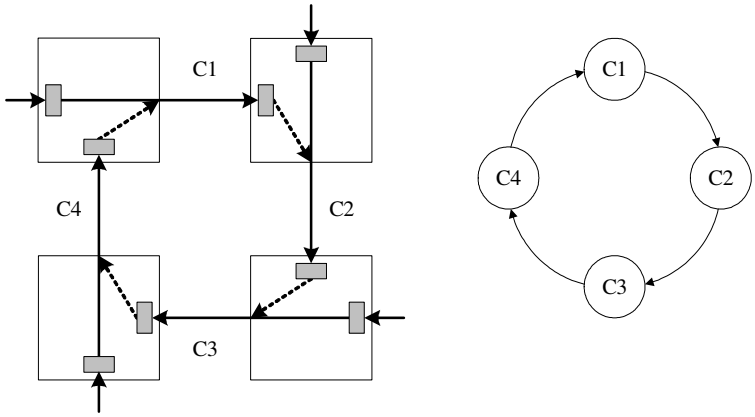
(2.8) $$R : C \times N \mapsto C$$

When source routing is used, the output of the routing function is an entire path, P, as in (2.6). The returned path is an ordered sequence $<c_1, c_2, \ldots, c_n>$ of network channels (alternatively it can be a sequence of nodes). When distributed routing is used, a routing function like (2.7) or (2.8) is evaluated once per hop of the packet. The output of the function is used to select the next channel the packet takes. Distributed routing can be node-based or channel based. Node-based routing uses a function of the form (2.7). It takes as input the current node and the destination node. Channel-based distributed routing uses a function of the form (2.8). The routing decision is based on the current channel and the destination node.

Distributed routing cannot implement every routing strategy which is possible with source routing. This is because little or no history is used from a packet to compute its next hop. In function (2.7), for example, nothing is known about the packet except the current node and the destination node. Function (2.8) adds information about the current channel, providing just enough history (where the packet comes from) to decouple dependencies between channels, which is important for avoiding deadlock.

Deadlock and livelock

Deadlock is a permanent condition in which a system cannot continue to function unless some corrective action is taken. A typical deadlock condition is waiting on an event that will never occur, where the reason is a circular resource dependency. Deadlock occurs in interconnection networks when a group of packets is unable to make progress because the packets are waiting on one another to release resources, usually buffers or channels. Such a situation is shown in Figure 2.16.a. The figure presents four routers (squares) and four packets (arrows) traversing them. Each packet traverses one router straight and enters a second router where it wants to make a right turn. To make the turn each packet has to wait (dashed arrow) for the requested channel to become free. The requested channels, denoted as c_1 to c_4, will become free only when some of the packets advance and release their channels, which will never happen. Thus, the four packets are deadlocked and will never make progress.



a) circular resource dependency b) resource dependency graph

Figure 2.16: Deadlock in an interconnection network

For the purpose of analysis of deadlock situations, resource dependency graphs are used [31]. The resource dependency graph of our example deadlock situation is given in

Figure 2.16.b. The nodes in the graph represent resources. In our situation these are channels. The edges of the graph represent resource dependencies; for a given resource (node), its output edges direct to the resources requested by the current resource owner, as here the owners are packets. For example, Figure 2.16.b presents a situation where the packet that holds channel c_1 requests (waits for) channel c_2. In resource dependency graphs, deadlock situations are recognized as cycles.

Two approaches are used in networking to cope with deadlock: *deadlock avoidance* and *deadlock recovery*. We will discuss each in turn.

Deadlock avoidance

Deadlock can be avoided by eliminating cycles in the resource dependency graph. One way to avoid the cycles is by restricting the routing function. A general framework for design of restricted deadlock free routing algorithms for mesh networks is proposed by Glass et al. [33]. It is known as *the turn model*.

The turn model defines a deadlock cyclic dependency in terms of the particular turns the packets have to take in the network to construct the dependency. Figure 2.17.a shows the eight possible turns in a two-dimensional mesh and the two simple cycles that can be created by combining these turns. By inspection, to avoid deadlock we must eliminate at least one turn in each of these two cycles. Three deadlock-free routing algorithms constructed by turn elimination are shown in Figure 2.17.b, c and d. In *west-first* routing shown in Figure 2.17.b, the two turns going in west directions are eliminated. A packet must take all its west hops before moving in any other direction. After turning from the west direction it may route in any other direction except west. In *north-last* routing shown in Figure 2.17.c, the two turns from north to west and to east are eliminated. A packet may move freely between the directions except north. Once a packet turns north it cannot route in any other direction. Finally, by eliminating the north-to-west and the east-to-south turns *negative-first* routing is derived. The east and north directions are considered as positive (+x and +y) horizontal and vertical directions, while west and south are considered as negative (-x and -y) horizontal and vertical directions. With this notion of direction, in negative-first routing a packet must move completely in the negative direction before changing to a positive direction. Once in a positive direction, the packet stays there until it reaches its destination.

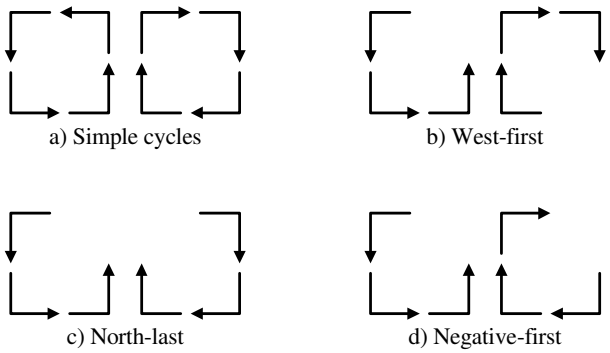


Figure 2.17: Turn model for a two dimensional mesh network

Figure 2.17 presents three basic cases of turn elimination resulting in deadlock-free routing. Other similar deadlock-free routing algorithms can be constructed by eliminating other pairs of turns [33].

Dimension-ordered routing can also be interpreted from the perspective of the turn model. As an example, in Figure 2.18, we represent the XY-routing in terms of the turn model. In XY-routing, a packet is first routed completely in the *x* direction before turning to the *y* direction. Only turns from horizontal to vertical directions are allowed. While the turn model eliminates only one turn in a simple cycle, the dimension-ordered routing eliminates two turns. Thus, the dimension-order routing is more restrictive than the general turn model.

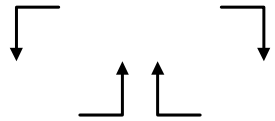


Figure 2.18: XY-routing (dimension ordered routing) as a more restricted version of the turn model.

The turn model and dimension-ordered routing restrict the path diversity which reduces the fault tolerance. In the case of dimension-ordered routing the path diversity is reduced to zero; only one path can be taken between source and destination. Additionally, these techniques cannot remove the channel cycles inherited in topologies such as the torus.

Another approach for eliminating cycles in the resource dependency graph and avoiding deadlock is by imposing a partial order of the resources and insisting that a packet is allocated resources in ascending order. With such a policy a deadlock cycle cannot occur, because to construct the cycle at least one higher-numbered resource holder must request a lower-numbered resource, which is not allowed.

Dally et al. [24] proposes a method for deadlock avoidance by resource ordering which is illustrated by the example given in Figure 2.19. The figure presents a four-node ring network. In each node the channels (and the buffers) are duplicated and divided into two classes, class A and class B. When a new packet is injected in the network it uses only the resources of class A, but after the packet crosses the border between the fourth and the first node, it can use resources only from class B. As shown in the figure, with such a resource allocation policy cyclic dependencies between resources are avoided.

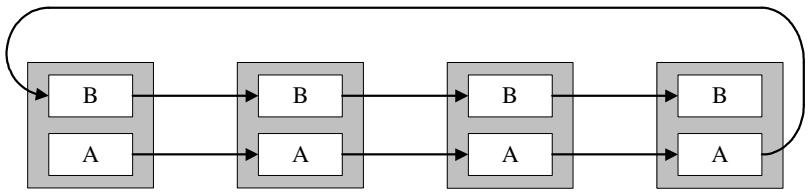


Figure 2.19: Deadlock avoidance by resource ordering

A disadvantage of the resource ordering approach is that the amount of network resources is increased in order to introduce resource classes and provide proper ordering. Also, an imbalance of resource usage is observed – while some resources are overloaded others are underutilised.

Deadlock recovery

The techniques discussed until now focus on eliminating the conditions that cause deadlock. These methods require a restricted routing function or additional resources to break the resource dependencies. Another approach for dealing with deadlock is to recover from deadlock when it occurs instead of avoiding it. *Deadlock recovery* relies on the fact that deadlocks will be infrequent and the average-case performance rather than the worst-case performance is considered important. Deadlock recovery algorithms involve two key phases: deadlock *detection* and *recovery*. In the detection phase, the deadlock configuration is recognized. The detection is usually accomplished using timeout counters. With each network resource is associated a counter, which is reset when data is sent through the resource. However, if the counter reaches a predetermined threshold, the resource is considered deadlocked and the recovery phase is started. The deadlock recovery may be regressive or progressive. In *regressive* recovery, the packets that are deadlocked are removed from the network. Notification to the sender of the dropped packet is required in order to initiate retransmission. In *progressive* recovery [13], the deadlocked packet is not dropped, but drained in a special escape buffer. Once in the escape buffer, the packet is routed using a deadlock free routing algorithm. In this way, the restrictive deadlock free routing is used rarely, only when a deadlock occurs, while in the rest of the time routing is given full freedom.

Livelock

Livelock is a situation in which a packet continues to move in the network but never reaches its destination. Livelock may occur with non-minimal routing algorithms that misroute packets. If there is no limit on the maximum number of times a packet may be misrouted, the packet may remain in the network indefinitely. One technique for avoiding livelock is to add a small amount of state information to each packet. The state can be a *misroute count*, which holds the number of times a packet has been misrouted. Once the count reaches a threshold, no more misrouting is allowed. A similar approach is to store an *age-based priority* in each packet. When a conflict between packets occurs, the oldest packet wins.

Having in mind the area constraints on the router implementation, increasing the number of the routing resources is not desirable. Among the reviewed deadlock solutions, only the turn model does not require additional resources. It simply restricts the routing algorithm which is inexpensive to implement. Although restricted, the routing algorithm can still be made adaptive and flexible. The disadvantage is that the turn model works only for mesh networks and not for tori.

2.3.6. Quality of Service (QoS)

Network design is usually focused on improving the average network performance and aims at achieving higher average throughput and lower average latency. But no matter how good the employed network techniques are, situations remain in which congestions occur and resources are allocated to one network user while another has to wait. Hence, different network users may experience different throughput, latency and in general a different quality of service. On the other hand, different parts of the network traffic may have different requirements about the services received from the

network. For example, some part of the traffic may be latency-sensitive, while another part is not. Some part of the traffic may tolerate data loss, while others do not.

It is useful to divide the network traffic into a number of traffic *classes* according to the services they require. The traffic classes fall into two broad categories: *guaranteed service* (GS) and *best effort* (BE) services. Guaranteed service classes are granted a certain level of performance as long as the traffic they inject complies with a set of restrictions. The restrictions usually set an upper bound on the volume of traffic the client can inject – the maximal offered throughput. In exchange, the network specifies guarantees about the services it provides, for example the maximal latency.

In contrast to the guaranteed traffic, the network does not give any guarantees about the services provided to the best effort traffic. Depending on the current traffic conditions, the best effort traffic may experience an arbitrary low throughput and high latency. The network will simply do its best to deliver the packet to its destination.

The network services are categorised into classes according to a number service characteristics. The service can be with or without losses depending on whether it guarantees that all data sent on the network are delivered. The service may or may not guarantee that data is delivered without fault. The service may or may not guarantee that data is delivered in the order it was sent. According to their quantitative characteristics services can be classified as low or high throughput, low or high latency, low or high jitter. When some of the quantitative characteristics are of importance the service provides a worst case bound for it and respectively falls in the category of guaranteed services. The services can be also classified according to the way they are accessed: whether they are granted or have to be requested, whether they require or not a connection to be established between source and destination.

2.4. Network-on-Chip solutions

Having discussed all the major techniques we are now well equipped to some recent network-on-chip solutions. We discuss what techniques they use for implementing the network and for providing guaranteed and best effort services. We discuss only the most mature solutions and techniques that are relevant to this thesis.

2.4.1. Circuit switching solutions

SoCBUS [84] is a network-on-chip solution that combines circuit switching and dropping flow control. The network uses a two-dimensional mesh topology. As in circuit switching, before sending the data a request is sent on the network to reserve channels to the destination. The request is routed using distributed minimal adaptive routing. In contrast to circuit switching, when the request is blocked it is not stored in the switch, but is dropped as with dropping flow control and a NACK signal is sent back to the sending node. The advantage of the dropping flow control in this case is that it makes the network deadlock-free, because all blocked requests are dropped. Circuit switching has a simpler and more area efficient implementation than packet switching techniques. The reported maximal clock speed of a SoCBUS switch is 1.2 GHz in 0.18 μm technology.

A circuit switching network can provide guaranteed services by keeping circuits permanently open. A circuit is opened once during the setup phase and then remains open. The opened circuit guarantees constant throughput and latency, because the resources it uses are allocated and not shared. But during the time the circuit is open the

physical channels it uses are reserved and cannot be used by other circuits. Since the number of physical channels is limited, only few connections can be opened simultaneously. On the other hand, when the circuits do not remain open but are closed and opened again when needed, latency cannot be guaranteed because the time for opening a connection is not bounded (the request may be blocked and dropped many times). Therefore, circuit switching is not an efficient solution for providing guaranteed services, especially in dynamic systems.

Wolkotte et al. [86] proposes a circuit switching network with improved capabilities for providing guaranteed services. The improvement consists in increasing the number of physical channels in the network. The network uses a two-dimensional mesh topology, but instead of one physical channel between the neighbouring routers there are four physical channels. Thus, this solution utilises the huge amount of wiring resources provided by the semiconductor technology. By increasing the number of physical channels, the number of circuits that can be opened simultaneously is also increased.

To simplify the design further, the network proposed by Wolkotte et al. does not implement mechanisms for opening and closing circuits (request, ACK and NACK signalling). Instead, each switch has a configuration interface through which connections are set between the switch input and output ports. The reported area of a switch with 4-bit wide channels, implemented in 0.13 μm technology is 0.05 mm^2 and the maximal clock frequency is 1 GHz.

The configuration interface is accessed through an additional serial wormhole network with ring topology [85]. Each router in the wormhole network is connected to the configuration interface of a circuit switch. The wormhole network is used to carry configuration messages to the circuit switches and to handle the best effort traffic in the system. To configure a circuit, configuration messages are sent over the wormhole network to all the switches along the path of the circuit.

2.4.2. Packet switching solutions

The Æthereal network-on-chip [34] provides guaranteed services (GS) and best effort (BE) services by combining two techniques: time-division multiplexing (TDM) and wormhole routing. The routers of the Æthereal network consist of two parts. One part handles the BE traffic and employs wormhole routing. The other part handles the GS traffic and employs TDM. The network has a global notion of time. The network channels are used on a timeslot basis. GS services are provided by timeslot reservation. A GS communication channel is constructed by reserving timeslots on the physical channels along the path between the source and the destination node. All the GS traffic in the network is scheduled in timeslots. The BE wormhole network uses the time slots left unused by the GS traffic without preliminary reservation.

The Æthereal network is also equipped with a network interface (NI) [66]. The NI is a component connecting the IP modules to the network routers. The functionality provided by the NI is related to partitioning messages into packets, message reassembling and end-to-end flow control.

The Æthereal network is available in three versions: GS-BE distributed programming architecture, GS-BE centralised programming architecture and GS centralised programming architecture. With the distributed programming architecture every router is equipped with a timeslot table which controls the forwarding of GS packets from input ports to output ports. The tables are configured by the senders by

sending a special configuration BE packet which travels from source to destination configuring the tables of all traversed routers. The area of this version of the router is 0.24 mm^2 (32-bit channels, 24 word buffers and 256 slots in 0.13 μm CMOS technology) and the maximal clock frequency is 500 MHz.

The GS-BE centralised programming architecture moves the timetables from the routers to the NIs. Now the tables control the exact time the GS packets are injected in the network. The GS packets traverse the network with a constant speed of one hop per cycle and follow predetermined paths. Hence, knowing the packet injection time we know exactly which timeslots on the network channels the packet utilises. The tables programming is preformed by a central system authority (root). The root configures the NIs either through memory mapped I/O interface or through special system GS or BE packets (ReserveSlot and FreeSlot). By moving the timetable to the NI the router area is reduced to 0.13 mm^2. The maximal clock frequency is 500 MHz.

The GS centralised programming architecture is the same as the GS-BE centralised but without the wormhole part. Thus this version does not support BE traffic but the router area is reduced to 0.03 mm^2 and the maximal clock frequency is 1000 MHz.

TDM has simple implementation and provides a straightforward way for achieving predictable networks (and systems) operation. However, TDM alone cannot handle BE traffic. A separate BE solution is required, which adds extra cost. A drawback of TDM is also the need for schedule computation. Every time the setup of the GS connections in the network is changed a new schedule has to be computed. Schedule computation is an NP complete problem and as the size of the network increase reconfiguring the network becomes a heavy and slow task. This inflexibility make a TDM network not the most suitable solution for a dynamic system. Furthermore, the global notion of time in a TDM network requires global clock distribution, which makes the network difficult to apply in the GALS (Globally-Asynchronous Locally-Synchronous) systems foreseen in the near future.

The area of the Æthereal router with five 32-bit channels, 24 word buffers and 256 slots in 0.13 μm CMOS technology is 0.24 mm^2. The router can operate at maximal clock frequency of 500 MHz.

The Æthereal network is also equipped with a network interface (NI) [66]. The NI is a component connecting the IP modules to the network routers. The functionality provided by the NI is related to partitioning messages into packets, message reassembling and end-to-end flow control. The size of the NI is 0.143 mm^2 in 0.13 μm technology and operates at a maximal clock frequency of 500 MHz.

The Æthereal solution does not consider a particular network topology but the network is generated automatically to satisfy the communication requirements of a target application as at the same time the network area overhead is minimised [36]. The generated network can be of any topology, regular or irregular. While such a network minimizes the area overhead and provides a scalable solution for IP core interconnection, it is not clear whether it structures the global on-chip wiring and helps to solve the signal integrity problem.

aSOC [52] is a framework for on-chip communication in heterogeneous tiled architectures. The topology of the network is a two-dimensional mesh. aSOC implements an advanced TDM scheme capable of handling more dynamic traffic patterns, but still keeps the hardware simple. Instead of a slot table, each router has a sequencer that allows switching between different timeslot schedules at run-time. A new schedule is not loaded at run time; all the schedules are loaded at configuration time and at run-time different schedules can be activated. Although this approach adds flexibility

to TDM, it still requires all communications to be scheduled at compile time. Implemented in 0.18 μm technology the network speed is 400 MHz.

Proteo [70] is packet switching virtual cut-through NoC. It is intended for heterogeneous systems. The network itself is heterogeneous. Its topology is hierarchical. A number of subnets with a ring topology are connected by a system-wide ring network. The subnets interconnect clusters of functionality-related components. The network is constructed from a number of parameterised components. By choosing the components parameters at design time, a network that meets the requirements of an arbitrary application can be constructed. The packets in the network are of limited length, which requires message splitting and reassembling at the network entrance and exit point.

In Nostrum NoC [56], guaranteed services are provided using a TDM related technique called temporally disjoint networks. The advantage is that no slot tables are needed. Instead, a specific slot reservation scheme called looped containers is used. To our knowledge, no implementation results are available for this network.

SPIN [10] is a packet switching wormhole network with distributed adaptive routing. The topology is indirect and employs a three stage Clos network [27]. While this network scales well in number of terminal nodes, it does not show structured and regular wiring. Therefore, it does not help in coping with the signal integrity problem. Implemented in 0.18 μm the network router has area of 0.24 mm^2 and operates at 200 MHz [12].

2.4.3. Clockless solutions

The NoC solutions presented below are also packet switching, but they are implemented using asynchronous design techniques and do not rely on a globally distributed clock. Therefore, they have the advantage of being directly applicable for GALS systems foreseen for the near future.

The MANGO network, proposed by Bjerregaard et al. [18], is a clockless NoC solution that employs virtual channels (VCs). The network provides guaranteed services (GS) as well as best effort (BE) services. The GS services are connection-oriented and are provided by means of VC reservation. The VCs at each physical channel are divided in two sets – one set of VCs serving the GS communications and one set serving the BE communications. The VCs from the GS set are used to carry GS connections. Each VC from the set can be statically allocated to at most one connection. Thus, connections do not share VCs and blocking is avoided. Data is sent over the connections without additional control information. The VCs from the BE set are used to carry connectionless source-routed data packets. The VCs from this set are shared and are allocated to packets dynamically. Therefore, packet blocking is possible and throughput guarantees cannot be given.

The two sets of VCs share the same physical channels, but in the router they are treated separately. The router in the MANGO network consists of two parts, a GS part and a BE part, which manage the data from the corresponding set of VCs. The GS part of the router multiplexes the GS data arriving on the input links to the output links. The multiplexing is controlled by a map stored in the GS part of the router. The map gives the correspondence between the input GS VCs and the output GS VCs, as this correspondence is considered static during a connection usage. The map is loaded in the GS part of the router through the BE part by sending a BE packet to the router. Thus, to set a connection, configuration BE packets have to be sent to all the routers along the route of the connection. The BE part of the router controls the data transfer on the BE

VCs. It implements the functionality of a virtual channel router. It retrieves the routing information from the received packet headers and forwards the packets according to this information.

The reported area of a MANGO router [18] with five 32-bit links and 8 VCs per link implemented using 0.12 μm CMOS standard cells is 0.188 mm^2. The performance of the router corresponds to 515 MHz.

Felicijan et al. [32] propose another clockless NoC that employs VCs, but instead of VC reservation, GS services are provided by means of VC priorities. The network has four VCs per physical channel. Three of them are used for guaranteed service connections and the BE traffic shares the fourth VC. The four VCs are assigned static priorities and the BE traffic uses the one with the lowest priority. Thus, packets using higher priority VC receive better services: higher throughput and lower latency. However, since packet blocking on the high priority VCs is still possible, the received service is difficult to predict and guarantee. To our knowledge, no implementation results are available for this network.

2.4.4. Summary

Table 2.2 summarises the techniques and the implementation results for the reviewed NoC solutions

Table 2.2: Summary of the reviewed NoC solutions.
na = information is not available

NoC	Provided services	Topology	Flow control	Routing	Area [mm2]	F [MHz]	Tech. [mm]
SoCBUS	BE	2-D mesh	Circuit switching with dropping	Distributed, minimal adaptive	na	1200	0.18
Wolkotte	GS, BE	2-D mesh, Ring	Circuit switching, cut-through (serial)	na	0.05	1000	0.13
Æthereal Distib.	GS, BE	Any	TDM, Wormhole	contention-free routing, source routing	0.24	500	0.13
Æthereal Centr.	GS, BE	Any	TDM, Wormhole	contention-free routing, source routing	0.17	500	0.13
Æthereal GS only	GS	Any	TDM	contention-free routing	0.03	1000	0.13
aSOC	GS	2-D mesh	TDM	na	na	400	0.18
Proteo	na	Hierarchical rings	na	na	na	na	na
Nostrum	GS, BE	2-D mesh	TDM	diflection routing	na	na	na
SPIN	BE	3-stage Clos network	Wormhole	Distriburted adaptive	0.24	200	0.13
MANGO	GS, BE	grid-type	Virtual channels	source routing	0.19	515*	0.12

Table 2.3 compares only the NoC solutions providing both GS and BE services. In all these networks, GS is provided by means of resource reservation. The approaches rely either on circuit reservation, time-slot reservation or VC reservation. To open a GS connection, resources (circuits, time slots or VCs) are reserved over network channels.

All the networks assume that the reservation is done centrally, by a global network authority. Therefore, they require a centralized system organization.

The centralised network resource management ease the application of deadlock avoidance solutions. A deadlock avoidance algorithm can be directly incorporated in the resource reservation and traffic planning in the system.

Although the networks differ in term of the techniques employed, in all of them opening a GS connection requires that a central authority reserves resources at network channels. From an algorithmic point of view, the problem of resource reservation in all NoCs presented in Table 2.3 is the same or at least quite similar. In all of them the network channels provide multiple resources of some type (lanes, time-slots, VCs) and an algorithm searches for resources to be reserved at the channels from source to destination, such that a path is constructed. Thus, we may expect that the task of opening a GS connection will have similar computation complexity for these networks. However, the approaches differ in whether the resources on network channels are ordered or not. While it is not of importance which circuits or VCs are reserved on a channel, for the time-slot reservation it is of importance which time-lots are reserved. For example, in the most restrictive case, if a time-slot *i* is reserved on one channel, then the time slot *i*+1 must be reserved on the next channel and so on, such that the data advances on the path with every time slot. Thus, the time-slot reservation (TDM) approach is more restrictive in resource reservation. As a result, providing a GS connection when most of the resources are occupied will be more difficult in a TDM network than in a network where GS are provided by means of VC or circuit reservation. Moreover, since TDM relies on a global notion of network time it is not suitable for application in globally-asynchronous locally-synchronous (GALS) system where global clock, and therefore global notion of time, is not available.

Table 2.3: Specifics of the networks providing GS and BE services

NoC	GS approach	Applicable in GALS systems	GS connection is configured by	Combining GS and BE
Wolkotte	Circuit reservation	yes	BE packets	Separate networks
Æthereal	Time-slot reservation	no	BE packets	Separate router parts
Nostrum	Time-slot reservation	no	The source node	Single router
MANGO	VC reservation	yes	BE packets	Separate router parts

The NoC solutions presented in Table 2.3 differ in the mechanisms used for configuring (opening) a GS connection; i.e. the actions that have to be taken to open a GS connection after resources have been reserved for it at a higher level. In most of the networks a GS connection is opened by BE packets that configure the routers along the connection. Either separate BE packets are sent to all the routers along the GS connection, or a single BE packet is sent through the routers along the connection. In any case, an explicit connection setup is required before using the connection. Since BE packets are used for that, no guarantees can be given for the duration of the setup period. In Nostrum a GS connection is opened directly by the sending node without sending an explicit BE packet. However, throughput guarantees can be claimed for a connection only during the network setup period when the system is started. Therefore, in Nostrum GS connections cannot be opened dynamically during the system operation

time. Only in Æthereal (centralised programming architecture) the configuration is done by GS packets and the time for opening a connection can be predicted.

Finally, Table 2.3 compares how the networks combine the GS and BE services in a single solution. Wolkotte uses two separate networks to carry GS and BE traffic. Æthereal and MANGO uses a single network, but in the routers both types of traffic are processed separately. In the Æthereal router BE and GS packets use different data paths (buffers) and in the MANGO router they are processed by separate sub-routers. Only in Nostrum both traffic types are processed by a unified router solution.

2.5. Conclusion

In this chapter we review the most important techniques for building interconnection networks. Two-dimensional mesh and torus topologies naturally fit the physical placement of the tiles on a chip and have the potential to provide a simple and regular global wire layout together. Among the reviewed flow control mechanisms, virtual-channel flow control provides a balance between performance and buffer size. Taking into account the area constrained NoC implementation, the most efficient solution to avoid deadlock is the restricted routing, using the turn model.

The main problem in the NoC design for our application domain is the provision of guaranteed services together with best effort services. The difficulty comes from the fact that the resources available for building a router are limited due to the design area constraints. In most of the reviewed NoCs, separate techniques are used for each service class. Either two separate networks are implemented to deal with each traffic class or the routers consist of two separate parts, each dedicated to a different traffic class. In the second case, both types of traffic share the same network channels, but inside the router they are processed by different parts of the router. As we will demonstrate in the rest of the thesis, this separate treatment of BE and GS is unnecessary. By unifying the two classes of traffic solution can be found.

The second problem in the NoC design is the network reconfiguration. The approach taken for network reconfiguration determines whether the network can be reconfigured at runtime and whether the time for reconfiguration can be predicted and guaranteed. Since we aim at dynamically reconfigurable system, runtime reconfiguration and predictable reconfiguration time are mandatory for out network. The solution we present satisfies this requirement.

The third problem that has to be considered when designing a NoC is the computation of the network configuration. In most of the reviewed network solutions it is assumed (some times silently) that the configurations loaded into the network are provided by some central network authority or that pre-computed configurations are used. We also assume a central system authority. However, in our case the configurations are computed at run-time and therefore the time for computing a configuration is a critical issue – it must be small and bounded. We will present a network solution that requires a configuration computing task of lower complexity that the task required by the statically scheduled (TDM) network solutions. Thus it is a solution more suitable for systems where the network configuration is computed at run-time.

Chapter 3

Network-on-Chip architecture*

This chapter proposes a router architecture and a resource reservation scheme, which when combined enable us to provide guaranteed as well as best effort network services in a virtual channel network. The concept is verified through a simulation and compared with other solutions.

3.1. Introduction

The review of networking techniques in Chapter 2 shows that virtual-channel flow control has the following two advantages over other flow control techniques: i) it allow the size of the network buffer to be reduced without significantly affecting the network performance, ii) the length of the network packet is not restricted by the size of the network buffers.

In Chapter 4 we shall see that a major part of the area of a network router of such a small scale as a NoC router is occupied by buffers. Therefore, for the area constrained router implementation it is crucial to minimise the router buffer size. However, with most flow control mechanisms the buffer size reduction has a negative effect on the network performance; the network throughput drops and the message latency increases. For some flow control mechanisms the small buffer size also implies impractical constraints on the packet length. With the buffer size that we aim in our design (of several data words) the maximal packet length would be only a few words, which would be rather inconvenient when handlings intensive data streams – the major part of our traffic.

The virtual channel flow control is the only flow control mechanism which enables us to reduce the total size of the router buffers, and thus to minimise the area, without sacrificing performance; the cost is only a small increase of latency while the packet length is unrestricted. This is the reason for us to choose virtual channel flow control for our Network-on-Chip (NoC). A network based on a virtual-channel flow control we shall call a *virtual channel network*.

In this chapter we look at the possible architectures for a router in a virtual channel network. Our objective is to define an architecture that allows providing of predictable communication services. We propose predictable router architecture and complement it with a method for providing guaranteed services (GS) and best effort services (BE) on a network level. Like most of the other NoCs, the approach we use for providing guaranteed services relies on a centralised system organisation. What distinct our

* Major parts of this chapter have been presented at the International Symposium on VLSI [6], at the International System-on-Chip Conference [5], at the EUROMICRO Symposium on Digital System Design [1] and at the PROGRESS Embedded Systems Symposium [3].

approach is the unified treatment of BE and GS traffic at a router level. Our router architecture does not include features that are specific for the one or the other type of traffic. The distinction between BE and GS traffic is made at a network level when resources are distributed between the traffic by a central system authority. Thus, the router architecture is simplified and the architectural choices are less traffic dependent.

In the following discussion we consider a network like the one shown in Figure 3.1. The network routers are arranged in a grid. Each router is connected to its neighbouring routers and to a local processing element (PE) by full-duplex channels. The constructed network topology is a two-dimensional mesh.

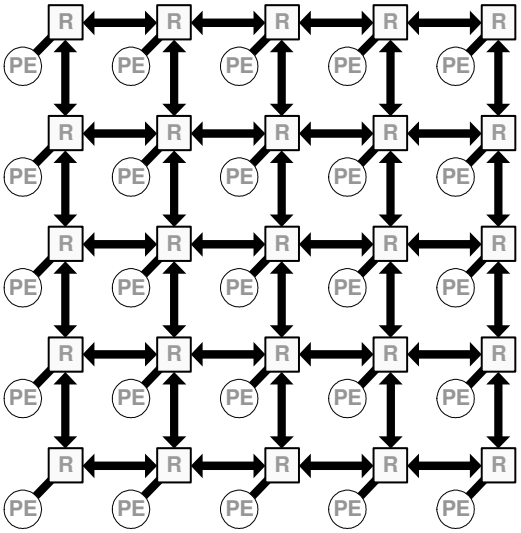


Figure 3.1: Network architecture

All routers in the network are identical. Each router has five input and five output ports – four connected to its neighbours and one to its local PE. The routers at the edge of the mesh are connected to fewer than five channels. The ports that are not connected to channels are either left unused or connected to external interfaces or peripheral devices. The only active network components are the routers. The PEs are not considered part of the network. They are present as the source and sink of data.

For more gradual explanation of the virtual channel router and its operation, we first introduce a wormhole router and then extend it with virtual channels.

3.2. Wormhole router architecture

Wormhole routers, as the name suggests, employ wormhole flow control [68]. Historically, the wormhole flow control was called wormhole routing, but actually has nothing to do with routing. The typical architecture of a wormhole router is presented in Figure 3.2. The router has a number of input and output ports connecting it to neighboring routers or a PE. In the considered example network, the routers have five input and five output ports. Each input port has a first-in-first-out (FIFO) buffer to store the incoming data. From the buffer the data is switched to an output port and transmitted to the next router. The switching is done by a fully connected switching fabric configured by a switch allocator.

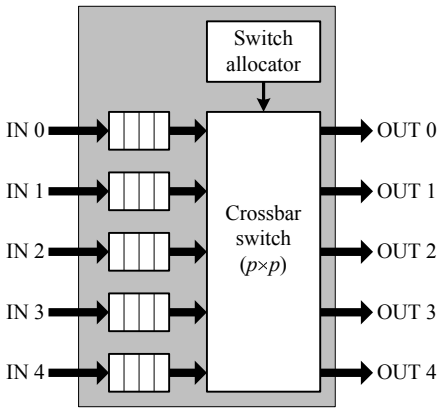


Figure 3.2: Architecture of a wormhole router

The data in a wormhole network is transported in packets. The packet encapsulates the transported data together with a small amount of control information. The packets are constructed of a series of smaller data units called flow control digits or *flits*. A flit is the atomic data unit that the flow control operates on; this is the data unit transmitted between two routers in a single flow control transaction. Flits are of constant size – one or more data words. Each flit is tagged as either: *header (H)*, *body (B)* or *tail (T)*. Packets are constructed by flits of different type. A typical packet format is shown in Figure 3.3. A packet starts with one or more header flits (H), followed by one or more body flits (B) and terminates with a tail flit (T). The sequence of header flits forms the packet header which constructs the beginning of the packet. The packet header carries routing information and control information used by routers. The sequence of body flits carries the transported data (the payload). The tail flit indicates the packet end.

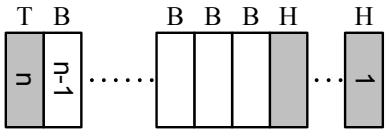


Figure 3.3: Packet format

To send data over the network, a sender PE constructs a packet and injects its flits in the network. Each router on the packet's route forwards the flits to the next router until the destination PE is reached. The destination PE receives the packet flit-by-flit.

In each traversed router, before it is forwarded the packet passes through two stages of processing. The first stage is the *routing* stage. In this stage the router examines the routing information in the packet header and determines the packet destination output port. In the second stage the packet is allocated a crossbar connection to the destination output port. This is the *switch allocation* stage. When the switch connection is allocated, all the packet flits are forwarded to the output port and the switch connection is released after the tail flit.

When the destination output port is currently occupied by another packet, the crossbar connection can not be provided and the packet blocks. The blocked packet waits until the port is released and the switch connection is allocated. While waiting the packet is stored in the network buffers. The first flits of the packet are buffered in the

router where the blocking occurred. The next part of the packet is buffered in the previous router and so on. To avoid data loss, the flow control provides a back pressure mechanism that blocks the output port of the previous router in case the buffer of the next router gets full. In that way, the blocking propagates back along the packet route to the packet tail or the source PE.

It is possible that several packets destined for the same output port have arrived on different input ports and are at a same time in the switch allocation stage. In such case *packet contention* occurs and so *arbitration* is needed to solve it. One of the contending packets is chosen according some arbitration policy and granted the output port while the other contending packets have to wait.

The arbitration and allocation is done by the switch allocator (see Figure 3.2). The switch allocator receives requests from the input ports, decides what switch connections to provide and configures the crossbar switch. Although it is not shown in Figure 3.2, the switch allocator interacts with the input ports too. Figure 3.2 is simplified to show only the data path in the router and does not show the control paths. If we look at each input port in more detail, we find not only a FIFO buffer, but also control logic which together with the buffer forms an input controller. The input controllers of all ports interact with the switch allocator as shown in Figure 3.4.

Besides the FIFO, the input controller consists of routing logic and port control logic. The routing logic examines the packet header and determines the packet destination output port. The port control logic stores the current state of the packet and interacts with the switch allocator to request an output port and to forward flits. The input controller also sends information about the available buffer space back to the previous router. This information is usually referred to as *credits*. The credits are used by the switch allocator in the previous router to prevent sending data to full buffers.

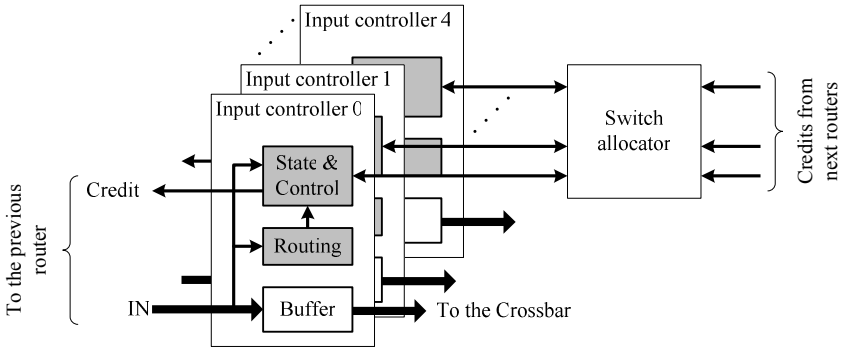


Figure 3.4: Input controller – structure and connection to the switch allocator

The switch allocator arbitrates between the requests coming from all input ports and controls the flit forwarding process. The maximal rate at which the requests can arrive and flits can be forwarded is determined by the time for transmitting a flit over a channel between two routers. Therefore, the maximal rate at which the switch allocator operates is determined by the flit transmission time. The flit transmission time depends on the physical channel rate and the flit size. A flit of size L bits is transmitted on channel of rate b bit/s for time L/b seconds. Thus, the larger the flit, the lower the switch allocator operating rate is. The flit size is used to adjust the speed of the router to the speed of the physical channel. This is needed when the router implementation is slower than the rate at which the data is delivered by the channels. The flit size also influences

the buffer size. As we shall see in Chapter 4, our router implementation is fast enough, so we can choose the smallest flit size to minimize the buffers. In our network, the size of the flit is a single word and a flit is transmitted over a network channel in a single clock cycle.

In a wormhole network, blocked packets are stored not in a single router but spread over several routers along the route. In each of the routers a packet occupies a FIFO buffer and its input channel. These channels cannot be used by other packets until released, so other packets may block waiting for them. These packets may, in turn, cause other packets to block and so on, the blocking spreads as a tree in the network. This is known as the effect of *tree blocking*. Tree blocking is the reason why wormhole networks have a low saturation throughput. Tree blocking increases the packet blocking probability and as a result the network channel utilisation and the network saturation throughput decreases. For a formal discussion on the performance of wormhole networks, refer to [30, 44].

The high packet blocking probability in wormhole networks causes low saturation throughput and low channel utilization. To improve the performance of wormhole networks, virtual channels should be employed.

3.3. Virtual channel router

Virtual-channel flow control is a technique that allows several logically independent channels to share the same physical channel. Employing virtual channels to improve the performance of a wormhole network was proposed by Dally [21]. The idea of virtual channels is shown in Figure 3.5. The figure shows two routers with a physical channel between them. In the transmitting router, several virtual channels (VCs) are multiplexed on the physical channel. In the receiving router the VCs are demultiplexed and buffered separately. Thus, the VCs time-share the physical channel. The multiplexing is controlled by an arbiter and the arbitration policy determines how the bandwidth of the physical channel is allocated to the VCs.

The packets traverse the network using VCs. When a packet blocks, it keeps the VCs it traverses occupied. However, the traversed physical channels are not blocked since other VCs on these channels can continue to carry data. Hence, packet blocking does not directly cause physical channel blocking and the effect of tree blocking is reduced. Therefore, employing VCs in a wormhole network improves the physical channel utilization and the network saturation throughput. The cost is only a small increase of the average packet latency; however, the worst case latency stays the same.

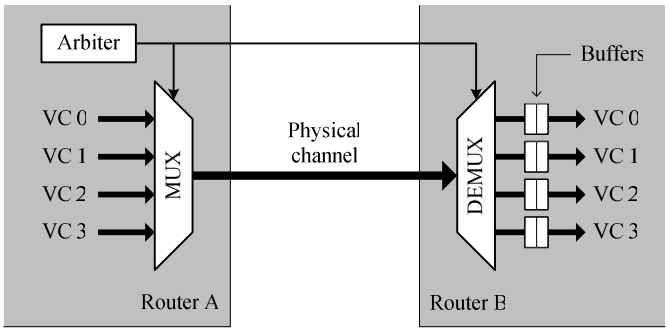


Figure 3.5: Implementing virtual channels (VCs) on a physical channel

Figure 3.6 presents two possible architectures for a virtual channel router. They differ in where the VC multiplexing takes place. In the architecture of Figure 3.6.a, the VCs are multiplexed immediately after the buffers, before the crossbar inputs. The crossbar is symmetric – in a router with p ports the crossbar size is $p \times p$ ports. Hence, we call this architecture a *symmetric architecture*. With the symmetric architecture the size of the crossbar does not depend on the number of VCs. In the second architecture shown in Figure 3.6.b, the VCs are directly connected to the crossbar without multiplexing. The crossbar is asymmetric – for v VCs per physical channel, the crossbar size is $pv \times p$ ports. Hence, we call this architecture an *asymmetric architecture*. With the asymmetric architecture the crossbar size depends on the number of VCs.

As we shall see, the asymmetric architecture simplifies the router control logic, but because of its larger crossbar it is practically excluded from consideration [21, 62]. However, in Chapter 4 we show that when appropriately implemented the area results for the asymmetric architecture can be competitive with those of the symmetric architecture.

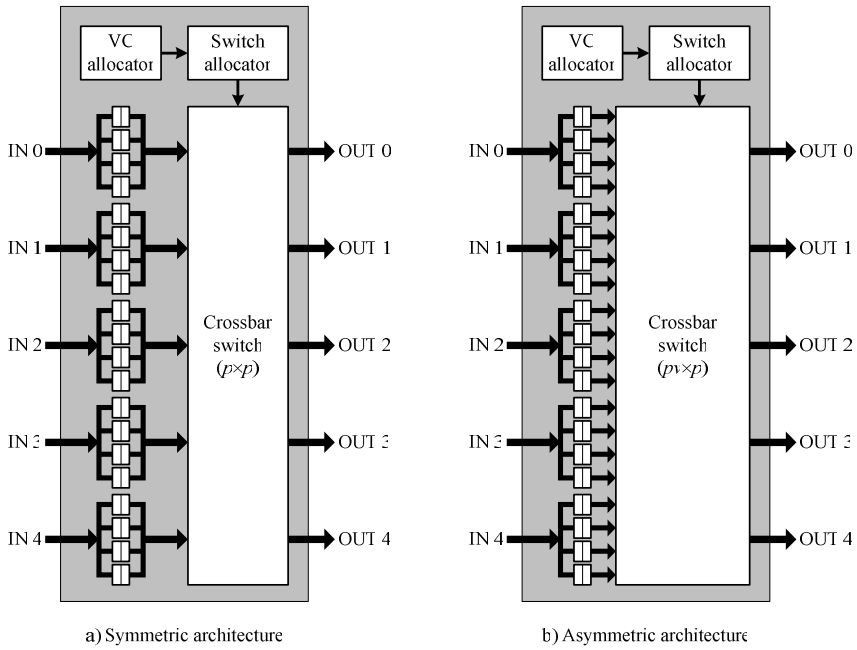


Figure 3.6: Architectures of a virtual channel router; p – number of ports; v – number of VCs per port

Compared to the wormhole router, the virtual channel router increases the number of buffers. However, the total amount of buffer space does not increase, because the buffers can be smaller. A given fixed amount of storage space per input port can be distributed among a number of VCs. For example, we can have a small number of VCs with large buffers or many VCs with a small amount of buffer space per VC; if only one VC is used the virtual channel router turns to a wormhole router. This trade-off was studied by Dally [21]. The results show that it is advantageous to have more VCs with a small buffer space, because in this way the network saturation throughput is increased. In general, there is little performance gained by making the VC buffers deep.

If the amount of buffer space per VC is fixed, the increase of the number of VCs linearly increases the total amount of buffer area per input port. However, the saturation throughput of the network does not increase linearly but logarithmically. Adding a small number of VCs first causes a large increase in throughput, but as more VCs are added the gain diminishes. Dally's results [21] suggest that for uniform traffic four to eight VCs per physical channel is adequate. For example, four VCs per physical channel result in 75% of the throughput of a network with 20 VCs. Further increase of the VCs to eight results in 80% of the throughput of a network with 20 VCs. Since the doubling the VCs doubles the buffer area but improves the throughput only by 5%, in our network we use four VC per physical channel.

Compared to a wormhole router, a virtual channel router has a more complex control. A new control block, called *VC allocator*, is introduced in the router architecture and a new packet processing stage, called *VC allocation*, is performed. The new packet processing stage takes place between the two old stages, the routing stage and the switch allocation stage. After the packet header is examined in the routing stage, the packet goes in the VC allocation stage. In the VC allocation stage, the packet is allocated a VC on the destination output port. Then, the packet goes in the switch allocation stage to be allocated a crossbar connection.

While in a wormhole router crossbar connections are allocated on a packet basis (for the duration of the packet transmission), in the virtual channel router crossbar connections are allocated on a flit basis. After each transmitted flit the packet has to compete again for a crossbar connection. That is because the output ports are time-shared between several VCs. In fact, crossbar connections are allocated not to packets, but to VCs. The switch allocator acts as the arbiter in Figure 3.5 and distributes the bandwidth of the physical channels between the VCs.

The VC allocation is done by the VC allocator upon request from the input controllers. In a virtual channel router every VC has an input controller. All input controllers interface with the switch allocator, as in Figure 3.4, but they also interface with the VC allocator. By allocating a VC to a packet, the VC allocator creates a correspondence between the VC the packet has arrived on and the VC the packet will depart on. This correspondence is stored as a router state for the duration of the packet and is used to control the switch allocator. The switch allocator has to provide connections between corresponding input and output VCs. These connections are provided for a single flit transmission. How often they are provided determines the transmission rate of the VCs, hence their bandwidth.

In a virtual channel router packets are allocated two types of resources – a VC and bandwidth of an output port. The allocation of these resources is done respectively by the VC allocator and the switch allocator. Compared to a wormhole router, a virtual channel router decouples the channel allocation from the bandwidth allocation. This allows the bandwidth distribution to be controlled without affecting the channel distribution policy.

3.4. Resource allocation in a VC router

The Quality-of-Service (QoS) experienced by the network packet is to a greate extend determined by the way in which network resources are allocated to packets. The resources of a virtual channel network are the *virtual channels* and the physical channel *bandwidth* allocated to the VCs. Here we explore the possibilities to make the allocation of these two resources predictable such that the service quality can be guaranteed.

We consider two types of network services – guaranteed services (GS) and best effort (BE) services. The traffic that uses GS is guaranteed service quality for any network condition. The service quality is measured in terms of communication throughput and packet latency and the quality guarantees are given in terms of a worst case bound for the throughput and latency. The quality provided by the BE services is not guaranteed but depends on the current network conditions. However, the BE services should provide fairness, which means that in any network condition all the BE traffic is treated equally.

The quality of the network service is, in fact, the network behaviour seen from the perspective of a network user. Consider a packet injected into the network by a network user. Traversing its path from source to destination, the packet uses network resources, i.e. VCs and bandwidth, provided by the traversed routers. The packet will experience best service if it always and immediately obtains the full capacity of the resources it requires for the time it needs them. However, because the network resources are shared between all the traffic in the network, the packet often has to contend for resources. Hence, the quality of the service the packet receives is determined by the presence of resource contention along the path and by the way these contentions are solved. To make the service predictable we must be able to control or predict these two factors – the presence or absence of contention and the contention solving.

The presence or absence of resource contention is influenced by the traffic distribution in the network and in time. While we do not have control over the traffic distribution in time, which is the data generation process in the data sources, it is possible to control the routes the traffic takes in the network by controlling the traffic routing policies. Thus, traffic routing is an option to control the presence of contention.

How the resource contentions are solved is determined by the resource arbitration policy. This is, whether the packet has to wait in a case of a conflict and, if so, for how long. The resource arbitration is done by the router allocators, which we will examine now in more detail. In a virtual channel router there are two allocators the *VC allocator* and the *switch allocator.*

3.4.1. VC allocation

A route has input ports and output ports and one each port there are VC. We refer to the VCs on the input ports as *input VCs* and to the VCs on the output ports as *output VCs*. Packets enter the router on the input VCs and leave the router on output VCs. The task of the VC allocator is to create a one-to-one mapping between input VCs and output VCs for the purposes of packet traversal disallowing converging VCs.

When a packet enters a virtual channel router, it is first allocated a VC on an output port. The procedure for that is the following. The input controller of the VC where the packet has arrived sends a request to the VC allocator. Then the input controller waits until the allocator grants the request.

The VC allocator receives requests from all input VCs, so it can receive up to *pv* requests at a time. A packet may request any VC on any output port[*]. Hence, an output VC can be requested by more than one packet and the VC allocator must arbitrate between the requests.

[*] To reduce the implementation complexity, most routers exclude the possibility of forwarding a packet in the direction it comes from. In our implementation we also do so. However, to avoid triviality, we do not consider this option when explaining the resource allocation.

To provide service guarantees, we must be able to provide and upper bound on the time a packet waits until it is allocated a VC. One attempt in this direction is to introduce traffic priorities. The GS traffic can be given a higher priority than the BE traffic. Thus, whenever a GS packet competes with BE packets, the GS packet wins and attains the requested VC first. While this scheme improves the service provided to the GS traffic at expense of the service provided to the BE traffic, it does not provide a bound on the VC allocation time for the GS traffic. A GS packet still may compete with other GS packets, in which case the time for VC allocation is difficult to predict. The allocation time depends on the number of competing packets, the number of arbitration cycles before the packet attains the VC and the time the packets occupy the VC.

An upper bound on the VC allocation time can be provided by ensuring that no contention occurs, in which case a packet is immediately allocated a VC. One option to avoid contention is to provide that competing requests from different input VCs arrive at different times. However, we do not have control over the packets arrival times and this option is not feasible for our network. The other option to avoid contention is to provide that an output VC is requested only by packets arriving on the same input VC. Such a condition can be provided by appropriate traffic routing. Since it is possible to control the traffic routing, this is a feasible option for our network.

When contention is avoided in all the routers traversed by the packet, the packet will never find a VC occupied and will never wait. The allocation time per router is known and the upper bound of the total allocation time along the path can be calculated. To avoid contention for all the VCs traversed by a packet we use *VC reservation*. All the packets from source to destination always follow the same path over reserved VC. It is stipulated that the reserved VCs are not used for other communications.

The VC reservation is done at a higher system level by a central routing function that finds routes for all the network traffic. The central routing function is a task running in the system on a general purpose processor which acts as a central system authority.

The paths over VCs are reserved when an application is started. Usually and application will need several paths for communication between its tasks running on different PEs. We can see these paths as *connections* and hence the GS services are connection oriented.

The VC reservation at higher system level does not instantiate the connection, but only reserves VCs for it, providing that these VC will not be used for other communications. The connections are instantiated by the PEs when the application is started. When configured the PE received descriptions of the paths it has to use for communication. Since we use source routing, these descriptions are directly used in the packet headers. To open a connection a PE sends a packet header and to close the connection it sends a tail. Thus, there is certain duality between a packet and a connection when considering the GS services in our network. However, the negotiation for the connection resources is done in advance at a higher system level. The resources are permanently reserved for the connection, no matter whether it is opened or not.

The BE traffic is transported on VCs that carry only BE traffic. This VCs may be shared between BE packets of many source-destination pairs. Hence, the BE packets may encounter contention and their VC allocation time is not bounded. Whether a VC is used for GS or BE traffic is decided centrally when the traffic routing is done.

In our router the VC allocator practically arbitrates only BE packets. The GS packets are never arbitrated but directly allocated output VCs, because conflicts between them never occur. The VC allocator consists of one arbiter per output VC [63], thus *pv* arbiters in total. Each arbiter solves the conflicts between the requests that may

arrive from all the input VCs, thus pv requests at maximum. Hence, each arbiter is of size pv:1 – it has pv request inputs, of which at most one is granted at a time.

3.4.2. Switch allocation

After the packet has been allocated a VC on the destination output port, it moves from the VC allocation stage to the switch allocation stage. In this stage the packet competes for a crossbar connection to the output port. Since the output ports are shared between the VCs, crossbar connections are allocated not for transmission of a whole packet but only for transmission of a single flit. After each forwarded flit, the packet has to compete for a crossbar connection again. Thus, the flits from the different VCs are interleaved on the output port.

The rate at which a VC is granted a crossbar connection to transmit flits determines the VC throughput and hence the throughput utilised by the packet traversing the VC. If flits from separate VCs are interleaved over a physical channel, then the VCs receive an equal share of the physical channel bandwidth. The share is inversely proportional to the number of VCs currently forwarding flits. This number is bounded by the maximum number of VCs on a physical channel. Therefore, the minimum throughput of a VC is also bounded.

Crossbar connections are allocated to the VCs and their packets by the switch allocator. The complexity and performance of the switch allocator depends on the architecture of the virtual channel router. The symmetric architecture from Figure 3.6.a requires a more complex switch allocator which performance is difficult to predict, while the asymmetric architecture from Figure 3.6.b simplifies the allocator and makes it predictable. We now explain the reasons for this dependency.

In the symmetric router architecture from Figure 3.6.a the VCs are multiplexed after the input buffers and the crossbar is of size $p \times p$. When allocating connections in this crossbar we must consider two constraints implied by the architecture: i) an input port can forward at most one flit at a time, ii) an output port can forward at most one flit at a time. To satisfy the first constraint, the allocator needs one arbiter per input port to select one among the possible v input requests. To satisfy the second constraint, the allocator needs one arbiter per output port to select one of the possible v requests to this output port. Remember that at this stage the VC allocation is already done and a one to one correspondence between VCs on the input and output ports has been established.

The architecture of the resulting switch allocator is shown in Figure 3.7. It has two arbitration stages – at the input ports and at the output ports. Each stage has p arbiters of size (v:1). An arbiter has v request inputs and v grant outputs. Whether a request is granted is indicated by activating the corresponding grant output as at most one grant output can be active at a time. Besides the arbiters, the switch allocator contains a fully connected switch controlled by the VC allocator that connects the corresponding request/acknowledge lines of the input and output arbitration stage (this switch serves only request/acknowledge signals and must not be confused with the router crossbar switch that switches the data).

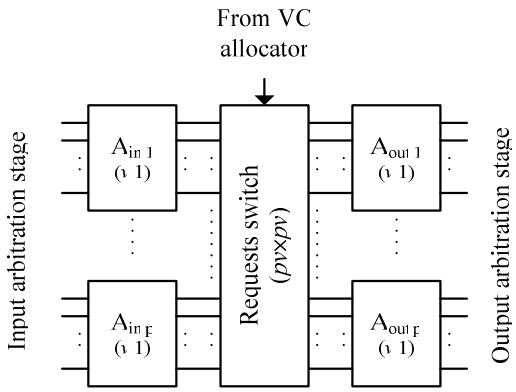


Figure 3.7: Complexity of a switch allocator for the router architecture from Figure 3.6.a

Achieving high throughput and fairness in a switch allocator of this type has been studied by McKeown [54]. He proposes the *i*SLIP allocator, which achieves fair arbitration by using round-robin arbiters and employing a special synchronisation policy between the input and the output arbitration stages. While the proposed allocator prevents packet starvation and achieves high throughput, it does not guarantee equal bandwidth distribution between the VCs. Thus, no prediction can be made about the throughput allocated to a VC. Another switch allocation solution is the *wrapped wave front arbiter* [28, 77]. This type of arbiter also prevents packet starvation, but it does not allocate bandwidth fairly to VCs and no prediction can be made about the VC throughput. We may conclude that the router architecture in Figure 3.6.a leads to a complicated switch allocation which does not allow for achieving predictable bandwidth allocation. Hence, service guarantees cannot be given.

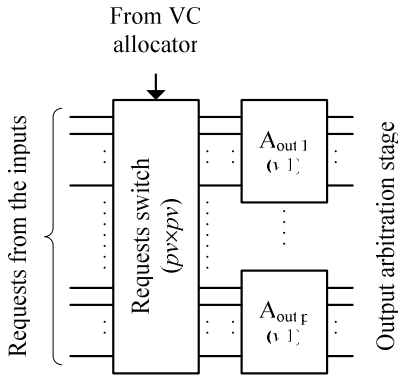


Figure 3.8: Complexity of a switch allocator for the router architecture from Figure 3.6.b

In the asymmetric architecture from Figure 3.6.b, the VCs are connected directly to the crossbar and VCs on a same input port can forward flits simultaneously. Hence, no arbitration on the input ports is needed. Now the only constraint that must be taken into account is that an output port can forward at most one flit at a time. The structure of the switch allocator for this architecture is shown in Figure 3.8. Compared to the allocator

in Figure 3.7, the input stage of arbitration is removed. Now only one arbiter is responsible for the grant of a given request. The decision of an arbiter does not depend on the decision of other arbiters. Each arbiter takes its own decision how to distribute bandwidth between the packets. When round-robin arbiters are used, the packets are served fairly, starvation is prevented and the physical channel bandwidth is equally distributed between the VCs. Knowing the number of VCs that are used on an output port, we can predict what throughput these VCs provide. Therefore, with the router architecture from Figure 3.6.b we can have a predictable switch allocation only by employing round-robin arbiters. This is the architecture we use for our router.

By combining predictable switch allocation and virtual channel reservation, the services provided by a router are made predictable. A network built of such routers will also be able to provide guaranteed services. In the following section we propose a method for providing GS in such network.

3.5. Providing service guarantees at a network level

Consider a virtual channel network with K virtual channels per physical channel. The physical channels are time-shared between the VCs on a flit-by-flit basis in a *round-robin* fashion. Thus, the physical channel bandwidth is equally shared between the VCs. Time slots are allocated only for VCs that have flits ready to be forwarded, the idle VCs do not use timeslots. Therefore, bandwidth is allocated only for VCs that currently transport data.

Let the network be defined as a graph $I=(N,C)$, where the graph vertices represent the network nodes and the edges represent the physical channels. All the physical channels have the same bandwidth b. If on a channel c_i there are k_i VCs currently transmitting data while the remaining K-k_i VCs are idle, then each of the k_i VCs is guaranteed a throughput of:

$$TH_i = \frac{b}{k_i} \tag{3.1}$$

This is the worst case throughput provided by the VCs. Whatever traffic load is applied to these k_i virtual channels their throughput will not fall below TH_i.

At a network level guaranteed services are provided on a connection basis. A *connection* is a path over the VCs between the source and destination nodes. The VCs traversed by the path are reserved and not used for other communications. Let a connection P traverse a sequence of H physical channels $<c_1, c_2,\ldots c_H>$. The minimal throughput of the connection, TH_P, is determined by the VC with the minimal throughput on the connection path.

$$TH_P = \min_{i\in[1,H]}\{TH_i\} \tag{3.2}$$

Assume the network is requested to provide a connection with a minimal throughput TH_R. According to (3.2) this connection must traverse only channels c_i where the throughput TH_i is greater than or equal to TH_R:

$$TH_R \le TH_i \tag{3.3}$$

Substituting (3.1):

(3.4) $$TH_R \leq \frac{b}{k_i}$$

or

(3.5) $$k_i \leq \left\lfloor \frac{b}{TH_R} \right\rfloor = k_R$$

Equation (3.5) provides a selection criterion for the channels traversed by the connection. To guarantee that the minimal connection throughput is at least TH_R, it must be provided that the connection traverses only physical channels where at most k_R VCs are occupied. k_R is a positive integer in the range $1 \leq k_R \leq K$. If in (3.5) $k_R = 0$, this means that the requested throughput exceeds the capacity of the physical channel ($b < TH_R$) so this throughput cannot be provided. The best we can do is to set $k_R=1$ and guaratee throughput b. If in (3.5) $k_R > K$, this means that the requested throughput is less than the granularity at which throughput is provided and we must fix $k_R=K$. The actual minimum throughput bound of the provided connection is:

(3.6) $$TH_P = \frac{b}{k_R}$$

Distributing physical channel bandwidth between the VC by means of round-robin arbitration makes the VC throughput predictable, but also implies that all VCs on the same physical channel guarantee equal throughput. While simple, this approach is not the most efficient. For example, two GS connections, one of high throughput and one of low throughput, with aggregated throughput that is less than the physical channel bandwidth, may not be fit on the same physical channel simply because they can be given only equal throughput guarantees. The bandwidth allocation can be made more flexible by employing *weighted round-robin* arbitration. A weighted round-robin arbiter distributes bandwidth between the VCs proportionally to weights assigned to the VCs. The weights can be chosen such that bandwidth is allocated more accurately to the throughput requests. However, introducing weights will complicate the arbiters and the router organisation.

The latency T of a packet traversing a GS connection is the sum of two components [27]: *head latency* T_h and *serialization latency* T_s. The packet head latency is the time it takes for the packet head to reach the destination, while the serialization latency is the time required for receiving the entire packet body at the destination, after the packet head has reached it. The head latency in our network is the aggregated time the header spends in the routers waiting for VC allocation. Since the VCs used by the GS connection are reserved and not used for other communications, the packets traversing the connection do not compete with other packets and VCs are allocated to them immediately. A GS packet head spends in a router a time t_r, which depends only on the router design and does not include waiting. This time is equal for all routers; it is the time needed to allocate a free output VC and move the head to the next router. The head latency for traversing a connection of H hops is then $T_h = H * t_r$. The serialization latency T_s depends only on the packet length L and the connection throughput TH_P, $T_s = L/TH_P$. The maximal latency of a packet of size L on a H hop connection with minimal throughput bound TH_P is then:

(3.7) $$T_{\max} = H * t_r + \frac{L}{TH_P}$$

3.6. System level support

Our approach for providing guaranteed services relies on VC reservation done at a system level. Guaranteed services are provided by means of GS connections, which are paths reserved over the VCs in the network. To provide a GS connection with a throughput TH_R, the system searches the network for a path over the VCs from source to destination node, such that all VCs satisfy (3.5). Thus, (3.5) is used as a path search criteria. The system reserves the VCs used in the path by changing their state from "free" to "occupied", so these VCs are not considered in the next path searches. When the GS connection is not needed anymore the path and its VCs are released and the VCs state is changed again to "free".

The path search process is referred to as routing and is performed by a routing function. Searching for a path, the routing function needs to know the state of all VCs in the network. Thus, it requires global information about the network state. This information can either be acquired every time it is needed or it can be stored centrally. To avoid traffic and time overhead, the information is stored centrally for our system. The routing function is discussed in more detail in Chapter 5.

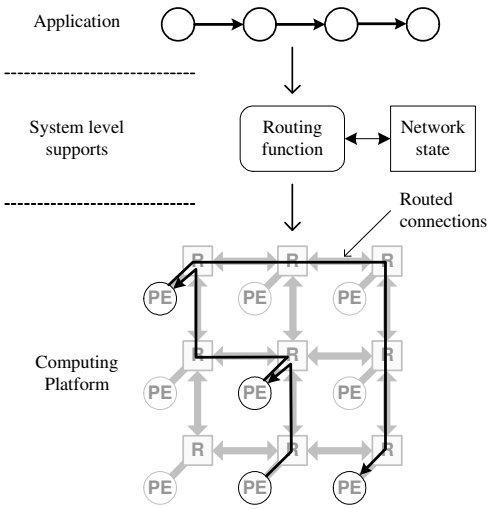


Figure 3.9: Starting an application on the system architecture

The routing function is called every time a connection is created. Since this adds time overhead, it is inefficient or even impossible to provide a new connection every time a GS packet is sent. Instead, connections are provided at a coarser communication level every time an application is started or changed, as depicted in Figure 3.9. The application is represented as a set of tasks that run on different processing elements and communication between the tasks. When the application is started, the routing function is called to provide GS and BE connections for the inter-task communications. These connections are then used for the duration of the application, which for the baseband

processing and multimedia applications considered in our system can be from seconds to hours. When the application is stopped, the connections it uses are released.

The routing function returns a path description in the form of a VC sequence to be traversed. In our network this description is used as a network address. We employ *source routing* which means that the network address defines not only the destination node, but also the path that is taken to reach it. The path descriptions of the connections used by an application are loaded as network addresses in the PEs where the application tasks run. A task uses the addresses given to it to send data to the other application tasks.

3.7. Simulations

To validate VC reservation as an approach for providing service guarantees, we performed a cycle-accurate simulation of a mesh network. The network size is 6-by-6 nodes. It is large enough to provide realistic traffic conditions (interferences between data streams) while it still keeps the simulation time acceptable. The simulation is performed in SystemC and takes a few days. The simulated network consists of routers that model the behaviour of our real router implementation for which the main ideas have been established now. For a detailed description of the implementation see Chapter 4; here we summarise the main parameters only. The router parameters are chosen to provide sufficient performance for the simulated traffic at acceptable router area cost. The network channels are 16 bit wide. A flit is a single word size and is transmitted over a physical channel in a single cycle. There are four VCs on a physical channel. The FIFO buffers are two words deep and can store two flits. The maximum time t_r for processing and forwarding a packet head for the GS packets is 4 clock cycles.

The latency of a GS packet is expressed in terms of clock cycles in the following way. If w is the physical channel width in bits and T_c is the clock period, then the physical channel bandwidth is $b=w/T_c$. From (3.6) we then get that $TH_P=w/(k_R*T_c)$. Substituting in (3.7), the maximum latency of a packet of length L bits on a channel with guaranteed throughput bound TH_P and length H hops is

$$T_{\max} = \left(4H + \frac{L}{w} k_R \right) T_c \tag{3.8}$$

For the simulation we assume a clock period T_c=3 ns corresponding to a router operation frequency of 333 MHz, which is in the operating range of our design (see Chapter 4). The physical channel bandwidth is then b=5.333 Gbit/s.

3.7.1. Setup

We simulate a mesh network of 6-by-6 nodes with a traffic load that mimics the traffic generated by streaming applications. Streaming applications typically have a simple pipeline structure – a number of tasks connected in a pipeline by communication channels. To construct a traffic pattern that models the pattern in a running system we use a ring graph, which can be thought of as a representation of serially connected streaming applications. The graph vertices represent application tasks and the graph edges represent the communication channels between the tasks. The number of tasks in the ring graph is 36, the same as the number of nodes in the simulated network. We randomly scatter the tasks over the network nodes, as each task is placed on a separate

node. The edges in the so mapped ring graph define the communication pattern used in the simulation. We consider random scattering to be the worst case strategy for running tasks on a multiprocessor system. In a real system the application tasks are placed on PEs such that the traffic locality is maximized, which leads to better traffic conditions.

The network nodes, where the tasks are mapped, do not do processing but only serve as source and sink of data. Each node generates both types of traffic, GS and BE, according to the constructed communication pattern. Thus, between two communicating nodes there are two communication channels, one GS channel and one BE channel. While the ring pattern is a realistic model for GS traffic, probably it is not the most realistic for BE traffic. Currently we do not know what traffic pattern is realistic for the BE traffic because it depends very much on the application specifics, the type and the placement of the PE in the system, etc. However, the purpose of the BE traffic in this simulation is to create heavy traffic conditions and to disturb the GS traffic. We choose to let the BE traffic follow the GS traffic pattern, so that each GS connection is accompanied by a BE connection, both following the same path. Thus, the GS and the BE traffic is concentrated on the same paths, maximizing the interference between the two traffic types.

During the simulation the intensity of the GS traffic is kept constant while the BE traffic intensity is gradually increased to the point of network saturation. The aim is to show that the guarantees given by the GS connections are not violated by any traffic condition. The traffic conditions are changed by changing the BE traffic intensity. During the simulation statistics are collected for the packet latencies, both GS and BE.

All network nodes generate traffic of same amount and granularity. For the GS traffic we use the traffic characteristics of a real high-throughput baseband processing application – a HiperLAN/2 receiver [67]. Every 4 μs a node generates a new packet with a 256 Bytes payload. This equals 512 Mbit/s average throughput per node or 18.4 Gbit/s aggregated throughput for the 36 nodes in the system. The BE traffic consists of packets with 10 Bytes payload. The BE packet generation period is gradually reduced in order to increase the BE traffic intensity.

In our simulation setup we guarantee that GS packet latency is less than 1/3 of the GS packets generation period. In that way a PE spends at most 1/3 of the time receiving packets, at most 1/3 of the time transmitting packets and uses the remaining time for processing. To guarantee that latency we provide GS connections with appropriate throughput. We do that by routing the communication channels according to the criteria discussed in Section 3.5.

To achieve a GS packet latency of 1/3 of the generation period (4μs/3=1.3 μs), the GS connections must have a throughput of at least 256B/(4μs/3)=1.536 Gbit/s. This is the throughput TH_R we request for the GS connections. It is found from (3.5) that k_R=3, hence a GS must traverse only physical channels where at most 3 VCs are used. The maximal message distance in a 6-by-6 mesh network is H_{max}=10 hops. According to (3.8) the maximal latency of a GS packet is 424 clock cycles or 1.272 μs, which is less than the required 1.3 μs. To provide the requested throughput, we route the GS and BE connections such that no more than three VCs are used on the physical channels (k_R=3).

3.7.2. Results

The simulation results are presented in Figure 3.10. The graph there illustrates how the latency of the GS and BE packets depends on the offered BE load. The offered BE load is given per PE, all PEs generate the same amount of data. The packet latency is

given over all connections in the network. For the GS packets we give the maximal and the mean latency. The horizontal line represents the 424 cycle latency guarantee for the GS packets.

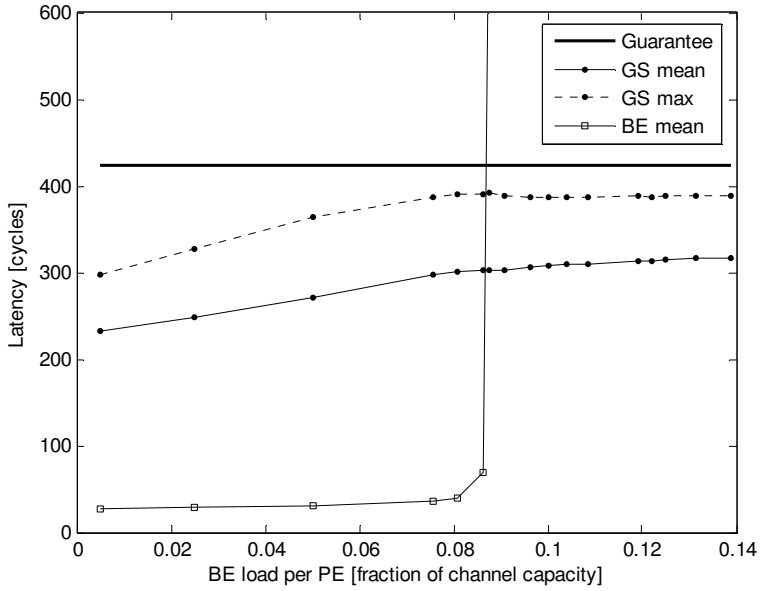


Figure 3.10: Packet latency vs. offered BE load; buffer size of 2 flits

When the offered BE load is low, the latency of the GS packets is smaller than the guaranteed. The reason is that the GS traffic utilizes the bandwidth not used by the BE traffic. The latency of the GS packets is higher than the latency of the BE packets because the GS packets (256B) are longer than the BE packets (10B). The GS packets latency is dominated by the serialization latency. With the increase of the BE load, the latency of the GS packets also increases and at some point the GS packets maximal latency reaches a point of saturation. A further increase of the BE load increases the GS packets mean latency, but not the maximal latency. The latency of a GS packet never exceeds the guaranteed latency. The maximal latency of the GS packets never reaches the latency bound because the bound is given for worst case conditions, assuming all the k_R VCs constantly transmit data. In our setup this is not the case – GS packets are transmitted 1/3 of the time.

The network saturates for BE traffic when the offered BE load reaches about 0.09 of the channel capacity. This throughput is rather low, but it is enough to handle the amount of BE traffic generated in our system. The traffic generated by the HiperLAN/2 receiver, and by baseband processing applications in general, is dominated by the GS traffic. About 90% of the generated traffic consists of GS streams and only 10% consists of fine granularity BE messages. For the HiperLAN/2 receiver, this can be estimated to an average of 512 Mbit/s GS traffic and 57 Mbit/s BE traffic per PE. Thus, the expected BE traffic is only 0.01 of the channel capacity b=5.333 Gbit/s or less then 15% of the BE saturation throughput in Figure 3.10. This means that the network will

operate in the left most part of the graph and the low saturation throughput for the BE traffic does not cause problems for the considered applications.

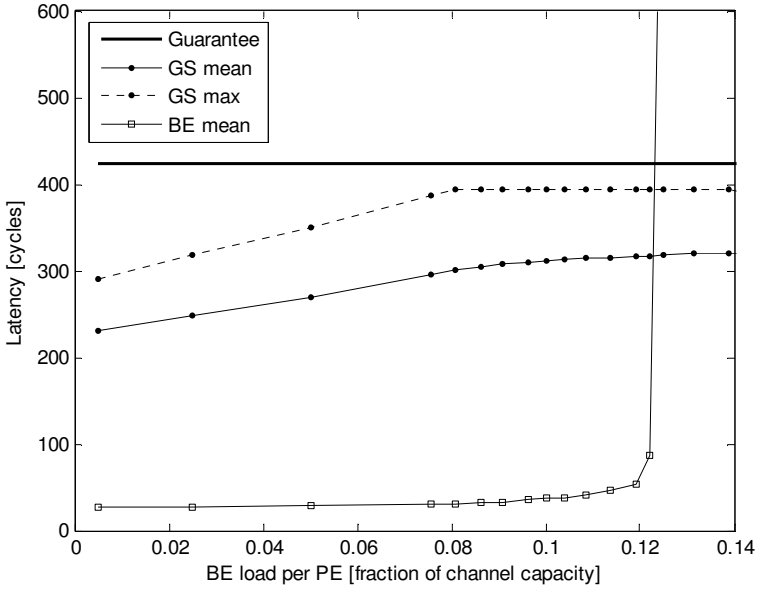


Figure 3.11: Packet latency vs. offered BE load; buffer size of 4 flits

The same simulation was repeated for a network with a buffer size of 4 flits. The results are shown in Figure 3.11. It shows that enlarging the buffers mainly results in an increase of the BE saturation throughput, from 0.09 to 0.12.

3.8. Comparison

Table 3.1 and Table 3.2 compare our virtual channel network-on-chip (VCNoC) with the NoCs reviewed in Chapter 2 which provide GS and BE services. Table 3.1 and Table 3.2 are copies of Table 2.2 and Table 2.3 extended with a row describing our VCNoC. The performance figures will be discussed in Chapter 4.

Table 3.1 compares the techniques employed by the networks. The MANGO network is the solution which is closest to our VCNoC. It employs virtual channel flow control and uses similar approach for providing service guarantees. The main implementation difference and certain advantage for MANGO is that it is implemented using asynchronous techniques, in contrast to our synchronous implementation. Thus MANGO shows that it is possible to design a virtual channel network using asynchronous design techniques.

The main functional difference between MANGO and VCNoC is in the way in GS connections are configured. In MANGO, the configuration of a GS connection is done by sending explicit configuration BE packets to all the routers along the GS connection. The same approach is used by the network of Wolkotte. Its main drawback is that the connection configuration time cannot be guaranteed because BE packets are used. By contrast, in our network a connection is configured by the source node simply by

injecting a packet header in the network. The header follows a preliminary reserved path to the connection destination node and the time it takes to reach it is bounded and guaranteed.

Table 3.1: Comparison of employed network techniques;
NA=Not Applicable, VCNoC=Virtual Channel NoC

NoC	Provided services	Topology	Flow control	Routing	Area [mm2]	F [MHz]	Tech. [mm]
Wolkotte	GS, BE	2-D mesh, Ring	Circuit switching, cut-through (serial)	NA	0.05	1000	0.13
Æthereal Distrib.	GS, BE	Any	TDM, Wormhole	contention-free routing, source routing	0.24	500	0.13
Æthereal Centr.	GS, BE	Any	TDM, Wormhole	contention-free routing, source routing	0.17	500	0.13
Æthereal GS only	GS	Any	TDM	contention-free routing	0.03	1000	0.13
Nostrum	GS, BE	2-D mesh	TDM	contention-free routing, diflection	--	--	--
MANGO	GS, BE	grid-type	Virtual channels	source routing	0.19	515*	0.12

In Nostrum GS connections are also opened by the source node, but guarantees can be claimed only during the network initialization when the system is started. In our network GS connections can be opened and closed at run-time.

In Æthereal, like our in network, the configuration is done by GS packets and hence the configuration time can be guaranteed.

Table 3.2: Comparison of network characteristics;
VCNoC=Virtual Channel NoC

NoC	GS approach	Applicable in GALS systems	GS connection is configured by	GS packetisation required	Combining GS and BE
Wolkotte	Circuit reservation	yes	BE packets	yes	Separate networks
Æthereal	Time-slot reservation	no	GS packets	yes	Separate router parts
Nostrum	Time-slot reservation	no	The source node	yes	Single router
MANGO	VC reservation	yes	BE packets	no	Separate router parts
VCNoC	VC reservation	yes	The source node	No	Single router

Besides the functional differences between MANGO and our VCNoC there are also architectural differences. In the MANGO network the router consists of two separate parts, one that processes the BE traffic and one that processes the GS traffic. A similar router structure is used in the Æthereal network. With such a router organization the GS and BE traffic use separate router resources, thus the total amount of needed resources increases. For example, in the MANGO router the number of VCs is doubled and divided in two sets, one set used for GS and one for BE traffic. The division of

resources also requires that a choice is made at design time about the amount of resources dedicated to both traffic types.

Our network router does not have separate parts for the different traffic classes. Both types of traffic are processed by a single routing solution and use common resources. Thus the total amount of resources is minimised. Whether a resource is used by GS or BE traffic is decided dynamically at run-time. There is no difference between the GS and BE traffic at router implementation level. Hence, no choices based on traffic specifics have to be made at design time. The amount of resources allocated to different traffic types is decided at run-time and can be dynamically adapted to the application demands.

3.9. Conclusion

In this chapter we present a solution for a virtual channel network-on-chip (VCNoC) able to support two traffic classes - guaranteed service (GS) and best effort (BE) traffic. We propose a router architecture and a resource reservation scheme, which in combination enable our network to provide GS as well as best effort services. Our network is one of the first VCNoC providing GS services and one of the few to provide an upper bound on the time for opening a GS connection. The router architecture we propose is unique with that it supports GS and BE traffic in an integral way instead of combining two separate architectural solutions.

Our VCNoC is applicable in systems where the GS traffic is dominated by intensive streams and the BE traffic is composed mainly of low intensity and fine granularity messages. This is the type of traffic expected in multi-processor system-on-chip for streaming applications (communication and multimedia applications). Compared to other networks, our network implies fewer restrictions on the traffic injected in it. Data streams do not need to be split into packets and later reassembled again, but are directly transported.

The resource reservation scheme we propose for providing GS services requires a central system organisation. This is the system organisation required also by the other proposed NoC solutions.

Chapter 4

Implementation*

In this chapter we discuss design and implementation issues of an asymmetric virtual channel router architecture. We propose an implementation that makes the implementation area competitive with the area and performance of the symmetric architecture.

4.1. Introduction

In Chapter 3 we choose an asymmetric architecture for a virtual channel router instead of the symmetric one, because the asymmetric architecture simplifies the router arbitration and makes the router performance predictable. However, the asymmetric architecture requires a larger crossbar switch and for that reason it is usually regarded as an area inefficient router solution. Since the network-on-chip (NoC) implementation is area constrained, it is important to clarify what the implementation area cost of a router is and to select the router parameters such that the required network performance is provided at a minimal area cost.

This chapter deals with the implementation of our network router. We simplify and optimise the implementation of the asymmetric router architecture to make it more area efficient. Firstly, we restructure the router implementation in order to avoid overlapping functionality and to reduce the router size. Secondly, we propose a simplified implementation for the VC allocator. The synthesis results reveal that the area and performance of the asymmetric architecture are competitive with those of the symmetric architecture (also implemented by us).

We also explore the scalability of the proposed architecture by comparing synthesis results for different combinations of route parameters, e.g. buffer size, number of VCs, and channel width. We discuss how the router performance and implementation area depends on the different parameters.

4.2. Implementation details

In this section we present implementation details of our virtual channel NoC (VCNoC). We discus the implementation of the asymmetric router architecture and propose an optimised implementation that minimises the area of a virtual channel router.

* Major parts of this chapter have been presented at the IEEE International System-on-Chip Conference [5] and at the EUROMICRO Symposium on Digital System Design [1].

4.2.1. Flit and packet format

The basic unit of information recognized by virtual channel flow control is the *flit*. Flits in our network are a single data word. The flit format is shown in Figure 4.1. It consists of *w* bits of transported data and a two-bit field coding the flit type. The flit types are: *Header flit* (*HF*), *Data flit* (*DF*), *Command flit* (*CF*) and *Tail flit* (*TF*). The header and the tail flits denote respectively the beginning and the end of a packet, while the data and control flits carry the packet payload. The command flits utilize the fourth state of the two-bit field for flit type. The network does not distinguish between data and control flits; it treats both as flits that transport payload. In this way the network can transport two types of data, for example data and commands, together in a packet practically at no additional cost. From our experience, we find this network feature useful and convenient to support the higher levels of the system organization.

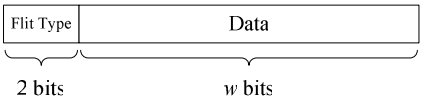


Figure 4.1: Flit format

Our network transmits a flit on a physical channel in a single clock cycle. As discussed in Chapter 3, making the flit longer is advantageous when the router operation rate is slower than the channel operation rate and high channel utilization is a primary goal. For networks of such a small scale like the on-chip networks this is not the case. Since the router implementation is fast and the network channels are inexpensive, high channel utilization is not a primary goal.

Data is transported over the network in packets constructed of flits. A packet consists of three parts: header, body and a tail. The packet header carries routing information. The packet body carries the transported data, or the payload. The packet tail just indicates the end of the packet. The packet is constructed of a sequence of flits of different type. The permitted flit sequence constructing a packet is:

```
Packet ::= Header Body Tail
Header ::= HF+
Body ::= (DF|CF|HF)*
Tail ::= TF
```

The packet header typically consists of multiple header flits. The number of header flits equals the number of routers traversed from source to destination. The header flits configure he traversed routers. In each router the first header flit of a packet is examined, routing information is extracted from it and the flit is discarded. The routing information determines on which output VC the packet is forwarded. Thus, the header flit adds one more VC to the packet path. The routing information must lead the packet to its destination. The flits arriving in the router after the first header flit, just follow the path. The packet body may contain any flit type except a tail flit. The tail flit terminates the packet and releases the reserved path. The packet body may contain any number of flits, including no flits at all, in which case no payload is transported.

4.2.2. Channel interface

The physical channels are the medium on which flits are transferred between the routers. The channel interface between two routers is presented in Figure 4.2. Let K be the number of VCs on a physical channel and w be the data width. The flits are transported on a unidirectional channel of a size 2+w bits which we call the *Flit Bus*. On which VC the current flit is being transmitted is indicated by the signal *VC_sel* of size $\lceil \log_2 K \rceil$ bits. The signal *Valid* indicates whether a flit is transmitted in the current clock cycle. The signal *Ready* goes in reverse direction. It returns the flow control credits from the receiving router to the transmitting router. The credits are carried in parallel. The signal consists of K bits, each bit corresponding to a VC. Each bit indicates whether there is free space in the receiving FIFO buffer. By sending the credits in parallel instead of coding and sending them sequentially on a narrower channel, we simplify the design avoiding the coding and decoding logic. We speed up the credit regeneration process, at the cost of utilizing the larger amount of the available wires. The total amount of wires needed to build the channel is:

$$num\_wires = K + \lceil \log_2 K \rceil + w + 2 + 1 \tag{4.1}$$

For example, with four VCs (K=4) and 16 bit data width, the number of wires needed to build the interface is 25.

The presented channel interface is parallel and unidirectional. There are two such channels between two neighbouring routers – one channel in each direction – to provide full-duplex communication.

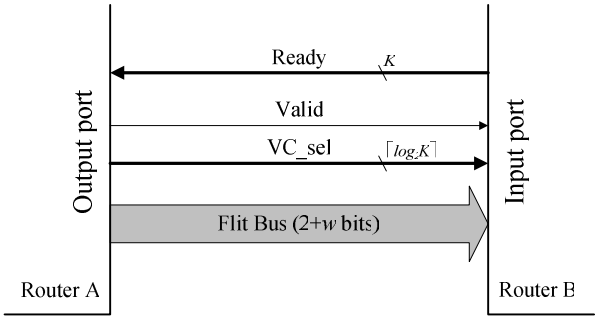


Figure 4.2: Channel interface between two routers (K = number of VCs, w = data width)

An example timing diagram of the channel interface is given in Figure 4.3. It shows a number of flits sequentially transmitted over the channel. It is assumed that there are four VCs per physical channel. In cycle i, *Flit1* is transmitted on VC 0. In that cycle the signal *Ready* indicates that there is free space in all VC FIFOs. Since enabled by the *Valid* signal, the flit is written in the FIFO of VC 0 on the rising edge at the end of the cycle. In the next cycle, i+1, another flit is transmitted on VC 0. In cycle i+2, the *Ready* signal indicates that there is no free buffer space in the FIFO of VC 0. The *Valid* signal is deactivated and no flit is transmitted in this clock cycle. In cycle i+3 the buffer space is available again and *Flit 3* is transmitted on VC 0. In cycle i+4 the buffer of VC 0 is full again, but a flit is transmitted on another VC for which buffer space is available, the VC 3.

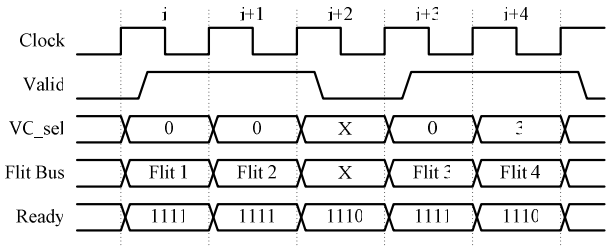


Figure 4.3: An example timing diagram of the channel interface

4.2.3. Input controller

For every VC at the router inputs there is a FIFO buffer and control logic –together they form an input controller (see Figure 4.4). There is an input controller for every VC at every input port of the router. When a flit arrives on an input port, it is demultiplexed to the respective VC and processed by its input controller. The input controller buffers the incoming flits and examines them. It stores the current VC state and interfaces with the VC allocator and the switch allocator to requests the resources required for forwarding the flits – a VC and a switch connection.

The functionality of our input controller is closely coupled to the functionality of the VC allocator and the switch allocator. To understand the complete role the input controller plays in the router operation, the router VC and switch allocation needs to be known. For that reason here we only introduce the architecture of the input controller and part of its functionality, while the full functionality will become clear later when the allocation is presented.

The architecture of our input controller is presented in Figure 4.4. The controller consists of a FIFO buffer, control logic *Ctrl*, two registers *ID* and *dest* and a comparator *cmp*. The control logic implements the main functionality of the input controller and controls the VC state. The two registers and the comparator take a part in the VC allocation and the switch allocation; their function will be explained later in Section 4.2.4.

Upon its arrival a flit is stored in the FIFO. The FIFO sends back information to the previous router whether there is free buffer space available; this is the signal *Ready[i]*, which directly drives one bit of the signal *Ready* in the channel interface. If a flit is received when there is no free buffer space, the flit is dropped. However, during normal operation the flow control does not allow flits to be sent when there is no free buffer space in the receiving router.

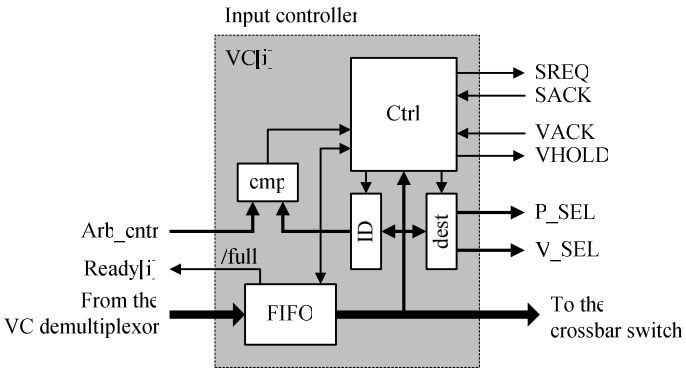


Figure 4.4: Architecture of an input controller

The head of the FIFO (its output) is constantly monitored by the control logic *Ctrl*. The control logic consists of a Finite State Machine (FSM) and some glue logic. The state diagram of the FSM is shown in Figure 4.5. Its states correspond to the possible VC states that basically repeat the stages of packet processing. After reset the VC is in an *Idle* state, which indicates that no packet is using the VC and the VC is free. In that state the FSM monitors the type of the flits at the FIFO head, waiting for a header flit to denote the beginning of a packet. All the flits of type different from *HF*, i.e., *DF*,*CF*,*TF*, are discarded from the network. The discarded flits are read out of the FIFO without being forwarded. The discarding prevents the VC from blocking when incorrect packet formats are received. When the first header flit appears at the FIFO head, the FSM moves to the next state *VC allocation* (VC alloc). At the same time the data carried by the header flit is loaded in the *ID* and the *dest* registers and the header flit is discarded.

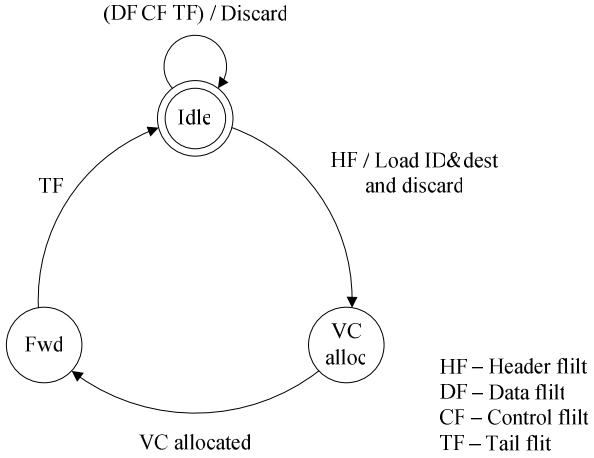


Figure 4.5: A virtual channel state diagram

The format of a header flit is shown in Figure 4.6. Its data part contains three fields. The fields *p_sel* and *v_sel* contain respectively the number of the output port and the VC on which the packet has to be forwarded. These fields are loaded into the destination register *dest*. They appear directly on the output signals *P_SEL* and *V_SEL* showing the allocators where the packet has to go. The size of the field *v_sel* and the

signal *V_SEL* is $\lceil \log_2 K \rceil$ bits. The size of the field *p_sel* and the signal *P_SEL* is $\lceil \log_2 P \rceil$ bits, where *P* is the number of router output ports. For our router with *K*=4 and *P*=5 the size of the fields *v_sel* and *p_sel* is respectively 2 and 3 bits. The field *ID* contains an identifier, which is loaded in the *ID* register of the input controller. The identifier takes part in the VC allocation and its exact function is explained in Section 4.2.4.

Figure 4.6: Header flit format

While it is in *VC allocation* state, the input controller neither forwards nor discards flits from the FIFO. Instead, it waits until the VC allocator allocates the virtual channel *v_sel* on output port *p_sel*. The input controller interfaces with the VC allocator through the signals *VACK* and *VHOLD*. The signal *VACK* notifies the input controller that the requested VC (selected by *P_SEL* and *V_SEL*) is currently free. The signal *VHOLD* is activated to indicate that the input controller currently holds the requested VC. The two signals together with the comparator *cmp* and the register ID take part in the VC allocation. Details about the VC allocation are given in the following subsection.

When the VC is allocated, the FSM moves to the next state *Forwarding* (Fwd). In the *Forwarding* state, the input controller forwards all the flits arriving in the FIFO to the VC selected by the content of the destination register *dest* (virtual channel *v_sel* on port *p_sel*). The controller interacts with the switch allocator through the signals *SREQ* and *SACK*. The request signal *SREQ* indicates that there is a flit in the FIFO to be forwarded. The switch allocator responds with the acknowledge signal *SACK* indicating that a crossbar connection is allocated and the flit can be forwarded in the current clock cycle. The acknowledge signal is issued only when there is free buffer space in the next router.

In the *Forwarding* state the control logic monitors the type of the flits being forwarded, waiting for a tail flit. When a tail flit is forwarded, the FSM moves to its *Idle* state waiting for the next packet.

4.2.4. VC allocator

Figure 4.7 presents a straightforward implementation [63] of the VC allocator as discussed in Section 3.4.1. The allocator operates as follows. When a packet arrives in an input controller *i* the controller activates its signal *REQ[i]* to request a VC on an output port. Which VCs is requested is indicated by the signal *SEL[i]*. The VC allocator receives the request and demultipexes it to the requested output VC. At each output VC there is an arbiter A_j that arbitrates between the possible multiple requests (up to *pv*). Hence, as shown the allocator contains *pv* demultiplexers of size 1:*pv* and of pv arbiters of size *pv*:1 (one of *pv*). However, Figure 4.7 does not show all the details of the VC allocator. In addition to what is shown, an acknowledge signal must be returned (multiplexed back) for each request and the arbiters must consider the current state of the output VCs – whether they are occupied of free. Hence the VC allocator is even more complex than what is shown in Figure 4.7.

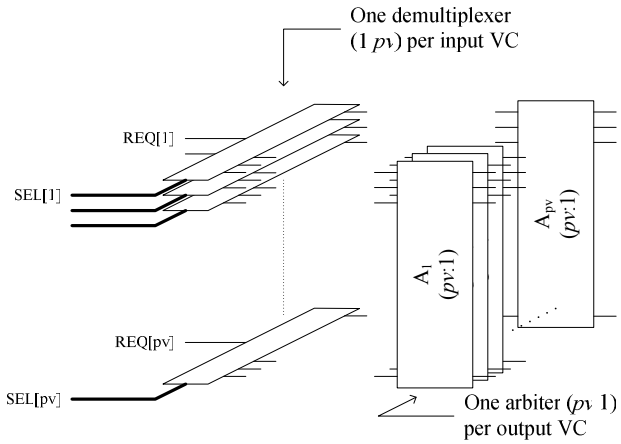


Figure 4.7: A straightforward implementation of a VC allocator

Since designed for the worst case, the arbiters are rather large – they arbitrate over the maximum of pv requests. All the arbiters together operate on $(pv)^2$ request signals in total. Out of these signals, at most pv can be active at the same time. For example, in a router with 5 ports (p=5) and 4 VC (v=4) only 20 request signals out of the total 400 can be active simultaneously. Therefore, the implementation shown in Figure 4.7 contains redundancy.

We propose an alternative implementation that reduces the redundancy and respectively the complexity of the allocator. In contrast with the straightforward implementation, we avoid the large arbiters at the output VCs. Instead, the arbitration takes place in the input controllers. Each input controller contains a comparator and a register named *ID* (see Figure 4.4). These are the additional components needed to perform the arbitration. When a new packet arrives, the *ID* register is loaded, as described in section 4.2.3, with an identifier. We guarantee at a higher system level that the packets competing for the same output VC have different identifiers. Which VC is requested is indicated by the signals *P_SEL* and *V_SEL* of the input controller. The signal *VACK* to the input controller indicates whether the requested VC is free. In the router there is a central counter that runs constantly. Its current value is supplied on input *Arb_cntr* of all input controllers. When *VASK* signals that the requested VC is free, the input controller compares the counter value with the ID identifier. If the comparison is successful, the controller assumes that it wins the arbitration and that the output VC is allocated to it. The controller indicates this decision by activation the signal *VHOLD* after which it proceeds with the packet forwarding. The signal *VHOLD* stays active until the whole packet is forwarded and the output VC is released.

The uniqueness of the identifiers guarantees that only one of the controllers competing for the same VC assumes that it wins the arbitration and activates its *VHOLD* signal. Hence, conflicts are avoided. The counter is sampled when the requested VC becomes free and in general the counter value is non-deterministic at that time. Hence, no preferences are given to any of the competitors and the arbitration is fair. It is possible, although with very small probability, that a packet has a bad luck and does not win the arbitration for a long time. Such situation is acceptable for the BE traffic since no guarantees are given. For the GS traffic such situation never occurs because no VCs are shared and arbitration is never needed.

It may happen that at the moment when the requested VC becomes free the counter value does not equal any of the packet IDs. In that case no VC is allocated at the first clock cycle, but since the counter value increments constantly with the clock, in several clock cycles it will reach some of the ID values and the VC will be allocated. If the counter is n bits long, the VC will be allocated in at most 2^n clock cycles. To provide that all ID values will be reached by the counter, the ID size must be less than or equal to the size of the counter. Because of the requirement for uniqueness, the size of the ID determines the maximal number of packets that may be competing for the same VC. Thus, the size of the ID is a trade-of between the maximal number of BE paths that can share the same VC and the maximal time for VC allocation (excluding waiting). Our experience shows that a 2 or 3 bit identifier satisfies the needs for VC sharing. When the identifier is two bits long, each input controller is equipped with a two bit ID register and a two bit comparator to perform the arbitration. Their complexity is less than the complexity of the arbiters in the straightforward design, discussed earlier. The worst case time for VC allocation is 4 cycles.

The externally loaded unique identifiers can be avoided by numbering all the input controllers and using the controller number instead of identifiers. However this will increase the worst case time for VC allocation to pv clock cycles.

To complete the VC allocator design, it remains to show how the signals *VACK* to the input controllers are generated. The signal *VACK* returned to an input controller indicates whether the VC selected by the controller signals *P_SEL* and *V_SEL* is currently free. The VC is free when none of the requesting input controllers has an active *VHOLD* signal.

The signals *VHOLD* from all input controllers are demultiplexed to the corresponding output VCs selected by the signals *P_SEL* and _SEL, in the same way as in Figure 4.7 the signals *REQ[i]* are demultiplexed to the VCs selected by the signals *SEL[i]*. At each output VC, instead of an arbiter there is an OR element that logically OR's the *VHOLD* signals. These are the *VHOLD* signals of all input controllers selecting that particular VC. If none of these signals is active, then the VC is free. The inverted output of the OR element is then the VACK signal. It is multiplexed back to the input controllers requesting the VC.

In the input controllers, arbitration takes place only when the *VACK* signal is activated, showing the requested VC is free. This happens only when all input controllers requesting the same output VC have their *VHOLD* signals deactivated. Since in a clock cycle only one of these controllers can win the arbitration, only one of their *VHOLD* signals can become active in a clock cycle. Activating one of the *VHOLD* signals deactivates the *VACK* signals and ends the arbitration until the output VC is released again and the *VHOLD* signal is deactivated. Therefore, at most one input controller may attain the VC at a time and conflict free arbitration is guaranteed.

4.2.5. Switch allocator

In Chapter 3 we chose the asymmetric router architecture which for ease of reference we show again in Figure 4.8. The architecture consists of three main blocks a VC, allocator, a switch allocator and a crossbar switch. (Besides that each input VC has an input controller, but for picture clarity we do not show them.). All the three blocks perform massive signal switching. As we saw in the previous section, the VC allocator contains a switch of size $(pv)^2$ for all the signals *VACK* and *VHOLD*. The switch allocator (see Figure 3.8) also contains a switch of size $(pv)^2$ for the signals *SREQ* and

SACK. The crossbars switch is of size p^2v and switches data from input to output ports. All this switching is mainly controlled by the signals *P_SEL* and *V_SEL* from the input controllers selecting where the packets are forwarded. Hence, we can expect a repeated functionality in the switching circuits in the three blocks.

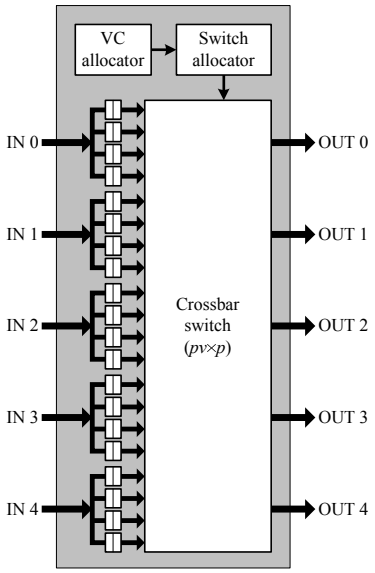


Figure 4.8: Selected router architecture

The three block are also closely functionally connected as they hierarchically control each other as seen in Figure 4.8.. The VC allocator controls the switch allocator, enabling only packets with already allocated VCs to be served. The switch allocator controls the crossbar switch, determining only conflict free configurations.

Because of the repeated functionality in the three blocks and because of the tight interface between them, it may be expected that if implemented as separate blocks like in Figure 4.8, the design will become inefficient in terms of area. Instead, we propose to integrate the large switches in the VC allocator, the switch allocator and the crossbar switch into a single switching structure. Thus we avoid the repeated functionality and the complex interfaces between the blocks.

The router structure we implement is shown in Figure 4.9. The figure shows only the control paths, while the data paths are omitted for clarity. The switching parts of the VC allocator, the switch allocator and the crossbar switch are merged in a single switching unit. The round-robin arbiters (RRA) of the switch allocator are shown separately outside the switch – one arbiter per output port. The switching unit has a crossbar structure. The rows in the crossbar correspond to input VCs and are connected to the input controllers. The columns in the crossbar correspond to output ports and are connected to the RRAs and the output ports of the router. Thus, the crossbar has pv rows and p columns.

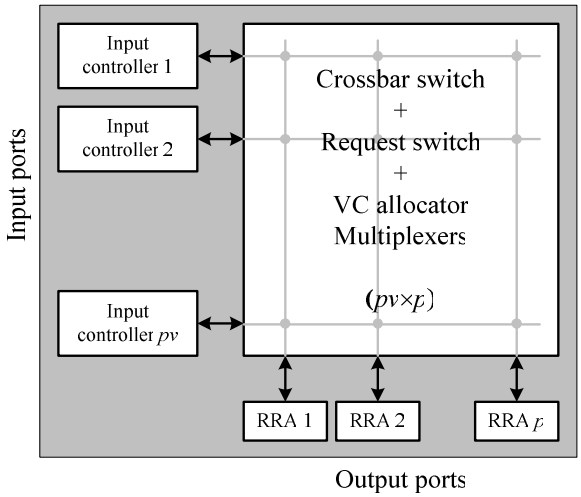


Figure 4.9: Implemented router structure. RRA – round-robin arbiter

Each cross point in the crossbar connects an input controller to an output port. The cross points, however, are not simple switching elements but contain also some control logic. Let us assume that there are four VCs per port. The structure of the cross point then is shown in Figure 4.10 and Figure 4.11. The two figures present the part of cross point functionality supporting the switch allocator and the VC allocator respectively. In the crossbar they are implemented together, but for clarity we present and discuss them separately.

The cross point functionality related to the switch allocator is shown in Figure 4.10. The signals from an input controller (see Figure 4.4) are connected to all cross points in the corresponding row. In each cross point a comparator *cmp1* compares the port select signal *P_SEL* with a constant *P* representing the crossbar column number. Thus, the comparator recognizes whether the input controller wants to forward data to that output port. If the comparison is successful, the cross point is activated. For a five-port router, the *cmp1* is simply a three-input gate.

In the active cross point, a demultiplexer *dmx* switches the request signal *SREQ* to one of four request lines running along the column. Each request line corresponds to an output VC and collects the switch allocation requests *SREQ* to that VC from all cross points in the column. The request lines are constructed as OR-chains running along the column. To which request line the signal *SREQ* is switched is determined by the signal *V_SEL*. Since only one input controller may use an output VC at a time, a request line may collect at most one *SREQ* signal.

The column request lines generate the signals *sreq0* to *sreq3* which show whether there is any request to the corresponding output VC. The signals *sreq0* to *sreq3* are processed by the round-robin arbiter (*RRA*) of the corresponding output port. The RRA is also connected to the output port and monitors the *Ready* signals from the next router. The *Ready* signals indicate whether buffer space is available in the next router. Thus, the arbiter considers only the request for VC with available free buffer space in the next router. Every clock cycle the arbiter grants one request, if any. The granted VC number is indicated by the signal *vc_sel* which is sent back along the crossbar column. The same signal drives the signal *VC_sel* at the router output port (see Figure 4.2) indication on

which VC flit is being sent. The arbiter also drives the signal *Valid* at the output port indicating that a request is granted and a flit is forwarded in the current clock cycle.

In the active cross points along the crossbar column, a comparator *cmp2* monitors whether the number of the VC currently granted by the arbiter (*vc_sel*) equals the VC requested by the cross point (*V_SEL*). When the comparison succeeds, the cross point connects the input data bus (*Flit*) to the output port bus and a flit is forwarded. An acknowledgement signal is sent to the input controller on the acknowledge line *SACK* running along the crossbar row. The acknowledgement line is constructed as an OR-chain that collects the acknowledgement signals from all cross points in the row. Since only one cross point can be active in a row, at most one acknowledgement signal is sent on the acknowledgement line.

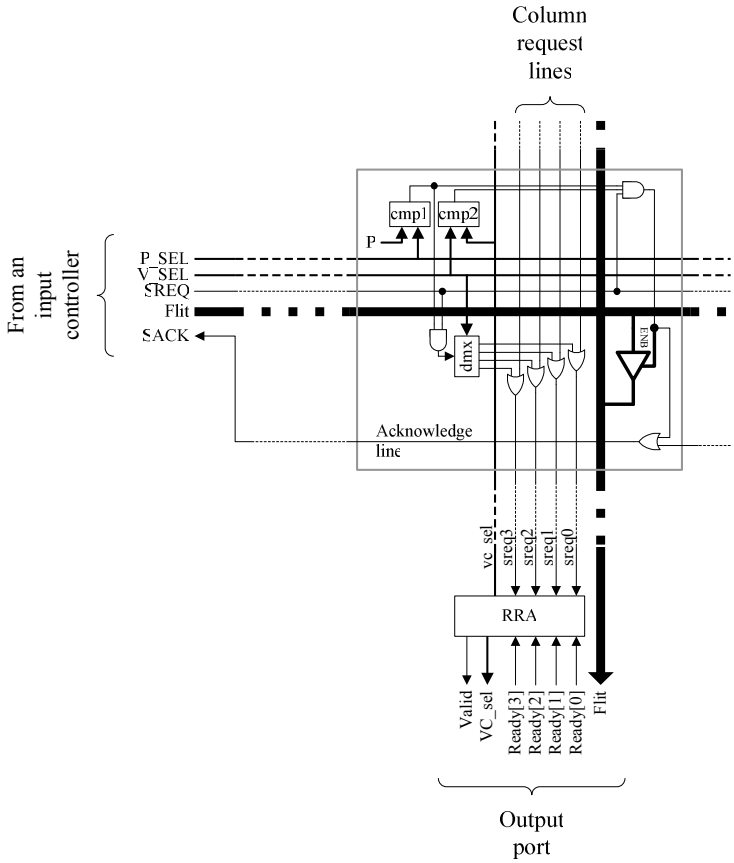


Figure 4.10: A cross point in the crossbar switch – only the functionality supporting the switch allocation and data switching

The cross point, as shown in Figure 4.10, switches only the forwarded data and the switch allocator signals, but not the VC allocator signals. The cross point functionality needed to support the VC allocation is shown in Figure 4.11. The comparator *cmp1* is the same comparator as in Figure 4.10, which activates the cross point. In an activated cross point, the signal *VHOLD* is switched to one of the VC state lines running along the column. To which state line the signal is switched is determined by the signal

V_SEL. The state lines are organised as an OR-chains that collect the signals *VHOLD* from all cross points in a column. The *VHOLD* signal shows whether the input controller currently holds the output VC it requests. Hence, the state line shows whether the corresponding output VC is occupied or free. Since an output VC is allocated to only one input controller, only one of the *VHOLD* signals switched to a VC state line will be active at a time.

At the end of the OR-chains the state lines are inverted and sent back along the column. The new signals, named *vack0* to *vack3*, indicate whether the corresponding VC is free. In the active cross points along the crossbar column, one of the signals *vack0* to *vack3* is selected, again by *V_SEL*, and sent along to the input controller along the *VACK* line. The *VACK* line is an OR-chain that collects the *VACK* signals from all cross points in the row. Since only one cross point can be active in a row, only one *VACK* signal is sent on the OR-chain. The *VACK* signal at the end of the line indicates to the input controller whether the output VC it requests is currently free.

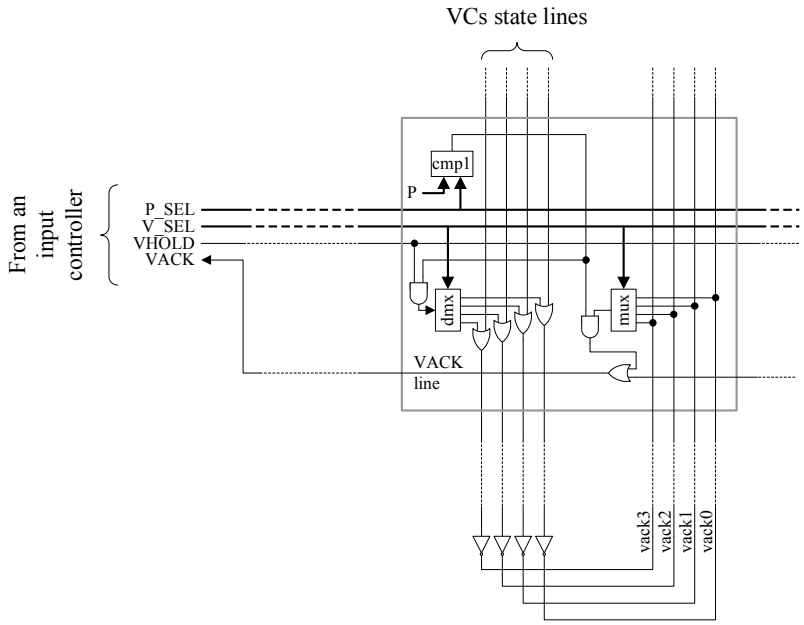


Figure 4.11: Additional functionality in a cross point needed for support of the VC allocation

By combining the functionality shown in Figure 4.10 and Figure 4.11 in a single cross point, we can build a crossbar capable of switching all the signals needed for the VC allocator, the switch allocator and data forwarding. The proposed design consists of many small components uniformly distributed in the crossbar. As we shall see in the next section, this integrated design results in a more area efficient implementation than a design, where the VC allocator, the switch allocator and the crossbar switch are built as separate, complex, but tightly interconnected blocks.

4.2.6. Round-robin arbiter

The switch connections for forwarding flits are allocated to input controllers by round-robin arbiters (*RRA*). Every clock cycle a round-robin arbiter grants one of the possible multiple requests on its inputs. The requests are granted in circular priority order as the request granted in the current cycle has the lowest priority in the next cycle.

We use the implementation of a round-robin arbiter proposed by Gupta [35]. To explain it, we first introduce the static priority arbiter shown in Figure 4.12.a. The static priority arbiter grants requests according to static priorities assigned to them. The requests arrive on the arbiter inputs r_i and the grant is issued on one of the outputs g_i. In the arbiter shown in Figure 4.12.a, the request inputs with lower i have higher priority. The arbiter is organized as a priority chain. The higher end of the chain, which has the highest priority, is set to constant '1'. A carry signal c_i is transmitted down on the chain links, showing whether a request of higher priority has been granted.

The variable priority arbiter, shown in Figure 4.12.b, is derived from the static one by connecting the priority chain in a ring. Switching (OR) elements controlled by the priority inputs p_i, are added between the links in the chain. The priority inputs set the current request priorities. At any time there is only one priority input that is active and its position determines the position of the request input with highest priority. The active priority input drives the corresponding OR element such that the priority ring is broken in a chain again. Thus, the higher end of the chain and the request priority order is determined dynamically by the priority inputs.

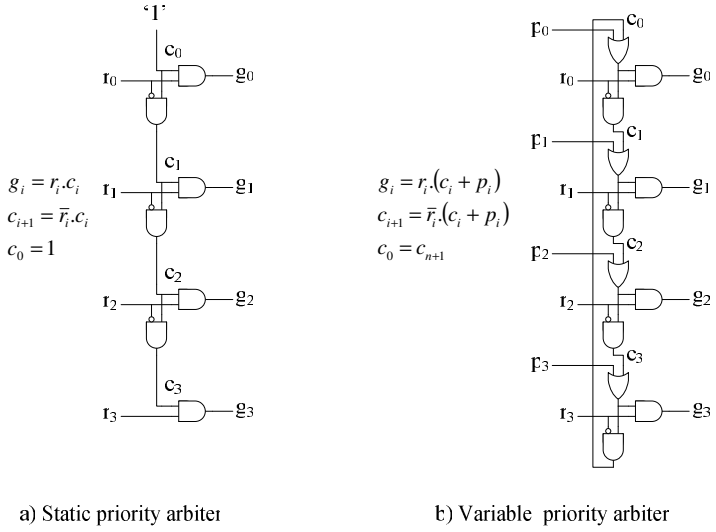


Figure 4.12: Priority arbiters

A Round-robin arbiter is constructed from the variable priority arbiter by using the priority control circuit presented in Figure 4.13. The current state of the priority inputs is a function of the last grant given by the arbiter. The current request priority state is stored in a vector of flip-flops. The state is changed only when a request is granted. The new priority state is derived by rotating the current grant vector g with one position, such that in the next cycle the currently granted request input has the lowest priority. The output *valid* indicates whether a grant is issued in the current cycle. The signal

vc_sel is derived by encoding the grant vector. Upon circuit reset one of the flip-flops must be set while the other flip-flops are cleared.

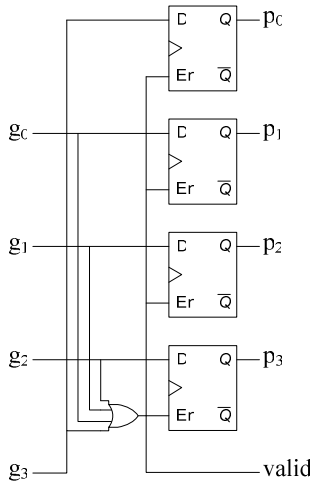


Figure 4.13: Priority control in a Round-robin arbiter

4.3. Synthesis results

The design of a router described in the previous section was modelled in VHDL and synthesised. The router design parameters: buffer size (B), number of VCs per port (V) and width of the network channel (W), are left as model parameters. We synthesised the router for a number of combinations for the design parameters values and observe how separate parameters influence the implementation results (area and maximal operating frequency). We also model the symmetric router architecture and use its synthesis result as a reference for comparison. Synthesis was preformed with the Synopsys Design Compiler® using the TSMC® 0.13µm library[*].

The area results for different combinations of parameters are presented in Figure 4.14. The maximal operating frequency for the same combinations of parameters is presented in Figure 4.15. In each of the three graphs shown in the figures, we vary one of the parameters B, V or W, while the others are fixed. As a basic router configuration we take B=2 flits, V=4, W=16 bits. The varied parameter can take the following values: B={2,4,8} flits, V={2,4,8}, W={8,16,32} bits. Each bar in the graphs in Figure 4.14 presents the cell area for the given router configuration. The area has three contributing parts: area taken by the input controllers, area taken by the crossbar and area taken by other circuitry which includes the RRA arbiters and some glue logic.

The first graph in Figure 4.14 presents the router area for three different buffer sizes. The buffer size influences only the area of the input controllers. We expect that the area for the input controllers increases linearly with the buffer size, while the area for the other router parts remains the same. This is confirmed by the results shown by the graph. Doubling the buffer area almost doubles the area of the input controllers. There is a small additive component to this area which is due to the additional state and control logic in the input controllers

[*] Both products are furnished with a license from Synopsys (Northern Europe) Limited.

Even for the smaller buffer size of two flits, a significant part of the area is occupied by the input controllers, dominated by the buffer area. Thus, the network buffers are expensive in terms of area and must be minimised. As discussed in Chapter 3, the performance gain from an increased buffer size is a higher saturation throughput for the best effort traffic. However, in the same chapter we saw that the intensity of the BE traffic in our network is small; even with 2-flit buffers the BE traffic utilises less than 15% of the saturation throughput. Thus, we minimise the buffer space without sacrificing performance.

The second graph in Figure 4.14 shows how the router area changes with the number of VCs. The number of VCs determines the number of input controllers, the number of crossbar inputs and the size of the arbiters. The number of input controllers and the size of the arbiters changes linearly with the number of VCs, so does their area. The number of crossbar inputs also increases linearly with the number of VCs. However, the number of request/acknowledge lines along the crossbar columns (the OR chains) is also determined by the number of VCs. Therefore, the area of the crossbar increases more than linearly but less than square with the number of VCs.

More VCs per router port means that more guaranteed service (GS) connections can be opened simultaneously in the network. Also the bandwidth of the physical channels is distributed at a finer grain. However, it must be noted in Figure 4.15 that with the increase of the VC number, the router performance deteriorates.

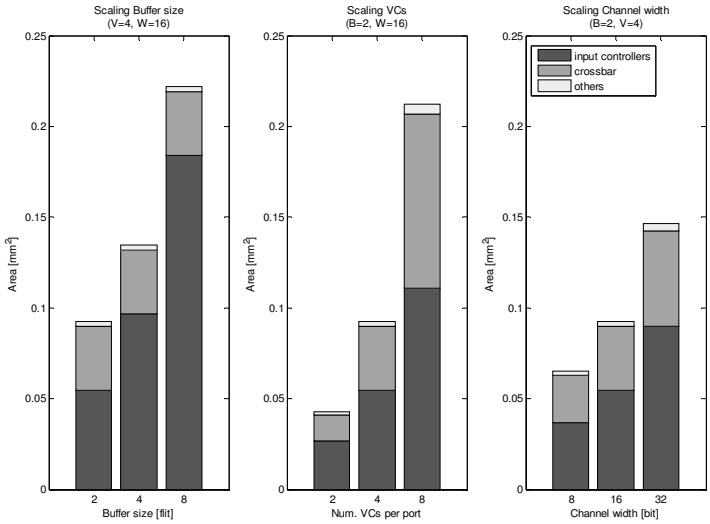


Figure 4.14: Area of a router for various values of the design parameters (B=buffer size, V=number of VCs per port, W=channel width)

The third graph in Figure 4.14 shows how the router area changes with the channel width. Channel widening increases linearly the FIFOs size and respectively the area of the input controllers. The amount of glue logic also increases linearly. In the crossbar, the number of bits per channel determines the number of tri-state buffers per cross point and the number of wires in the columns and rows. Since the tool reports only cell area, we observe only the linear dependency between the channel width and the number of tri-state buffers. In general, crossbars tend to be wire dominant However, in our design

we merge the crossbar with two control blocks (VC allocator and switch allocator) and so introduce control functionality uniformly distributed among the cross points and the crossbar area. Thus, we expect that our crossbar is not wire dominant and the reported cell area is near the actual crossbar area.

While the channel widening increases the amount of data transmitted in a clock cycle, it does not noticeably affect the router operating frequency. Therefore, widening the physical channels effectively increases the network throughput. Doubling the channel width doubles the router performance, but the router area increases less than two times. Therefore, considering the performance-area ratio, the channel width is beneficial way of increasing router performance.

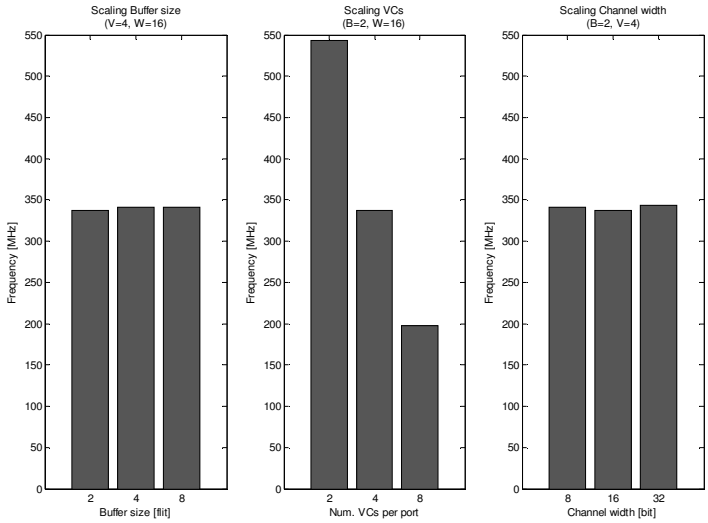


Figure 4.15: Maximal operating frequency of a router for various values of the design parameters (B=buffer size, V=number of VCs per port, W=channel width)

In Figure 4.15 we see how the maximal router clock frequency depends on the router parameters. The strongest dependency is on the number of VCs. The reason is that the number of VCs directly influences the length of the router critical paths. The critical paths are the paths of the *SREQ* signals (see Figure 4.10), which propagate from an input controller to the cross point, then over the column request lines to the RRA. The arbiter returns back the signal *vc_sel* to the cross point where it is switched over the row acknowledge line and returned to the input controller as the *SACK* signal. All this path is propagated in a single cycle and its propagation delay determines the router maximal clock frequency. Since the number of VCs determines the number of cross points in the crossbar columns, it determines the length of the column request lines and their propagation delay. The line's propagation delay, which is dependent on the number of OR elements in the chains, increases linearly with the number of VCs. The delay of the arbiter also depends linearly on the number of VCs. The delay of the other parts of the critical path is not directly dependent on the number of VCs. Therefore we expect the delay of the router critical paths to depend linearly on the number of VCs. The maximal clock frequency is inversely proportional on this delay.

The delay of the critical paths and its dependency on the number of VCs can be reduced by replacing the OR-chains with cascaded ORs. The cascaded OR is faster than an OR-chain, but it uses more OR elements. Since we focus on the router area reduction we experimented with OR-chains.

The buffer size and the channel width do not directly influence the length and the delay of the router critical paths. For that reason they do not affect the maximal clock frequency noticeably.

Finally, we compare the implementation results of our asymmetric router architecture design and canonical design of symmetric architecture (see Figure 3.6). The two architectures use the same router configuration: B=2, V=4, W=16. For the symmetric architecture design the VC allocator, the switch allocator and the crossbar switch are implemented as separate blocks.

Figure 4.16 compares the implementation area and the maximal clock frequency of the architectures. The symmetric architecture has a relatively small crossbar area, while its arbitration ('others') takes more than 50% of the router area. This is because the allocators are implemented inefficiently as separate blocks and also because the switch allocator for this architecture is more complex. However, the main disadvantage of this architecture is that it due to its complex arbitration it cannot provide service guarantees.

The asymmetric architecture design we proposed in this chapter, in spite of its larger crossbar, reduces the overall router area compared to the symmetric architecture. The crossbar size increases, but the arbitration area is decreased considerably. And what is more important, this architecture can provide guaranteed services. Although the clock frequency of the asymmetric architecture is slightly smaller, speed of both architectures is comparable.

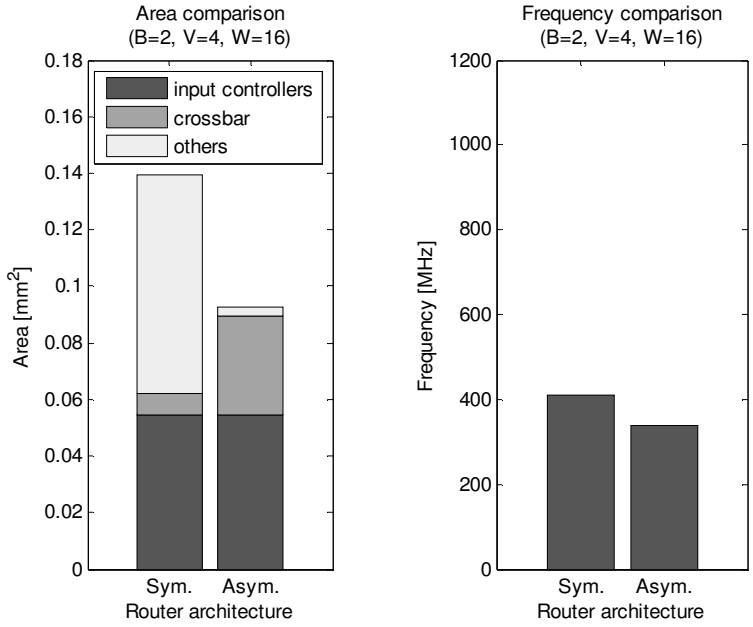


Figure 4.16: Area and performance comparison between symmetric and asymmetric architecture of virtual channel router

Table 4.1 compares the implementation results of our router with the results of the other NoCs providing GS and BE services. For similar design parameters the area of our router is comparable with the area of the other networks. It is slightly smaller but also slower than the other packet switching networks. The circuit switching solution of Wolkotte is much smaller and faster than all the packet switching solutions, but it alone is not able to provide BE services.

Table 4.1: Router implementation results of the NoCs providing GS and BE services

NoC	Flow control	Area [mm2]	F [MHz]	Tech. [mm]	Configuration
Wolkotte	Circuit switching, cut-through (serial)	0.05	1000	0.13	4-bit channel 4 channels per port, BE network not included
Æthereal Distib.	TDM, Wormhole	0.24	500	0.13	32-bit channel, 1-flit GS and 8-flit BE buffers per port, 3-word flits
Æthereal Centr.	TDM, Wormhole	0.17	500	0.13	
Æthereal GS only	TDM	0.03	1000	0.13	
MANGO	Virtual channels	0.19	515*	0.12	32-bit channel, 2x48 VCs per port, 2-flit buffers, 1-word flits
VCNoC	Virtual channels	0.15	350	0.13	32-bit channel, 4 VCs per port, 2-flit buffers, 1-word flit

4.4. Conclusion

In this chapter we discuss the design and implementation of the asymmetric virtual channel router architecture we employ in our network. Since it is considered to be too extravagant in implementation area, the asymmetric architecture is usually disregarded and the traditional symmetric architecture is preferred. However, here we show that for routers of such small scale as the network-on-chip (NoC) routers, an appropriate implementation can make the asymmetric architecture competitive in area and performance with the symmetric one. Moreover, it is competitive with the other NoC router solutions reviewed in Chapter 2. The reason why we insist in using the asymmetric architecture is that, in contrast to the symmetric one, it is able to provide predictable performance.

We propose an optimised design for an asymmetric architecture which avoids repeated functionality, complex interfaces and makes the implementation more uniform. All this helps in reducing the overall router area and making it smaller than that of a symmetric router architecture.

The analysis of the implementation results confirms the observation made for other NoC designs that in terms of area buffers are the most expensive router component. The router maximal operating frequency is sensitive only to the number of VCs. For that reason and because of the area constraints, the number of VCs in a router should be minimised. What is the required minimum number of VCs from a functional point of

view is discussed in the next chapter. Among the router parameters, the network channel width is the most effective way for increasing the router throughput.

Chapter 5

Evaluation of the virtual channel reservation approach*

Virtual channel reservation is a simple approach for providing guaranteed throughput services in a virtual channel network-on-chip. However, its performance is limited by the number of virtual channels per physical channel. In this chapter we explore the limits of the approach and investigate how these limits depend on the routing algorithm, traffic locality, network topology and network size. We also estimate the implementation overhead of virtual channel reservation.

5.1. Introduction

In Chapter 3 we propose virtual channel reservation as an approach for providing guaranteed services (GS) in our virtual channel NoC. To provide guaranteed services we reserve a path of VCs from source to destination and the data sent over this path can rely on a guaranteed throughput. Such paths we call *GS connections* and they are reserved at run-time when the application that uses them is started.

The GS connections stay reserved for the lifetime of the application, which for our target applications (streaming DSP applications) can be from seconds to hours. During that time the reserved VCs cannot be used for other communications. Since there is a limited number of VCs per physical channel, only a limited number of GS connections can be reserved simultaneously, respectively provided to applications. Therefore, to apply virtual channel reservation we must be sure that the network can meet the system demands for GS connections.

In this chapter we investigate under what conditions the virtual channel reservation can satisfy the demand for GS connections in our system. We assume a fixed number of virtual channels. As we saw in Chapter 4, the router area and performance are sensitive to the number of VCs, which does not give us much freedom to vary with the number of VCs. Given a fixed number of VCs per physical channel, there are a number of factors determining the maximal number of connections that can be open simultaneously and these are: required bandwidth, traffic locality, connection routing, network topology and network size. For different combinations of these factors we test whether or not the requested connections can be provided. The test is done by means of simulations in which a routing function tries to find and reserve paths for the approximated maximal

* Major parts of this chapter have been presented at the International Worksop on Applied and Reconfigurable Computing [7].

number of GS connections. Our goal is to identify the cases for which the proposed NoC can satisfy the system demands and the virtual channel reservation can be applied safely.

Applying the virtual channel reservation requires support at system level. In this chapter we also estimate what is the system overhead for this support, in order to show that the VC reservation can be used at run-time.

5.2. Network and GS services

Our network is constructed by interconnecting a two-dimensional array of PEs in a grid; each PE is equipped with a router as the neighbouring routers are connected by two *physical channels*, one in each direction. All physical channels have the same bandwidth. The network is a virtual channel network [21], which means that each physical channel is shared between *K virtual channels* (VCs). The VCs dynamically share the bandwidth of the physical channel. The VCs currently transporting data receive equal shares of the physical channel bandwidth, while the idle VCs do not receive bandwidth.

To provide GS we use virtual channel reservation – an approach which is introduced in Chapter 3 and which we briefly repeat here. Let the network be defined as a graph $I=(N,C)$, where the vertices represent the network nodes and the edges represent the physical channels. All physical channels have bandwidth b.

Assume that on a physical channel c_i, there are k_i VCs transporting data, while the remaining $K-k_i$ VCs are idle. Then, because of the bandwidth sharing each of the k_i VCs is guaranteed throughput of:

$$TH_i = \frac{b}{k_i} \tag{5.1}$$

Thus, if we know the number of occupied VCs, we can predict what the minimal throughput of a VC will be. Furthermore, if we can control the number of occupied VCs, we can guarantee a minimal VC throughput. The virtual channel reservation exploits this network feature. By controlling how the VCs in the network are used we provide guaranteed network services.

With the virtual channel reservation approach guaranteed services are provided on a connection basis. The network provides *connections* with guaranteed minimal throughput (GS connections). A GS connection is just a path reserved over the VCs from source to destination. Such a connection can guarantee a requested throughput TH_R if all the VCs on its path guarantee a throughput that is greater than or equal to TH_R. According to (5.1) this means that for all physical channels $<c_1, c_2, \ldots c_H>$ traversed by the connection it must be provided that:

$$k_i \le \left\lfloor \frac{b}{TH_R} \right\rfloor = k_R \tag{5.2}$$

This is a necessary and sufficient condition for a GS connection to provide a minimal throughput TH_R (for more details see Chapter 3).

The virtual channel reservation approach requires support at system level – a central routing function that reserves the VCs and provides connections. The function runs as a

system routine on a central general purpose processor. The routing function is called every time a new connection is required and the function output is used for network reconfiguration. To find a path with a given throughput TH_R the routing function searches the network for VCs that satisfy (5.2).

Connections are requested and provided at run-time when a new application is started. When an application is started by a central system authority, the routing function is called to find paths for all GS connections required by the application. Since the routing function is used at run-time, it must be fast.

The number of GS connections that can be open simultaneously in the network is limited. The main constraint of course is the number of virtual channels, but there are number of other factors influencing the limit:

- *routing function* – the routing function allocates the VCs to the connections, thus it distributes resources to the connections. The way the allocation is done may influence the maximum number of connections that can be opened simultaneously.
- *traffic locality* – the traffic locality determines the average distance between source and destination; therefore, it determines the average number of VCs per connection, which in turn influences the maximum number of connections.
- *network topology* – the network topology determines the number of physical channels in the network. Hence the topology also influences the limit.
- *network size* – the number of nodes in the network determines the number of physical channels for a given topology and so it may also influence the limit.

The factors listed above define a five-dimensional design space which we explore by judiciously selecting points in each dimension. All the dimensions except the traffic locality and network topology are independent of each other.

Traffic locality

Locality is a traffic characteristic, which is a result of the application mapping. The mapping places the application graphs over the network graph. Thus, traffic locality depends on the mapping algorithm, the topology of the applications graphs and the network graph topology. To provide traffic patterns with different locality characteristics we simulate the operation of a mapping function. We use different mapping scenarios to map a fixed topology application graphs on fixed topology network graphs. In that way we create what we call a spatial model of the GS traffic, discussed in Section 5.4.

Number of VCs

We do not experiment with the number of VCs but fix it to a constant value. Although more VCs allow more connections to be opened simultaneously, the number of VCs cannot be increased arbitrarily because this increases the router area and lowers router clock frequency (see Chapter 4). We fix the number of VCs per physical channel to 4 (K=4). For this number the router area is less than 0.1 mm^2 (for 16-bit channels and 2-flit buffers) and the minimal throughput per VC is 1400 MBit/s, which is enough to support high throughput communications like HiperLAN/2 (512 Mbit/s). Increasing the number of VCs to 8 doubles the area and reduces the minimal VC throughput to 800 Mbit/s, which might be a tight bound for high throughput applications. The number 4 is

also motivated by the result of a study about the trade-off between performance and buffer area of a virtual channel router, presented by Dally [21]. According to (5.1), with K=4 a VC can guarantee throughput of b, $b/2$, $b/3$ and $b/4$.

Routing function

To test whether the NoC can satisfy the system demand for connections, we have to test whether the routing function can provide paths for the maximal number of requested connections. We do this by means of simulation experiments. We simulate only the operation of the routing function and not the network itself. The routing function takes as an input two nodes, a source-destination pair, between which a connection is needed and returns as a result a path for the connection. The network is represented by its topological graph and the routing function is run to find paths in the graph for a number of connections that approximates the maximal number of connections demanded by the system. The number of connections and their distribution in the system are provided by a spatial model of the GS traffic in the system.

To see how the routing function performs in different conditions, simulation experiments are performed for different combinations of factors listed above. Given a fixed number of VCs, we experiment with the other factors influencing the maximum number of connections. We experiment with two routing algorithms – one that does load balancing trying to uniformly distribute traffic in the network and another that does not do load balancing. Experiments are made for traffic with different locality. We construct traffic models that approximate worst case locality, best case locality and intermediate case locality.

Topology

The experiments are conducted for the most popular grid topologies shown in Figure 5.1 – mesh and torus (see Chapter 2). Later we also make energy cost estimations where the length of the network channels is important. For that reason we include also the folded torus topology. The folded torus has the same graph topology as torus, but its nodes are reshuffled in the plane and the network channels have different physical length.

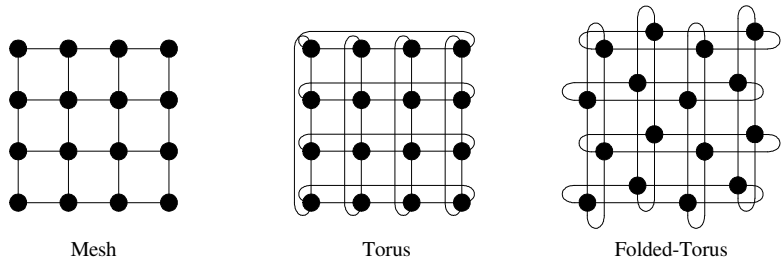


Figure 5.1: Network topologies used in the experiments

Network size

To take into account the effect of the network size, experiments are performed with networks of two sizes – 10-by-10 nodes and 16-by-16 nodes. Considering the available chip area and the size of the processing elements, these are system sizes feasible in the today and near future semiconductor technologies.

We continue with a more detailed introduction of the routing function and an estimation of the overhead it entails. Later we describe the traffic model we use to simulate the routing function. Finally, we describe the simulation experiments and present the results.

5.3. Routing function

The task of the routing function is to find paths for GS connections. A GS connection is described by its source node S, destination node D and requested throughput TH_R. The routing function has the form R(S,D,TH_R). It takes as an input a connection description and returns a description of a network path from S to D, or fails if the connection cannot be provided. The returned path guarantees a throughput of at least TH_R. The path is described as an ordered sequence <vc_1, vc_2, …, vc_n> of virtual channels vc_i reserved for the connection.

5.3.1. Operation

When searching for a path that can provide a specified throughput TH_R, the routing function may use only physical channels that satisfy (5.2) or in other words, physical channels where the number of occupied VCs is less than or equal to k_R. When the routing function decides to route a path over a given physical channel c_i, it reserves one of the free VCs on that physical channel. This changes the throughput of the other occupied VCs, if any, on the same physical channel. Therefore, searching for a path, the routing function must be aware that it does not violate the guarantees given to the paths already routed. The routing function must use only channels satisfying the following two *GS routing criteria*: if the path is routed through the physical channel, then i) (5.2) will hold for the path currently being routed and ii) (5.2) will still hold for the paths already routed through the same physical channel. Here we discuss only the basic functionality of the routing function. More advanced options like changing of existing paths are considered as future work.

To apply the GS routing criteria, the routing function must know the state of the VCs on every physical channel. The VC state indicates whether a VC is occupied, and if so, what throughput it guarantees, or what the k_R of the path used is. The constraint k_R may take integer values between 1 and K.

The state of the VC j on a channel c_i is stored in a *state variable* r_{ij} that may take values 0, 1, … K. Value 0 indicates that the VC is not occupied, while for the occupied VCs r_{ij} stores their constraint k_R:

$$r_{ij} = \begin{cases} 0 & \text{if } vc_{ij} \text{ is not occupied} \\ k_R & \text{if } vc_{ij} \text{ is occupied} \end{cases} \tag{5.3}$$

The state variables of all VCs in the network construct the network state. Thus the routing function is of type:

$$R : state \times S \times D \times TH_R \rightarrow state \times path$$

Given the network state, we can find the number k_i of occupied VCs on a given physical channel c_i by counting the number of VCs on c_i which state $r_{ij} \neq 0$, $0 \leq j \leq K\text{-}1$. The routing function needs to know k_i when deciding whether it can route through a given

physical channel c_i. The routing function checks whether k_i satisfies the routing criteria. The first criterion is to guarantee the throughput of the connections being routed, which is:

$$k_i + 1 \leq k_R \tag{5.4}$$

The second criterion is to guarantee the throughput of the connections already routed through the same physical channel, which is:

$$k_i + 1 \leq r_{ij} \mid r_{ij} \neq 0,\ 0 \leq j \leq K - 1 \tag{5.5}$$

When a physical channel satisfies these criteria, the routing function may use any free VC on it.

The routing function searches the network state for VCs that satisfy the routing criteria and uses these channels to construct a path between the source S and destination D. When the path is found, the routing function sets the state of all VCs used in the path to k_R (r_{ij}:=k_R). In this way it reserves the VCs, since they are not considered in following path searches. When the path is not needed any more (e.g. the application using it terminates) the state of the used VCs are freed and their state is set to "free" (r_{ij}:=0).

In this chapter we discuss only routing of GS traffic. However, in the real system both types of traffic, BE and GS traffic, are handled. This requires a small modification of the network state and the routing function. The network state needs one extra bit per VC to identify whether the VC is used to carry BE or GS traffic. When a VC is used for BE traffic, its state shows the number of BE paths that share the VC. The routing function increments or decrements the VC state value when routing or deleting a path through the VC. When routing BE traffic, the function must avoid deadlock. As discussed in Chapter 2, the most inexpensive way for that is to apply the Turn Model, which restricts the turns the paths can make.

Finding a route in a network is equivalent to finding a path between two nodes in a graph. The network topology is represented as a graph I=(N,C) and a path searching algorithm is run on that graph. Among all possible paths the shortest is preferable, because shorter network routes result in less network traffic and less energy for communication. Therefore, the routing function is based on an algorithm for the shortest path search in graphs. Actually, the algorithm runs on a sub-graph I'=(N,C') which is derived by removing all channels in I that do not satisfy the GS routing criteria. In the course of operation the routing algorithm thus ignores the channels in I that do not satisfy the GS routing criteria.

5.3.2. Algorithms

We experiment with two shortest path search algorithms: *Breadth-first search* (BFS) and *Dijkstra's algorithm* (DA) [20]. Breadth-first search is a basic shortest path search algorithm that works on non-weighted graphs. It is also used as routing algorithm in the IBM SP2 system [74]. We use BFS to implement a simple routing function that does not do load balancing. The algorithm finds shortest paths in terms of the number of edges. Thus, the routing function based on BFS finds paths that are minimal in terms of physical distance. However it does not take into account the current state of the network – physical channels with or without load are treated equally as long as they satisfy the GS routing criteria. The computational complexity of the algorithm is linear in the

number of network nodes $O(N)$. The memory complexity of the algorithm is also linear in the number of network nodes.

Dijkstra's algorithm is a more advanced shortest path search algorithm that works on weighted, directed graphs with non-negative weights. We use it to implement a more sophisticated routing function that tries to balance the traffic by distributing the communication over the network. The algorithm finds shortest paths in terms of a minimal weighted sum. In our network, the weight we assign to an edge is one plus the number of occupied VCs on the corresponding physical channel; thus the weight of channel c_i is $(1+k_i)$. In the sum, one stands for a unit of physical distance and k_i is the weight representing the number of occupied VCs. Thus, searching for a minimal weight path, DA prefers to use physical channels with fewer VCs occupied, so we may expect that DA will distribute communications in the network more uniformly than BFS.

The graph weights change dynamically. Every time a connection is routed the state of the reserved VCs is changed, which increases the weights of the physical channels traversed by the connection. Reversely, when the connection is deleted, the VCs are released and the weights are reduced. Hence, the weights reflect the current network state and the routing algorithm adapts its decision to this state.

A naive implementation of Dijkstra's algorithm leads to a computational complexity of $O(N^2)$, but with an optimised version $O(C \log_2 N)$ can be achieved [20]. The memory complexity of the algorithm is linear in the number of nodes.

Here we experiment only with BFS and DA algorithms to find out whether the load balancing improves the performance of the routing function. However, the A* (A star) algorithm [37] can be used to reduce the run time of the routing function. The A* algorithm performs a directed search in a graph. Since our network topology is known in advance, we can use this knowledge to direct the algorithm in its search for a shortest path. Thus, we can reduce the number of explored nodes and respectively to reduce the average time for finding a connection. We include the A* algorithm in our future work.

5.3.3. Overhead

To measure the time overhead due to the routing, we measure the worst case execution time of the routing function on an ARM processor – a general purpose processor popular for a low-power SoC. The worst case execution time of the routing function gives the maximal time for routing a connection. We implement in C two routing functions, one based on BFS and one based on DA, and run them on an ARM7 simulator to measure their worst case execution time. For DA we use the naïve implementation of complexity $O(N^2)$ because our intention here is to get an indication about the magnitude of the execution times. Since in the simulations presented in Section 5.6 DA does not prove to be beneficial to the performance of the routing function, we do not make further efforts to optimise DA.

While, searching for a path, both algorithms progressively explore network nodes until the destination node is reached. The number of explored nodes determines the number of iterations the algorithm makes. Thus, to create a worst case execution condition, we force iteration over all the nodes. Since a mesh and a torus topology have the same number of nodes, the topology does not influence the worst case execution time. We measure the time assuming a mesh topology. Each algorithm is run once to explore all network nodes and its execution time is recorded. To see how the overhead

scales with the network size, measurements are taken for two network sizes – 10-by-10 and 16-by-16.

The results are presented in Table 5.1. Even at a modest clock frequency of 100 MHz, BFS can provide a connection in less than a 1 ms for both network sizes. Typically our applications require 5 to 10 connections [67, 87]. Hence, providing all connections will take several milliseconds. Therefore, when an application is started, the time overhead due to the routing is of an order of milliseconds, which is tolerable given that the lifetime of GS connections ranges from seconds to hours.

With DA the maximal time for providing a connection increase to 1.7 ms and 10 ms for the two network sizes. Then, the time overhead for starting an application can be tenths to hundred milliseconds. This is also tolerable but it is almost perceptible to the user. Increasing the clock frequency to several hundreds of MHz will reduce the DA overhead, but will not improve its scalability with the number of nodes. The scalability of DA is poor because of the high complexity of the used algorithm, $O(N^2)$. The poor scalability will be a problem when the system size grows. Therefore, if the routing function is based on DA, a more optimised implementation of the algorithm with complexity lower than $O(N^2)$ must be used.

Table 5.1: Time overhead for routing a single connection in a network with 4VCs per physical channel; WCET = Worst Case Execution Time

Time overhead	Network size	Algorithm	
		BFS	DA
WCET [cycle]	10x10	28720	172055
	16x16	74355	1020995
WCET [µs] @100MHz	10x10	287	1721
	16x16	744	10210

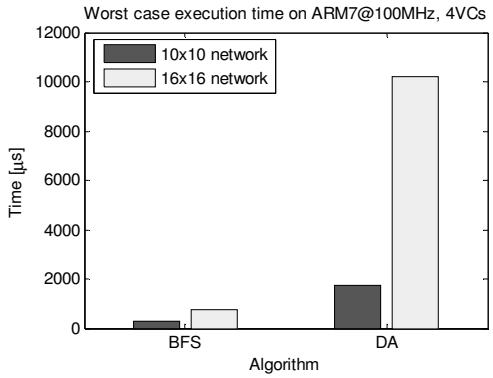


Figure 5.2: Scalability of the routing function execution time with the network size

To see how the time overhead for routing a connection scales with the number of VCs, we measure the worst case execution time of the routing functions for 4 and 8 VCs per physical channel in a network of fixed size 10-by-10 nodes. The results are presented in Table 5.2 and Figure 5.3.

The number of VCs influences the execution time of the algorithms only through the GS routing criteria used as a channel selection rule. When the algorithm explores a node, it applies the channel selection rule to the physical channels of the node. Applying the rule to a physical channel means examination of the state of all VCs on that physical

channel. Hence, by increasing the number of VCs we linearly increase the time for applying the rule and respectively the time for exploring a node. Since the algorithm iterates over the network nodes, when increasing the number of VCs we linearly increase the time for a single iteration and the algorithm execution time. Here we examine the worst case execution time and both algorithms iterate over all network nodes (100 nodes for a 10-by-10 network). Hence, both algorithms perform the same number of iterations and we expect that a change in the number of VCs will have the same impact on the execution time of both algorithms.

The results in Table 5.2 and Figure 5.3 confirm our expectation. The increase in the number of VCs from 4 to 8 leads to a similar increase of the execution time for both algorithms. This increase is about 1.1 μs (110 cycles) per iteration or about 0.1 ms in total. Because this value is small compared to the total execution time, and because a large number of VCs is not possible due to their high area cost and negative impact on the performance, we do not consider the number of VC a serious time overhead related issue.

Table 5.2: Time overhead for routing a single connection in a 10×10 network; WCET = Worst Case Execution Time

Time overhead	Num. VCs	Algorithm	
		BFS	DA
WCET [cycle]	4	28720	172055
	8	39724	183801
WCET [μs] @100MHz	4	287	1721
	8	397	1838

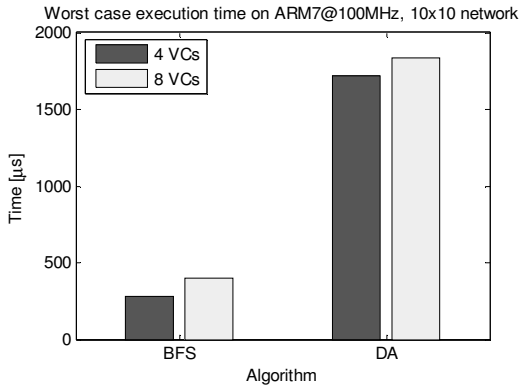


Figure 5.3: Scalability of the routing function execution time with the number of VCs

The memory overhead for supporting the routing function is presented in Table 5.3. The function implementation requires memory for storing the function code, function variables and memory for storing the network state. It shows that both algorithms have similar memory requirements. Most of the required memory is for storing the network state. The network state is the same for both algorithms and consists of the state of all VCs in the network. Assume *w* bits are needed to store the state of a VC. Then, per physical channel *Kw* bits are required.

The size of the network state for the two network topologies is calculated in the following way. In a torus network every node is connected to five channels – four to

neighbouring routers and one to the local PE. Therefore, to store the state of a N-node torus network $5NKw$ bits of memory are needed. In a mesh network of size $n \times n = N$ nodes, there are $(n-2)^2$ nodes connected to 5 channels, $4(n-2)$ nodes connected to 4 channels and 4 nodes on the corners connected to 3 channels (always including the channel to the local PE). Thus, to store the state of a mesh network $[5(n-2)^2+4*4(n-2)+4*3]Kw$ bits are needed. The network state memory presented in Table 5.3 is for a network with 4 VCs per physical channel (K=4) and a state of one byte per VC (w=8). Given four VC per physical channel, the VC state variable can take 5 values – from 0 to 4 (see Section 5.3.1). Thus, three bits are enough to store a VC state and the total network state given in Table 5.3 can be compressed. However, this will complicate the access to the state variables.

The memory required for the support of a routing function can reach several KBytes, which may exceed the size of the local memory of a PE. However, the routing function runs on a central general purpose processor, which besides routing performs other system function. To support the system, this processor cannot rely only on a local memory but is connected to a larger external memory of size of hundreds of MBytes [94]. This is where the network state is stored. Compared to that memory size, the memory overhead due to the routing is acceptable. The external memory is slower than the internal one, but the ARM system we consider here runs at 100 MHz which is also slow and due to caches the access time of the external memory is not a bottleneck.

Table 5.3: Routing function memory overhead

Memory overhead	Network size	Algorithm	
		BFS	DA
Code size [byte]	-	512	484
Algorithm variables [byte]	10x10	400	300
	16x16	1024	768
Network state* [byte] for mesh topology	10x10	1840	1840
	16x16	4864	4864
Network state* [byte] for torus topology	10x10	2000	2000
	16x16	5120	5120

*-Assuming 4 VCs per physical channel

We investigate two options for implementing the routing function – BFS and DA. While both algorithms have similar memory complexity, DA has a higher computational complexity than BFS. To decide which algorithm is preferable, we have to know also how the algorithms perform, in terms of maximal number of provided connections. To find out that, we simulate the operation of the routing function with a traffic model that estimates the maximal number of requested connections.

5.4. Spatial model of the GS traffic

The performance of the routing function is tested against a model of the GS traffic in our system. The model reflects the special aspects of the streaming traffic in the system – it approximates the maximal number of GS connections requested by the system and the traffic pattern these connections follow. Furthermore, the model allows for experiments with different degrees of traffic locality.

The GS traffic in our system is generated by streaming applications. Streaming applications typically have a simple pipeline structure represented by a pipeline graph.

At a certain moment in time a number of streaming applications are running simultaneously in the system. Hence, there are number of pipeline graphs scattered over the PEs. The edges of the pipeline graphs represent the required GS connections. To model such traffic we use a graph with a ring topology whose nodes are scattered over the PEs. A large ring graph can be considered as many short pipeline graphs connected serially. Scattering the nodes of the ring graph over the PEs, scatters the short pipeline graphs in the same way we expect to find in the real system. The only difference is that in a real situation some of the applications may communicate with peripheral devices instead of with other applications. However, most likely these peripheral devices will be placed at the border of the PE array where they are connected to the on-chip network. Therefore, the ring graph nodes assigned to border PEs can be considered as peripheral devices which are sources and sinks of data.

To approximate the maximal number of GS connections demanded by the system, we assume that the number of vertices in the ring graph is equal to the number of PEs in the system. Every graph vertex then is mapped on a separate PE, which means that every PE in the model generates and consumes a stream. Hence, the model assumes single task processors in the system, which is the case for the majority of the PEs in our system. The traffic model produces as an output a list with the connections in the mapped ring graph. The connections are described by their source and destination PE. The list consists of N connections, one connection per PE. During the simulations, the traffic model produces such lists and for each list the routing function is called to find paths for all the connections.

The actual PEs where an application will run is determined by the spatial mapping [72]. The positions of these PEs determine the communication distances between the application tasks. Thus, the spatial mapping has a strong influence on the communication locality. We use the spatial mapping to enforce specific locality characteristics in the traffic patterns generated by our traffic model. The traffic model is constructed by mapping of the ring graph on the array of PEs. To model the traffic locality we use three different strategies for mapping the ring graph. The three strategies produce mappings that approximate respectively the *best*, the *worst* and an *intermediate* case of traffic locality.

The three mapping strategies use the same algorithm for choosing the PEs, but differ in a locality parameter given to the algorithm. The algorithm operates on the ring graph in the following way. The graph vertices are mapped sequentially in the order they appear in the graph. For every next vertex, a PE is chosen randomly among those PEs which are at a distance less than or equal to d hops from the PE where the previous graph vertex is mapped. Here d is a parameter of the algorithm that sets a diameter for the preferred network distance. If there is no free PE within that distance, then a PE is chosen randomly among all free PEs. In this way the majority of the communicating PEs are close to each other, but still there are some long distance communications in the system. We expect that to be near to the real conditions in the system because not always the particular circumstances will allow communicating tasks to be mapped close to each other, e.g., inter-application communications, dependencies on the PE type or hot-spot areas in the system

The three mapping strategies differ only in the value of the parameter d. The first strategy tries to maximize the traffic locality; it sets the parameter d to 1 and approximates *best* case locality. The second strategy approximates *worst* case locality. It sets the parameter d to the diameter of the network (the longest network distance, which

in a 10-by-10 network is 18 hops). Therefore, the worst case mapping strategy uniformly scatters the graph nodes over the PEs and no locality should be expected.

The third strategy sets d to an intermediate value 4, so we call this the *intermediate* case of locality. The intermediate locality models the situation expected in a real system where the mapping procedure tries to reach maximal locality but does not always succeed, because the nearby PEs might be occupied by already running applications or be of the wrong type. The distance range from 1 to 4 is where we expect the average communication distance to fall in a system that makes optimisation efforts towards improving its traffic locality. That is because the applications we consider usually consist of 5-10 nodes, so they can form clusters of 5-10 occupied PEs. Within a range of 4 hops, up to 40 other nodes can be reached, which is more than enough to jump across clusters to find a free node around. So we can infer that communication distances of four and more hops will not be observed so often.

The traffic model we construct by scattering the vertices of the ring graph is a subset of the class of permutation traffic, well known in the domain of interconnection networks [27]. In permutation traffic patterns, each source s sends all its data to a single destination d chosen by a permutation of the nodes, $d=\pi(s)$. The difference with our model is that we force specific locality characteristics in the generated traffic patterns.

By using the three mapping strategies, our traffic model can produce traffic patterns with different locality characteristics – best, worst and intermediate locality. The traffic patterns are randomised, but with specific locality characteristics depending on the used mapping strategy. The mapping strategy influences only the average distance of the connections generated by the traffic model.

The distribution of the distances of the connections generated by our traffic model for the three different cases of locality is presented in Figure 5.4 to Figure 5.7. The four figures present results for mesh and torus topologies and for network sizes of 10-by-10 and 16-by16 nodes. Figure 5.4 presents the distance distribution for a mesh topology of size 10-by-10. The three graphs correspond to the three locality cases. Although no locality is expected in the case of worst locality, the distances are not uniformly distributed but follow the distance distribution in the mesh topology [64]. The locality preserving mapping strategies (the intermediate and best case locality) give preference to distances smaller than or equal to d, so these distances appear with higher probability than the others. For example, for intermediate traffic locality, 97% of the communications are at a distance less than or equal to 4 and only 3% at a distance greater than 4.

Since the distribution of distances between the vertices in a torus and mesh graphs differ, changing the topology from mesh to torus changes the shape of the distance distribution. The change is most clearly seen for worst case locality traffic. Increasing the network size changes the network diameter and the mean of the distance distributions, but does not change their shape.

The result of the mapping shows that even with the simple mapping strategy we use a high degree of locality can be achieved. In the left graph of the figures where we aim at the highest possible locality, the mapping algorithm manages to map on nearest neighbour PEs 88% of the graph nodes. This is because of the simple application structure we assume. Hence, the simple pipeline structure of streaming applications simplifies the application mapping.

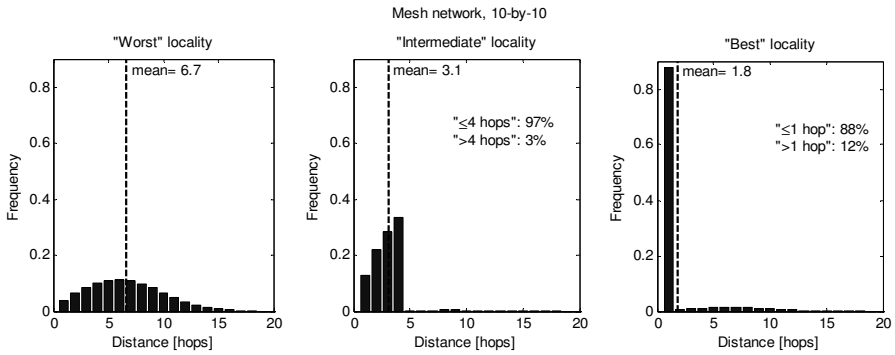


Figure 5.4: Distance distribution in a 10-by10 mesh network and ring communication pattern

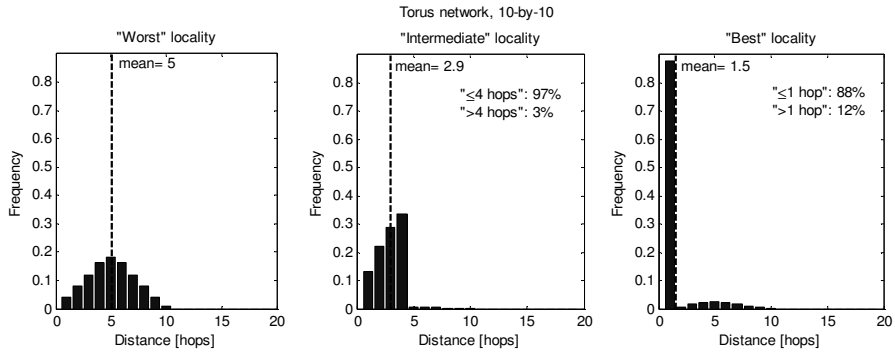


Figure 5.5: Distance distribution in a 10-by10 torus network and ring communication pattern

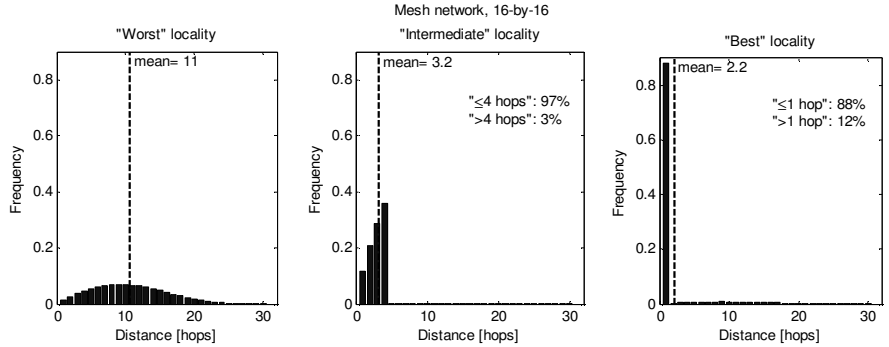


Figure 5.6: Distance distribution in a 16-by16 mesh network and ring communication pattern

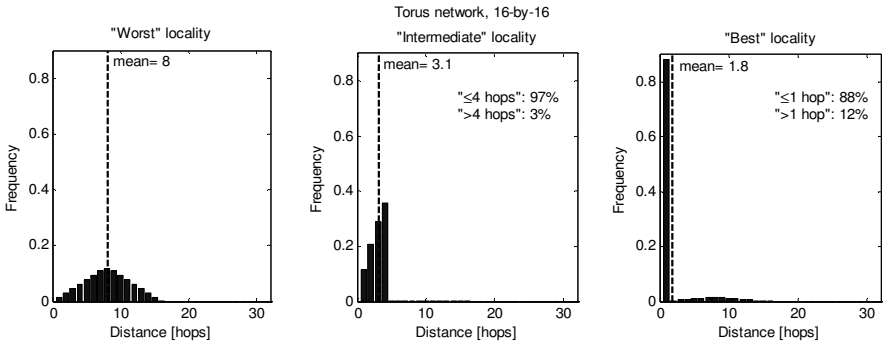


Figure 5.7: Distance distribution in a 16-by16 torus network and ring communication pattern

5.5. Simulation experiments

We perform simulation experiments to test how different factors influence the routing function performance and respectively the applicability of the virtual channel reservation. A simulation experiment consists of two steps. In the first step the traffic model generates a set of N connections and in the second step the routing function is called to find a path for each connection. The output of the simulation experiment is a set of the paths found by the routing function. First of all, we are interested whether the routing function can find paths for all N connections. In case all the paths are found, the experiment is considered *successful*. We also collect information about the actual length of the paths found. All N connections are GS connections and request the same throughput TH_R. In this way we test the limit for maximal requested throughput. Since the network supports four VCs per physical channel, TH_R may take values $b/4$, $b/3$, $b/2$ or b.

Experiments are conducted for all combinations of factors influencing the performance of the routing function:

- routing algorithm – we experiment with the two routing functions based respectively on Breadth-first search (BFS) algorithm and Dijkstra's algorithm (DA), discussed in Section 5.3
- traffic locality – we experiment with traffic patterns with different locality characteristics: worst, intermediate and best, discussed in Section 5.4
- network topology – we use mesh and torus, two network topologies most popular for on-chip and multiprocessor networks
- network size – we experiment with two network sizes 10-by-10 and 16-by-16 nodes or 100 and 256 nodes.

To asses the performance of the routing function for average traffic conditions, we perform 1000 experiments for each combination of factors. In each experiment changes only the traffic pattern according to which the connections are distributed. The patterns are randomised, but with similar locality characteristics. Each experiment sets a *sample* in the space of the possible traffic patterns. We count the number of the successful samples. The path length results collected for the successful samples are averaged. The number 1000 was chosen empirically as the smallest value, for which a further increase

does not change the results noticeably. The results of the performed simulation experiments are presented and discussed in the following section.

5.6. Simulation results

In this section we present and discuss the results of the conducted simulation experiments. We compare how the different factors influence the performance of the routing function in order to decide which of them are of importance and can be used for improvement and which can be neglected.

5.6.1. Number of successful samples

Figure 5.8 shows how many of the 1000 samples taken for each combination of factors in a 10-by-10 network are successful. In other words, it shows in how many of the 1000 experiments the routing function succeeds to route all the connections. The three graphs correspond to the three cases of traffic locality, each graph presenting the results for mesh and torus topology. Of interest to us are the cases in which all 1000 samples are successful because in these cases the network can satisfy the system demands for GS connections; therefore, the virtual channel reservation approach can be safely applied. The best result would be to have in all the cases all 1000 samples are successful. However, because of the limited number of VCs, in some of the cases this cannot be achieved. In the cases when not all samples are successful, the routing function cannot always provide all requested GS connections and the virtual channel reservation approach will limit the system operation.

In Figure 5.8 we see that for worst case traffic locality the virtual channel reservation approach can be safely applied if the requested throughput TH_R is restricted to $b/4$ for mesh topology and up to $b/3$ for torus topology. The torus topology helps to improve the performance in such traffic conditions by increasing the throughput limit from $b/4$ to $b/3$. This is because the torus has smaller diameter, which shortens the average length of GS connections. Introducing traffic locality improves the performance of the virtual channel reservation approach by increasing the limits on the TH_R to $b/2$ for intermediate locality and b for best locality. The reason is that traffic locality restricts the length of the GS connections, so fewer VCs are reserved per connection. The VCs that are left free may be used either for opening more connections or to increase the throughput demand per connection. When the traffic shows locality, the improvement achieved by replacing the mesh with the torus topology is not significant, because the GS connections are short because of the locality anyway.

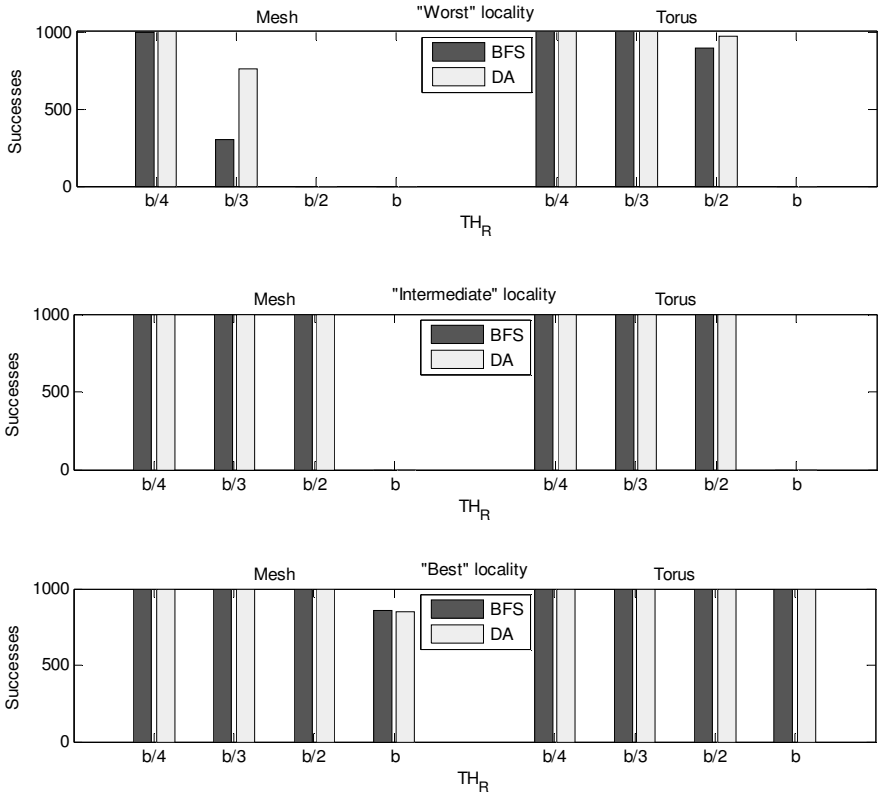


Figure 5.8: Number of successful samples in a 10-by-10 network

The two routing algorithms do not change significantly the performance of the routing function. The DA and BSF perform almost equally in all cases, although DA tries to do load balancing and BFS does not. The most important observation is that in all the cases the demand for GS connections is either satisfied or not satisfied, no matter which algorithm is used. The reason that both algorithms perform similarly, although they put different efforts in routing, is the traffic pattern in the system which is a result of the simple application structure. Every node in the network generates and consumes one stream. Hence, the sources and sinks of data are uniformly distributed in the network and so are the connections. Thus, with this applications structure and this system organization, the traffic in the system is almost balanced and it does not make much difference whether the routing algorithm performs load balancing or not.

Among the three factors – locality, topology and routing algorithm – the traffic locality is the one with strongest influence on the routing performance, while the routing algorithm does not influence the performance significantly.

The results show also that 4 VCs per physical channel provide enough network resources for applying the virtual channel reservation in a 10-by-10 network if the throughput requests are restricted to *b*/4 (in a mesh network) or *b*/3 (torus network). By increasing the traffic locality the throughput restriction can be increased to *b*/2 or *b*.

By requesting different throughput (b/4, b/3, b/2 or b) we restrict the number of VCs, k_i, used on a physical channel to 4, 3, 2 or 1 (see (5.2)). Therefore, the results for the different throughput requests can be interpreted also as results for a different number of number of VCs per physical channel.

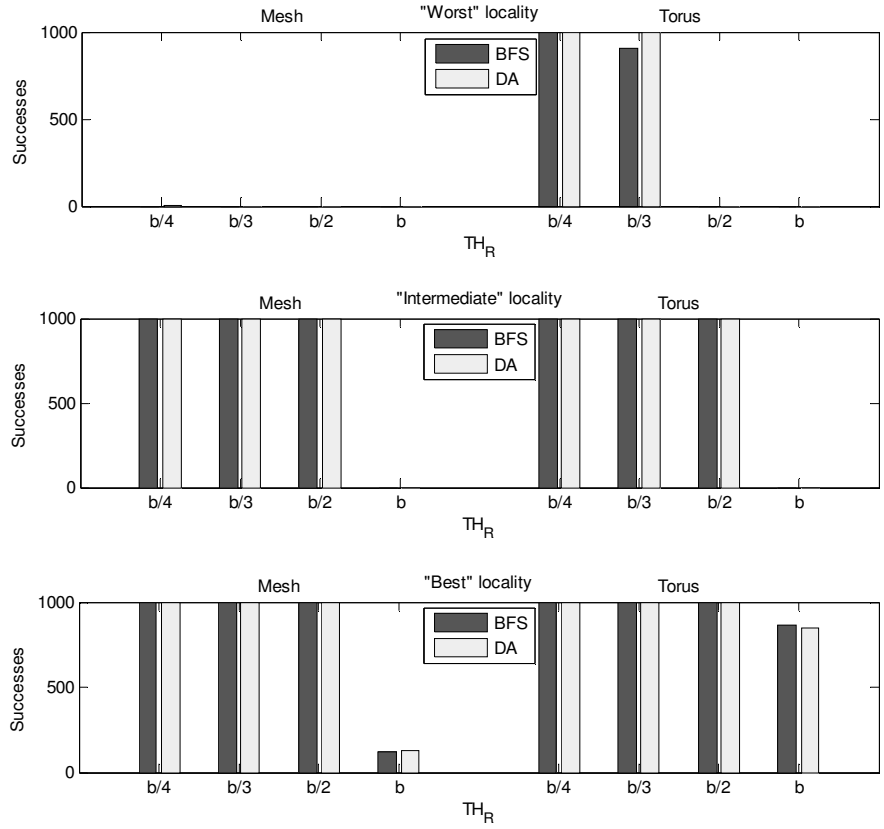


Figure 5.9: Number of successful samples in a 16-by-16 network

The number of successful samples in a network of size 16-by-16 is shown in Figure 5.9. Compared to a 10-by-10 network, the results deteriorate for traffic with no locality, while for local traffic the results are similar. The reason is that when the traffic does not show locality, the average communication distance increases with the increase of the network size. In contrast, for traffic that shows locality the communications are mostly local and the communication distances do not depend strongly on the network size.

The results show that the virtual channel reservation cannot be used in 16-by-16 network when traffic does not show locality, because the network cannot provide the requested GS connections. To apply the approach in a larger network, the number of VCs per physical channel can be increased. However, this solution is costly in terms of area. The other options are to use a network topology with a higher connectivity, like the torus, or to increase the communication locality. The last option is most profitable,

because, as we shall see, improving the traffic locality also helps to reduce the communication energy cost and generally improves the traffic conditions.

Again, and for the same reason as before, the routing algorithms do not show significant performance differences.

5.6.2. Detour cost

The routing function tries to route GS connections using shortest paths, but this is not always possible because some VCs along the shortest path might be occupied. In such a case the routing function takes a detour – a path which is not minimal. *Detour cost* is defined as the difference between actual path length and the distance between source and destination (the minimal path length). The better routing algorithms manage to route the traffic using shorter paths and therefore, with less detour cost.

The detour cost in a 10-by-10 network is shown in Figure 5.10. The presented figures give the sum of the detour cost of all 100 connections in the traffic pattern. The cases when routing is not possible (see Figure 5.8) and no data is available are marked with 'x'. In most of the cases the sum detour cost is less than ten hops, which is negligible compared to the sum distances of the 100 connections (at least 100 hops). This means that both algorithms manage to find short paths for most of the connections.

The detour cost exceeds 10 hops only in the cases when not all GS connections can be routed (see Figure 5.8). In these cases the routing function runs into a situation when the network resources are almost exhausted and finding a direct path is almost impossible. When routing the last few (2-3) connections, the function takes long detours. So the contribution to the detour cost shown in the figure comes mainly from few connections that are routed last, while the rest of the connections use almost minimal paths.

The detour cost becomes large only in the cases when not all requested connections can be routed (see Figure 5.8 and Figure 5.9), in which cases the routing function operates near its limits and the VC reservation cannot be effectively used. For the cases where the routing function freely routes all the connections, which are the cases where the VC reservation is intended to be used, the detour cost is negligible. Therefore, we can conclude that in cases of practical importance the detour cost is negligible.

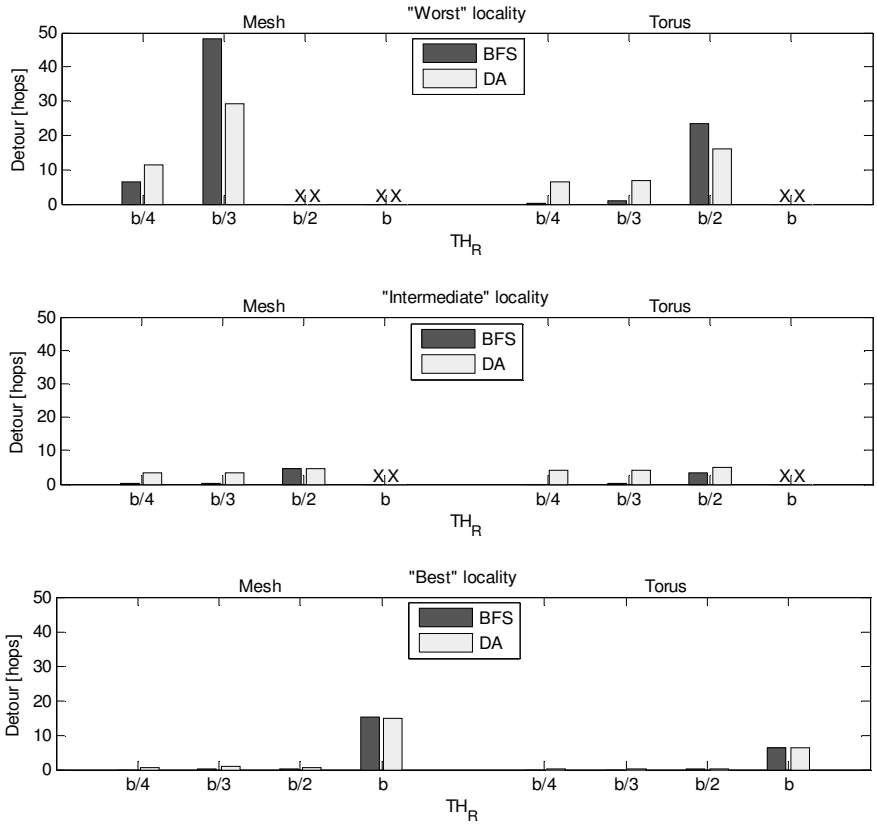


Figure 5.10: Detour cost in a 10-by-10 network

Figure 5.11 presents the detour cost in a 16-by-16 network. For the cases where the virtual channel reservation is applicable, the detour cost is again less than 10 hops and therefore negligible.

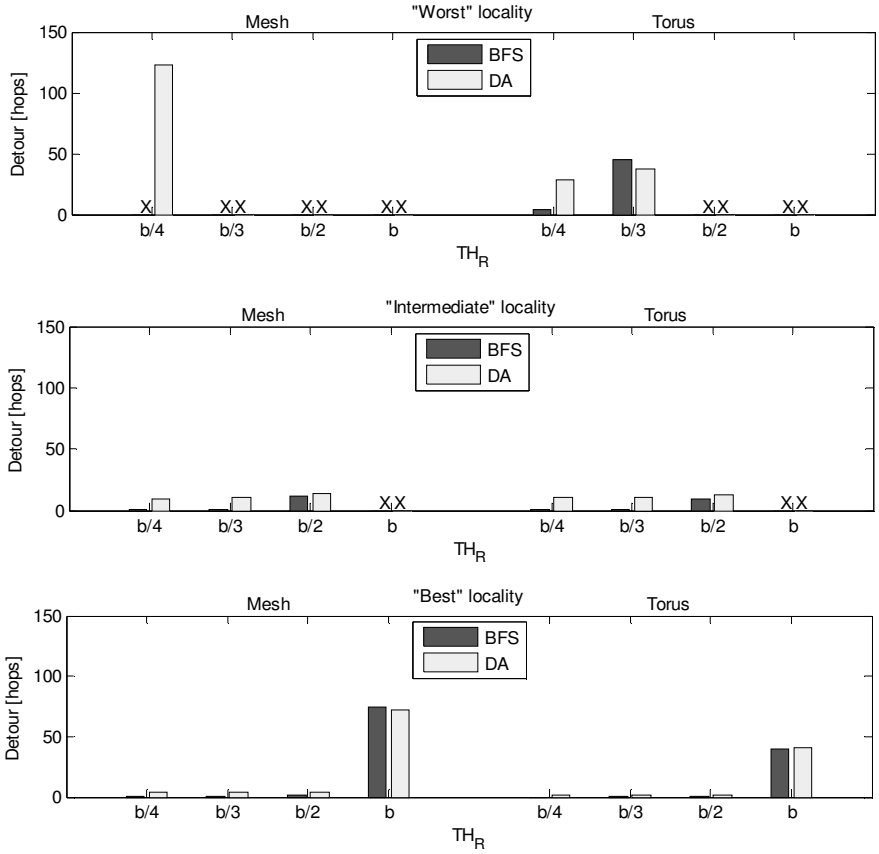


Figure 5.11: Detour cost in a 16-by-16 network

5.6.3. Communication energy cost

Wolkotte et al. [9] perform gate level power simulations with the VHDL model of our virtual channel router. Wolkotte estimates the average energy cost for traversing a router and uses this result to construct an energy model of the virtual channel network. He also constructs an energy model of his circuit switching network [9]. Here we use the same energy models to estimate and compare how the factors *routing algorithm*, *traffic locality*, *network topology* and *network size* influence the average communication energy cost for both networks – our virtual channel network and the Wolkotte's circuit switching network.

The energy models estimate the average energy cost for transporting a bit in the network, in [pJ/bit]. Both models have the form:

$$E_{ps} = E_R\left(N_{hop} + 1\right) + \left(0.39 + 0.12 l_{wire}\right) N_{hop} \tag{5.6}$$

Here l_{wire} is the length of a physical channel in mm. N_{hop} is the average network distance in number of hops. E_R stands for the energy cost for traversing a router. The energy cost

E_R for the virtual channel router and the circuit switch is derived by gate level power simulations for 0.13 µm technology and takes values E_{R_PS} = 0.98 [pJ/bit] and E_{R_CS} = 0.37 [pJ/bit]. The second term in the model estimates the energy for traversing the wires between two routers (the physical channels).

We use the energy models to estimate the average communication energy cost for three topologies – mesh, torus, and folded torus (see Figure 5.1). For the size of a PE we assume 1.5x1.5 mm or 2.25 mm^2 – the size of Montium processing tile [40]. The PEs are arranged in a two-dimensional array and interconnected using different topologies. The different topologies result in different channel length and different average communication distance. In a mesh, the length of the physical channels equals the edge length of the PE edge, so l_{wire} is 1.5 mm. In a torus topology, the wraparound channels cross the entire array of PEs. Thus, in a 10-by-10 network and a 16-by-16 network the length of the wraparound channels is respectively 15 mm and 24 mm. In a folded torus, the channels in the middle of the network cross two PEs, so they are 3 mm long. To take into account that the wraparound channels in a torus have a different length, (5.6) is modified to contain two terms that capture the energy contribution of the regular channels and the wraparound channels. For a 10-by-10 torus network the modified equation is:

$$E_{ps} = E_R\left(N_{hop}+1\right)+\left(0.39+0.12*1.5\right)\left(1-p\right)N_{hop} + \left(0.39+0.12*15\right)pN_{hop} \quad (5.7)$$

The network distance N_{hop} is replaced by the mean communication distance calculated from the simulation results. During the experiments we collect information about the utilization of the regular channels and the wraparound channels. This information is used to calculate the coefficient p in (5.7). The weight p stands for the fraction of hops that traverse a wraparound channel.

Figure 5.12 presents the results for the average communication energy cost in a 10-by-10 network. The left graph presents the results for the virtual channel network and the right graph presents the results for the circuit switching network. The results show that by exploiting traffic locality, the average communication energy cost in the system can be reduced by 50% to 70% for the different topologies. The reason is that with local traffic the average number of traversed channels and routers is smaller, which reduces the energy spent for communication.

In a condition of worst case locality, the torus topology reduces the energy cost compared with a mesh topology because the smaller network diameter of the torus keeps the average communication distance shorter. When the traffic shows locality and therefore shorter communication distances, the smaller torus diameter is not advantageous any more.

Because the length of the physical channels in a folded torus is doubled compared to torus, it should be expected that the energy cost also increases. However, the results show that changing the network topology from torus to folded torus does not change the communication energy cost notably. That is because for traffic without locality, the same amount of energy used for traversing the longer channels in the folded torus is used for traversing the wraparound channels in the torus. In other words, the average aggregated channel length traversed by the messages is equal in both topologies and no difference in the energy cost is seen. For local traffic a difference can be seen but it is

small, because the communication energy cost is dominated by the energy for traversing the routers.

The routing algorithm influences the communication energy cost by the detour cost – higher detour cost entails more energy for communication. However, the detour cost is negligibly small (see Section 5.6.2) and the influence of the routing algorithm on the communication energy cost is insignificant.

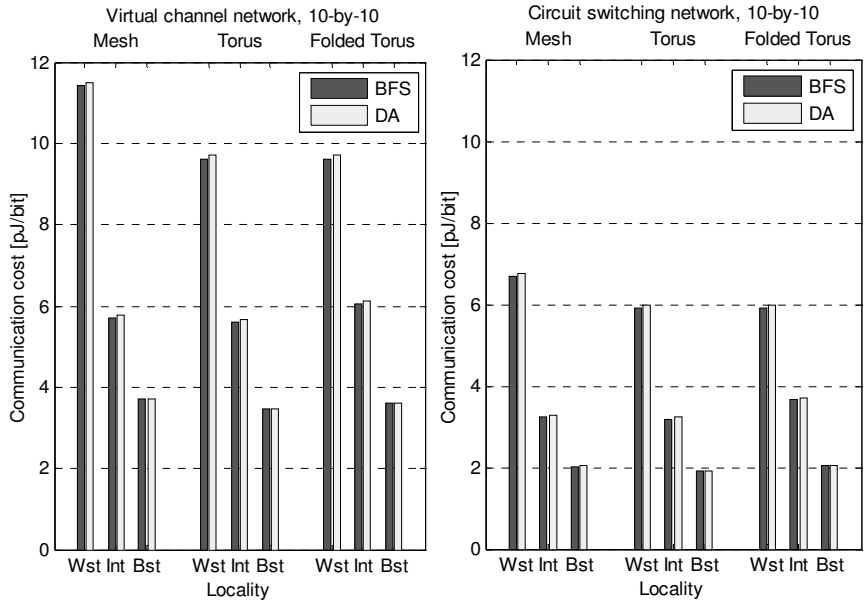


Figure 5.12: Average communication energy cost in a 10-by-10 packet switching and circuit switching networks

The right graph in Figure 5.12 presents the energy cost results for a circuit switching network of size 10-by-10. The energy cost for the circuit switching network is smaller compared to the results for the virtual channel network. That is because the circuit switches are simpler than the routers in the virtual channel network and the energy cost for their traversal is lower. However, the circuit switches are also less flexible (see Chapter 2)

Figure 5.13 presents the energy cost results for a network of size 16-by-16 nodes. The results differ from those for a 10-by-10 network, mainly for worst case traffic locality. That difference is due to the dependency of the average communication distance on the network size – increasing the network size increases the communication distances and respectively, the energy cost. For local traffic this dependency is weak and changing the network size does not noticeably change the energy cost. No data for the BFS are present for worst locality in mesh because there are no successful samples for 16-by-16 network (see Figure 5.9).

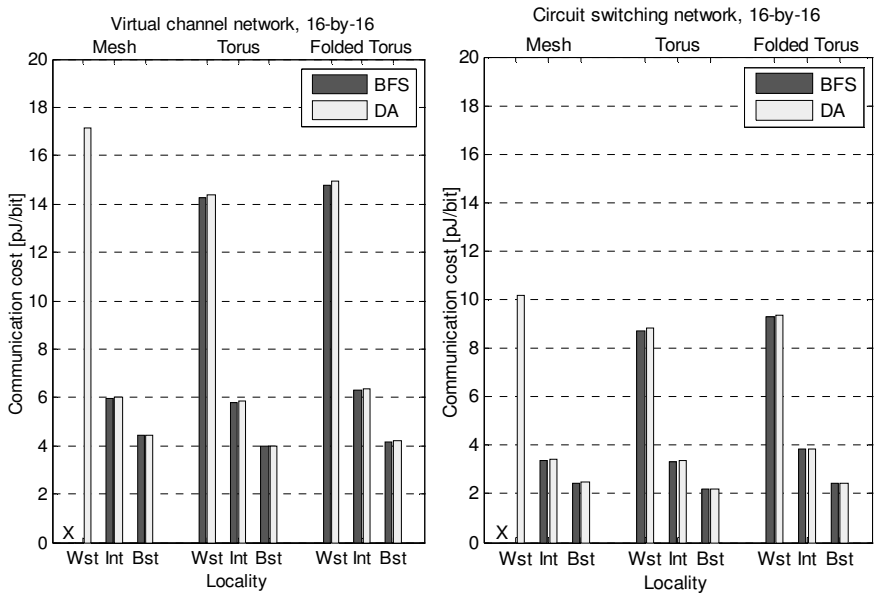


Figure 5.13: Average communication energy cost in a 16-by-16 packet switching and circuit switching networks

5.6.4. Performance in the presence of BE traffic

So far we have been discussing the performance of the routing function assuming that the system requests only GS. Indeed, the GS traffic accounts for 90% of all the traffic in our system, but still there is 10% BE traffic that must also be served by the network. The BE traffic has much lower resource requirements than the GS traffic. The reason is not only that BE traffic uses a smaller fraction of the total traffic, but also the fact that the BE traffic may share VCs. In contrast with the GS traffic where each GS connection uses a separate, reserved path over the VCs, the BE traffic may share paths – several BE connections may share a single VC for traversing a physical channel. Therefore, to support the BE traffic at most one VC per physical channel is allocated for BE connections.

Let us again consider a network with four VCs per physical channel (K=4), but this time one VC per physical channel is used for BE traffic and the remaining three VCs can be used by GS connections. In such a network the GS connections use physical channels where the number of occupied VCs, k_i, is 2, 3 or 4. Therefore, the throughput guaranteed by a GS connection can be respectively $b/2$, $b/3$ or $b/4$ (see (5.1)). Results for the number of successful samples in such a network are presented in Figure 5.14 and Figure 5.15. The main conclusion that can be drawn is that when BE traffic is present, virtual channel reservation cannot be used in mesh topology without traffic locality. The other option is to increase the number of VCs to 5. The influence of the different factors on the results is the same as for the results for GS traffic only.

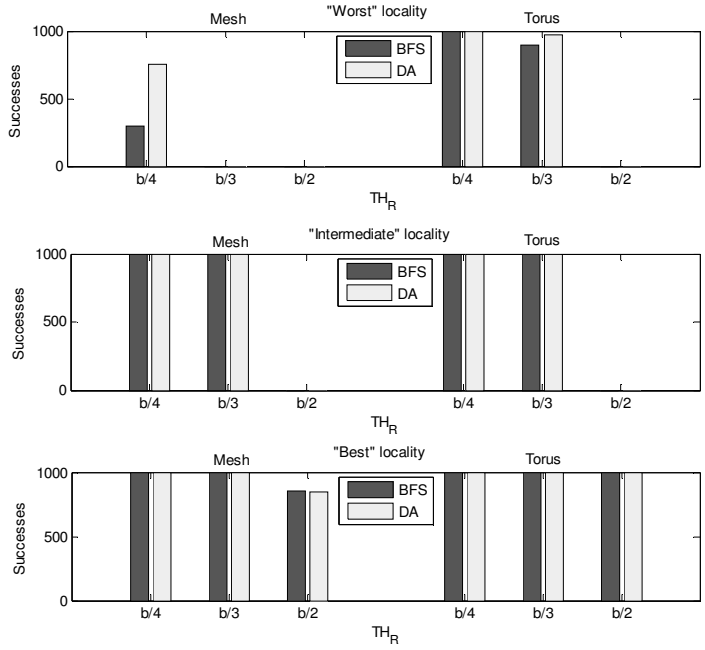


Figure 5.14: Number of successful samples in a 10-by-10 network with BE traffic present

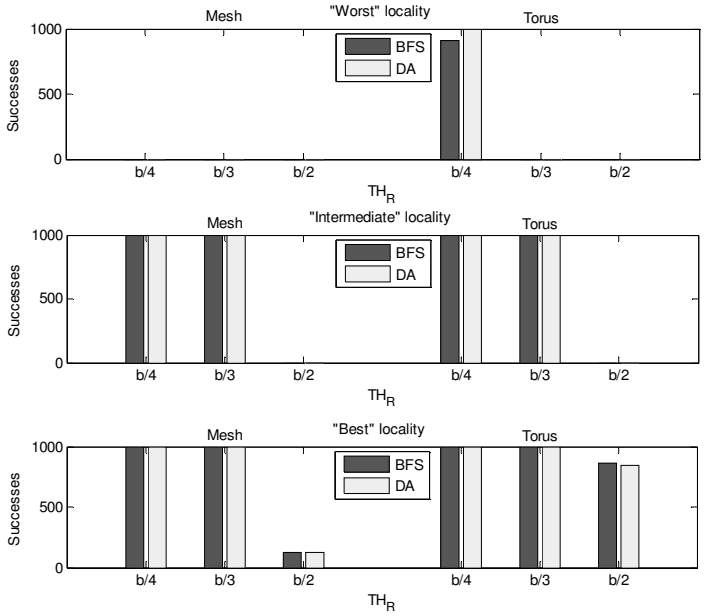


Figure 5.15: Number of successful samples in a 16-by-16 network with BE traffic present

5.7. Conclusion

In this chapter we evaluate how the performance of the virtual channel reservation approach depends on several factors: traffic locality, routing function, network topology and network size. For different combinations of these factors we investigate whether virtual channel reservation can provide the guaranteed communication services demanded by the streaming DSP applications in our system. We also estimate the overhead that has to be introduced at system level to support the virtual channel reservation. The average communication energy cost in the network is also estimated.

The results show that virtual channel reservation can be applied in a network with four virtual channels per physical channel. However, the performance of the approach depends on the traffic locality. Traffic locality weakens the effect of the network topology and the network size on the performance, thus making the system more scalable. By exploiting traffic locality the average communication energy cost can be reduced by 50% to 70%. The results also show that for our applications traffic locality is not difficult to achieve in systems of size 10-by10 and 16-by-16 nodes. Using a straightforward mapping strategy we manage to achieve average distance of only two hops, as 88% of the communications in the system are with neighbouring nodes.

Comparing the influence of the different factors, the network topology comes at a second place. A network topology with a lower diameter and higher connectivity is beneficial for the system performance when the system traffic does not show locality. However, in a presence of traffic locality the network topology loses its influence. The same is true for the network size. Hence, if the system manages to maintain the locality, then simpler network topology can be used.

The routing algorithm has the weakest influence on the routing performance. With or without load balancing, the routing function performs in a similar way. Hence, the Breadth-first search algorithm is preferred over Dijkstra's algorithm because of its linear complexity.

The overhead introduced by the routing function is small enough to allow application of virtual channel reservation at run-time. A routing function based on the Breadth-first search algorithm routs a connection in less than a millisecond. It takes only a few milliseconds to provide all the connections requested by an application in our target domain.

We may conclude that the virtual-channel reservation approach is fast and efficient enough to be applied for providing service guarantees at run-time. A direction for improvement of our work is to investigate the possibilities for run-time traffic optimisation, e.g. rerouting connections. Furthermore, the performance of the routing function can be improved by employing a more advanced algorithm like A*.

Chapter 6

Network integration*

This chapter discusses the how our network is integrated in the system and how predictable system operation is achieved. Instead of the traditional fully-static scheduled system organisation we use an alternative approach which reduces the application scheduling complexity such that scheduling can be done at run-time as required in our dynamic system.

6.1. Introduction

Until now we have discussed our NoC solution separately from the system that it serves. We presented how the NoC operates and what communication services it provides. In this chapter we move our attention to the operation of the entire system and examine how the NoC integrates in it. However, an elaborate discussion of the high level system organisation is beyond the scope of this thesis, so our intention is only to describe how the system uses the network and to identify the costs incurred by employing the NoC.

Since our multiprocessor system is dynamically reconfigurable, the run-time overhead incurred by employing the network must be small enough not to compromise the dynamic behaviour of the system. Since the system is intended for real-time applications, it must be able to provide the applications with integral performance guarantees by providing computation and communication guarantees. Starting a new real-time application at run-time is the most time critical system task for a dynamic system; the system must perform this task fast and transparently, as at the same time performance has to be guaranteed to the application. For that reason, we discuss mostly how the system starts new applications and how it guarantees application performance.

In most network-based SoCs proposed so far, predictable system operation is achieved by employing some form of fully-static scheduling. In general, starting an application on a statically scheduled systems is an NP-complete problem [69, 73]. Such a high-complexity problem is acceptable for these SoCs since they are statically configured and their configuration is computed at compile-time. However, since our system is dynamic, part of the configuration is computed at run-time [72] and to keep the time overhead low the computation must be of low complexity.

Instead of fully-static scheduling, in our system we use a different approach called self-timed scheduling [73]. In combination with the simple structure of our streaming application and the guaranteed services provided by the NoC, this system organisation

* Major parts of this chapter have been presented at the EUROMICRO conference on Digital System Design [4] and at the Communicating Process Architectures Conference [8].

reduces the computational complexity for scheduling of applications. Starting a real-time application in our system requires only solving a small system of linear inequalities. To the best of our knowledge, our system is the first NoC based SoC to employ self-timed scheduling for achieving predictable system performance at low run-time cost.

6.2. System operation

As discussed in Chapter 1, our system consists of an array of heterogeneous processing elements (PE). Each PE is connected to the NoC and communicates with other PEs only through the network. The system is centralised – there is one PE that acts as a central authority and starts application tasks on the other PEs in the system. This happens dynamically, at run-time, while some applications are already running in the system. Since the system is controlled by the central PE, when we talk about an action taken by the system we mean the action taken by the central PE.

At the system level the NoC is represented by the communication services it provides; these are the guaranteed services (GS) and the best effort services (BE) of our NoC. As discussed in Chapter 5, the system requests communication services by calling a central routing function. For example, the system may call the function to provide a GS connection between two PEs for the communication between the tasks running on these PEs. (In Chapter 5 we argued that providing such a connection takes less than a millisecond.) The routing function runs on the central PE, so requesting a connection simply means calling a routine which does not involve network communication.

6.2.1. Starting an application

In general, to run an application on a multiprocessor system, four steps have to be performed, as indicated in Figure 6.1: partitioning, compiling, mapping and scheduling. In the first step, *partitioning*, the application is partitioned into tasks that will run in parallel on separate PEs. In the second step, *compiling*, the separate tasks are compiled for the target PE types. In the third step, *mapping*, the actual PEs where the tasks will run are selected among the available suitable PEs in the system. In the fourth step, *scheduling*, the proper timing behaviour of the application is provided such that its performance requirements are met.

There are two criteria for application partitioning. The first one is grouping into separate task parts of the application with computation demands that match the capabilities of different specialised PEs. By matching computational demands with PE capabilities the system performance and efficiency are improved. The second criterion is minimisation of the communication between tasks. Application partitioning is an aspect of parallel programming, which is a difficult and widely researched topic that is beyond the scope of this thesis.

During compilation, besides the usual goals of code size and speed optimization, the compiler is expected to provide information about the execution time of the compiled task (in clock cycles). This information is needed during scheduling for predicting the tasks timing behaviour. Compiler technology and execution time analysis are also beyond the scope of this thesis.

When mapping an application efficiently, the optimisation criterion is improving the communication locality by mapping communicating tasks on neighbouring PEs. Finally, the objective of the application scheduling is to provide such a performance for

the application tasks and communications between them that the required overall performance is guaranteed.

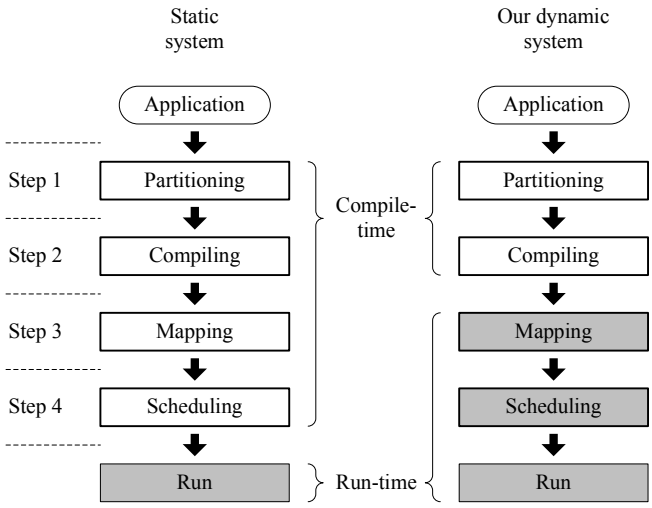


Figure 6.1: Steps performed when starting an application

All the NoC based systems proposed so far are static (statically configured) [34, 52], which means that the four steps described above are performed at compile time. In a static system the system configuration is generated and loaded into the system once, before the system start up, and then it is used for the entire operation period of the system. Performing all the preparations at compile time is advantageous, as the time and resource limitations for computing the configuration is minimal. Hence, in static systems the computational complexity of the four steps performed for starting an application is not a critical issue.

In a dynamic system like our, however, applications are started at run-time so not all the four preparation steps can be performed at compile time. As shown in Figure 6.1, in contrast with static systems, our dynamic system performs mapping and scheduling at run-time. Hence, these two steps must be performed fast and efficiently. Since mapping and scheduling are performed by the SoC it self, therefore using limited computational power and resources compared with the resources available at compile time, the mapping and scheduling algorithms must be as simple as possible.

Fortunately, the structure of our applications simplifies mapping and scheduling in our system. As discussed in Chapter 1, the applications in the system have a simple pipeline structure. The simple application structure eases the mapping task compared to applications with a more irregular structure since the mapping deals with fewer dependencies between the tasks, thus with fewer constraints. Hence, we expect mapping in our system to be faster and to achieve high locality. The latter was demonstrated in Chapter 5 for a homogeneous system (all PEs are uniform). The mapping experiments there show that the mapping procedure manages to reduce the distance of 88% of the communications to one hop and to achieve an average communication distance in the system of less than 2 hops (see Figure 5.4 to Figure 5.7).

6.2.2. Scheduling approaches

Compared to a fully-static system, in our system application scheduling is simplified thanks to a combination of applications with a simple structure and the specific approach we take for providing predictable system operation. In a system with *fully-static* scheduling all the computation and communication in the system is driven by a global schedule; the start time and duration of all tasks and communications is fixed by the schedule. To schedule a new application, the system has to recompute the global schedule for all applications. This may be a simple task when the system is not busy (no other applications are running), but in general when the system has been running for some time, computing a schedule is a NP hard problem [69, 73]. Such a complexity is already difficult to cope with in static systems where scheduling is performed at compile time, but it is not acceptable for our dynamic system.

In our system we employ a technique, called *self-timed* scheduling [73], for achieving predictable system operation. This technique does not require computation of a global schedule, it relaxes the scheduling constraints compared to fully-static scheduling and thus reduces the scheduling complexity.

Although applying a static schedule is sufficient for providing predictable system and application performance, fully-static scheduling applies more restrictions on the system operation than is necessary for achieving predictability. For example, if we consider a video application, the required fixed frame rate at the output of the application does not imply that a given pixel in the frame must be ready at the exact time as is provided by the static schedule. It only implies that the pixel must be ready any time before the frame release time. The fixed execution times in a fully-static system also cause difficulties when the execution times are data dependent. To fit such an application into a static schedule, the variations have to be compensated by introducing data buffers that are appropriately sized for the specific application.

In contrast with fully-static scheduling, self-timed scheduling does not really schedule the system operation, but the computation and communication is data driven; tasks and communications are started when the required data is present and space is available for storing the result. Self-timed scheduling only requires that the upper bound on the task execution times and the communication times are known. In our system these are not difficult to obtain because of the specific class of applications we are interested in. The communication times are a direct function of the provided network throughput guarantees and the amount of communicated data. Since the PEs are single task processors and the tasks running on them do not interfere with other tasks, the task execution times are the same us times provided by the compiler. The worst case execution time of tasks running on a shared processor can also be derived [83].

The cost to pay for employing self-timed scheduling is a small amount of additional hardware that provides the data driven operation. The additional hardware required constitutes of handshake circuits that implement blocking read and blocking write for data exchange between the NoC and the PEs. However, since a self-timed system naturally handles variable (data dependent) execution times, it avoids the application dependent buffering required in fully-static systems, so on the other hand self-timed scheduling simplifies the architecture.

Another reason to prefer self-timed scheduling over fully-static scheduling is that fully-static scheduling is difficult to apply in GALS (Globally-Asynchronous Locally-Synchronous) systems. To work to a global schedule, the system must have a global notion of time, which is missing in GALS systems because no global clock is

distributed there. In contrast, self-timed scheduling does not require global notion of time and so is suitable for GALS systems. The ability to apply the chosen scheduling approach in a GALS system is advantageous since it is expected in the near future that more and more systems will be designed as GALS systems because of the cost of the global clock distribution in the future semiconductor technologies (see Chapter 1).

6.3. Self-timed operation

We continue by presenting how streaming applications run on our self-timed system. Figure 6.2 presents a streaming pipeline application mapped on our system. The application pipeline consists of *n* tasks, denoted as P_1 to P_n, running on separate PEs. The processed data items are transported between the PEs by the NoC. The NoC and the PE exchange data through the PE local memory (MEM); the received data items are loaded in the MEM and after processing they are read from the MEM and transmitted to the next PE. The task running on a PE reads from the MEM the arrived data items, processes them and stores the results back into the MEM. The data exchange between a PE and the NoC is handled by a network interface (NI) unit which implements the blocking read and write to the MEM needed to provide the self-timed behaviour. For example, when the input data buffer reserved in the MEM for arriving data items is full, the NI stops receiving and holds the next data item blocked in the network. This respectively blocks the data item transmission in the previous PE. Similarly, the data transmission is blocked when the output data buffer is empty and there is not yet a next data item ready to be transmitted. In the same way a task running on a PE blocks when the input data buffer in the MEM is empty or when the output data buffer is full.

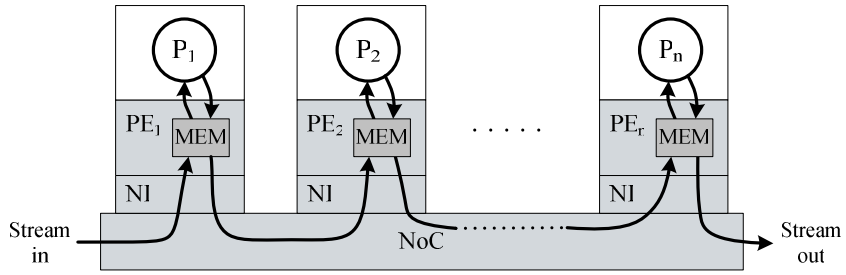


Figure 6.2: A streaming application mapped on a multiprocessor architecture

The performance of the application is determined by the time the tasks need to process a data item and by the time needed to communicate a data item between two PEs (provided no blocking occurs). These times we call respectively processing and communication time. To predict the application performance, all the processing and communication times of the application must be known. When these times are data dependent, their upper bound must be known.

Besides the processing and communication times, the application performance also depends on whether processing and communication in a PE can be performed simultaneously. The parallelism between processing and communication is restricted by the MEM. Since all of the three operations receiving, processing and transmitting a data item requires access to MEM, these operations can be performed in parallel only if MEM supports parallel access. Based on the simultaneous access supported by the MEM we can distinguish the following three cases of parallelism in a PE:

i). single access – the MEM allows only one access at a time and so only one operation can be performed at a time. The operations receive, process and transmit are performed sequentially.

ii). double access – the MEM can be accessed by two entities simultaneously. The processing and communication are performed in parallel.

iii). triple access – the MEM can be accessed by three entities simultaneously. Receiving, processing and transmitting are performed in parallel.

To allow multiple accesses, the MEM can be implemented either as multi-port memory or as multiple banks of single port memories. Thus, the three cases above correspond to the following memory organisations: single-port MEM, dual-port MEM and triple-port MEM or MEM consisting of single, two or three memory banks. In any case, the cost of enabling more parallelism is a higher memory cost in terms of area and energy. If separate memory banks are used, the number of banks and respectively the area is proportional to the number of memory ports. In case of a single bank multi-port memory, the area increase is even higher because besides of the larger memory size needed, the memory complexity increases too. The energy cost of a memory access scales linearly with the memory size [81].

To predict the application performance in a self-timed system (for the purposes of scheduling) we build an application model that captures all the aspects that influence the applications performance – the processing and communication times, the effect of blocking due to the self-timed operation and the parallelism enabled by the PEs memory organization. For modelling we use a standard modelling technique called homogeneous synchronous data flow (HSDF) graphs, which we now briefly present.

6.4. HSDF graphs and MCM analysis

Homogeneous Synchronous Data Flow (HSDF) [50, 73] is a model of computation suitable for describing parallel DSP applications. The model is based on a special type of directed graphs, called HSDF graphs. A vertice of an HSDF graph is called an *actor*; it models some activity. An actor is characterised by an execution time given as a label of the actor. A graph edge represents a dependency between the actors at the end points of the edge. The actors interact by exchanging *tokens* over the connecting edges. In principle an edge behaves like an unbounded FIFO buffer where the tokens are stored.

When there is at least one token preset on each input edge of an actor, the actor is executed (also called fired). After a time period equal to its execution time the actor produces one token on each of its output edges. To provide that a second execution cannot start before the first one has finished the actor is assigned a self-edge with a single token.

Figure 6.3 shows an example HSDF graph that models a bounded FIFO buffer with a capacity of two data items. The two actors model respectively the FIFO write and FIFO read operations. Writing into the FIFO takes time ET_1 and reading from the FIFO takes time ET_2 – these are the actors execution times. The data items that are written into the FIFO arrive as tokens on the input edge IN and the data items that are read depart as tokens on the edge OUT. The number of tokens on the cycle between the two actors corresponds to the buffer capacity. Each token on the upper edge corresponds to an empty buffer space and each token on the lower edge corresponds to a full buffer space. When a token arrives on the edge IN, the actor *Write* is executed, consuming an empty buffer space and producing a full buffer space. Subsequently, executing *Read* consumes a full buffer space and produces an empty buffer space and a data item on the

OUT edge. In the FIFO model shown, the data are read out of the buffer immediately after they are written. In a more practical example additional input edges that enable the execution of the actors may be present.

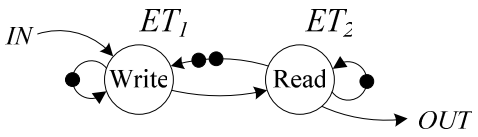


Figure 6.3: HSDF model of a FIFO buffer of capacity two data items

Given an HSDF graph, we can derive its throughput in terms of the number of tokens processed per time unit by applying a standard analysis technique for synchronous data flow models called Maximum Cycle Mean (MCM) analysis [15, 73]. MCM analysis examines all simple cycles in an HSDF graph *G* and determines their *cycle mean.* The cycle mean of a simple cycle is defined as the ratio between the sum of the execution times of all the actors on the cycle and the number of tokens on the cycle:

$$cycle\_mean = \frac{\sum_{i \in c} ET_i}{tokens(c)}, \tag{6.1}$$

Here *c* is a simple cycle in *G*, the sum is taken over all actors that belong to *c*. *tokens(c)* gives the number of tokens on the cycle *c*. MCM analysis consists of calculating the cycle mean for every simple cycle in the analysed graph *G* and selecting the maximum calculated cycle mean. Thus, the MCM of a graph *G* is:

$$MCM_G = \max_{c \in G} \left[\frac{\sum_{i \in c} ET_i}{tokens(c)} \right], \tag{6.2}$$

Here the *max* function is taken over all simple cycles in *G*. The cycle with the maximal cycle mean is called the *critical cycle*, and its cycle mean determines the graph throughput. The throughput of the analysed graph in [token/s] is the reciprocal of the maximum cycle mean:

$$TH_G = \frac{1}{MCM_G} \tag{6.3}$$

As an example we calculate the throughput of the FIFO model from Figure 6.3. The graph in Figure 6.3 contains three cycles – two self edges with one token and one cycle with two tokens. The means of these cycles are ET_1, ET_2 and $(ET_1+ET_2)/2$ and the MCM is therefore $max(ET_1, ET_2, (ET_1+ET_2)/2)$. The graph throughput is then TH = $1/max(ET_1, ET_2, (ET_1+ET_2)/2)$.

HSDF graphs have two important properties [16]: periodicity and monotonicity. The periodicity property means that after a transient period in the beginning, the execution of a strongly connected HSDF graph will exhibit periodic behaviour. The monotonicity property means that the throughput of a HSDF graph is a non-decreasing

function of the execution time of the actors, or in other words, decreasing these execution times may only lead to equal or higher throughput. In our models the execution times can vary, but we label the actors always with the worst case execution times. Hence, according to the monotonicity property, by applying the MCM analysis we derive the worst case throughput of the graph.

Applying MCM analysis requires finding all the simple cycles in the HSDF graph, which depending on the graph might not be a simple task to perform at run-time. However, in the case of our system, cycles are found and analysed already at compile time, as at run-time only derived results are used. Thus, the more complicated part of the modelling process is performed off-line and it is not an obstacle for the real-time system operation.

6.5. Predicting throughput of an application

We use the HSDF model and apply MCM analysis for predicting the throughput of streaming applications running on our system. We need this prediction in the scheduling step (see Figure 6.1) to guarantee the throughput of the application being started. After the mapping the application is modelled as a HSDF graph using the information about the memory organization of the PEs where the application runs and the actual execution times of the tasks.

In this section we present how the applications are modelled and their throughput is derived. We first model a single application task running on a PE, considering the three options for the PE memory organisation. Then we extend the model for a complete application.

6.5.1. Throughput of a single application task

Consider a single application task running on a PE. As discussed earlier, we have three options for the PE memory organization: single-port, dual-port and triple-port memory. For each of the three cases we construct an HSDF model of a task and then we apply MCM analysis on the model to derive the task throughput.

To model a task as an HSDF graph we need three actors: one for receiving a data item, one for processing and one for transmitting (these are the three basic operations performed by the PE). We refer to these actors as the receiving, processing and transmitting actors. To the receiving and transmitting actors we also refer as the communication actors.

We adopt the following notation. The processing actor of task i is denoted as P_i and its processing time is denoted as PT_i. The sending actor, which is the communication actor with the next task, is denoted as C_i and its sending time is denoted as CT_i. Respectively, the receiving actor, which is the communication actor with the previous task, is denoted as C_{i-1} and its receiving time is denoted as CT_{i-1}. The ratio between the communication and processing times depends on the complexity of computation and the amount of communicated data. For our system we expect the processing and communication times to be of the same scale.

Single-port memory organisation

A single-port memory allows only one of the three actors to be executed at a time. Following the data dependencies, the natural order in which the actors are executed is: receiving, processing and transmitting. After each data item this cyclic pattern is

repeated for the next data item. We model this task behaviour with the HSDF graph of Figure 6.4. The graph contains one cycle with one token. The token in the cycle is circling around the actors allowing only one of them to execute at a time. The token can be interpreted as a grant for memory access – the actor that currently has the token has access to the memory. The three self edges provide that an actor fires only after its previous execution finishes.

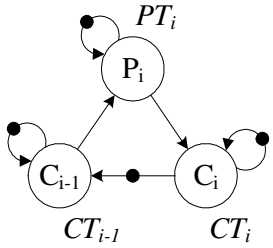


Figure 6.4: An HSDF model of a task running on a PE with a single-port data memory

Applying MCM analysis we derive the throughput of the graph. From equations (6.2) and (6.3) we find the MCM and the throughput of the graph in [token/s]:

$$MCM_1 = \max\left[CT_{i-1}, PT_i, CT_i, \left(CT_{i-1} + PT_i + CT_i\right)\right] = \\ = CT_{i-1} + PT_i + CT_i \tag{6.4}$$

$$TH_1 = \frac{1}{CT_{i-1} + PT_i + CT_i} \tag{6.5}$$

Since a grant for memory access is given for the time of processing/communicating a single data item, this result is interpreted as the worst case throughput in [data item/s] of the task *i* running on a PE with single-port data memory. From (6.5) it follows that to guarantee a lower bound TH_R on the stage throughput ($TH_R \leq TH_1$), we must have that:

$$CT_{i-1} + PT_i + CT_i \leq \frac{1}{TH_R} \tag{6.6}$$

This result is illustrated by the time-diagram in Figure 6.5. The diagram represents one period of the task operation. In this period one data item is received, processed and sent. To provide a throughput of TH_R, the operation period must be less than or equal to $1/TH_R$, which is expressed by (6.6). The task operates as follows. The processing of a data item (PT_i) starts immediately after the data item is received (CT_{i-1}). Because of blocking induced by self-timing a waiting period ($W_{i,1}$) may be introduced after the processing and before transmitting the data item (CT_i) when the next stage is not yet ready to receive. A waiting period ($W_{i,2}$) may also be introduced after transmitting the data item when the previous stage is not immediately ready to send the next data item.

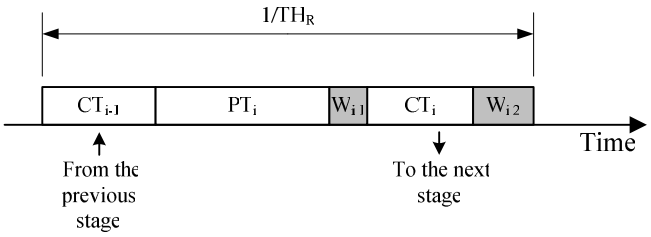


Figure 6.5: One period of the operation of a PE with single-port memory

Dual-port memory organisation

On a PE with a dual-port memory two of the three actors can be executed at a time. The PE is organized such that grants for the memory ports are given to the actors in a cyclic fashion following the data dependencies. This behaviour is modelled also by the graph in Figure 6.4, but this time with two tokens on the main cycle, each token corresponding to a memory port. An example time-diagram of the PE operation is presented in Figure 6.6. While the PE is processing the current data item, the NI transmits the previously processed data item and then receives the next data item. Because of blocking waiting times $W_{i,1}$, $W_{i,2}$ and $W_{i,3}$, may be introduced.

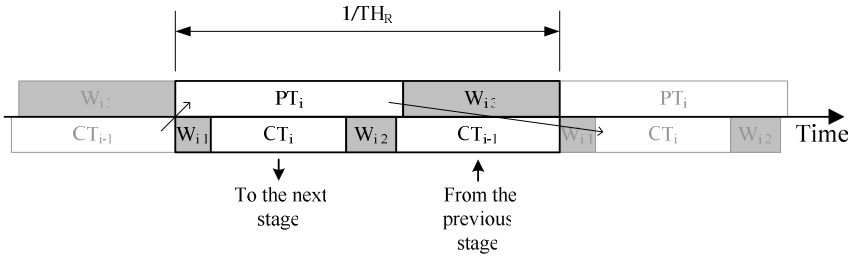


Figure 6.6: One period of the task operation on a PE with dual-port memory

Applying MCM analysis to the graph with two tokens, the throughput is:

$$TH_2 = \frac{1}{\max\left[CT_{i-1}, PT_i, CT_i, \left(\frac{CT_{i-1} + PT_i + CT_i}{2}\right)\right]} \tag{6.7}$$

This is the worst case throughput in [data item/s] of the task *i* running on a PE with dual-port memory. From (6.7) it follows that to guarantee a lower bound TH_R on the stage throughput ($TH_R \leq TH_2$), the following system of inequalities must hold:

$$\begin{cases} \dfrac{CT_{i-1} + PT_i + CT_i}{2} \le \dfrac{1}{TH_R} \\ CT_{i-1} \le \dfrac{1}{TH_R} \\ PT_i \le \dfrac{1}{TH_R} \\ CT_i \le \dfrac{1}{TH_R} \end{cases} \tag{6.8}$$

The first inequality in (6.8) states that the time for receiving, processing and transmitting a data item must be less than two times the data arrival period. This is two times longer than the time constraint in the single-port memory case. A comparison between Figure 6.5 and Figure 6.6 shows that with a dual-port memory the PE has more time to process a data item and hence can work at a lower frequency than with a single-port memory. With a dual-port memory it is possible to achieve 100% PE utilization, which is not possible with the single-port memory because of the alternation of processing and communication. In the case of a dual-port memory the communication times can also be longer than in the case of single-port memory. This means that communication channels with lower throughput are needed to communicate the same amount of data. Thus, the communication throughput demands are reduced and the NoC traffic conditions are relaxed.

The organisation scheme of PE with a dual-port memory given here is one of several possible; however it is the one achieving higher throughput.

Triple-port memory organisation

When a task is running on a PE with a triple-port memory, all three actors can execute at the same time. The task operation in this case is illustrated by the example time-diagram in Figure 6.7. The waiting times shown in the figure may be introduced due to blocking.

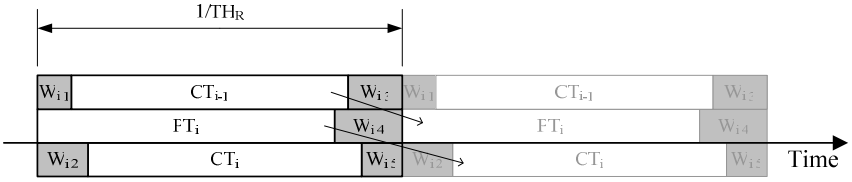


Figure 6.7: One period of the operation of a PE with three-port memory

The HSDF graph that models the task operation is the same as in the previous two cases, but now the graph cycle contains three tokens. Applying MCM analysis to this graph, the task throughput is:

$$TH_3 = \frac{1}{\max\left(CT_{i-1}, PT_i, CT_i\right)} \tag{6.9}$$

Equation (6.9) gives the worst case throughput in [data item/s] of task *i* running on a PE with triple-port data memory. From (6.9) it follows that to guarantee a lower bound TH_R on the task throughput ($TH_R \leq TH_3$), the following system of inequalities must hold:

$$\begin{cases} CT_{i-1} \leq \dfrac{1}{TH_R} \\ PT_i \leq \dfrac{1}{TH_R} \\ CT_i \leq \dfrac{1}{TH_R} \end{cases} \tag{6.10}$$

The triple-port memory relaxes the constraints on the execution times even further than the dual-port memory case. The communications can be extended along the entire period $1/TH_R$ and the required communication throughput can be reduced to the minimum. It is also possible to achieve 100% PE utilisation.

6.5.2. Comparison

In all three cases of PE memory organization a task is modelled as a simple HSDF graph and predicting the task performance requires only a straightforward analysis. Since the graphs depend only on the PE memory organization and not on the specifics of the application, the modelling and the analysis can be done off-line. What needs to be done at run-time for predicting the task performance is simple: i) choosing the analysis result associated with the memory organization of the PE where the task runs and ii) substituting the actual processing and communication times.

Table 6.1 summarises the task performance results. For the three cases of memory organisation the table gives the task worst case throughput and the constraints on the execution times implied by a given application throughput bound TH_R.

Table 6.1: Summary of the results for a single pipeline stage running on a PE

Mem	Worst case throughput	Constraints
Single-port	$TH_1 = \dfrac{1}{CT_{i-1} + PT_i + CT_i}$	$CT_{i-1} + PT_i + CT_i \leq \dfrac{1}{TH_R}$
Dual-port	$TH_2 = \dfrac{1}{\max\left[CT_{i-1}, PT_i, CT_i, \left(\dfrac{CT_{i-1} + PT_i + CT_i}{2}\right)\right]}$	$\dfrac{CT_{i-1} + PT_i + CT_i}{2} \leq \dfrac{1}{TH_R}$ $CT_{i-1}, PT_i, CT_i \leq \dfrac{1}{TH_R}$
Triple-port	$TH_3 = \dfrac{1}{\max(CT_{i-1}, PT_i, CT_i)}$	$CT_{i-1}, PT_i, CT_i \leq \dfrac{1}{TH_R}$

Note: (CTi-1, PTi, CTi ≤ 1/THR) stands for (CTi-1 ≤ 1/THR and PTi ≤ 1/THR and CTi ≤ 1/THR);
THR – requested throughput

Comparing the worst case throughput we have:

(6.11) $$TH_1 < TH_2 \leq TH_3,$$

which confirms the intuitively expected result that increasing the number of MEM ports improves the PE performance. However, as discussed earlier, the memory area and energy cost increase proportionally.

PE utilization

Considering system efficiency, it is desirable that the PEs are highly utilized. The PE utilization ρ is defined as the fraction of the time the processor is busy processing data items. For a periodic task running on a single-task PE, the PE utilization can be defined as the ratio between the period PT_i in which the PE processes a data item and the data arrival period. For a real-time application with a throughput TH_R the average data inter-arrival period is $1/TH_R$. Thus:

(6.12) $$\rho = TH_R * PT_i$$

If two PEs are running the same task, the one with higher utilisation has more time to perform the task. Hence, it can operate at a lower clock frequency which saves energy. Therefore, higher PE utilisation is desirable also because it improves the system power efficiency.

The maximal PE utilization that can be achieved for the different memory organizations is derived by substituting (6.12) in the constraint inequalities given in Table 6.1. The results are presented in Table 6.2.

Table 6.2: Maximal achievable processor utilization

Mem.	Maximal processor utilisation
Single-port	$\rho \leq 1 - TH_R(CT_{i-1} + CT_i)$
Dual-port	$\rho \leq 1$, if $(CT_{i-1} + CT_i) \leq 1/TH_R$ $\rho \leq 2 - TH_R(CT_{i-1} + CT_i)$, otherwise
Triple-port	$\rho \leq 1$

A 100% PE utilisation can be achieved with a dual-port memory, but only with appropriate constraints on the communication times. Employing three-port memory increases the memory cost, but only reduces the communication throughput demands. A 100% PE utilization cannot be achieved with a single-port memory.

The PEs memory organisation is a system design time choice which influences the throughput and the utilisation of the system. The information about the PEs memory organisation is used at run-time for predicting the throughput of applications.

6.6. Throughput of the whole application

A model of the whole application is constructed by combining the separate models of the application tasks. Since the applications have a pipeline structure, the task models are just concatenated in a pipeline. In principle, the throughput of the application is determined by the throughput of the slowest task in the pipeline.

Figure 6.8 presents an example model of a pipeline of three tasks running on PEs with single-port memory. The concatenation of the task models is done by merging the sending and receiving actors of consecutive stages, which assumes that the sending and receiving start and finish at exactly the same time. Such an assumption neglects the network delay due to the network buffering. Although the network buffering is reduced to minimum, the network still buffers a small amount of data (few words) and this introduces a small delay between the moments when sending and receiving start at both ends of a communication channel. However, neglecting this delay is justified when the pipeline throughput is considered. The pipeline throughput is determined by the operation rates of its stages. Introducing a delay in the inter-stage communications brings offset in the stages operation periods, but does not change the operating rate of the stages. Therefore, neglecting the propagation delay in the inter-stage communications does not change the pipeline throughput result.

Communication delay should be taken into account when the pipeline latency is critical. To express the delay explicitly in the application HSDF graph, the graphs of the separate tasks are connected serially by introducing a new actor between the sending and receiving actors. The execution time of the new actors equals the delay of the communicate channels between the tasks.

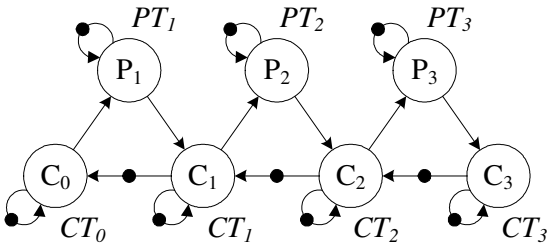


Figure 6.8: An HSDF model of an application of three tasks running on PEs with a single-port data memory merging the send of the previous with the receive of the next

The throughput of the application in Figure 6.8 is found by applying equation (6.5) for each of the three tasks in the pipeline:

$$TH_1 = \frac{1}{\max\limits_{i\in[1,3]}\left[CT_{i-1} + PT_i + CT_i\right]} \tag{6.13}$$

Respectively, the constraint inequalities implied by a given throughput bound TH_R are derived by applying inequality (6.6) for each of the three tasks:

$$CT_{i-1} + PT_i + CT_i \le \frac{1}{TH_R}, \text{ for } i\in\{1,2,3\} \tag{6.14}$$

If the system of inequalities (6.14) is satisfied we guarantee that the application throughput is greater than or equal to TH_R.

When the application runs on PEs of different type, then the equations and constraint inequalities associated with the corresponding PE types are selected. Thus, to predict the application performance at run-time we only select the appropriate equations and substitute in them the actual execution times. To guarantee the application

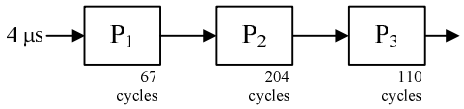
performance, at run-time we select the appropriate constraint inequalities and solve the constructed system of inequalities to find the required upper bounds for the processing and communication times. These bounds are then provided by the system by requesting proper communication services from the NoC and by configuring the PEs clock frequencies.

In conclusion, scheduling an application at run-time on a self-timed system requires solving a small system of linear inequalities. Solving such system is a task of square complexity which in combination with the small system size (less than ten variables) will result in much lower overhead than the overhead for computing a global schedule in a fully-static scheduled system (an NP hard problem). Therefore, because of its lower runtime overhead self-timed system organisation is more suitable for dynamic systems than the full-static organisation.

6.7. Example

We illustrate with an example how applications are scheduled on a self-timed system. For the example we use again the HiperLAN/2 receiver which we schedule on our system. How the HiperLAN/2 receiver is partitioned to run on our system is discussed in [67]. The application is partitioned in a pipeline of three tasks shown in Figure 6.9. The tasks are compiled for Montium processing tiles [39] and the processing times per data item of 256 bytes reported by the compiler are given in the figure (in clock cycles). The receiver has been mapped on Montium tiles and our objective now is to schedule it, or in other words to guarantee its real-time operation.

The HiperLAN/2 receiver processes information received on a wireless communication channel. Every 4 μs a new data item arrives on the input of the receiver and the receiver must be ready to process it. Thus, the data inter-arrival period defines the real-time constraint on the receiver operation. To be able to process all arriving data items, the receiver must have a throughput of at least TH_R = 1/4 μs = 250 [data item/ms].



P1 - Frequency offset correction
P2 - Inverse OFDM
P3 - Equalization, Phase offset correction and De-mapping

Figure 6.9: Pipeline of a HiperLAN/2 receiver

To guarantee this throughput for the receiver, we first construct a system of constraint inequalities. We select the inequalities associated with the memory organisation of the used PEs. In our case, the Montium tiles have single port memories, so we construct the system by combining three inequalities (6.6):

$$CT_{i-1} + PT_i + CT_i \leq \frac{1}{TH_R}, \text{ for } i \in \{1,2,3\} \tag{6.15}$$

If we provide that (6.15) is satisfied, then the receiver throughput is always at least TH_R and the receiver real-time operation is guaranteed.

The Montium tiles are set to run at a fixed clock frequency of 100 MHz. Hence, the actual task execution times are PT_1=0.67 µs, PT_2=2.04 µs and PT_3= 1.1 µs. Substituting this in (6.15) together with TH_R=250 [data item/ms] we have:

$$\begin{cases} CT_0 + CT_1 \le 3.33\mu s \\ CT_1 + CT_2 \le 1.96\mu s \\ CT_2 + CT_3 \le 2.9\mu s \end{cases} \quad (6.16)$$

Any solution of the system of inequalities (6.16) gives a set of worst case communication times for which the required application throughput TH_R is guaranteed. One possible solution is: CT_0=2.35µs, CT_1=0.98µs, CT_2=0.98µs, CT_3=1.92µs. In practice, additional optimisation constraints can be considered when deriving the communication times. For example, minimising the mean communication throughput requested for the application. Having the communication times and the size of the communicated data items we calculate the throughput of the communication channels that has to be requested from the NoC. The communication channels are requested by calling the central routing function (see Section 5.3) which provides a channel in less than a millisecond.

Additional efforts can be made also for improving the PE utilisation. With the processing times given above and TH_R=4 µs, the utilisation of the tiles is ρ_1=17%, ρ_2=51%, ρ_3=28%. The utilisation can be improved by adjusting the PEs clock frequencies and the communication throughput. However, since the tiles use a single-port memory, it is not possible to achieve 100% tile utilisation.

6.8. Conclusion

We presented how our Network-on-Chip (NoC) can be integrated in a system such that predictable system operation is provided. The main difference between our network and other networks is in the way in which predictable operation is achieved. While other NoC based systems use fully-static scheduling leading to high to scheduling complexity, we employ self-timed scheduling. In contrast to fully-static system, a self-timed system in combination with the simple structure of our applications reduces the scheduling complexity, which allows application scheduling to be done at run-time as required for our dynamic system. Self-timed organisation is also more suitable for GALS systems. Our network is directly fit for self-timed system organisation. To the best of our knowledge our system is the first NoC based systems that employs self-timed scheduling.

The PE memory organisation is a system design time choice which influences the performance and utilisation of a self-timed system. We discussed three cases of memory organisation: single-, dual- and triple-port memory organisation. The more memory ports, the higher the application performance and the PE utilisation. However, the memory cost (in terms of area and energy) also increases with the number of ports. The lowest cost at which 100% PE utilisation can be achieved is using dual-port memory which doubles the memory size compared to single-port memory.

The results presented in this chapter confirm that the NoC solution we propose is able support the overall operation of our dynamic real-time system. It is possible to

achieve a predictable operation and to guarantee performance at a system level as the overhead for providing performance guarantees is low enough to allow dynamic system operation.

Chapter 7

Conclusion

The relentless development of semiconductor technology in the last four decades has resulted in integrated circuits (ICs) with higher and higher transistor density, thus allowing chips to accommodate more and more functionality. Today we can integrate a complete system on a single die; such a system is known as a System-on-Chip (SoC). However, the high density reached in the recent years cause problems due to the smaller dimensions of IC features. These problems complicate the SoC design and begin to hamper the technology utilisation. The main problems, as discussed in Chapter 1, are:

- the signal integrity problem – low performance global wires,
- the design productivity gap – high design complexity,
- the clock distribution problem – expensive global clock distribution.

In this thesis we investigate a new approach for organising the global communications of a SoC. This approach provides a general solution for the above three problems. We propose a network-on-chip (NoC) architecture – an on-chip communication solution that matches the on-chip global communication requirements with the constraints of the underlying VLSI technology. We investigate the NoC concept in the context of a specific class of SoC architectures – a multiprocessor SoC for streaming Digital Signal Processing (DSP) applications.

The novelty of our NoC is its ability to operate in a dynamic environment while providing guaranteed services (GS) as well as best effort (BE) services. We evaluate our solution by examining it from different perspectives: from low level implementation details, via its architecture, to the integration into a high level system organisation. The results confirm that we succeeded in finding a practical NoC solution which has a run-time overhead that is low enough to support dynamic system operation.

In Chapter 1 we define three research questions to guide us through our investigation. The first question is:

1. *What network techniques are appropriate to minimize the network overhead while maintaining satisfactory performance?*

It asks for appropriate networking techniques for building a NoC which match the constraints of the underlying technology with the system requirements. In search for an answer we review the available techniques and focus on virtual channel flow control. Virtual channel flow control is an efficient flow control mechanism which requires a small buffer size, hence has a small router area, and has an acceptable performance for our application domain. It also allows for an arbitrary packet length, which is convenient when handling streaming traffic. Our NoC solution is one of the first to employ virtual channel flow control (Chapter 2).

The main challenge for the network design is to provide support for GS and BE traffic. While most NoCs proposed so far achieve that by combining two separate

network solutions, one supporting GS and one supporting BE traffic, by using virtual channels (VCs) we manage to provide a single integrated solution capable of supporting both traffic types. To achieve this result we propose a router architecture with a predictable performance. We combine the router architecture with a virtual channel reservation scheme that extends the predictability at the network level. Thus, our network is able to provide GS as well as BE services (Chapter 3).

Our second research question is:

2. *What is the overhead and the performance of a NoC architecture?*

This asks for a study into the actual NoC performance and all the costs incurred by employing our NoC. This includes the implementation cost as well as the exploitation costs, e.g. system level support.

Although VCs complicate the network routers, we demonstrate that by using appropriate design techniques we can achieve an acceptable area overhead and satisfactory performance. We propose an efficient router design which reduces the implementation area of a virtual channel router, making it comparable to the area of other NoC solutions. With our design, the area of a router in 0.13 μm technology ranges from 0.05 – 0.25 mm^2 depending on the router parameters. Compared to the area of the processing elements in the system, the network area overhead is estimated at only 5 – 15%. The achieved channel throughput is several Gbit/s, enough to satisfy the system demands in our application domain. The implementation results also show that the router area is most sensitive to the size of the buffers while the router performance is most sensitive to the number of VCs (Chapter 4).

Our virtual channel reservation approach for providing service guarantees requires a centralised system coordination, as do all other proposed NoC solutions. However, the central support required by our solution is lighter compared to the other solutions. Our approach reduces the typical NP-complexity of the support tasks to at most quadratic complexity. The light weight support enables our approach to be applied at run-time, for a dynamic environment. To the best of our knowledge our NoC is the first one that is able to provide guaranteed services requested at run-time (Chapter 5).

The third question we define is:

3. *What is the optimal use of the NoC?*

This question addresses the overall system operation. To answer it we investigate the operational aspects of our network and how the network can be integrated into a system.

The application of virtual channel reservation for providing service guarantees is restricted to a system where the applications have a rather simple structure and where they generate simple traffic patterns, which is the case in our target application domain. For systems where the applications have a simple structure, the virtual channel NoC is an area efficient solution because it requires less buffer space than other networking techniques. However, with more complex application structures the traffic patterns get more complicated and thus need more network resources. Satisfying these needs more virtual channels are needed, and therefore the virtual channel NoC rapidly increases in area, its performance deteriorates and so it loses its advantages. For systems where the applications have a complex structure, TDM (time-division multiplexing) is a more area-effective approach for providing guaranteed services.

The performance of the virtual channel reservation approach is strongly influenced by the communication locality. Locality makes the performance of the approach less sensitive to the network size, thus it improves the scalability. Furthermore, by optimising the communication locality the communication energy cost can be reduced up to three times (Chapter 5).

Having a NoC that is able to provide service guarantees at run-time is a necessary but not a sufficient condition for building a predictable system for a dynamic environment. To find out whether such a system is feasible, we investigate whether the multiprocessor system can be organised such that seamless integration and efficient use of the NoC are achieved, and whether the overhead for system support allows dynamic operation. The results present evidence that a self-timed system organisation in combination with the simple structure of our applications reduces the overhead for system scheduling. Compared to the fully-static approach used so far in NoC based systems, self-timed organisation reduces the computational complexity for scheduling real-time applications from exponential to polynomial. This makes self-timed systems more suitable for dynamic operation than the fully-static systems. To the best of our knowledge, the system we propose is the first NoC based SoC that employs self-timed scheduling (Chapter 6).

Future work

We may conclude that the NoC we propose is a practical communication solution for a dynamic multiprocessor system for streaming DSP applications. It is a solution that can be built at an acceptable cost. Nevertheless, there are many possibilities to enhance the solution and prepare it to handle the future growth of the system size.

By providing structured, regular and scalable chip layout our NoC solves the signal integrity problem and facilitates the migration of complex SoC designs to the next generations of semiconductor technologies. However, to secure the successful NoC application in future, scalability must also be provided on a higher system level. The performance of our NoC solution, as well as any network solution, is dependent on the traffic locality. In our system traffic locality is provided by the application mapping function. Hence, it needs to be investigated how different mapping strategies behave when the system size grows and how mapping can provide better traffic locality in future. The same applies for the traffic routing strategies, although routing performance is significantly improved by the traffic locality. In this respect, the relation between application mapping and traffic routing must be investigated in order to find out whether information exchange between them can improve the overall network performance and scalability.

In our work we discussed only the basic operation of a dynamic system, i.e. starting and terminating applications at run-time. Obviously, much more can be done at run-time for improving the system and network efficiency. For example, it needs to be investigated what strategies for run-time system load optimisation and network traffic optimisation can be applied and how these strategies can be implemented to run as background optimisation routines.

By providing system modularity our NoC solution helps narrowing the design productivity gap. Thanks to the modularity, modifying or adding system modules leads to minimal disturbances in the rest of the system. However, to further facilitate the modules integration we have to define a standard interface between the network and the modules. Standard module interfaces have already been established for buses and it

needs to be examined whether they can be directly applied or adapted for NoC based design.

While our NoC provides the two basic classes of services needed, guaranteed and best effort services, there is a variety of higher level network services that can be built on top of these two basic classes. By providing such higher level services the network can further simplify the modules integration. Hence, it needs to be investigated what higher level services can be built to match these services to the services needed by the applications and the system.

One aspect of the NoC integration which needs special attention is whether the NoC can facilitate post manufacturing system testing. The systems being built today are constantly increasing in size and complexity and that makes the system testing and verification a formidable task. As a part of the system the NoC also needs to be tested, so it is desirable for this test to be as simple as possible. Hence, it is worth investigating whether particular network features can simplify this test. Furthermore, serving as a global communication infrastructure, the NoC provides communication access to most of the system modules and so plays an important role in the module testing. This role needs to be investigated to find out whether the NoC can facilitate the testing of system modules.

What has been said for the post manufacturing test is also true for the application development and debugging. The higher application complexity and the system parallelism make application development and verification a complex task. Although applications are most often developed in a system simulation environment, practical experience shows that application debugging on a real system can never be completely avoided. Understanding the reason of an unexpected application malfunctioning can be very difficult in a large multiprocessor system and any help in this direction is desirable. As a global communication infrastructure the NoC may be able to improve visibility of the system internals by making them observable. That is another network aspect that needs to be investigated.

Considering the NoC implementation, a definite direction for improvement is to replace the current synchronous design with an asynchronous one. This will enable our system to operate in Globally-Asynchronous Locally-Synchronous systems and thus to avoid the clock distribution problem. The transition from a synchronous to an asynchronous design is made possible by the fact that our NoC employs only techniques that do not rely on global clock distribution.

The NoC concepts we discuss in this thesis are the first steps in a new direction for SoC design. The NoC based systems are still in their infancy and have a long way to go before they establish themselves as a reliable practical solution. However, our work and the work of others confirm that the direction is right and motivates us to continue investigating the potential of NoC based multiprocessor systems. The knowledge we gain studying these systems prepares us to face the next generations of very large SoC designs.

Bibliography

Publications

[1] N. Kavaldjiev and G. J. M. Smit, "An energy-efficient Network-on-Chip for a heterogeneous tiled reconfigurable System-on-Chip," in *Proceedings of the IEEE EUROMICRO Symposium on Digital System Design (DSD'04)*, Rennes, France, 2004, pp. 492-498.

[2] N. Kavaldjiev and G. J. M. Smit, "A Survey of Efficient On-chip Communications for SoC," in *Proceedings of the PROGRESS 2003 Embedded Systems Symposium*, Nieuwegein, the Netherlands, 2003, pp. 1-12.

[3] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "Two Architectures for On-chip Virtual Channel Router," in *Proceedings of the PROGRESS 2004 Embedded Systems Symposium*, Nieuwegein, the Netherlands, 2004, pp. 96-102.

[4] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "Throughput of Streaming Applications Running on a Multiprocessor Architecture," in *Proceedings of the IEEE EUROMICRO Symposium on Digital System Design (DSD'05)*, Porto, Portugal, 2005, pp. 350-353.

[5] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "A Virtual Channel Router for On-chip Networks," in *Proceedings of the IEEE International System-on-Chip Conference (SOCC'04)*, Santa Clara, California, USA, 2004, pp. 289-293.

[6] N. Kavaldjiev, G. J. M. Smit, P. G. Jansen, and P. T. Wolkotte, "A Virtual Channel Network-on-Chip for GT and BE traffic," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'06)*, Karlsruhe, Germany, 2006, pp. 211-216.

[7] N. Kavaldjiev, G. J. M. Smit, P. T. Wolkotte, and P. G. Jansen, "Providing QoS Guarantees in a NoC by Virtual Channel Reservation," in *Proceedings of the International Workshop on Applied and Reconfigurable Computing (ARC 2006)*, Delft, The Netherlands, 2006, pp. 299-310.

[8] M. H. Wiggers, N. Kavaldjiev, G. J. M. Smit, and P.G. Jansen, "Architecture Design Space Exploration for Streaming Applications Through Timing Analysis," in *Proceedings of the Communicating Process Architectures Conference*, Eindhoven, the Netherlands, 2005, pp. 219-233.

[9] P. T. Wolkotte, G. J. M. Smit, N. Kavaldjiev, J. E. Becker, and J. Becker, "Energy Model of Networks-on-Chip and a Bus," in *Proceedings of the International Symposium on System-on-Chip (SoC'05)*, Tampere, Finland, 2005, pp. 82-85.

References

[10] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE'03)*, Munich, Germany, 2003, pp. 70-73.

[11] S. B. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," *IEEE Transactions on Computers*, 38 (4), 1989, pp. 555-566.

[12] A. Andriahantenaina and A. Greiner, "Micro-network for SoC: implementation of a 32-port SPIN network," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, 2003, pp. 1128-1129.

[13] K. V. Anjan and T. M. Pinkston, "An efficient, fully adaptive deadlock recovery scheme: DISHA," in *Proceedings of the 22nd annual international symposium on Computer architecture*, S. Margherita Ligure, Italy, 1995, pp. 201-210.

[14] K. A. Arvind, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek, "RAMP: Research Accelerator for Multiple Processors - A Community Vision for a Shared Experimental Parallel HW/SW Platform," University of California, Berkeley, Technical Report UCB//CSD-05-1412, September 2005.

[15] F. L. Baccelli, G. Cohen, and G. J. Olsder, *Synchronization and linearity: an algebra for discrete event systems*. New York: John Wiley & Sons, 1993.

[16] M. Bekooij, O. Moreira, P. Poplavko, B. Mesman, M. Pastrnak, and J. v. Meerbergen, "Predictable embedded multi-processor system design," in *Proceedings of the 8th International workshop on software and compilers for embedded systems (Scopes 2004)*, Amsterdam, the Netherlands, 2004, pp. 77-91.

[17] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *Computer*, 35 (1), 2002, pp. 70-78.

[18] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip," in *Proceedings of the Design, Automation and Test in Europe Conference and Exposition (DATE 2005)*, Munich, Germany, 2005, pp. 1226-1231.

[19] D. M. Chapiro, "Globally-Asynchronous Locally-Synchronous systems," Ph.D. thesis, Stanford University, 1985.

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, 2 ed. Cambridge, Massachusetts: MIT Press, 2001.

[21] W. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, 3 (2), 1992, pp. 194-205.

[22] W. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Transactions on Computers*, 39 (6), 1990, pp. 775-785.

[23] W. Dally, F. Labonte, A. Das, P. Hanrahan, J.-H. Ahn, J. Gummaraju, M. Erez, N. Jayasena, I. Buck, T. Knight, and U. Kapasi, "Merrimac: Supercomputing with Streams," in *Proceedings of the ACM/IEEE conference on Supercomputing (SC'03)*, Phoenix, Arizona, 2003, pp. 35-42.

[24] W. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, 36 (5), 1987, pp. 547-553.

[25] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the 38th Conference on Design Automation (DAC'01)*, Las Vegas, NV, 2001, pp. 684-689.

[26] W. J. Dally, L. R. Dennison, D. Harris, K. Kan, and T. Xanthopoulos, "The Reliable Router: A Reliable and High-Performance Communication Substrate for Parallel Computers," in *Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, 1994, pp. 241-255.

[27] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[28] J. G. Delgado-Frias and G. B. Ratanpal, "A VLSI crossbar switch with wrapped wave front arbitration," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50 (1), 2003, pp. 135-141.

[29] A. M. Despain and D. A. Patterson, "X-Tree: A tree structured multi-processor computer architecture," in *Proceedings of the 5th annual symposium on Computer architecture*, 1978, pp. 144-151.

[30] J. Draper and G. Joydeep, "A comprehensive analytical model for wormhole routing in multicomputer systems," *Journal of Parallel and Distributed Computing*, 23 (2), 1994, pp. 202-214.

[31] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems*, 4 (12), 1993, pp. 1320-1331.

[32] T. Felicijan and S. Furber, "An Asynchronous On-Chip Network Router with Quality-of-Service (QoS) Support," in *Proceedings of the IEEE International System-on-Chip Conference (SOCC'04)*, Santa Clara, California, 2004, pp. 274-277.

[33] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," *Journal of the ACM (JACM)*, 41 (5), 1994, pp. 874-902.

[34] K. Goossens, J. Dielissen, and A. Radulescu, "Ethereal network on chip: concepts, architectures, and implementations," *IEEE Design & Test of Computers*, 22 (5), 2005, pp. 414-421.

[35] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, 19 (1), 1999, pp. 20-28.

[36] A. Hansson, K. Goossens, and A. Radulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on*

Hardware/Software Codesign and System Synthesis, Jersey City, NJ, USA, 2005, pp. 75-80.

[37] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics SSC4*, (2), 1968, pp. 100–107.

[38] A. Hemani, T. Meincke, S. Kumar, A. Postula, T. Olsson, P. Nilsson, J. Oberg, P. Ellervee, and D. Lundqvist, "Lowering power consumption in clock by using globally asynchronous locally synchronous design style," in *Proceedings of the 36th Design Automation Conference*, 1999, pp. 873-878.

[39] P. M. Heysters, "Coarse-Grained Reconfigurable Processors," Ph.D. thesis, University of Twente, 2004.

[40] P. M. Heysters, G. J. M. Smit, and E. Molenkamp, "A Flexible and Energy-Efficient Coarse-Grained Reconfigurable Architecture for Mobile Systems," *Journal of Supercomputing*, 26 (3), 2003, pp. 283-308.

[41] R. Ho, K. Mai, and M. Horowitz, "Efficient on-chip global interconnects," in *Proceedings of the Symposium on VLSI Circuits*, 2003, pp. 271 - 274.

[42] R. Ho, K. W. Mai, and M. A. Horowitz, "The future of wires," *Proceedings of the IEEE*, 89 (4), 2001, pp. 490-504.

[43] P.-C. Hu and L. Kleinrock, "A dynamic timeout scheme for wormhole routing networks," in *Proceedings of the IEEE International Conference on Communications*, 1997, pp. 1406 - 1410.

[44] P.-C. Hu and L. Kleinrock, "An Analytical Model for Wormhole Routing with Finite Size Input Buffers," in *Proceedings of the 15th International Teletraffic Congress*, 1997, pp. 549-560.

[45] A. Jantsch and H. Tenhunen, *Networks on Chip*: Kluwer Academic Publishers, 2003.

[46] A. B. Kahng, "Design technology productivity in the DSM era," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2001)*, 2001, pp. 443-448.

[47] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks*, 3, 1979, pp. 267-286.

[48] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*, vol. 2. New York: John Wiley & Sons, Inc., 1976.

[49] E. A. Lee, "Building Unreliable Systems out of Reliable Components: The Real Time Story," EECS Department, University of California, Berkeley, Technical report UCB/EECS-2005-5, October 2005.

[50] E. A. Lee and D. G. Messerschmitt, "Synchronous Data Flow," in *Proceedings of the IEEE*, 1987, pp. 1235-1245.

[51] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *IEEE Transactions on Computers*, 34 (10), 1985, pp. 892-901.

[52] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12 (7), 2004, pp. 711-726.

[53] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for *k*-ary *n*-cubes," *IEEE Transactions on Computers*, 40 (1), 1991, pp. 2-12.

[54] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Transactions on Networking*, 7 (2), 1999, pp. 188-201.

[55] J. D. Meindl, "Interconnect opportunities for gigascale integration," *IEEE Micro*, 23 (3), 2003, pp. 28 - 35.

[56] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed Bandwidth Using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE'04)*, Paris, France, 2004, pp. 890-895.

[57] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Electronics*, 38 (8), 1965, pp. 114-117.

[58] J. Muttersbach, T. Villiger, and W. Fichtner, "Practical design of globally-asynchronous locally-synchronous systems," in *Proceedings of the 6th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2000)*, Eilat, Israel, 2000, pp. 52-59.

[59] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, 26 (2), 1993, pp. 62-76.

[60] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, *Interconnect-Centric Design for Advanced SOC and NOC*: Kluwer Academic Publishers, 2004.

[61] L.-S. Peh and W. J. Dally, "Flit-reservation flow control," in *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture (HPCA-6)*, 2000, pp. 73-84.

[62] L.-S. Peh and W. J. Dally, "A delay model for router microarchitectures," *IEEE Micro*, 21 (1), 2001, pp. 26-34.

[63] L. S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA-7)*, 2001, pp. 255-266.

[64] M. J. Pertel, "Mesh Distance Formulae," California Institute of Technology, Pasadena, CA, Technical report Caltech-CS-TR-92-05, March 1992.

[65] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Communications of the ACM*, 24 (5), 1981, pp. 300-309.

[66] A. Radulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P. Wielage, and K. Goossens, "An efficient on-chip NI offering guaranteed services, shared-memory abstraction, and flexible network configuration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24 (1), 2005, pp. 4-17.

[67] G. K. Rauwerda, P. M. Heysters, and G. J. M. Smit, "Mapping Wireless Communication Algorithms onto a Reconfigurable Architecture," *Journal of Supercomputing*, 30 (3), 2004, pp. 263-282.

[68] C. L. Seitz, "The Hypercube Communications Chip," Department of Computer Science, California Institute of Technology, Display File 5182: DF:85, March 1985.

[69] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, *Scheduling and load balancing in parallel and distributed systems*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1995.

[70] D. Siguenza-Tortosa, T. Ahonen, and J. Nurmi, "Issues in the development of a practical NoC: the Proteo concept," *Integration, the VLSI Journal*, 38 (1), 2004, pp. 95-105.

[71] L. T. Smit, "Energy-Efficient Wireless Communication," Ph.D. thesis, University of Twente, 2004.

[72] L. T. Smit, J. L. Hurink, and G. J. M. Smit, "Run-time Mapping of Applications to a Heterogeneous SoC," in *Proceedings of the International Symposium on System-on-Chip*, 2005, pp. 78-81.

[73] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*. New York, Basel: Marcel Dekker, 2000.

[74] C. B. Stunkel, D. G. Shea, B. Aball, M. G. Atkins, C. A. Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, "The SP2 high-performance switch," *IBM System Journal*, 34 (2), 1995, pp. 185-204.

[75] H. Sullivan, T. R. Bashkow, and D. Klappholz, "A Large Scale, Homogenous, Fully Distributed Parallel Machine," in *Proceedings of the 4th annual symposium on Computer architecture*, 1977, pp. 118-124.

[76] D. Sylvester and K. Keutzer, "A global wiring paradigm for deep submicron design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19 (2), 2000, pp. 242-252.

[77] Y. Tamir and H. C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, 4 (1), 1993, pp. 13-27.

[78] N. Thepayasuwan and A. Doboli, "Layout conscious bus architecture synthesis for deep submicron systems on chip," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE 2004)*, Paris, France, 2004, pp. 108 -113.

[79] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proceedings of the Design Automation Conference*, 1998, pp. 732-737.

[80] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in *Proceedings of the Thirteenth annual ACM symposium on Theory of computing*, Milwaukee, Wisconsin, USA, 1981, pp. 263-277.

[81] H.-S. Wang, L.-S. Peh, and S. Malik, "A power model for routers: modeling Alpha 21364 and InfiniBand routers," *IEEE Micro*, 23 (1), 2003, pp. 26-35.

[82] T. Whitney and G. Neville-Neil, "SoC: Software, Hardware, Nightmare, Bliss," *ACM Queue*, 1 (2), 2003, pp. 24-31.

[83] M. H. Wiggers, M. J. G. Bekooij, P. G. Jansen, and G. J. M. Smit, "Efficient Computation of Buffer Capacities for Multi-Rate Real-Time Systems with Back-Pressure," in *Proceedings of the International Conference on Hardware-Software Codesign and System Synthesis (CODES)*, Seoul, Korea, 2006, pp.

[84] D. Wiklund and L. Dake, "SoCBUS: switched network on chip for hard real time embedded systems," in *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003, pp. 78-85.

[85] P. T. Wolkotte, G. J. M. Smit, and J. E. Becker, "Energy-efficient NoC for best effort communication," in *Proceedings of the 15th International Conference on Field Programmable Logic and Applications (FPL 2005)*, 2005, pp. 197-202.

[86] P. T. Wolkotte, G. J. M. Smit, and G. K. Rauwerda, "An Energy-Efficient Reconfigurable Circuit Switched Network-on-Chip," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Denver, Colorado, USA, 2005, pp. 155-161.

[87] P. T. Wolkotte, G. J. M. Smit, and L. T. Smit, "Partitioning of a DRM receiver," in *Proceedings of the 9th International OFDM-Workshop*, Dresden, Germany, 2004, pp. 299-304.

[88] H. Zhang, V. George, and J. M. Rabaey, "Low-swing on-chip signaling techniques: effectiveness and robustness," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8 (3), 2000, pp. 264 - 272.

Electronic sources

[89] ARM, "AMBA Specification," Rev. 2.0, 1999, http://www.arm.com/

[90] IBM, "The CoreConnect™ Bus Architecture," 1999, http://www.ibm.com/

[91] OCP-IP, "Opeen Corree Protocol Datasheet," 2004, http://www.ocpip.org/

[92] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors (ITRS)," 2002, http://public.itrs.net/

[93] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors (ITRS)," 2005, http://public.itrs.net/

[94] Texas Instruments Inc., "TMS320DM6443 Digital Media System-on-Chip," Rev. C, 2006, http://www.ti.com

[95] VSIA, "On-Chip Bus Attributes Specification," Rev. 1, 2001, http://www.vsi.org/

Titles in the IPA Dissertation Series

J.O. Blanco, "The State Operator in Process Algebra", Faculty of Mathematics and Computing Science, TUE, 1996-01

A.M. Geerling, "Transformational Development of Data-Parallel Algorithms", Faculty of Mathematics and Computer Science, KUN, 1996-02

P.M. Achten, "Interactive Functional Programs: Models, Methods, and Implementation", Faculty of Mathematics and Computer Science, KUN, 1996-03

M.G.A. Verhoeven, "Parallel Local Search", Faculty of Mathematics and Computing Science, TUE, 1996-04

M.H.G.K. Kesseler, "The Implementation of Functional Languages on Parallel Machines with Distrib. Memory", Faculty of Mathematics and Computer Science, KUN, 1996-05

D. Alstein, "Distributed Algorithms for Hard Real-Time Systems", Faculty of Mathematics and Computing Science, TUE, 1996-06

J.H. Hoepman, "Communication, Synchronization, and Fault-Tolerance", Faculty of Mathematics and Computer Science, UvA, 1996-07

H. Doornbos, "Reductivity Arguments and Program Construction", Faculty of Mathematics and Computing Science, TUE, 1996-08

D. Turi, "Functorial Operational Semantics and its Denotational Dual", Faculty of Mathematics and Computer Science, VUA, 1996-09

A.M.G. Peeters, "Single-Rail Handshake Circuits", Faculty of Mathematics and Computing Science, TUE, 1996-10

N.W.A. Arends, "A Systems Engineering Specification Formalism", Faculty of Mechanical Engineering, TUE, 1996-11

P. Severi de Santiago, "Normalisation in Lambda Calculus and its Relation to Type Inference", Faculty of Mathematics and Computing Science, TUE, 1996-12

D.R. Dams, "Abstract Interpretation and Partition Refinement for Model Checking", Faculty of Mathematics and Computing Science, TUE, 1996-13

M.M. Bonsangue, "Topological Dualities in Semantics", Faculty of Mathematics and Computer Science, VUA, 1996-14

B.L.E. de Fluiter, "Algorithms for Graphs of Small Treewidth", Faculty of Mathematics and Computer Science, UU, 1997-01

W.T.M. Kars, "Process-algebraic Transformations in Context", Faculty of Computer Science, UT, 1997-02

P.F. Hoogendijk, "A Generic Theory of Data Types", Faculty of Mathematics and Computing Science, TUE, 1997-03

T.D.L. Laan, "The Evolution of Type Theory in Logic and Mathematics", Faculty of Mathematics and Computing Science, TUE, 1997-04

C.J. Bloo, "Preservation of Termination for Explicit Substitution", Faculty of Mathematics and Computing Science, TUE, 1997-05

J.J. Vereijken, "Discrete-Time Process Algebra", Faculty of Mathematics and Computing Science, TUE, 1997-06

F.A.M. van den Beuken, "A Functional Approach to Syntax and Typing", Faculty of Mathematics and Informatics, KUN, 1997-07

A.W. Heerink, "Ins and Outs in Refusal Testing", Faculty of Computer Science, UT, 1998-01

G. Naumoski and W. Alberts, "A Discrete-Event Simulator for Systems Engineering", Faculty of Mechanical Engineering, TUE, 1998-02

J. Verriet, "Scheduling with Communication for Multiprocessor Computation", Faculty of Mathematics and Computer Science, UU, 1998-03

J.S.H. van Gageldonk, "An Asynchronous Low-Power 80C51 Microcontroller", Faculty of Mathematics and Computing Science, TUE, 1998-04

A.A. Basten, "In Terms of Nets: System Design with Petri Nets and Process Algebra", Faculty of Mathematics and Computing Science, TUE, 1998-05

E. Voermans, "Inductive Datatypes with Laws and Subtyping -- A Relational Model", Faculty of Mathematics and Computing Science, TUE, 1999-01

H. ter Doest, "Towards Probabilistic Unification-based Parsing", Faculty of Computer Science, UT, 1999-02

J.P.L. Segers, "Algorithms for the Simulation of Surface Processes", Faculty of Mathematics and Computing Science, TUE, 1999-03

C.H.M. van Kemenade, "Recombinative Evolutionary Search", Faculty of Mathematics and Natural Sciences, UL, 1999-04

E.I. Barakova, "Learning Reliability: a Study on Indecisiveness in Sample Selection", Faculty of Mathematics and Natural Sciences, RUG, 1999-05

M.P. Bodlaender, "Scheduler Optimization in Real-Time Distributed Databases", Faculty of Mathematics and Computing Science, TUE, 1999-06

M.A. Reniers, "Message Sequence Chart: Syntax and Semantics", Faculty of Mathematics and Computing Science, TUE, 1999-07

J.P. Warners, "Nonlinear approaches to satisfiability problems", Faculty of Mathematics and Computing Science, TUE, 1999-08

J.M.T. Romijn, "Analysing Industrial Protocols with Formal Methods", Faculty of Computer Science, UT, 1999-09

P.R. D'Argenio, "Algebras and Automata for Timed and Stochastic Systems", Faculty of Computer Science, UT, 1999-10

G. Fabian, "A Language and Simulator for Hybrid Systems", Faculty of Mechanical Engineering, TUE, 1999-11

J. Zwanenburg, "Object-Oriented Concepts and Proof Rules", Faculty of Mathematics and Computing Science, TUE, 1999-12

R.S. Venema, "Aspects of an Integrated Neural Prediction System", Faculty of Mathematics and Natural Sciences, RUG, 1999-13

J. Saraiva, "A Purely Functional Implementation of Attribute Grammars", Faculty of Mathematics and Computer Science, UU, 1999-14

R. Schiefer, "Viper, A Visualisation Tool for Parallel Program Construction", Faculty of Mathematics and Computing Science, TUE, 1999-15

K.M.M. de Leeuw, "Cryptology and Statecraft in the Dutch Republic", Faculty of Mathematics and Computer Science, UvA, 2000-01

T.E.J. Vos, "UNITY in Diversity. A stratified approach to the verification of distributed algorithms", Faculty of Mathematics and Computer Science, UU, 2000-02

W. Mallon, "Theories and Tools for the Design of Delay-Insensitive Communicating Processes", Faculty of Mathematics and Natural Sciences, RUG, 2000-03

W.O.D. Griffioen, "Studies in Computer Aided Verification of Protocols", Faculty of Science, KUN, 2000-04

P.H.F.M. Verhoeven, "The Design of the MathSpad Editor", Faculty of Mathematics and Computing Science, TUE, 2000-05

J. Fey, "Design of a Fruit Juice Blending and Packaging Plant", Faculty of Mechanical Engineering, TUE, 2000-06

M. Franssen, "Cocktail: A Tool for Deriving Correct Programs", Faculty of Mathematics and Computing Science, TUE, 2000-07

P.A. Olivier, "A Framework for Debugging Heterogeneous Applications", Faculty of Natural Sciences, Mathematics and Computer Science, UvA, 2000-08

E. Saaman, "Another Formal Specification Language", Faculty of Mathematics and Natural Sciences, RUG, 2000-10

M. Jelasity, "The Shape of Evolutionary Search Discovering and Representing Search Space Structure", Faculty of Mathematics and Natural Sciences, UL, 2001-01

R. Ahn, "Agents, Objects and Events a computational approach to knowledge, observation and communication", Faculty of Mathematics and Computing Science, TU/e, 2001-02

M. Huisman, "Reasoning about Java programs in higher order logic using PVS and Isabelle", Faculty of Science, KUN, 2001-03

I.M.M.J. Reymen, "Improving Design Processes through Structured Reflection", Faculty of Mathematics and Computing Science, TU/e, 2001-04

S.C.C. Blom, "Term Graph Rewriting: syntax and semantics", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2001-05

R. van Liere, "Studies in Interactive Visualization", Faculty of Natural Sciences, Mathematics and Computer Science, UvA, 2001-06

A.G. Engels, "Languages for Analysis and Testing of Event Sequences", Faculty of Mathematics and Computing Science, TU/e, 2001-07

J. Hage, "Structural Aspects of Switching Classes", Faculty of Mathematics and Natural Sciences, UL, 2001-08

M.H. Lamers, "Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes", Faculty of Mathematics and Natural Sciences, UL, 2001-09

T.C. Ruys, "Towards Effective Model Checking", Faculty of Computer Science, UT, 2001-10

D. Chkliaev, "Mechanical verification of concurrency control and recovery protocols", Faculty of Mathematics and Computing Science, TU/e, 2001-11

M.D. Oostdijk, "Generation and presentation of formal mathematical documents", Faculty of Mathematics and Computing Science, TU/e, 2001-12

A.T. Hofkamp, "Reactive machine control: A simulation approach using χ", Faculty of Mechanical Engineering, TU/e, 2001-13

D. Bosnacki, "Enhancing state space reduction techniques for model checking", Faculty of

Mathematics and Computing Science, TU/e, 2001-14

M.C. van Wezel, "Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects", Faculty of Mathematics and Natural Sciences, UL, 2002-01

V. Bos and J.J.T. Kleijn, "Formal Specification and Analysis of Industrial Systems", Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e, 2002-02

T. Kuipers, "Techniques for Understanding Legacy Software Systems", Faculty of Natural Sciences, Mathematics and Computer Science, UvA, 2002-03

S.P. Luttik, "Choice Quantification in Process Algebra", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2002-04

R.J. Willemen, "School Timetable Construction: Algorithms and Complexity", Faculty of Mathematics and Computer Science, TU/e, 2002-05

M.I.A. Stoelinga, "Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems", Faculty of Science, Mathematics and Computer Science, KUN, 2002-06

N. van Vugt, "Models of Molecular Computing", Faculty of Mathematics and Natural Sciences, UL, 2002-07

A. Fehnker, "Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems", Faculty of Science, Mathematics and Computer Science, KUN, 2002-08

R. van Stee, "On-line Scheduling and Bin Packing", Faculty of Mathematics and Natural Sciences, UL, 2002-09

D. Tauritz, "Adaptive Information Filtering: Concepts and Algorithms", Faculty of Mathematics and Natural Sciences, UL, 2002-10

M.B. van der Zwaag, "Models and Logics for Process Algebra", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2002-11

J.I. den Hartog, "Probabilistic Extensions of Semantical Models", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2002-12

L. Moonen, "Exploring Software Systems", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2002-13

J.I. van Hemert, "Applying Evolutionary Computation to Constraint Satisfaction and Data Mining", Faculty of Mathematics and Natural Sciences, UL, 2002-14

S. Andova, "Probabilistic Process Algebra", Faculty of Mathematics and Computer Science, TU/e, 2002-15

Y.S. Usenko, "Linearization in μCRL", Faculty of Mathematics and Computer Science, TU/e, 2002-16

J.J.D. Aerts, "Random Redundant Storage for Video on Demand", Faculty of Mathematics and Computer Science, TU/e, 2003-01

M. de Jonge, "To Reuse or To Be Reused: Techniques for component composition and construction", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2003-02

J.M.W. Visser, "Generic Traversal over Typed Source Code Representations", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2003-03

S.M. Bohte, "Spiking Neural Networks", Faculty of Mathematics and Natural Sciences, UL, 2003-04

T.A.C. Willemse, "Semantics and Verification in Process Algebras with Data and Timing", Faculty of Mathematics and Computer Science, TU/e, 2003-05

S.V. Nedea, "Analysis and Simulations of Catalytic Reactions", Faculty of Mathematics and Computer Science, TU/e, 2003-06

M.E.M. Lijding, "Real-time Scheduling of Tertiary Storage", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2003-07

H.P. Benz, "Casual Multimedia Process Annotation -- CoMPAs", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2003-08

D. Distefano, "On Modelchecking the Dynamics of Object-based Software: a Foundational Approach", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2003-09

M.H. ter Beek, "Team Automata -- A Formal Approach to the Modeling of Collaboration Between System Components", Faculty of Mathematics and Natural Sciences, UL, 2003-10

D.J.P. Leijen, "The λ Abroad -- A Functional Approach to Software Components", Faculty of Mathematics and Computer Science, UU, 2003-11

W.P.A.J. Michiels, "Performance Ratios for the Differencing Method", Faculty of Mathematics and Computer Science, TU/e, 2004-01

G.I. Jojgov, "Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving", Faculty of Mathematics and Computer Science, TU/e, 2004-02

P. Frisco, "Theory of Molecular Computing -- Splicing and Membrane systems", Faculty of Mathematics and Natural Sciences, UL, 2004-03

S. Maneth, "Models of Tree Translation", Faculty of Mathematics and Natural Sciences, UL, 2004-04

Y. Qian, "Data Synchronization and Browsing for Home Environments", Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e, 2004-05

F. Bartels, "On Generalised Coinduction and Probabilistic Specification Formats", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2004-06

L. Cruz-Filipe, "Constructive Real Analysis: a Type-Theoretical Formalization and Applications", Faculty of Science, Mathematics and Computer Science, KUN, 2004-07

E.H. Gerding, "Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications", Faculty of Technology Management, TU/e, 2004-08

N. Goga, "Control and Selection Techniques for the Automated Testing of Reactive Systems", Faculty of Mathematics and Computer Science, TU/e, 2004-09

M. Niqui, "Formalising Exact Arithmetic: Representations, Algorithms and Proofs", Faculty of Science, Mathematics and Computer Science, RU, 2004-10

A. Loh, "Exploring Generic Haskell", Faculty of Mathematics and Computer Science, UU, 2004-11

I.C.M. Flinsenberg, "Route Planning Algorithms for Car Navigation", Faculty of Mathematics and Computer Science, TU/e, 2004-12

R.J. Bril, "Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets", Faculty of Mathematics and Computer Science, TU/e, 2004-13

J. Pang, "Formal Verification of Distributed Systems", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2004-14

F. Alkemade, "Evolutionary Agent-Based Economics", Faculty of Technology Management, TU/e, 2004-15

E.O. Dijk, "Indoor Ultrasonic Position Estimation Using a Single Base Station", Faculty of Mathematics and Computer Science, TU/e, 2004-16

S.M. Orzan, "On Distributed Verification and Verified Distribution", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2004-17

M.M. Schrage, "Proxima - A Presentation-oriented Editor for Structured Documents", Faculty of Mathematics and Computer Science, UU, 2004-18

E. Eskenazi and A. Fyukov, "Quantitative Prediction of Quality Attributes for Component-Based Software Architectures", Faculty of Mathematics and Computer Science, TU/e, 2004-19

P.J.L. Cuijpers, "Hybrid Process Algebra", Faculty of Mathematics and Computer Science, TU/e, 2004-20

N.J.M. van den Nieuwelaar, "Supervisory Machine Control by Predictive-Reactive Scheduling", Faculty of Mechanical Engineering, TU/e, 2004-21

E. Abraham, "An Assertional Proof System for Multithreaded Java -Theory and Tool Support- ", Faculty of Mathematics and Natural Sciences, UL, 2005-01

R. Ruimerman, "Modeling and Remodeling in Bone Tissue", Faculty of Biomedical Engineering, TU/e, 2005-02

C.N. Chong, "Experiments in Rights Control - Expression and Enforcement", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2005-03

H. Gao, "Design and Verification of Lock-free Parallel Algorithms", Faculty of Mathematics and Computing Sciences, RUG, 2005-04

H.M.A. van Beek, "Specification and Analysis of Internet Applications", Faculty of Mathematics and Computer Science, TU/e, 2005-05

M.T. Ionita, "Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures", Faculty of Mathematics and Computing Sciences, TU/e, 2005-06

G. Lenzini, "Integration of Analysis Techniques in Security and Fault-Tolerance", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2005-07

I. Kurtev, "Adaptability of Model Transformations", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2005-08

T. Wolle, "Computational Aspects of Treewidth - Lower Bounds and Network Reliability", Faculty of Science, UU, 2005-09

O. Tveretina, "Decision Procedures for Equality Logic with Uninterpreted Functions", Faculty of Mathematics and Computer Science, TU/e, 2005-10

A.M.L. Liekens, "Evolution of Finite Populations in Dynamic Environments", Faculty of Biomedical Engineering, TU/e, 2005-11

J. Eggermont, "Data Mining using Genetic Programming: Classification and Symbolic Regression", Faculty of Mathematics and Natural Sciences, UL, 2005-12

B.J. Heeren, "Top Quality Type Error Messages", Faculty of Science, UU, 2005-13

G.F. Frehse, "Compositional Verification of Hybrid Systems using Simulation Relations", Faculty of Science, Mathematics and Computer Science, RU, 2005-14

M.R. Mousavi, "Structuring Structural Operational Semantics", Faculty of Mathematics and Computer Science, TU/e, 2005-15

A. Sokolova, "Coalgebraic Analysis of Probabilistic Systems", Faculty of Mathematics and Computer Science, TU/e, 2005-16

T. Gelsema, "Effective Models for the Structure of pi-Calculus Processes with Replication", Faculty of Mathematics and Natural Sciences, UL, 2005-17

P. Zoeteweij, "Composing Constraint Solvers", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2005-18

J.J. Vinju, "Analysis and Transformation of Source Code by Parsing and Rewriting", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2005-19

M.Valero Espada, "Modal Abstraction and Replication of Processes with Data", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2005-20

A. Dijkstra, "Stepping through Haskell", Faculty of Science, UU, 2005-21

Y.W. Law, "Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2005-22

E. Dolstra, "The Purely Functional Software Deployment Model", Faculty of Science, UU, 2006-01

R.J. Corin, "Analysis Models for Security Protocols", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2006-02

P.R.A. Verbaan, "The Computational Complexity of Evolving Systems", Faculty of Science, UU, 2006-03

K.L. Man and R.R.H. Schiffelers, "Formal Specification and Analysis of Hybrid Systems", Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e, 2006-04

M. Kyas, "Verifying OCL Specifications of UML Models: Tool Support and Compositionality", Faculty of Mathematics and Natural Sciences, UL, 2006-05

M. Hendriks, "Model Checking Timed Automata - Techniques and Applications", Faculty of Science, Mathematics and Computer Science, RU, 2006-06

J. Ketema, "Bohm-Like Trees for Rewriting", Faculty of Sciences, VUA, 2006-07

C.-B. Breunesse, "On JML: topics in tool-assisted verification of JML programs", Faculty of Science, Mathematics and Computer Science, RU, 2006-08

B. Markvoort, "Towards Hybrid Molecular Simulations", Faculty of Biomedical Engineering, TU/e, 2006-09

S.G.R. Nijssen, "Mining Structured Data", Faculty of Mathematics and Natural Sciences, UL, 2006-10

G. Russello, "Separation and Adaptation of Concerns in a Shared Data Space", Faculty of Mathematics and Computer Science, TU/e, 2006-11

L. Cheung, "Reconciling Nondeterministic and Probabilistic Choices", Faculty of Science, Mathematics and Computer Science, RU, 2006-12

B. Badban, "Verification techniques for Extensions of Equality Logic", Faculty of Sciences, Division of Mathematics and Computer Science, VUA, 2006-13

A.J. Mooij, "Constructive formal methods and protocol standardization", Faculty of Mathematics and Computer Science, TU/e, 2006-14

T. Krilavicius, "Hybrid Techniques for Hybrid Systems", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2006-15

M.E. Warnier, "Language Based Security for Java and JML", Faculty of Science, Mathematics and Computer Science, RU, 2006-16

V. Sundramoorthy, "At Home In Service Discovery", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2006-17

B. Gebremichael, "Expressivity of Timed Automata Models", Faculty of Science, Mathematics and Computer Science, RU, 2006-18

L.C.M. van Gool, "Formalising Interface Specifications", Faculty of Mathematics and Computer Science, TU/e, 2006-19

C.J.F. Cremers, "Scyther - Semantics and Verification of Security Protocols", Faculty of Mathematics and Computer Science, TU/e, 2006-20

J.V. Guillen Scholten, "Mobile Channels for Exogenous Coordination of Distributed Systems: Semantics, Implementation and Composition", Faculty of Mathematics and Natural Sciences, UL, 2006-21

H.A. de Jong, "Flexible Heterogeneous Software Systems", Faculty of Natural Sciences, Mathematics, and Computer Science, UvA, 2007-01

N.K. Kavaldjiev, "A run-time reconfigurable Network-on-Chip for streaming DSP applications", Faculty of Electrical Engineering, Mathematics & Computer Science, UT, 2007-02