An Asynchronous Time-Division-Multiplexed Network-on-Chip for Real-Time Systems

Kasapaki, Evangelia

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

An Asynchronous Time-Division-Multiplexed Network-on-Chip for Real-Time Systems

Evangelia Kasapaki

Kongens Lyngby 2015
PhD-2015-361

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Matematiktorvet, building 303B,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3351
compute@compute.dtu.dk
www.compute.dtu.dk PhD-2015-361

Summary (English)

Multi-processor architectures using networks-on-chip (NOCs) for communication are becoming the standard approach in the development of embedded systems and general purpose platforms. Typically, multi-processor platforms follow a globally asynchronous locally synchronous (GALS) timing organization. This thesis focuses on the design of Argo, a NOC targeted at hard real-time multi-processor platforms with a GALS timing organization.

To support real-time communication, NOCs establish end-to-end connections and provide latency and throughput guarantees for these connections. Argo uses time division multiplexing (TDM) in combination with a static schedule to implement virtual end-to-end circuits. TDM is a straightforward way to provide guarantees and to share the resources efficiently, and it has an efficient hardware implementation. Argo supports a GALS system organization, and additionally it explores more flexible timing within its structure, to address signal distribution issues, using a network of asynchronous routers.

NOCs consist of a switching structure of routers connected by links, with network interfaces (NIs) that connect the processors to the switching structure. Argo uses a novel NI design that supports time-predictability, and asynchronous routers that form a time-elastic network. The NI design integrates the DMA functionality and the TDM schedule, and uses dual-ported local memories. The routers combine the router functionality and asynchronous elastic behavior. They also use a gating mechanism to reduce the energy consumption. The combination of the NI design and the router design supports the formation of end-to-end paths in the NOC, from the local memory of a sending core to the local memory of a receiving core. These end-to-end paths do not require any dynamic arbitration, buffering, flow control, or clock synchronization, in the routers or the NIs.

This thesis explores the implementation of the individual components of Argo, as well as several complete instances of the Argo NOC. The implementations target both FPGA technology and 65 nm CMOS technology. It is shown that (i) the NI design is scalable and four to five times smaller than previously published NIs for similar NOCs, (ii) the router design is power efficient and two to three times smaller than equivalent router designs, and (iii) the overall Argo NOC is around four times smaller than other TDM NOCs. Argo is an important part of the T-CREST paltform and used in a number of configurations.

The flexible timing organization of Argo combines asynchronous routers with mesochronous NIs, which are connected to individually clocked cores, supporting a GALS system organization. The mesochronous NIs operate at the same frequency, possibly with some skew, while the network of asynchronous routers absorbs this skew within certain limits. The elasticity of the asynchronous network is explored, answering the question of how much skew the Argo NOC can absorb. A qualitative analysis studies the parameters affecting the elasticity and its limits. A quantitative analysis models the Argo behavior using timed-graph models and worst-case timing separation of events analysis to evaluate the elasticity of Argo. The results show that the skew absorbed by the network of routers can be two or more cycles, depending on the frequency applied at its endpoints, the NIs.

Overall this thesis presents the design and implementation of Argo, and the analysis of its elastic behavior. It shows that Argo provides hard real-time guarantees in a straightforward way, it has an efficient implementation and it is time-elastic.

Resumé (Dansk)

Multi-processor arkitekture der bruger intra-chip netværk (eng: NOCs) til kommunikation er ved at blive en standard tilgangen i udviklingen af indlejrede systemer og generelle platforme. Typisk er tidssynkroniseringen i en multi-processor platform organiseret i et globalt asynkron lokalt synkron (GALS) domæne. Denne afhandling fokusere på designet af Argo, der er et NOC målrettet til multi-processor platforme til hårde realtids systemer med en GALS organisering af tidssynkroniseringen.

For at supportere realtids kommunikation, etablere NOCs end-to-end forbindelser og tilvejebringer latens og gennemløbs garantier for disse forbindelser. Argo bruger tidsmultiplexing (eng: TDM) i kombination med en statisk tidsplan til at implementere virtuelle end-to-end kredsløb. TDM er en ligefrem måde hvorpå garantier og effektiv deling af ressourcer kan tilvejebringes med en effektiv hardware implementering. Argo supportere systemer organiseret ved hjælp af GALS og derudover kan Argo bruges til at udforske en struktur med en mere fleksibel tidssynkronisering, der adressere signal distributions problemer med et netværk af asynkrone routere.

NOCs består af en kommunikations struktur af routere forbundet af links, med et netværks interface (NI), der forbinder processorne til kommunikations strukturen. Argo bruger et nyt NI design der supportere tidsforudsigelighed og asynkrone routere der udgøre et tidselastisk netværk. NI designet integrer DMA funktionaliteten og TDM tidsplanen, og bruger to-ported lokale hukommelser. Routerne kombinere routingsfunktionalitet og den asynkrone elastisk opførsel. Routerne bruger også en gating mekanisme til at reducere energi forbruget. Kombinationen af NI designet og router designet supportere kommunikations kannaler gennem NOC'et, fra den lokale hukommelse hos en sendende kerne til den lokale hukommelse hos en modtagende kerne. Disse kommunikations kannaler kræver ingen dynamisk arbitrering, buffering, flow kontrol eller klok synkronisering i hverken routere eller NI'er.

Denne afhandling udforsker implementationer af de individueller komponenter i Argo, såvel som en række af komplette instanser af Argo NOC'en. Implementationerne henvender sig både til FPGA teknologi og 65 nm CMOS teknologi. Det bliver vist at (i) NI designet er skalerbart og fire til fem gange mindre end tidligere publiceret NI designe af lignende NOC'er, (ii) router designet er energi effektivt og to til tre gange mindre end lignende router design, og (iii) i alt

er Argo NOC'et cirka fire gange mindre end andre TDM NOC'er. Argo er en vigtig del af T-CREST platformen og brugt i et antal af konfigurationer.

Argos fleksible tidssynkroniseret organisation kombinere asynkrone routere med mesokrone NI'er, som er forbundet til individuelt klokkede kerner, der supportere GALS system organisation. De mesokrone NI'er operere ved den samme frekvens, muligvis med tidsforskydelse af klokken, men netværket af asynkrone routere kan absorbere denne tidsforskydning op til en vis grænse. Elasticiteten af det asynkrone netværk bliver udforsket for at svare på spørgsmålet, hvor meget tidsforskydning af klokken kan Argo NOC absorbere. En kvalitativ analyse studere hvordan parametre påvirker elasticiteten og dens grænser. En kvantitativ analyse modellere Argos opførsel ved hjælp af timed-graph modeller og en analyse af værste tilfælde af tidsseparation mellem begivenheder for at evaluere elasticiteten af Argo. Resultatet viser at den tidsforskydelsen af klokken som netværket af routere kan absorbere er to eller flere klok cykler, afhængigt af den frekvens der påvirker NI'erne.

Samlet set præsentere denne afhandling design og implementation af Argo og analysen af dennes elastiske opførsel. Afhandlingen viser at Argo levere hårde realtids garantier på en ligefrem måde, samt har en effektiv implementation og en elastisk tidsopførsel.

Preface

The work of this thesis was conducted at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements of the Ph.D. program. The work of this thesis was partially funded by the European Union's 7th Framework Programme project Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST) under grant agreement no. 288008. The work of this thesis comprises work package 3 of the T-CREST project. The result of this thesis is part of the T-CREST platform and is avaliable under the BSD open source licence.

This work was supervised by Professor Jens Sparsø and Associate Professor Martin Schoeberl. Some of the results presented in this thesis were obtained in collaboration with Rasmus Bo Sørensen, Christoph Müller, and Ioannis Kotleas. The text of the thesis indicates whenever results of such a collaboration are presented. The integration of the T-CREST platform was a collective effort of the whole T-CREST project group. In addition, Joachim Rodrigues and Oskar Andersson from the Department of Electrical and Information Technology of Lund University provided help with the EDA tools.

The thesis describes the design and implementation of the Argo NOC and an analysis of its elastic timing. The thesis consists of ten chapters organized in four parts. The first part introduces the subject and presents the state of the art. The second part presents the design and implementation of the Argo NOC. The third part presents the analysis of its elasticity. The fourth part concludes the thesis.

The thesis does not contain any material that has been accepted for the award of any other degree or diploma in my name, in any university or other institution and, to the best of my knowledge does not contain any material previously published by another person, except where due reference is made in the text of the thesis.

Lyngby 27 March 2015

Evangelia Kasapaki

Publications and Technical Reports

Journal Publications

1. M. Schoeberl and S. Abbaspour and B. Akesson and N. Audsley and R. Capasso and J. Garside and K. Goossens and S. Goossens and S. Hansen and R. Heckmann and S. Hepp and B. Huber and A. Jordan and E. Kasapaki and J. Knoop and Y. Li and D. Prokesch and W. Puffitsch and P. Puschner and A. Rocha and C. Silva and J. Sparsø and A. Tocchi. T-CREST: Time-predictable Multi-Core Architecture for Embedded Systems. In Journal of Systems Architecture, 2015. published online, to appear in print.

2. E. Kasapaki and M. Schoeberl and R. B. Sørensen and C. T. Müller and K. Goossens and J. Sparsø. Argo: A real-time network-on-chip architecture with an efficient GALS implementation. In IEEE Transactions on VLSI Systems, 2015, to appear.

Conference Publications

1. E. Kasapaki, and J. Sparsø. The Argo NOC: Combining TDM and GALS. In *ECCTD, 2015*, Aug 2015.

2. C. Müller, E. Kasapaki, R. Sørensen, and J. Sparsø. Synthesis and layout of an asynchronous network-on-chip using standard EDA tools. In *NORCHIP, 2014*, pages 1–6, Oct 2014.

3. I. Kotleas, D. Humphreys, R. Sørensen, E. Kasapaki, F. Brandner, and J. Sparsø. A loosely synchronizing asynchronous router for TDM-scheduled NOCs. In *Networks-on-Chip (NoCS), 2014 Eighth IEEE/ACM International Symposium on*, pages 151–158. IEEE, Sept 2014.

4. E. Kasapaki and J. Sparsø. Argo: A time-elastic time-division-multiplexed NOC using asynchronous routers. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 45–52. IEEE Computer Society Press, May 2014.

5. E. Kasapaki, J. Sparsø, R. Sørensen, and K. Goossens. Router designs for an asynchronous time-division-multiplexed network-on-chip. In *Proc. of Euromicro Conference on Digital System Design (DSD)*, pages 319–326, Sept. 2013.

6. J. Sparsø, E. Kasapaki, and M. Schoeberl. An area-efficient network interface for a TDM-based network-on-chip. In *Proc. Design Automation and Test in Europe (DATE)*, pages 1044–1047, Mar. 2013.

7. M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Proc. IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, pages 152–160, May 2012.

Technical Reports

1. E. Kasapaki and R. B. Sørensen. Argo programming guide.

2. T-CREST Project. D3.1 - Survey of time-predictable and asynchronous NOCs, and their WCET analysis. `http://www.t-crest.org/page/results`, 2014.

3. T-CREST Project. D3.2 - Simulation model of the self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

4. T-CREST Project. D3.3 - Hardware implementation of the self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

5. T-CREST Project. D3.4 - Report documenting the hardware implementation of the self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

6. T-CREST Project. D3.5 - Report on impact of asynchronicity on predictability of the NOC. `http://www.t-crest.org/page/results`, 2014.

7. T-CREST Project. D3.6 - FPGA implementation of self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

8. T-CREST Project. D3.7 - Analysis report on FPGA implementation of self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

9. T-CREST Project. D3.8 - Integration report of the full system implemented in an FPGA. `http://www.t-crest.org/page/results`, 2014.

Acknowledgements

First of all I would like to thank my supervisor Jens Sparsø for his invaluable help and support, for sharing his experience and ideas and for the inspiring discussions. I would also like to thank my co-supervisor Martin Schoeberl for his feedback and his encouragement in reaching for the best. I am grateful to both of them for their time and for the opportunities they have given me.

I also appreciate the brainstorming and the interesting discussions with my colleagues Rasmus, Christoph, Luca, Ioannis, Florian, Alexander, and Sahar.

I am always grateful to my parents for their love and support in my every pursuit.

I would also like to thank Wolfgang for his constant help and support in every aspect of this path.

I would also like to thank my friends Aliki and Maria for being close even from far.

Furthermore, I would like to thank the fellow PhD students and friends Domi, Laura, Alessio, Fontas, Stavros for walking the same path and showing me the way. I enjoyed all the fun moments and new experiences we shared together.

Moreover I would like to thank Nikos for always being there for a good discussion or a chat, Søren for helping me find my way in Denmark, Nils for all the dances and for his friendship, Aris and Spyridoula for being there for a break and whenever needed. And finally thanks to all the dance people that shared a dance with me and made Denmark a warmer place.

This thesis was funded by the European Union's 7th Framework Programme project Time-predictable Multi-Core Architecture for Embedded Systems (T-CREST) under grant agreement no. 288008. I would like to thank the partners of the project for the interesting discussions during this work.

Contents

Part I.

Introduction and State of the Art

1
Introduction

Digital systems are an integral part of everyday life; they exist as embedded systems in commonly used devices. Consumer electronics, household appliances, telecommunication systems and industrial, transportation, and medical applications are some of the areas that use embedded computational systems. The functionality of these systems is often critical for the safety of people, as in transportation and medical applications. In addition, the timing of these systems is important, as in transportation. Thus, besides the requirements for efficiency, these systems also have real-time requirements for their operation.

Real-Time systems are systems with real-time requirements from their environment. They are built to guarantee the response of each operation within strict timing constraints. Efficiency is not the primary goal. To provide timing guarantees, it must be possible to analyze them and to provide worst-case guarantees of their operation and timing.

According to Moore's Law, increasing computational needs drive demands for higher performance systems. The high performance processors built nowadays do not satisfy these computational needs. The current approach to providing higher performance systems is to integrate a number of processors in a single system on a chip. *Multi-processor Systems-on-Chip (MP-SoC)* facilitate the construction of larger systems in a composable way, the reuse of individual components, the reduction of design-time, and high performance. Such systems require a communication medium that satisfies the communication requirements, which increase as the system gets larger. Networks-on-chip is the most promising approach to deal with the increasing communication requirements.

Furthermore, advances in silicon technology and manufacturing processes lead to integration of larger systems in smaller areas. The effects of long wires and signal distribution in large systems cause critical problems for the functionality of these systems. An important problem is the clock distribution over the entire system. Problems of clock skew and jitter demand complex solutions for correct operation.

The focus of this thesis is the implementation of a Network-on-Chip for a MPSoC platform that is suitable for real-time systems. This implies that it must be analyzable and able to offer real-time guarantees. In addition, as determined by technology advances, it must be efficient, matching the current computational needs, and it must be able to handle signal distribution issues over the chip area.

1.1. Real-Time Systems

Embedded systems commonly have real-time requirements.

> *Real-time systems* are those systems in which the correctness of the system depends not only on the logical results of computation but also on the time at which the results are produced [155].

While *general purpose* systems focus on performance improvement, *real-time* systems focus on the time-predictability and analyzability of the system. The goal is to provide guarantees on the worst-case behavior, rather than optimized performance. A special subcategory is *hard real-time* systems. In such systems the consequences of failing to meet a deadline are severe [155]. Hard real-time systems are the context of this thesis.

Real-time requirements refer to both hardware and software. The development of such systems puts emphasis on the analyzability of the entire system. The standard approach for computational systems emphasizes optimizing the average-case performance. However techniques aiming at performance optimization often make the system difficult to analyze or have a negative effect on the worst-case execution. Real-time systems should be developed in a way that they are easy to analyze and are time-predictable, and consequently they are able to provide worst-case execution-time (WCET) guarantees. A metric for optimization is to optimize the worst-case performance instead of the average-case.

Approaches that satisfy the WCET guarantees focus on controlling the execution of tasks over time. This may include analysis of task data flows, or time scheduling of tasks.

1.2. MPSoC and NOCs

The increasing computational needs and Moore's Law have led to exploration of architectures that take advantage of modularity and reuse. According to International Roadmap of Semiconductors [161] the silicon process leading to higher transistor density, the demands for higher performance, and the demands for decreasing time-to-market, drive the development of integrated systems with many processing cores on a single chip, i.e., *Multi-processor systems-on-Chip (MPSoCs)*. MPSoC systems allow modularity and reuse, thereby reducing the time-to-market and increasing productivity. In addition, they offer higher computational power compared to single high-cost and high-performance processors.

MPSoCs aim at increasing the performance of systems. They provide architectures that take advantage of reuse and modularity and provide better performance. While they meet their goal in terms of design productivity and computational needs, limitations arise with regard to the on-chip communication. As systems become larger with more processing cores to satisfy the

computational needs, the communication becomes a bottleneck. Additionally, the evolution of silicon technology with regard to transistor sizes, i.e., deep submicron technologies and nanotechnologies, leads to high-density integration and allows bigger systems to be placed on a chip, further increasing the on-chip communication. The challenges in deep sub-micron and nanotechologies, including increased leakage power, temperature susceptibility, signal integrity, and parameter variability, make it even more important to find an efficient and reliable way of implementing the communication. The above factors indicate the need for efficient communication.

Traditionally, buses have been used to facilitate on-chip communication. However, they do not meet the current demands for performance and scalability. In large systems, with many processing cores demanding access to a shared bus, the bus becomes a bottleneck. *Networks-on-Chip (NOCs)* satisfy the needs for communication performance, scalability and flexibility.

1.3. Timing Organization

The increasing size of systems integrated on a single high-density chip raises signal distribution issues. Signals such as clock and reset that need to be distributed over the entire chip area pose a big challenge for most systems. Traditional globally synchronous systems fail to maintain the performance scaling improvements. Approaches to meeting the clock distribution challenge include extensive clock-tree synthesis to achieve maximally simulteneous arrival of the clock to every part of the chip and more flexible timing organizations of the system, such as mesochronous or asynchronous timing. These flexible timing organizations offer timing elasticity, such that they tolerate clock variations or entirely eliminate the clock. When applied to an entire system, all these techniques carry an overhead. The most widely used approach, especially in MPSoCs, is the *Globally-Asynchronous Locally-Synchronous (GALS)* organization. GALS systems consist of a number of synchronous islands that communicate with asynchronous protocols. This organization is ideal for the modular construction of MPSoCs. However, the problem of signal distribution is still apparent for NOCs, which span the entire chip area. Therefore, the clock distribution within the NOC is still a challenge, requiring a more flexible solution for the NOC itself.

1.4. Goals and Challenges

The focus of this thesis is a NOC suitable for real-time systems and for GALS system organization. The goal is to design and implement a NOC that is: (i) time-predictable, such that it provides guarantees for real-time requirements, (ii) time-elastic, such that it is tolerant to signal distribution and synchronization issues, and (iii) has an efficient implementation.

A particular challenge with this goal is to combine time-predictability and time-elasticity, two intuitively contradicting features, in one design. Traditional approaches offer solutions that achieve one or the other feature, and combinations include the elements of both solutions with an increased cost for integration. Our approach deals with both features in one solution, while also facing the challenge of finding an efficient implementation. It acieves that by rethinking

the design and applying each feature in different levels of the design as it will be explained in detail in 4.

1.5. Contributions

This thesis contributes the design, the implementation, and the elasticity analysis of an asynchronous time-predictable NOC, named Argo. The contributions of this thesis fall within two domains: Firstly the design and implementation of the Argo NOC, and secondly the analysis of its elastic behavior.

Design and Implementation:

The first domain to which this thesis contributes is the design and implementation of Argo. The contributions include the design of the NOC's building blocks, i.e., router and network interface (NI), the integration of them in the Argo NOC, the novel timing organization, and the implementation of Argo.

- **Router:** A contribution of this thesis is a novel NI design. This supports the time-predictability of the NOC by applying a TDM communication scheme. The NI is implemented in ASIC and FPGA technologies, and evaluated in terms of area and scalability. The implementation results show that the NI is very efficient compared to alternative NI designs in the literature.

- **NI:** A second contribution is the design and implementation of an asynchronous router, using the time-division multiplexing (TDM) scheme. The contribution involves exploration of several designs, their implementation in ASIC and FPGA technologies, and evaluation. The ASIC implementation includes both synthesis and layout of the designs. The final asynchronous Argo router is area-efficient, time-elastic, power-efficient on idle traffic, and with comparable speed to the equivalent synchronous designs.

- **NOC:** The integration of the above components into the entire Argo NOC involves a number of design decisions. The combination of the design decisions both in the router and NI lead to an efficient Argo NOC design. This novel design creates direct paths from the local memory of a processing core to the local memory of a remote processing core, thereby allowing the data to traverse the NOC without the need for any arbitration, buffering, end-to-end flow control, or synchronization. The contibution of this thesis includes implementation of the integrated Argo NOC in ASIC and FPGA technologies. The ASIC implementation includes synthesis and layout. The FPGA technology offers the possibility to prototype. Argo is shown to be very area-efficient, while being time-predictable.

- **Timing Organization:** Another contribution of the design refers to the timing organization of the NOC. Argo supports GALS system organization. The novel contribution of this thesis refers to the combination of time-predictable NIs with a time-elastic asynchronous network of routers. This timing organization is a novel approach that allows

the NIs to operate mesochronously while the elastic asynchronous network absorbs skew in signal distributions. This organization offers time-predictable communication that is highly tolerant to signal distribution issues.

Elasticity Analysis:

The second domain to which this thesis contributes is an analysis of the elastic timing behavior of the Argo NOC. The contribution includes an elasticity analysis of the network of asynchronous routers and its safe encapsulation in a time-predictable environment of mesochronous NIs.

- **Qualitative Analysis:** A contribution of this thesis is the understanding of the principles of the elastic network of asynchronous routers. This thesis contributes by providing insight into the elastic behavior and exploring its limits. A qualitative analysis contributes a relation of elasticity to performance. The exploration of the limits of the elasticity sets the elasticity bounds for safe operation.

- **Quantitative Analysis:** The last contribution of this thesis is a model analysis of the Argo NOC structure. This thesis contributes in describing the elastic behavior of Argo in timed-graph models, and analyzing them with an algorithmic/mathematical tool. The results of this analysis are supported by the implementation of an Argo instance. The quantitative analysis shows the wide limits of Argo elasticity.

1.6. Thesis Organization

The thesis is organized in four parts. Part I sets the scene for the thesis. Chapter 1 introduces the topic and the main aspects that are dealt with in the thesis. Chapter 2 presents the background and context of the thesis. The chapter includes background on real-time systems, multi-processor systems-on-chip, networks-on-chip, networks-on-chip from the real-time aspect, and asynchronous design. In addition Chapter 2 presents the T-CREST multi-core platform, which is the context for the network-on-chip developed in this thesis. Chapter 3 reviews the state-of-the-art in real-time projects, the approaches in real-time NOCs and in timing organization of NOCs. Furthermore, it analyzes in more detail some representative NOC designs in the area of real-time NOCs.

Part II of this thesis presents the design and implementation of the Argo NOC. Chapter 4 presents the basic principles of Argo. It deals with the overall NOC architecture and the overall timing organization. This includes the initialization process, the scheduling of the communication, and the real-time properties of the NOC obtained through a WCET analysis. Chapter 5 presents the Argo NI design and its implementation. This includes the basic ideas that lead to an efficient design, its architecture and functionality, and its interaction with the asynchronous network of routers. The chapter also presents the results of the NI implementation. Chapter 6 presents the asynchronous Argo router design and its implementation. The chapter also presents the alternative router designs that were explored in this work. Finally, implementation results of all the design are presented and compared. Chapter 7 presents the Argo NOC

implementation. This includes simulation, and implementation of various instances in various technologies, the challenges, and the results from this implementation. Additionally, this chapter presents the implementation of the entire T-CREST platform and ways to use the Argo NOC at the application level. As the Argo NOC code is open-source, the chapter provides information on how to access the code.

Part III of this thesis presents the elasticity analysis of Argo. Chapter 8 presents the elastic timing behavior of Argo and the qualitative analysis of the elasticity of Argo. Chapter 9 presents the quantitative analysis of the elasticity in Argo. This includes the description of the Argo NOC in two different timed-graph models and a analysis through the timing separation of events approach.

Finally, part IV gives an overview of the work done in this thesis and the insights gained. Chapter 10 presents the conclusions of this work. This inlcudes a review of the contributions, assessment of the results, potential open issues and overall conclusions.

2
Background

This chapter provides the background knowledge which this thesis builds on. It describes the basis concepts of real-time systems and the various parameters related to them, and presents MPSoCs and the challenges they involve in more detail, as they were briefly touched upon in Chapter 1. It also presents the T-CREST project, as the MPSoC platform that is the context for the Argo NOC, and provides a more in-depth presentation of NOCs, explaining terminology and the basic concepts. After presenting the background on real-time systems and NOCs, this chapter focuses on real-time NOCs and their timing properties. Finally, this chapter provides some background material on asynchronous design, required for the understanding of the principles and concepts used in the development of the asynchronous Argo NOC.

2.1. Real-Time Systems

As mentioned in Chapter 1,

> *real-time systems* are those systems in which the correctness of the system depends not only on the logical results of computation but also on the time at which the results are produced [155].

A common basis for all real-time systems is the *predictability* of the system. A system is predictable if it can guarantee that an operation can be completed within its requirements [154]. For a system to be *time-predictable*, it should provide a way to analyze whether the operations executed on the system meet their deadlines. To do so, the operations of the system are organized in *tasks* with specific deadline requirements and known *worst-case execution times (WCETs)*.

Many classifications of real-time systems, based on different parameters affecting them, have been presented in the literature. Stankovic et al. [154] define five dimensions, according to which they classify real-time systems. These dimensions are: (i) the granularity of the

deadline and the laxity of the tasks, (ii) the strictness of the deadline, (iii) the reliability requirements, (iv) the size and coordination of components, and (v) the environment. Kopetz [77] makes the following distinctions: (i) hard and soft real-time systems, (ii) fail-safe and fail-operational, (iii) guaranteed response and best effort, (iv) resource-adequate and resource-inadequate, (v) event-triggered and time-triggered.

Some of the parameters affecting a real-time system and the corresponding classifications are explained below:

- With respect to the strictness of the deadline of an operation or task, real-time systems are divided into *soft* and *hard* real-time [154, 77]. Hard real-time systems require their tasks to be completed within strict deadlines. In soft real-time systems, tasks may complete, even after the required deadline. A system may also execute a combination of hard and soft real-time tasks.

- With respect to the reliability and the effect of failure, some operations are defined as *critical* [154] or *safety-critical*. Missing a deadline of a critical task has severe consequences. Resources are therefore often dedicated to critical tasks, even if they are not used for the remaining operation. A system may solely execute critical tasks or have tasks of different criticality levels. To avoid the severe consequences after a failure, some real-time systems detect and identify the failure, such that they make a transition to a safe state or operational state with minimal consequences, gracefully degrading. The former type of system is called *fail-safe* and the latter *fail-operational* [77].

- With respect to the guarantees given, a real-time system can be classified as *guaranteed response* or *best effort* [77]. A guaranteed response system can provide analytic guarantees, based on load and fault worst-case assumptions. A best effort system provides the best achievable response in its current condition.

- With respect to the provision of resources a real-time system can be classified as *resource-adequate* or *resource-inadequate* [77]. Resource-adequate systems offer sufficient resources for every possible scenario. As this may result in a waste of resources, some real-time systems are resource-inadequate but rely on probabilistic analysis and dynamic allocation of resources. However, safety-critical systems need to be resource-adequate.

- With respect to the size of the system, the coordination of components may cause complications [154]. Tasks or components of a system need to operate in coordination. The completion of one task may depend on the completion of others. *Isolating* the execution of tasks simplifies the predictability analysis, but as the size of a system increases the interactions become more complex.

- With respect to the way in which operations are executed in a system, real-time systems are classified as *event-triggered* or *time-triggered* [77]. In event-triggered systems, operations are initiated by given events. In time-triggered systems the operations are initiated at specified points in time.

- With respect to the environment and the interaction of the system with it, real-time systems are classified as *static* or *dynamic* [154]. The environment of a system can be

deterministic and well-defined but in many cases the environment is dynamic. Systems that operate in deterministic environments can be static. To interact with a dynamically changing environment, a system needs to adapt to the changes of the environment. Such a system is dynamic.

A system is composed of a number of components at the hardware and software level. The hardware platform, the operating system, the compiler, and the application tasks comprise different layers of the system. Stankovic et al. [154] refers to two approaches to achieve predictability. In the *layer-by-layer* approach, every layer of the system is required to be predictable. In such a system, all components need to provide WCET guarantees, e.g. for the use of hardware resources, or for the execution of application tasks. This method requires careful planning and scheduling of all operations involved, as well as their interactions, such that everything is guaranteed. In the *top-layer* approach, an effort is made to ensure that the top layer is time-predictable and that the rest of the layers support the necessary actions, so that predictability is achieved at the top level.

The operation of real-time systems often relies on scheduling and careful planning of the sequences of the tasks. The scheduling is done statically or dynamically. Static scheduling evaluates the requirements of the tasks executed on the system and generates a static schedule of the tasks. It considers all tasks and their requirements to be known. This type of scheduling is good for systems in static environments, or in systems with static behavior. Dynamic scheduling considers the current condition of the system and evaluates a schedule based on this. Scheduling can be done off-line or on-line. Schedulability analysis can also be applied to analyze whether a system running a set of tasks is able to meet its requirements [156]. Liu et al. [82] deal with scheduling algorithms for multiple tasks on a single processor for hard-real-time systems. Ramamritham et al. [134] explore the scheduling and operating system support for real-time systems, and also scheduling algorithms for real-time multiprocessor systems [133].

The Argo NOC presented in this thesis was developed in the context of a hard real-time system. The approach to predictability followed in the system is the layer-to-layer approach. It uses isolation of operations and provides guaranteed responses in the operations executed in the NOC. Argo follows off-line static scheduling of communication. This also includes analysis for the adequecy of resources, as the requirements are analyzed to generate the schedule. The operations executed in the system are time-triggered.

2.2. Multi-Processor Systems-on-Chip

To meet the increasing computational needs and to exploit modularity and reuse, the dominant design approach in embedded systems is to use multi-processor systems. A typical multi-processor architecture is illustrated in Figure 2.1. It consists of several *processing elements (PEs)* or *processing cores*, organized in grids or clusters. The architecture includes a memory system. This can be a single off-chip shared main memory (DRAM), or multiple on-chip memory units, owned privately by the cores, or shared among clusters of cores. The memory is organized in a hierarchy that consists of the main memory, caches in the cores, and possibly local private memories. In addition, the cores typically contain DMA controllers, implementing data transfers separately from the core operation. In this way, computation is separated from

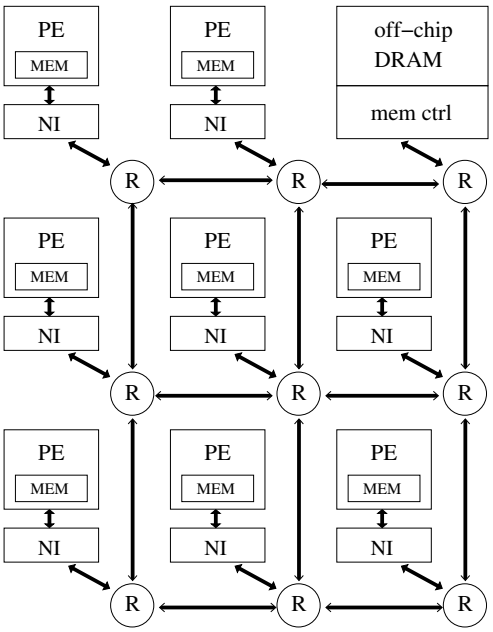


Figure 2.1.: Example multi-processor system.

communication. The cores and the shared memory are connected with an interconnection medium. This medium may be a shared bus or a network. The network, as shown in Figure 2.1, is typically a structure of routers (R) connected through network interfaces (NI) with the cores. Challenges in the development of such systems include optimization of communication, memory access, real-time guarantees, and software development for parallelization [68].

2.2.1. On-chip Communication: Networks-on-Chip

The focus of this thesis lies on the challenges of on-chip communication. Traditionally, buses have been used for this purpose, but they do not meet current demands on performance and scalability. In large systems with many processing cores demanding access to a shared bus, the bus becomes a bottleneck. Benini in [19] suggests that the interconnection technology will be the limiting factor in future SOCs. Interconnection networks, and in particular *Networks-on-Chip (NOCs)* offer a better and more flexible solution for meeting the communication requirements. NOCs satisfy the needs for scalability and flexible sharing of the communication network [57], [19]. They offer reliable and efficient communication, in terms of power and clock synchronization [18]. They are scalable, modular, testable, and reliable [147]. Arteris presents a comparison between a traditional bus example design and a NOC architecture [8]. NOCs are also supported as a better solution over direct wiring connections, for their advantages in structure, performance, modularity and reliability [36] [176]. NOCs and their fundamentals will be presented in more detail in Section 2.4.

2.2.2. Timing Organization and GALS

The increasing performance demands and the evolution of silicon technology has led to integration of larger systems with multiple components on a single chip. Therefore, the distribution of a single clock signal to the entire, dense chip area is becoming an increasingly challenging process due to parameter uncertainty and variability [161]. Different synchronization options are presented by Messerschmitt [94].

Globally synchronous systems require the concurrent arrival of the clock to every part of the system. In large multi-processor systems, with long wires, skew and jitter effects make this an impossible task. Techniques to deal with these effects include building buffer networks to balance the clock delays in different parts of the chip [136].

Another approach to dealing with clock distribution and synchronization is *mesochronous* timing organization. The global clock arrives with some phase difference, i.e., *skew*, to various areas of the chip. Thus, one global frequency is maintained throughout the chip, but with different phase differences in different areas. With mesochronous timing organization, the various parts of the system operate on the same frequency but with a bounded phase difference. Parts with clock skew are synchronized using mesochronous synchronizers [105], [93].

Entirely *asynchronous* timing organization is also possible. However, the implementation of an entire multi-processor system asynchronously is a challenging task. As the asynchronous design process is not well-supported by EDA tools, the development and testing of a large system is a practical challenge.

A different approach, widely used in recent years, to timing organization is the *Globally-Asynchronous Locally-Synchronous (GALS)* organization [78]. The design consist of a number of parts operating in independent clock domains, and therefore locally synchronous. The individual clock domains communicate asynchronously, so the system is globally asynchronous. A NOC-based MPSoC platform naturally supports a GALS timing organization where the components of the system operate within different clock domains [78]. The most common way to resolve synchronization among the different clock domains is with FIFO based synchronizers. Some solutions are presented in [30], [29], [17].

2.3. The T-CREST Platform

This thesis was conducted within the FP-7 project T-CREST. The T-CREST project aims at the development of a real-time multi-core platform. The goal is to redesign each of the components of the architecture with a focus on analyzability and time-predictability. The typical design approach for general purpose systems is to optimize the average case, possibly leading to a poor worst-case performance. The design strategy of the T-CREST project is to optimize the WCET instead of average-case execution time and to achieve a low WCET bound. The project deals with the design of the hardware platform, the compiler, the WCET analysis tools, and the development of industrial test cases to evaluate the platform.

The T-CREST platform consists of a number of Patmos processors connected by the Argo NOC, and a shared memory, accessed through a second NOC. The architecture of T-CREST platform is shown in Figure 2.2. Patmos [143] is a RISC-style, time-predictable processor. It includes caches designed with an emphasis on the WCET analysis (method cache [42], data

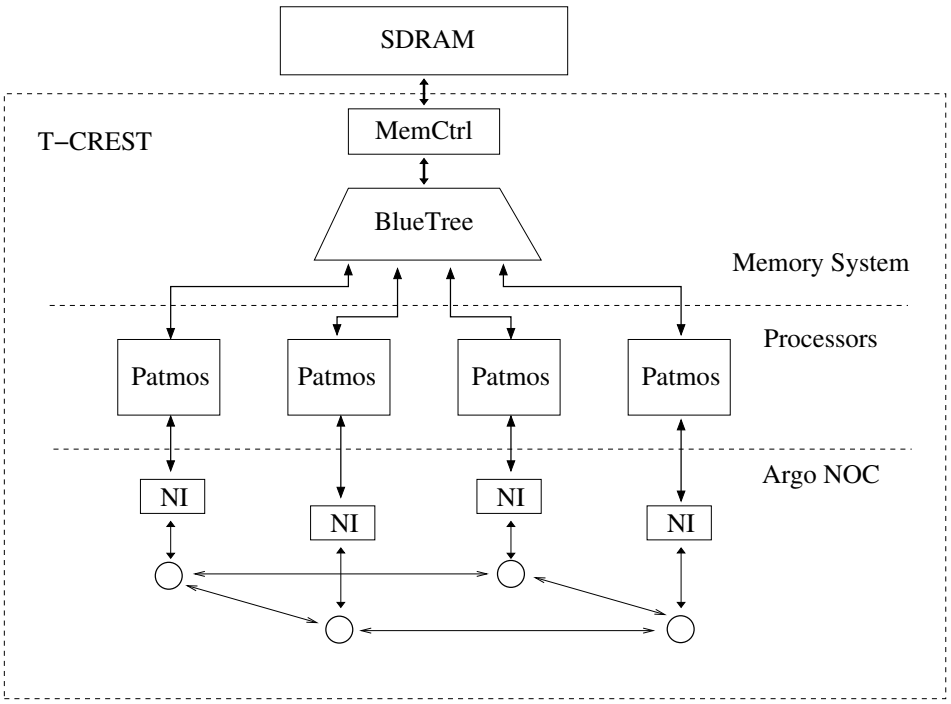


Figure 2.2.: T-CREST platform architecture.

cache, stack cache [2]), each optimized for the specific sets of data which it holds. For each processor there exists a local, private *scratch-pad memory (SPM)*. The Argo NOC connects the Patmos processors, implementing message-passing between them, and is the focus of this thesis. Argo offers end-to-end channels for DMA-controlled transfers of data from an SPM to a remote SPM. A second NOC, the BlueTree [50], a tree-based NOC, provides access to the shared off-chip main memory. Access to the memory is controlled by a memory controller [5], [79], [52].

2.4. NOCs in More Depth

As requirements for high performance lead to multi-core architectures integrated on a single chip (MPSoC), the requirements for on-chip communication increase as well. As presented in Section 2.2.1, a bus is not a suitable medium for large scale communication. NOCs aim at dealing with this communication challenge. As shown in many publications (e.g. [36], [44], [135]), NOCs are the most promising solution for the current communication needs. They offer flexible communication, scalability and favor reuse and composable architectures [36], [57], [19]. This section focuses on NOCs, discussing their basic concepts and architecture.

2.4.1. Basic Concepts

A NOC is a shared medium that provides the infrastructure to implement communication paths among cores connected to this infrastructure. It consists of switching nodes and links connecting the nodes. These comprise the shared resources that are shared among a number of

communication channels. From an architectural point of view, the basic building blocks of a NOC are: (i) the switching nodes or *routers*, (ii) the *links*, and (iii) the *network interfaces (NI)* or *network adapters* that connect the processing cores to the network of routers and links, as shown in Figure 2.1. Routers and links comprise the switching network and NIs bridge the processing cores and the switching network of routers. From a conceptual point of view, the basic concepts of NOCs are their topology, the ways data are transferred throughout the network (routing scheme, flow control), and the quality of service which they offer.

Protocol Stack

A relationship between the NOC design and the OSI protocol stack is often proposed ([19], [135], [22]). This section does not provide a precise OSI mapping but aims at building an understanding of the functionality and the services which a NOC offers. Benini et al. [19] propose a micronetwork stack adaptation of the OSI protocol stack to the services that MPSoCs connected by a NOC implement. According to the micronetwork stack, the architecture and the control of the NOC relate mainly to the *data link layer*, the *network layer*, and the *transport layer*. The *physical layer* of the stack corresponds to the physical wiring in the NOC.

The data link layer relates to services implemented by the routers and the links, concerning the propagation of data from router to router through a link. As the physical layer relates to the physical wires of the link, ways to ensure reliable use of the link are required. These include flow control over the link, and error detection or synchronization functionality implemented in the routers or links. The network layer relates to the network formed by the routers and links. It considers the way the nodes are connected, i.e. topology, the way the data is transferred within the network structure, i.e. routing scheme, etc. The transport layer relates to functionality offered at the endpoints of the network to the processing cores. These services are implemented typically in the NIs and include end-to-end flow control and reliable transfer of entire messages, without knowledge of the internal structure of the routers.

Data Organization

Data is organized in units and the different layers in the above layer abstraction operate on different granularity units. Cores at the end of the NOC consider *transactions* or *messages* as the basic communication data units. At the NIs, *messages* are organized into *packets*. A *Packet* is the basic data unit for an end-to-end transmission. Packets consist of a number of *flits (flow-control digits)*. A *flit* is the smallest data unit to which flow-control and routing are applied, as explained further in this section. Flits can be further divided into *phits (physical digits)*. A *phit* is the data unit that propagates through a physical pipeline stage in one clock cycle. The transport layer is responsible for translating messages to packets. Services of the network layer are applied at the granularity of *packets* or *flits*. Services of the data link layer are applied at the granularity of *flits* or *phits*. Services of the physical layer operate at the granularity of *phits*.

Topology

The *topology* of a NOC refers to the arrangement of routers in the network structure and the way they are connected. The topology can be ring, tree (binary or fat), mesh, torus or bi-

torus, star, and many hybrid and custom structures [38]. The mesh and torus structures can be two-dimensional (2D) or N-dimensional for arbitrary N, and they are the most common [137]. Routers in mesh structures are connected in four dimensions with other routers. The example network shown in Figure 2.1 follows the mesh topology. Tori follow the same principle, with wrap-around at the endpoints of the mesh. Bi-tori use bi-directional links. Æthereal [54] and aelite [59] are examples of mesh topologies while Nostrum [96] follows a bi-torus topology. Proteo [146], Octagon NOC [69], and the time-triggered NOC [140] follow a ring topology. SPIN [57] and Butterfly FT [118] follow a fat-tree topology. Pande et al. [119] compare a number of topologies.

Routing scheme

The *routing scheme* refers to the way in which the path which a packet follows from source to destination, over a number of nodes (hops) of the switching structure, is chosen. There are several routing algorithms based on the requirements and focus of an NOC. Routing decisions may be made at the source node, (*source routing*), or in the switching nodes, (*distributed routing*). In source routing the routing information of the packet is attached to the packet at the source and travels along with the packet thoughout the route. In distributed routing the decision is made and executed in the switching nodes. There are *minimal* and *non-minimal* algorithms, in terms of whether the chosen route is the shortest or not. Based on the outcome of the routing algorithm, the algorithm may be *deterministic* or *adaptive*. Deterministic routing algorithms always result in the same route for the same source-destination pair. The adaptive algorithms consider the state of the network while making the route selection dynamically [44]. These algorithms reduce congestion in the network, but are more complex to implement, thus they introduce a high implementation cost. Static routing is also possible, by analyzing the behavior of the NOC and providing static connections.

An example of an algorithm that is deterministic and minimal is the dimension-order routing. According to this, packets are routed in every dimension in strict order. XY-routing is often used in 2D mesh or tori. In XY-routing, packets are routed first in the X and then in the Y dimension. Hu et al. [64] and Siebenborn et al. [145] analyze the communication to provide static routing decisions. Neeb et al. [110] compare deterministic and adaptive routing algorithms. Hu et al. [63] and Ascia et al. [9] further explore routing techniques.

Flow control

Another characteristic of a NOC is the *flow control* or *switching technique*. Flow control refers to the way data is forwarded through the network [44], [135]. It can be viewed as a resource allocation problem or a contention resolution problem [38]. The resources involved can be bandwidth, buffering or control. Flow control is applied on the level of routers, i.e. from one router to the next, or at the level of NIs, i.e. end-to-end flow control.

There are two main switching techniques that control the propagation of data from router to router; *circuit-switching* and *packet-switching*. In *circuit-switching*, connections are set before transmitting the data, such that end-to-end paths, i.e., circuits, are established and the data travels unblocked from source to destination. Consequently, circuit-switching requires no buffering. In contrast to this, *packet-switching* allocates bandwidth and buffering to data units,

i.e., packets or flits. Flow control in packet-switching is applied at the granularity of *packets* or *flits* and it requires corresponding buffering resources in the routers.

In more detail, depending on the granularity of the packet-switching mechanism, we can distinguish between mechanisms applied when using packet granularity and flit granularity. *Store-and-forward* and *(virtual) cut through* are packet-switching flow-control mechanisms applied with packet granularity. *Wormhole switching* or *wormhole routing* is applied at the granularity of flits. In store-and-forward, a router stores a packet and waits until the entire packet arrives before forwarding it to the next node. In virtual cut-through, the router forwards the data as soon as the header arrives. Both techniques forward the packets as long as there is enough space for an entire packet in the next router. In wormhole routing a flit is forwarded upon arrival if the next router has available buffering space for a flit. Thereby, a packet transmission is pipelined through the network and a router needs to store only one flit. Most NOCs are packet-switching NOCs with wormhole routing and varying buffering configurations, e.g. [4], DSPIN [99], QNOC [27], XPIPES [21], Hermes [103], etc. Bufferless packet-switching NOCs are also possible; these drop the packets that cannot be forwarded to the next hop. Such NOCs include techniques for detecting and resending the packet if it is dropped. A way to control the availability of buffering space over the next hops is by using *credit-based* schemes as in QNOC [27] and MANGO [25]. *Virtual-channels (VCs)* are used to multiplex the use of physical links. MANGO [25] is a example of a NOC using VCs. Mello et al. [92] evaluate the use of VCs in Hermes [103].

An alternative approach for flow-control is *virtual circuit-switching*. Virtual circuit-switching multiplexes the allocation of resources to data units. *Time-division multiplexing (TDM)* allocates resources to virtual circuits over fixed-duration time-slots, such that end-to-end circuits are formed in specific time-intervals. Nostrum [96], Æthereal [54], and Aelite [59] are based on TDM schemes. *Spatial division multiplexing* is another way of multiplexing multiple virtual circuits over a link. Leroy et al. [81] propose a NOC that statically allocates subset of links to virtual circuits.

A different aspect of flow-control, concerned with the allocation of resources at the end-points,i.e. in the NIs, is *end-to-end* flow-control. A widely used type of end-to-end flow control is *credit-based* flow control. According to this scheme, credits related to the available buffering space at the end-points are exchanged between source and destination. Data transmission at the source is initiated when sufficient credits are received from the destination. Since this is an end-point functionality, it is commonly implemented in the NIs [131].

Quality-of-Service

Another feature of a NOC is the *Quality-of-service (QoS)* which it offers. Aspects of the QoS can be throughput, latency, and data integrity. Two main categories appear with respect to the guarantees they offer to the connections which they implement: Best-effort NOCs (BE) and Guaranteed Service NOCs (GS) [135]. BE NOCs provide no guarantees. They are optimized for the average-case scenario and aim at providing easy, efficient and fair sharing of the network resources. Their main focus is to minimize the latency of a packet transmission under the given conditions, without any guarantee. GS NOCs provide guarantees on throughput and latency. They are optimized for worst-case scenarios and aim at providing guarantees to connections by reserving resources. Most NOCs in the literature aim at providing BE services. However,

some NOCs support the combination of the two services in one NOC [135]. Æthereal NOC [54] implements both services. BE NOCs using priorities aim at providing "softer" guarantees. Examples of such NOCs are the QNOC [27], and the NOC proposed by Felicijan [47]. NOCs based on circuit-switching or virtual circuit-switching, i.e. TDM, provide "hard" GS. Examples are aelite [59], MANGO [25], and Nostrum [96].

2.4.2. Architecture

As mentioned at the beginning of this section the basic building blocks of a NOC are: (i) the *routers* or switching nodes, (ii) the *links*, and (iii) the *network interfaces (NIs)*.

Routers

A typical router provides storage space and control functionality. The functionality it offers ranges from simple switching input to output ports to complex routing and arbitration. The router functionality depends on the specific features which the NOC implements, e.g. the routing scheme, the flow control, the quality of services, etc. A typical packet-switching router supports buffering, routing, allocation of resources, arbitration, and switching. The latency of a packet or flit to propagate through the router depends on the pipeline stages of the router, i.e. the functionality it implements. Various router designs have been published in the literature and their stages range from single to multiple. The typical pipeline stages of a wormhole router include buffering, at the input or output ports, switching of inputs to output ports, and arbitration of output ports. In addition, in the case of distributed routing, the router pipeline implements the routing algorithm. Furthermore, if multiple VCs share a link, a VC allocator allocates VC resources. Peh et al. [122] present the micro-architecture of a typical wormhole router in more detail and a delay model for the router. In bufferless flow control schemes, such as circuit-switching, buffers are eliminated.

The area of a router depends mainly on its buffering capabilities, and additionally on the functionality it implements. As mentioned by Salminen et al. [137], and as shown in many NOC examples (e.g. [75], [146]), the buffers occupy 50-90% of the router area. The buffer area depends on the topology, the flow control scheme, and the flit width. The topology determines the number of ports. The buffers are placed at input ports, output ports, or both. Multiple VCs sharing a link may be associated with separate buffers. The depth of the buffers is also a design decision for the NOC, as well as the flit width. However, the router is highly dependent on the specific features of the NOC which it is a part of, and therefore the router architecture, area and latency vary, depending on the details of the NOC.

Links

Links connect routers to form the network structure. They may be uni-directional or bi-directional, may be shared among several channels, and may be simple physical wires or implement more complex functionality. In the basic case they are simple wires. Alternatively, they address issues of synchronization, link level flow-control, reliability, additional buffering, pipelining and encoding. For example, links in the Aelite [59] and in Spidergon STNoC [33] implement mesochronous synchronization.

Network Interface

The network interface (NI) bridges the processing cores with the network of routers. The NIs translate the processor transactions to the network-specific stream of packets. They implement functionality of the transport layer [19], [33], [22], and they decouple computation from communication [135]. NIs consist of a *front end* and a *back end* (alternatively *shell* and *kernel* [33]), as illustrated in Figure 2.3. The front end implements the interface to the cores. This is a bus-style read-write transaction interface, based on some standard protocol such as AMBA AXI [7] or OCP [3]. The front end transforms the processor transactions into a form of connection-oriented packet-streaming interface, and encapsulates the services provided by the network. The back end implements the interface to the network of routers. This is a packet-stream interface and is specific to the NOC. It implements functions that are related to the NOC functionality, like splitting and re-assembling of packets, routing, buffering and end-to-end flow control.

The NI design architecture is related to the OSI stack model. It follows a layered design approach to separately implement services intended for the different stack layers. The back end corresponds to the services of the network and the front end to services of the transport and session layer.

There are few papers in the literature addressing the design of NIs, and reporting implementation results. Saponara et al. [138], Radulescu et al. [131], Hansson et al. [59], and Copola et al. [33] are a few exceptions reporting implementation results of NIs. Radulescu et al. [131] present the implementation of an NI for the Æthereal NOC. Hansson et al. [59] present the NI implementation of the aelite NOC. Copola et al. [33] addresses the design of a NI for the Spidergon NOC developed by ST Microelectronics, and Saponara et al. [138] reports details of the NI for the same NOC. Saponara et al. [138] compares against a number of other NI designs, including an NI for the Æthereal NOC [131]. The area measures that Saponara et al. [138] report for typical NIs range from 7 to 50 kgates, where a kgate is 1000 minimum size two-input NANDs. Common to all the designs is that the NI back end includes a large amount of buffering, which is the reason for their large area.

2.4.3. Core-to-Core Communication

In most multi-processor platforms, processing cores contain some cache memory and some local scratchpad memory (SPM). Figure 2.3 shows a typical processing core and NI architecture. The processing cores contain instruction and data caches, and a local SPM. The SPM of a processing core contains the message to be transfered to the local SPM of another processing core. Typically, DMA controllers implement background DMA-driven block transfers from core to core. They are typically placed in the processing cores as shown in Figure 2.3 and their role is to offload the core from the data-transfer operations across the network. In this way they isolate the program execution from the network-related operations, making WCET analysis easier. The functionality of a DMA controller includes the transfer of the message data from the local SPM of the core to the local NI. The NI is responsible for organizing the data into packets and inject them to the network of routers.

For a message transfer in a multi-processor platform with a typical NI architecture and a mesochronous network of routers, data crosses several layers of functionality, and several clock

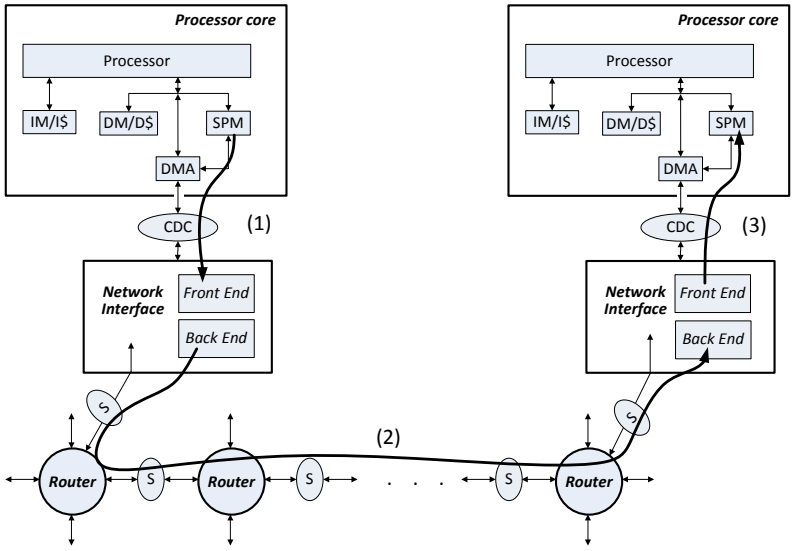


Figure 2.3.: Typical NI architecture and end-to-end communication datapath.

domains from the sending SPM, through the network of routers, and into the receiving SPM. The transfer is illustrated in Figure 2.3 and the following steps describe the process:

1. At the source end, the DMA controllers transfer the data from the local SPM to the front end of the NI, across a clock-domain. A clock-domain crossing (CDC) is required in the interface between the core and the NI. The NIs buffer the data as a number of connection-oriented channels, and organize them into packets for the different channels.

2. Several channels request access to the network of routers and arbitration among them is required. The back end of the NI performs arbitration and injects the packets into the network of routers. The packets travel across the network of routers to the receiving NI, through synchronizers placed on every link. The synchronization along the path and the limited buffering capabilities in the NIs call for end-to-end flow control among the NIs.

3. At the receiving end, the NIs restructure the messages from the packet streams, and the DMA controllers transfer the data from the front end of the NI to the local SPM, across another clock-domain (CDC).

The above steps of the data transfer show that an amount of buffering is required in the NIs. In addition, arbitration and end-to-end control among NIs is required. Clock-domain-crossings at the source and receiving ends, and mesochronous synchronizers along the router paths, add to the complexity of the architecture. Even with TDM that achieves low complexity in the routers, the overhead from arbitration, end-to-end flow control, buffering, and synchronization is large. The Argo architecture entirely eliminates this overhead by a number of design decisions.

2.5. Real-Time NOCs and WCET

Most of the NOCs published in the literature are designed to support BE traffic, and the focus is typically on minimizing average-case latency and maximizing bandwidth utilization. The fact that a NOC is a shared communication medium comprising multiple independently-arbitrated resources (routers, links) may severely complicate timing analysis. The latency that the NOC adds to a read or write transaction towards the main memory or a local memory in a remote node is very difficult to analyze. In a multi-processor platform that consists of several processing core and contains several caches, local memories, and a shared main memory, the traffic in the NOC depends on the execution history of all processing cores. This makes it extremely hard to compute the WCET of a given application executing on a given processing core, in order to guarantee the real-time behavior of this application.

A NOC for a real-time multi-processor platform must provide guarantees on bandwidth and latency for individual processor-to-memory or processor-to-processor transactions. This requires some form of end-to-end connection. There are essentially two ways of achieving this: (i) circuit switching, which can be physical or virtual, e.g. with TDM, and (ii) non-blocking routers with rate control.

Pure circuit-switching (SoCBUS [171], 4S project NOC [174]) establishes connections that own resources exclusively. Therefore real-time guarantees are easily achieved. However, this may result in low utilization of resources. TDM (Æthereal [54], Aelite [59], Nostrum [96]) is another way to implement circuit switching. TDM implements virtual circuit switching, which may result in better resource utilization. The alternative to circuit-switching is to use non-blocking routers with rate control (MANGO [24], Kalray [40]). Packets are arbitrated locally at the routers, where buffering is required. Rate control is applied either locally or at the source to achieve guarantees.

The proposed solutions are analyzed in detail in Section 3.2. In this section we focus on the NOC contribution to the WCET of a transaction and identify the parameters of the NOC design that affect the WCET.

2.5.1. WCET Analysis

There has been a lot of work in the literature on WCET analysis techniques and on scheduling algorithms [82] that consider the WCET of tasks for single processor systems. Scheduling algorithms consider a set of tasks with deadlines and WCET values and evaluate a specific task execution schedule. The WCET analysis of a transaction in the system considers the specific schedule of instructions that are executed, and takes into account the individual components of the hardware platform, i.e. the processor, the cache, and the main memory. To derive the timing of a remote read instruction, the WCET analysis considers the processor pipeline, the time to access the remote memory, and the time to transfer the data.

In a multi-processor platform, as shown in Figure 2.1, analyzing the WCET of a task executing on a processing core is in principle similar. However, the complexity is higher, as the accesses to remote memories will experience some additional latency resulting from interfering traffic in the NOC. The latency which a remote memory transaction experiences in a multi-processor system comes from the latency due to traversing the NOC and the latency to access the remote memory. The latency for the NOC traversal depends on the NOC design,

and on interference from traffic in the NOC. The latency for the memory access involves a possible wait time before access to the memory is granted and the time it takes to access the memory. The latter depends on the memory design and implementation. This section focuses on the latency that the NOC contributes to the execution time of a transaction. To analyze the timing for a remote network access, we identify the parameters affecting it.

The nodes of a multi-processor platform compete for network resources. The communication traffic generated by all the nodes and the outcome of this competition depend on the task execution history in all the nodes of the system. This makes the analysis very difficult in practice. We consider systems that are by design time-predictable and analyzable; in this case the execution history is analyzable and generates predictable communication traffic. Interference from unpredictable communication traffic is out of the scope of real-time multi-processor systems.

Another interference to be noted is the local interference of instructions executed on the same processing node. This is a local issue handled by the processor in question. For a real-time processor, the task execution is analyzable and time-predictable, thus, the local interference is expected to be incorporated in the WCET analysis of the processor.

Additionally, shared memory complicates the analysis of communication time. It is a point where all traffic towards memory converges, so arbitration is needed to grant access to the various memory transactions. This may produce a bottleneck, depending on the traffic in the overall system and the arbitration scheme, and which may severely affect the execution time of a transaction. However, since arbitration is a function of the memory controller, it lies outside the scope of this thesis and will not be taken into consideration.

The focus of this section is to analyze the way in which a NOC impacts the execution time of the instructions of an application. Thus, the scope is limited to estimating the latency of traversing the NOC when the processor accesses resources in remote nodes. The latency of accessing these resources depends on their design.

2.5.2. Impact of NOC on the WCET

An application executed in a processor may contain various sources of network accesses, each potentially affecting the execution time of the application. Sources of network traffic can be cache misses, originating from the instruction/data/stack caches accessing the main memory. Another possible source can be instructions bypassing the caches and directly accessing the main memory. The main memory can also be accessed by block transactions handled by DMAs, producing another source of network accesses. Finally, DMAs can initiate block transfers through the network to and from local memories of other processing cores. The various sources of network accesses generate various patterns of network traffic. The different NOC designs facilitate this network traffic in a different way. However, we identify three parameters that affect the timing of the above mentioned network accesses. These are: (i) the waiting time to acquire access to the network, (ii) the throughput of the network, i.e. the rate in which packets are inserted in the network, and (iii) the latency of one packet to traverse the network from one point to another. In the following, we provide an analysis of the various network accesses and evaluate their timing in regard to the above network parameters.

With regard to the network accesses listed in this section, we distinguish two types of network traffic in terms of the amount of data to be transmitted through the network: Single-word

transactions, coming from load or store instructions bypassing caches, or block transactions, initiated either by DMAs or by cache misses. With respect to the type of transaction, i.e. read or write, four different cases can be identified: (i) single-word read transactions, (ii) single-word write transactions, (iii) block-read transactions, and (iv) block-write transactions.

Depending on the type of transaction, the latency introduced by the NOC consists of two parts: (i) the time for the request to traverse the network, and (ii) the time for the response to traverse the network in the other direction. More specifically, all cases may experience some waiting time ($T_{wait,req}$) in order to get access to the network before the transaction request is sent through the network (taking time T_{req}). Depending on the case, a reply might need to be sent back. This would also require some waiting time for gaining access to the network ($T_{wait,reply}$) and some time to transfer the reply through the network (T_{reply}) back to the requesting node. In general, the network's contribution to the latency of a transaction is then:

$$T_{NOC} = T_{wait,req} + T_{req} + T_{wait,reply} + T_{reply} \tag{2.1}$$

In the following, we take a closer look at each of the four cases, further explaining each timing parameter. The timing parameters do not include the front end services of the NI, i.e. interaction with the processing cores, but rather the network of routers and links and the back end functionality of the NIs.

Single-word read transactions: These consist of a request for data and a reply with the data. T_{req} represents the time of a read request, i.e. the transmission latency of one packet from one end-point to another. T_{reply} corresponds to a single word of data as a response to a read request. It also involves a point-to-point packet transmission latency, similar to T_{req}. Parameters $T_{wait,req}$ and $T_{wait,reply}$ depend on the specific network design. Thus, the time interference from the network for single work read transactions, T_{sr}, is determined by the (two-way) transmission latency of one packet and the waiting times to acquire access to the network.

$$T_{sr} = T_{wait,req} + T_{latency} + T_{wait,reply} + T_{latency} \tag{2.2}$$

Single-word write transactions: These consist of a write request along with the data to be written and optionally a completion acknowledgment. T_{req} includes the time for a write request, and for the data to be transferred from one end-point to another. As a single-word transaction, it corresponds to the point-to-point transmission latency of one packet. Depending on the write scheme, an acknowledge reply is considered or not. In the unacknowledged-write schemes, parameters $T_{wait,reply}$ and T_{reply} are ignored. In the acknowledged-write schemes, parameter T_{reply} corresponds to a point-to-point packet transmission latency. Thus, the time interference from the network in the cases of acknowledged writes, T_{swa}, and unacknowledged writes, T_{sw}, is determined by the transmission latency of one packet and the waiting time to acquire access to the network.

$$T_{swa} = T_{wait,req} + T_{latency} + T_{wait,reply} + T_{latency} \tag{2.3}$$

$$T_{sw} = T_{wait,req} + T_{latency} \tag{2.4}$$

Block read transactions: These consist of a request for data and a reply. T_{req} is a read request and corresponds to a point-to-point packet transmission latency. T_{reply} represents the

time for a block of data to be transferred. This depends on the number of packets in the block of data (n) and on the rate at which packets can be injected into the network (*throughput*). An additional point-to-point packet transmission latency is needed for the last packet of data to traverse the network. Thus, the time interference from the network in block read transactions, T_{br}, is determined by the throughput, the size of the block of data and the waiting time to acquire access to the network.

$$T_{br} = T_{wait,req} + T_{latency} + T_{wait,reply} + n/throughput + T_{latency} \tag{2.5}$$

Block write transactions: They consist of a write request along with the data to be written, and optionally a completion acknowledgment. T_{req} includes the time required to send the write request and the transmission of data to be written. This depends on the *throughput* and the number of packets in the block of data (n). Depending on the write scheme, an acknowledge reply is considered or not. In the unacknowledged-write schemes, parameters $T_{wait,reply}$ and T_{reply} are ignored. In the acknowledged-write schemes, parameter T_{reply} corresponds to a point-to-point packet transmission latency. Thus, the time interference from the network in the cases of acknowledged block writes, T_{bwa}, and unacknowledged block writes, T_{bw}, is determined by the throughput, the size of block of data, and the waiting time to acquire access to the network.

$$T_{bwa} = T_{wait,req} + n/throughput + T_{latency} + T_{wait,reply} + T_{latency} \tag{2.6}$$

$$T_{bw} = T_{wait,req} + n/throughput + T_{latency} \tag{2.7}$$

2.6. Asynchronous Design Perspective

The increasing computational requirements lead to larger, composable systems, like MPSoCs. At the same time advances in technology with decreasing transistor sizes and denser chip areas impose greater challenges in the design process. The synchronous design has been the typical design style. However, clock distribution challenges lead to the exploration of different design styles, in the direction of more flexible timing organizations. As presented in Section 2.2.2, various timing organizations are explored. GALS design is the dominant trend, maintaining the synchronous design principle locally. A re-orientation towards an entirely flexible organization is offered by the asynchronous design style. This section provides an introduction to the basic asynchronous design principles needed for an understanding of the Argo NOC design. The interested reader is refered to [151], [15] for more details.

2.6.1. Why Asynchronous?

The typical design method in circuit design is the synchronous. The operation of a synchronous circuit relies on a clock signal. The functionality is organized in combination logic stages separated by registers. The propagation of data from one stage to the next is synchronized with a global clock signal.The computation is done in a pipelined fashion and is dependent on the fact that all stages are synchronized to the same clock signal. Synchronous design is simple and efficient as it relies on a global clock signal. To facilitate the design process, and reduce

the time-to-market, electronic design automation (EDA) tools have been developed to support the design process with a focus on the synchronous design method. However, decreasing transistor sizes allow bigger circuits to be placed in smaller areas. This makes the distribution of a global clock signal to all parts of the circuit a challenging task. The presence of skew, jitter, crosstalk, and the process variations, call for a more robust way of organizing the computation and propagating data through the circuit pipeline.

As maintaining global synchronicity becomes a challenge, the asynchronous design style gains more attention. Like in the case of synchronous design, asynchronous circuits are organized in combinational logic stages. In contrast to synchronous design, however, the asynchronous circuits do not rely on a global clock. This eliminates the clock distribution issues. The propagation of data from one stage to the next is done through handshaking. This implies a fine-grained control mechanism applied locally between stages. This control mechanism requires handshake control signals and some logic to implement the handshake protocol. Several asynchronous protocols and data encodings have been proposed and Section 2.6.3 presents the different options. The multiplicity of options creates a diverse design space for asynchronous circuits.

There are several potential advantages of asynchronous circuits. Some of these advantages are as follows:

- No clock distribution issues: Synchronous circuits need to distribute the global clock signal in the entire circuit, and handle the various phase differences that appear in different parts of the circuit. The principle of asynchronous circuits is that they operate on handshaking instead of a global clock signal and therefore avoid all clock distribution issues.

- High performance [89], [113]: Synchronous circuits are designed based on the global worst-case performance of their combinational logic stages. As all stages need to be synchronized to a global clock, the period of this clock needs to be sufficiently long to include even the slowest logic stage. Asynchronous circuits use local handshaking, which is determined by the local delays.

- Low power consumption [165], [166], [49]: Synchronous circuits consume power constantly, as every part of the circuit needs to be clocked even when not processing any data. Clock-gating mechanisms are implemented as additional functionality to reduce the power consumption. In addition the clock tree that drives the clock consumes considerable power. The data-driven nature of asynchronous circuits allows them, as an inherent mechanism, to save power by being idle.

- Less electromagnetic interference (EMI) [49], [121]: Synchronization of all parts of a synchronous circuit to a global clock creates high electromagnetic emissions, i.e. high noise, at the frequency of the clock. As asynchronous circuits perform local handshaking, they produce control signal ticks at various times and therefore do not suffer from high electromagentic noise.

- Robustness [89], [111]: Synchronous circuits are clocked based on their worst-case conditions. Thus they need to take into account delays from variations in the fabrication

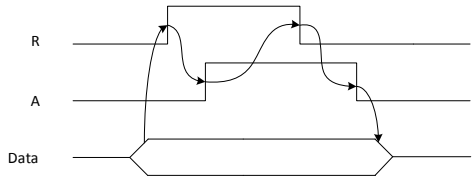


Figure 2.4.: Timing diagram showing a complete handshake cycle in a four-phase protocol.

process, in the supply voltage and in the temperature. As the asynchronous circuits are event-triggered, they adjust to these variations, in dynamically changing conditions.

- Composability and modularity [88], [107], [158], [153]: Asynchronous circuits implement local handshaking. This allows the control to be implemented locally. Stages operating with the same handshake protocol can be composed in a plug-and-play way. In addition, asynchronous design includes methods that are insensitive to delays, therefore, the asynchronous circuits built according to these methods can be easily composed without any timing considerations.

2.6.2. Challenges in Asynchronous Design

Asynchronous circuits have many potential advantages, but they also face challenges. One of the big challenges is the complex design process, which contrasts with the simplicity of the synchronous design process. The design, the implementation, the debugging, and the testing are not straightforward processes and require deeper knowledge and experience from the designer, and these are also required for an asynchronous design to take advantage of the listed benefits. Moreover, the design process is not supported by suitable EDA tools. The EDA tools that are widely used today are intended for a synchronous design process. Therefore, they are not able to accurately handle the steps in the design process of asynchronous circuits. There has been an effort to develop EDA tools for asynchronous circuits to automate or facilitate the design process, but the design flow has not been entirely automated and still requires some manual work from the designer to complete the process.

2.6.3. Handshake Protocols

There are several comunication protocols to synchronize the communication between blocks of computation. Asynchronous communication is based on handshake protocols. This section presents the most commonly used asynchronous handshake protocols.

Four-phase

The four-phase protocol requires two additional signals, besides the data, to implement the handshake, i.e. a request and an acknowledge signal. The four-phase protocol is a return-to-zero (RZ) protocol, as both signals have to be reset to zero between handshake cycles. For a complete handshake cycle four actions take place. Initially, all handshake signals are set to ‘0’.

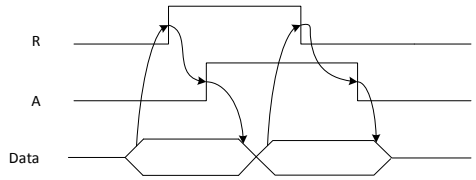


Figure 2.5.: Timing diagram showing two complete handshake cycles in a two-phase protocol.

(1) When the data are available, a rise transition of the request signal initiates the handshake. (2) A rise transition of the acknowledge signal indicates the reception of the request and the data. (3) The request signal is reset to '0'. (4) To complete the handshake the acknowledge signal is also reset to '0'. Different variations of the protocol allow the data to change as soon as the acknowledge is raised, or required them to stay stable until the falling of the acknowledge and the completion of the handshake. Figure 2.4 illustrates a timing diagram of a complete four-phase handshake cycle.

Two-phase

The two-phase protocol, like the four-phase protocol, requires two additional signals, i.e. the request and the acknowledge signal. The protocol is a non-return-to-zero (NRZ) protocol or transition-signaling protocol, as no reset operation is required and the actions are represented as transitions of the signals and not specific values. For a complete handshake in a two phase protocol two actions take place. Initially, all handshake signals are set to the same value. (1) When the data are available, a transition of the request signal, either positive or negative, initiates the handshake. (2) A transition of the acknowledge signal, in the same direction

of the request transition, completes the handshake. After this, the data are allowed to change. A timing diagram showing the signal transitions of two complete handshake cycles in the two-phase protocol is shown in Figure 2.5. The two-phase protocol uses two transitions to complete the handshake cycle instead of four the four-phase protocol uses. Therefore, it appears more promising in terms of handshake cycle period and power consumption.

2.6.4. Data encodings

There are several comunication protocols in asynchronous design. They combine a data encoding with a handshake protocol to synchronize the communication. This section presents the most commonly used data encodings.

Bundled Data

The bundled-data protocol considers bundle channels of data and control signals. The bundle channels consist of the normal data lines and two additional lines for the handshake protocol signals, i.e. the request and the acknowledge signal. The bundle channel implements a handshake protocol that can be either a four-phase or a two-phase protocol.

Encoding d.t	d.f	Values
0	0	reset
0	1	value ‘0’
1	0	value ‘1’
1	1	invalid

Table 2.1.: Dual-rail encoding with four-phase handshake protocol.

The bundled-data protocol relies on a timing assumption. The assumption is that the data should be ready when the request signal rises and should be kept stable until the acknowledge signal completes the handshake. This assumption is illustrated in the timing diagrams in Figures 2.4 and 2.5. In a pipelined design this means that the combinational logic in a stage should complete the evaluation of the data for the next stage before the request arrives at the receiver. To satisfy this timing assumption delay elements in the request line match the delay of the combinational logic stage.

Multi-Rail / Delay Insensitive

The multi-rail or one-of-N protocol considers the data in an N-bit encoding and an acknowledge signal. Each bit of data is encoded in N bits such that the completion of the computation and the validity of data is encoded within the data lines. In other words the request signal is encoded in the data lines. Since the validity of data is encoded within the data, no special timing assumption is required. For this reason, these protocols are called *delay insensitive (DI)*, as delays in the circuit do not affect the operation of protocol [167]. However, the encoding of data in multiple rails introduces more area and power consumption.

The multi-rail encodings, similar to the bundled-data channels, may also appear with a four-phase or a two-phase handshake protocol. The simplest encoding is the 1-of-2 encoding, or dual-rail. *Dual-rail with four-phase* and *dual-rail with two-phase* handshake protocols are the most commonly used. The latter is also refered to as *level encoded dual rail protocol (LEDR)* as it is based on signal level transistions. The two protocols are presented in the following.

Dual rail with four-phase handshake protocol: In dual-rail encoding, or 1-of-2, a bit of data d is represented with two wires/rails. One rail, $d.t$, holds the true value of the signal and the other rail, $d.f$, holds the false value. With two rails there exist the combinations appearing in Table 2.1 with the corresponding meaning. Every handshake cycle includes a normal phase and a reset phase. During the normal phase the value of data is transmitted in its dual form in $d.t$ and $d.f$, and the acknowledge signal is raised. During the reset phase the reset value (see Table 2.1) is transmitted over the two rails followed by a fall transition in the acknowledge signal. A request for a new handshake is encoded in the data, as one of the two rails switches value following the reset phase.

Level Encoded Dual Rail protocol (LEDR): In level-encoded dual rail, a bit of data d is also represented with two wires/rails. One rail $d.t$ holds the true value of the data and the other, $d.p$ holds the parity value. In level encoded dual-rail there is no reset phase. Instead of a true and a reset phase, an odd and an even phase alternate. In both phases the true rail, $d.t$, holds the true value of the data. The parity rail, $d.p$ holds the parity value in the odd phase and the true value

Encoding		**Values**
d.t	d.p	
0	0	value ‘0’ (even)
0	1	value ‘0’ (odd)
1	0	value ‘1’ (odd)
1	1	value ‘1’ (even)

Table 2.2.: Level encoded dual-rail (LEDR) encoding protocol.

Inputs		**Output**
A	B	Y
0	0	0
0	1	previous state
1	0	previous state
1	1	1

Table 2.3.: Truth table for C-element.

in the even phase. The data information is encoded as shown in Table 2.2. A request for a new handshake is encoded in the data, as either the true or the parity rail switches value between phases.

Level encoded transition signaling (LETS) with more rails are also possible, with 1-to-4 LETS encoding also being used and explored [90].

2.6.5. Muller C-element

A component that is widely used in asynchronous circuits is the Muller C-element or just C-element [108]. Such circuits often rely on the detection of transitions in a set of signals and synchronization on their transitions. The C-element holds a state and it maintains its state until all input signals change to the opposite state, whereupon the output changes to match the inputs. The truth table of the C-element appears in Table 2.3.

C-elements are either implemented with standard CMOS gates or are custom made. The implementation of C-elements has been studied and comparisons between different implementations appear in the literature [144], [46], [13], [104]. In this work we follow an implementation of the C-elements with standard CMOS gates, as a simple SR latch.

2.6.6. Timed-Graph Models

Asynchronous circuits are event-driven systems that operate on the principle of concurrent occurrences of events that are related in a cause and effect relation. In other words, the occurrence of an event triggers the occurrence of another event. To specify this behavior, directed graphs models are commonly used. *PetriNets (PNs)* are a graph tool to describe asynchronous and concurrent systems [109]. They are widely used and have many subclasses, include simplifications, interpretations, and variations to describe accurately the behavior of specific systems. Yet they form a basis for graph representations of many systems’ behavior. In this work we

focus on a subclass of PetriNets, the *Marked Graphs (MG)*, and in particular, an extension of them to include timing specification, the *Timed-PetriNets* [31]. The specific models we use are an extended version of the Signal Transition Graph (STG) model [32], and the Full Buffer Channel Net (FBCN) model [14]. This section provides specifications for PetriNets and the subclasses used in this work and gives a background for understanding the models.

PNs are bipartite, directed and weighted graphs formally defined as follows [109]:

Definition 2.6.1. A PetriNet is a 5-tuple $PN = (P,T,F,W,m_0)$ where:

- $P = \{p_1, p_2, ..., p_m\}$ is a finite set of places,
- $T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions,
- $F \subset (P \times T) \cup (T \times P)$ is a set of arcs (flow relation),
- $W : F \rightarrow \{1,2,3,...\}$ is a weight function,
- $m_0 : P \rightarrow \{0,1,2,3,...\}$ is the initial marking,
- $P \cap T = \oslash$ *and* $P \cup T \neq \oslash$.

In a more intuitive way, the nodes of the graph are of two types, *places*, p, and *transitions*, t. The arcs connect places to transitions, (p,t), or transitions to places, (t,p). Arcs are annotated with integer *weights*, $w(p,t)$ and $w(t,p)$, respectively. A place can hold a number of integer tokens. A *marking* m (state) of the PetriNet is an assignment of tokens to places. PetriNets have an *initial marking* m_0. In accordance with the *flow functions* (arcs), a transition t is enabled when each of its input places p contains at least as many tokens, as the weight of the arc connecting them, $w(p,t)$. A transition fires by consuming $w(p,t)$ tokens from each of its input places (p,t) and producing $w(t,p)$ tokens on each of its output places (t,p). A way to interpret a PN is to consider transitions as events, their input places as preconditions and their output places as post-conditions.

Both STGs and FBCNs belong to a subclass of PN, named *Marked Graphs (MGs)*. In MGs for each place in the graph there exists exactly one input transition and one output transition [109]. According to the MG specification, and for simplicity, the input and output place of each transition can be ignored in the graph representation. Thus, the transitions are connected directly with an arc that represent their relation.

Timed-PetriNets are PNs extended with a function that assigns timing values or ranges of values to the transitions or places of the PN, introducing timing constraints on the semantics of transition firings [31], [20], [168]. *Timed-STGs* are a subclass of Timed-PetriNets. The specification of STG and FBCN models that are used in this work are presented in more detail in Chapter 9, where they are used to model Argo.

Performance analysis of concurrent systems modeled in PNs relies on the evaluation of the cycle time of a PN. A cycle time is the time of the critical cycle of the graph. A cycle c is defined as the sequence of places and transitions connected by arcs, where the beginning node is the same as the ending node. The cycle weight is the sum of delays of the all the arcs of the cycle $d(c)$ divided by the sum of the tokens placed along the places in the cycle $t(c)$, $d(c)/t(c)$. The critical cycle(s) are the ones with the highest weight and this weight is the cycle time of the system. In concurent systems the cycle time determines the performance of this system.

3
Related Work

This chapter reviews the state-of-the-art in real-time multi-processor systems, in real-time NOCs, and in timing organizations, as those are the main topics which this thesis explores. In particular, this chapter presents work in real-time multi-processor platforms, and research projects that focus on real-time systems. It presents the approaches for implementing real-time NOCs and reviews some representative NOC designs, analyzing their timing behavior. As one of the main challenges in the current state-of-the-art in NOCs is the global signal distribution and timing organization, this chapter presents proposed solutions for more flexible timing organizations in NOCs.

3.1. Real-time Platforms and Projects

Several research projects focus on the area of real-time architectures. Some of these are MERASA, parMERASA, JEOPARD, PREDATOR, ALL-TIMES and Scalopes/CoMPSOC.

The FP-7 project MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analyzability) project [164] aims at multi-core designs for hard real-time systems along with software support and the tools for timing analysis of multi-core systems. The MERASA platform is a multi-threaded architecture using TriCore processors, connected by a bus. Both hard real-time and non-hard real time threads run on the platform. The processors include dynamically partitioned caches and scratchpad memories. In addition, the project involved development of an analyzable real-time memory controller, and adaptation of the WCET analysis tools OTAWA [12] and RapiTime2 [159] for use in the project platform. The ParMERASA project [163] follows MERASA with parallelization of applications and software support. In contrast to MERASA it uses a NOC for communication instead of a bus.

In the FP-7 project JEOPARD (Java Environment for Parallel Real-time Development), a Java environment (architectures, software, tools) for development of real-time systems was

explored. JEOPARD focused mainly on the development of tools. However, it also involved exploration of the real-time Java processor JOP [141] with dedicated caches optimized for specific purposes, and of TDMA-based memory access arbitration and its WCET analysis [124].

The FP-7 project PREDATOR aims at reducing the uncertainty of real-time system behavior and minimizing the cost of this uncertainty. Within this perspective, its goals are to study systems behavior, to create awareness of time-predictability and its problems, and to provide disciplines for system design that improves predictability properties and performance analysis [162].

The FP-7 project ALL-TIMES [58] focuses on timing analysis for safety-critical embedded systems. The project aims at developing a framework of timing analysis tools and methods. The goals are to develop complete methodologies, enhance the interoperability of existing tools and develop new ones, such that it provides integrated tool-chains.

The Scalopes project aims at supporting the development and evolution of low-power, multi-core platforms. The CoMPSOC platform [53] and tool flow [56] were developed within the project. The CoMPSOC platform has a scalable tile-based architecture [101], [102] with composability and predictability properties. Communication in the CoMPSoC platform is based on a network-on-chip. The Æthereal [54], and the aelite [59] NOCs have been developed within the CoMPSOC platform. Aelite has been a source of inspiration for the Argo NOC.

3.2. Real-Time NOC Approaches

A NOC used in a real-time system has to be analyzable and time-predictable. This implies that it must be able to provide latency and bandwidth guarantees. However, a typical NOC consists of multiple shared resources, implementing a number of message transactions, making it difficult to analyze individual transactions. To provide guarantees for the individual transactions, end-to-end connections need to be defined, such that each communication of a connection can be isolated from the overall network traffic and analyzed individually. As proposed by Goossens et al. [55], there are two design approaches to end-to-end connections. These are (i) non-blocking routers with rate control, and (ii) circuit-switching. Circuit switching builds end-to-end connections either through physical circuit switching or virtual circuit switching. The alternative solution is to use non-blocking routers with rate control. The rate control can be applied in several ways, e.g. with priorities, arbitration schemes, or by using complex mathematical analysis methods to control the communication flow. This section reviews the solutions proposed in the literature, giving an overview of the state-of-the-art in real-time NOCs. A number of specific NOC designs representative of each approach are explained in more detail along with their timing behavior in Section 3.4.

Physical circuit switching is a straightforward way to offer real-time guarantees. NOCs implementing circuit-switching are the NOC used in the 4S-platform [174], SoCBUS [171] and the NOC based on spatial division multiplexing presented by Leroy et al. [81]. The 4S-platform NOC [174] forms circuits by statically connecting input to output ports in the routers and assigning a portion of the links, i.e. a number of wires, to a circuit. Leroy et al. [81] follow the same concept of assigning a subset of the links to circuits. The connections are defined at initialization and owned exclusively, potentially leading to low utilization. This

approach is efficient for small numbers of connections [81]. SoCBUS establishes the circuits through a dial-up mechanism. The connections are reconfigurable but the dial-up attempt is not guaranteed to succeed. Overall, circuit-switching is a straightforward way to offer real-time guarantees, yet it is not scalable or flexible, due either to wasting resources on one hand or to inability to setup a connection on the other. Argo implements virtual circuit switching through time division multiplexing (TDM). The end-to-end circuits are statically defined, and thus there is no possibility of failure at runtime. In addition, in Argo the connections share the resources more efficiently, in a time-multiplexed fashion.

Virtual circuit-switching is another way to implement end-to-end connections. TDM involves multiplexing the use of the shared resources over time. Time is organized in fixed duration time-slots and the end-to-end connections are defined by a repeating schedule. Examples of NOCs using the TDM scheme are the Æthereal [54], the aelite [59], the Nostrum [96], and the TTNoC [140], [120]. In these TDM NOCs, the connections are defined in a static schedule. Sharing of resources is based on the application and communication requirements and since the communication is statically defined, it is straightforward to provide guarantees. In addition, since the circuits are based on a static schedule, the routers do not need to implement arbitration and flow control, and do not need to provide buffering. The result is very simple router designs. Argo follows the same approach, based on TDM, but in contrast to alternative TDM NOCs, Argo uses an asynchronous implementation offering elastic timing properties.

The alternative approach to circuit switching in order to establish end-to-end connections is by using non-blocking routers with rate control [175]. The rate is controlled in a number of ways, based on an arbitration scheme, on priorities, or on more complex techniques for analysis of the traffic flow. MANGO NOC [25] and the Kalray NOC [39] follow this approach. MANGO uses round-robin arbitration at the routers. Kalray uses static paths for the connections. Kalray enforces at the source point a throughput rate over a time interval, by constraining the traffic flow of a connection when a throughput limit is reached. In these NOCs, connections share the links but require buffers in the routers. In addition, they require complex arbitration and flow control at the routers, resulting in a large crossbar implementation. The high requirements for buffering and the complex crossbar implementation result in a costly router design. A typical MANGO router is ten times larger than an aelite router [55, 150]. The Argo router avoids all arbitration and buffering, resulting in a very efficient router.

Approaching the problem from another angle, research has been done on methods for analyzing and evaluating the performance of a NOC, or for controlling the flow of communication. Queuing theory [112], [115] uses probability distributions, and performs statistical analysis to analyze the traffic load, and evaluate the performance of the NOC. Queuing theory was applied to evaluate the performance of the Spidergon NOC [100]. Traffic flows of end-to-end connections are used in schedulability analysis [177], [67] to evaluate throughput, and to control the flow. Scheduling techniques attempt to give guarantees by enforcing throughput limits or by assigning priorities to traffic flows [27]. Network calculus [80], [11], [85] is a complex mathematical tool to estimate arrival curves for traffic flows in order to derive performance estimates, and set rates for the traffic flows [130]. These techniques assume buffering capacities, and hardware components to implement the arbitration or the rate control.

3.3. Timing Organization in NOCs

The timing uncertainties and clock distribution challenges motivate systems with more flexible timing organization. The structural way of building multi-processor platforms connected by a NOC allows the systems to be naturally organized in a GALS style, by having components in confined areas of the chip operate within a clock domain. A NOC implements the communication between those clock domains that are placed in different areas of the chip, so it must span the entire chip area, making the synchronization and timing organization within the NOC even more challenging.

The synchronization challenges of the NOC, even within a GALS design, trigger the exploration of NOC designs with flexible timing organization. A first step towards a more flexible timing organization is a *mesochronous* timing organization of the NOC [37]. A mesochronous NOC operates on one clock frequency, but its components (NI, routers, links) may exhibit a bounded phase difference. The typical way to achieve synchronization is to use synchronization elements between synchronous components. DSPIN [117] and aelite [55], [59] use bi-synchronous FIFOs [116], [169] on the links between routers. The bi-synchronous FIFOs are elastic buffers that bridge the clock phase difference between neighboring routers, and they are needed in every link. This incurs a significant area and power overhead.

Mesochronous synchronizers use FIFOs in their implementation. The bi-synchronous FIFO presented in [116] uses a five-stage FIFO and a token-ring. The full-custom FIFOs presented in [169] are implemented as asynchronous latch pipelines and improve the area overhead. The latter, used in aelite, is reported to halve the area of the bi-synchronous FIFO [59, Ch. 8]. Work in [84] presents FIFOs optimized specifically for mesochronous systems. Other approaches merge this FIFO into the buffers of the router [86] to minimize the overhead. However, the overhead cannot be entirely eliminated.

A more flexible design would be an entirely asynchronous one, consisting of asynchronous routers and links. CHAIN [10], MANGO [25], QNOC [43], and ANOC [16] explore asynchronous implementations. However, it is difficult to combine elastic timing and real-time guarantees. From the asynchronous NOCs, only MANGO offers real-time guarantees and therefore is suitable for real-time systems.

3.4. Real-Time NOC Designs

This section analyzes in more detail a number of real-time NOC designs that are representative of the proposed solutions existing in the literature as mentioned in Section 3.2. This section also analyzes their WCET following the analysis approach described in Section 2.5.

3.4.1. SoCBUS

SoCBUS [171, 139, 170] is a pure physical circuit-switching NOC. We present the key features below.

Key Features

SoCBUS is a packet-switching network. It uses the concept of a packet connected circuit (PCC) to dynamically setup physical end-to-end connections, i.e. circuits. After a circuit is setup it owns the resources of this circuit, and, thus it operates as a pure circuit-switching NOC. To establish a connection with the PCC concept, a packet traverses the switching network and locks the resources along the path. The circuits can be used when the entire circuit from end-to-end is locked and acknowledged. There are four phases for the setup procedure: (i) a packet traverses the network, as in a packet-switching network and temporarily locks the resources it traverses (routers, links). (ii) An acknowledge signal is sent back along the reserved for this connection path. If the attempted connection fails to be established, a negative acknowledge is sent back, freeing the temporarily locked resources. (iii) When the acknowledge is received at the source of the circuit, the reserved circuit can be used for data transfers. (iv) When a packet is received at the destination, it is acknowledged back. This acknowledge also cancels the circuit if it is no longer needed.

SoCBUS uses five-ported routers to implement a two-dimensional mesh. Each router is connected to a processing core through a wrapper. The wrappers have the role of the NI, i.e. to bridge the processing core interface with the packet-based interface of the routers. They implement clock-domain crossings and provide buffering for the packets to be transmited. The links connecting the routers are bi-directional and also include control information for mesochronous clocking and for acknowledgments. Mesochronous timing is used in the links to cover clock skew and delay uncertainty in gates and wires.

For the setup procedure, a packet is routed with dynamic arbitration through the packet-switching network, based on a minimum path algorithm. At every router, the packet is forwarded to one of the available routers according to the destination address. If there are no available routers, the circuit setup process fails and a negative acknowledge is sent back, releasing the reserved resources. The attempt is repeated again at a later point in time. A successful connection reserves a circuit that can be used in its full bandwidth.

SoCBUS in a later version uses the packet of the setup process to transmit short messages without reserving and tearing down connections.

WCET Properties

When a connection is established, it exclusively owns all the resources it has reserved (routers, links, buffers), and thus real-time guarantees are trivially achieved. These connections have no waiting time and a high througphput. On the other hand, short messages have the high cost of the setup and tearing down of the connection compared to the latency of the packet transmission. This makes SoCBUS suitable for data streaming applications but not for single packet transactions. Since connections are exclusively reserved, they are not efficiently shared among connections, potentially resulting in low utilization.

It is also possible that a required connection fails to be setup due to lack of resources. Unsuccessful connection attempts lead to overloading the network with unused traffic and wasting bandwidth. This dynamic behavior may cause changes in the execution time of tasks in an unpredictable way, compromising the real-time properties. Moreover, since the success of a

connection setup procedure is not guaranteed, SoCBUS is unsuitable for hard real-time platforms.

Argo uses the available resources in a time-multiplexed way. Resources are reserved for a circuit on a time-slot basis and do not stay locked for a long period of time. In this way Argo achieves better utilization of the resources. Additionally, there is no possibility of failure of a connection attempt. Resource allocation is evaluated before runtime, and thus it is known whether the resources are sufficient to meet the requirements.

3.4.2. 4S project Network

The NOC developed by the University of Twente for the 4S project (Smart ChipS for Smart Surroundings) [174], [173] is a pure physical circuit-switching network. We present the key features below.

Key Features

The 4S project NOC is a reconfigurable circuit-switching NOC that offers GS. The connections are formed by statically connecting inputs to outputs in the routers. An end-to-end circuit owns a portion of the link lines exclusively. The connections are predefined and configured at startup. Reconfiguration requires a restart of the network. The concept this NOC is based on is the allocation of portions of the links. The links are organized in fixed-width portions (lanes). The number and the width of the lanes in a link is fixed for the network and is decided at design time. An end-to-end connection owns a number of lanes and therefore a portion of the bandwidth.

The router of this NOC consist of a crossbar, configuration tables, and a crossbar configuration unit. The crossbar switches input lanes to ouput lanes, based on the information stored in the configuration tables. The router consists of one pipeline stage to register the outputs of the router after the crossbar switch. The configuration unit configures the tables with the predefined routing information during the startup. A data-converter unit has the role of the NI and connects the router to a processing core. The data-converter translates the processor messages to lane-width data untis (phits), performs packetization at the sender end, and reassembly of messages at the receiver end, generating and decoding the corresponding packet headers, respectively. Additionally, it provides the required buffering for the end-to-end connections.

Data is organized in lane-width phits at the data-converter. Each router consists of one pipeline stage and the links are wires without pipelining. Thus for a packet to traverse a hop, it requires as many clocks as the lane-width phits which it consists of.

A window counter mechanism is used to implement end-to-end flow control and prevent buffer overflow at the end-points. Each end-point maintains a counter that reflects the buffering capabilities of the other end. An acknowledge signal indicates the reception of the data and updates the information in the counters.

A second network is proposed to handle the BE traffic of the system and for configuring the circuit-swicthing GS network [172].

WCET Properties

Circuit-switching networks allocate resources to circuits to provide guaranteed services. In the 4S-platform NOC, the resources, i.e. lanes, are exclusively owned by circuits. With this method, there is no waiting time to access the network, and it provides high throuhput to the end-to-end connections. The latency and throughput depend on the width of the lanes, a design feature of the NOC. Moreover, it supports a limited number of connections. The circuits do not share the resources efficiently, leading to low utilization and low flexibility.

Argo achieves a better utilization of resources, as the resources are not exclusively reserved for specific circuits. Argo reserves the links in a time-multiplexed fashion providing more efficient sharing. The links are dedicated to a circuit for the duration of a time-slot.

3.4.3. Nostrum

Nostrum, [96], [97], is a virtual-circuit switching NOC. It implements virtual circuits through the TDM mechanism. We present the key features below.

Key Features

Nostrum is a packet-switching network that supports both GS and BE traffic. It implements virtual circuit switching in a time-multiplexed fashion, using the concepts of temporally disjointed networks (TDN), and looped containers (LC). TDNs are used to isolate traffic of different flows and the LCs to facilitate the TDM mechanism. For BE traffic, routing decisions are made dynamically in the routers. For GS, virtual circuits are predefined and information is stored in lookup tables in the routers. Deflection routing ensures that the delay to traverse a router is fixed, and packets are not stalling in the routers.

Nostrum architecture is a mesh of five-ported routers, known as switches. A switch is connected to four other switches in the four directions, and with a processing element (known as a resource). The interface of the switch to the resource is implemented with two components: the resource network interface (RNI), and the network interface (NI). The RNI has the role of the front-end of the typical NI, translating the resource communication protocol to the NOC protocol. The Nostrum NI implements the back end functionality of the typical NI, offering a number of communication services which are then utilized by the RNI.

The use of TDNs ensures isolation between traffic of different types (GS and BE), as well as between different virtual circuits. TDNs can be thought of in terms of different coloring of neighboring routers or buffering/pipeline stages: Two neighboring routers or stages have different colors, and packets are routed from a router or stage of one color to another of a different color. In this way different colored partitions are created. The number of partitions or TDNs depends on the topology of the network and the buffering stages of the network. In every time-slot, packets travel from TDNs of one color to TDNs of a different color. In this way, packets that reside in TDNs of different colors during a specific time-slot can never interfere.

The use of LCs is a way of providing bandwidth to a virtual circuit. A virtual circuit is implemented as a looped path including the source and destination point of the virtual circuit.

LCs assigned to a virtual circuit travel along this looped path, potentially carrying data. The number of LCs assigned to a virtual circuit reflect the bandwidth of this circuit.

The number of TDNs is a design feature of the network. The virtual circuits are predefined, static, and configured into the network at startup. The LCs in a virtual circuit can be reconfigured at runtime and adjusted according to the requirements. The maximum badwidth is determined by the number of TDNs.

The combination of TDNs and LCs impelement time-division-multiplexing. LCs are equivalent to time-slots in a TDM network. A looped path can accommodate multiple virtual circuits by time-interleaving their LCs. The virtual circuits are still independent and provided with guarantees, similar to using different time-slots in a TDM schedule. The maximum number of TDNs in a specific network determines the maximum number of virtual circuits that can share a link, and consequently the number of LCs available for reservation. Thus, TDNs can be seen as the repeated period of time-slots in a TDM network.

A specific mechanism for end-to-end flow control is not considered in Nostrum. As traffic is predefined and operated within the time multiplexing mechanism, the end-points are assumed able to handle the traffic. However, sufficient buffering should be considered at the endpoints, such that no overflowing of data occurs. The evaluation of the buffering space requires some analysis and results in a corresponding cost.

WCET Properties

In Nostrum, resources are shared among predefined virtual circuits and BE traffic. For each virtual circuit, bandwidth is guaranteed. The number of virtual circuits that exist in a network is a static design decision, depending on topology and other architectural characteristics. Bandwidth guarantees can be adjusted at run-time by issuing and removing LCs in a virtual circuit. The round-trip time in a virtual circuit is a multiple of the maximum number of TDNs, going through this virtual circuit. The size of a packet is assumed to be one flit.

The latency of a single packet is proportional to the number of hops in the path it traverses. For a packet to be transmitted there is also some initial waiting time until an LC for this virtual circtuit arrives at the source point. This waiting time is a fraction of the TDM period. The throughput is inversely proportional to the TDM period. The bandwidth guarantees offered are a fraction of the total bandwidth. The timing is affected by the topology properties of the entire network, and the overall traffic is taken into consideration in the timing behavior. This leads to a predefined and predictable timing behavior well-suited for real-time systems.

Argo also implements virtual circuits following the TDM approach. Argo defines the virtual circuits as end-to-end connections, while Nostrum defines looped paths that contain the endpoints of the connection. This approach considers forward and reverse path for each circuit, potentially wasting the bandwidth of the reverse path if the communication is one-way. In addition, Argo supports only GS, maintaining a very simple router design. Nostrum also support BE traffic with the design complexity it implies, since arbitration, buffering, and flow control are required.

3.4.4. Æthereal/Aelite

Æthereal and aelite [54], [55], [60] and [59], are also based on the virtual circuit-switching approach through TDM. We present the key features below.

Key Features

Æthereal and Aelite belong in the same family of NOCs. They follow the virtual circuit switching approach based on TDM. Æthereal supports both GS and BE. Aelite supports only GS and has very lightweight routers.

Both Æthereal and Aelite are based on TDM, so time is divided into fixed-duration timeslots. Within a timeslot a flit is forwarded from one hop to the next. A hop can be a router or a link if the links provide some additional functionality. End-to-end virtual circuits are defined, and routing and time scheduling are evaluated before runtime. The static TDM schedule evaluated before runtime is configured into the NOC at startup. During the operation of the NOC the TDM schedule is repeated periodically.

End-to-end virtual circuits are defined as end-to-end connections through a number of hops. The TDM schedule defines the overall traffic in the NOC, such that in one time-slot the use of a resource is dedicated to one virtual circuit. Guarantees are given to virtual circuits by assigning them a number of timeslots within a TDM schedule period.

Since the traffic is predefined, no contention appears in the routers. In every time-slot a flit from one input port in the router is forwarded to an output of the router, without blocking or waiting. In this way, no arbitration and no buffering is required in the routers. The Æthereal router stores tables with routing information, as it is defined in the TDM schedule. Aelite uses source routing, eliminating the routing tables from the routers and resulting in a very lightweight router.

Æthereal additionally provides BE services. A separate component in the Æthereal router is responsible for BE traffic and the links are shared. The BE router part implements wormhole-routing with source-routing and link level flow control. The Aelite router compared to Æthereal is very lightweight since it avoids the overhead for the BE traffic. In addition, due to the source-routing, the routing tables are placed in the NIs instead of the routers.

At the endpoint of the NOC, NIs connect the routers with processing cores. The NIs implement protocol translation from the read/write transaction protocol of the processing cores to streams of packets. They also split the data in packets and flits at the sender's end and reassemble the messages at the receiver's end. The NIs are also responsible for buffering packets for the end-to-end virtual circuits and implementing end-to-end flow control. Both Æthereal and Aelite apply credit-based flow control, to prevent buffer overflow at the endpoints. The destination point sends credits to the source points indicating the buffering available at the receiving end.

TDM relies on the timeslot mechanism and the global synchronization of all parts of the network, as they need to follow a global schedule. To avoid the need for global synchronization, Aelite uses mesochronous links. They are implemented as simple FIFOs. They adjust clock skew but imply an additional hardware cost.

Æthereal and Aelite perform scheduling at the level of flits. In addition, the flit size matches the pipeline depth of the Aelite router and the timeslot duration which is three cycles. This re-

lates the hardware resources to the scheduling, simplifying the scheduling. On the other hand, this bounds the hardware resource granularity to three stages, such that for every additional functionality, e.g. link pipelining, a multiple of three should be added.

WCET Properties

Æthereal and Aelite are based on the concept of TDM, i.e. time organization in fixed-duration timeslots and repeated execution of time-slot schedules. This implies that a TDM schedule is repeated with a fixed period. For a virtual circuit, considering a period of time slots, a number of slots are assigned to the circuit, reflecting the reserved bandwidth for this circuit. The time properties for a message transaction depend on the TDM schedule as well as on design parameters of the NOC. These are the pipeline depth of a router, the duration of a time slot in cycles, and the number of flits comprising a packet. In both Æthereal and Aelite, the above parameters are chosen to be identical, with a value of 3. In this way the scheduling is simplified but the hardware resources are bound to the scheduling.

Æthereal and Aelite provide a latency of a single packet transmission proportional to the hops in the traversed path. A waiting time, which depends on the TDM period, has to be considered at the beginning of transmission. Throughput of a virtual circuit is inversely proportional to the TDM period and depends on the number of reserved time-slots for this circuit. A universal view of the network must be considered to evaluate the TDM schedule, and consequently affects all the timing parameters. On the other hand, the universal view and the well-defined behavior of the TDM schedule offers great predictability, making TDM networks highly suitable for real-time systems.

Argo is also a TDM NOC that follows the same principle as Æthereal and Aelite. It was inspired by Aelite and it is similar in its timing behavior, and its properties. In contrast to Aelite, Argo follows an asynchronous implementation of the routers and, thus providing a more flexible time organization while still offering the same strong time-predictability.

3.4.5. MANGO

MANGO [24], [25], [26], [23], is an asynchronous NOC, using non-blocking routers with rate control. We present the key features below.

Key Features

MANGO implements end-to-end virtual circuits using routers with dynamic arbitration, enforcing rate control. It supports both GS and BE traffic. A MANGO router consists of a BE router and a GS router, and the two types of traffic share the links in the NOC. The sharing of the links is done by implementing multiple Virtual Channels (VCs) over a link. Another feature of MANGO is that it is implemented entirely as an asynchronous circuit.

To provide GS, MANGO supports end-to-end virtual circuits. Multiple virtual circuits share the links of the network. Every link is divided locally into a number of VCs. For every VC, a separate physical buffer exists in the router output port that leads to this link. Every end-to-end virtual circuit using the link is assigned to a VC, and consequently to the corresponding buffer for this VC. A crossbar switch forwards the packets from the input ports to the corresponding

output ports, and specifically to the corresponding VC buffer. This implies a high complexity in the crossbar and high buffering requirements, and results in a large router implementation. For example, in a 2D-mesh MANGO NOC with 8 VCs sharing each link, and five-ported routers, the crossbar switch has five input ports and 5x8 = 40 output ports, and each output port has its own VC buffer.

The MANGO router consist of a BE router and a GS component which are implemented as separate components. A number of VCs are assigned to end-to-end virtual circuits and a number to BE traffic. The crossbar performs input-to-output switching statically. The static connections are setup at initialization. In this way, a virtual circuit is a sequence of VC buffers connected by crossbar switches. A credit-based flow control scheme on the links prevent the VC buffers from overflowing.

Arbitration modules at the output ports of the routers perform admission control and determine the flow rate on the link. The arbitration scheme is applied locally and determines the bandwidth and latency of the end-to-end virtual circuits. A fair-share scheme or a priority-based scheduling scheme (ALG, [25]) are used in MANGO. The first provides equal guarantees to all VCs sharing the link. The second grants link access first to the VCs with higher priority.

The MANGO network is implemented as an asynchronous design. The routers use bundled-data and the links are delay insensitive, implemented as two-phase dual-rail pipelines.

Network adapters (NA) connect the routers with the processing cores. They bridge the processing cores interface, implementing a OCP protocol, with the packet-based interface of the router. The MANGO NA also bridges the asynchronous domain of the network or routers and links with the clock-domain of the processing core.

WCET Properties

MANGO follows an entirely asynchronous implementation. It supports a GALS architecture, resolving the problems of timing synchronization and clock distribution throughout the chip, while providing real-time guarantees. MANGO applies flow control locally within the routers, as it implements fine-grained arbitration and rate control in the routers. Therefore, it does not require end-to-end flow control. Due to the fine-grained arbitration the routers in MANGO become more costly. They have increased area since they support separate buffering for each VC. The timing properties depend on the number of VCs that share a link and the link arbitration scheme.

The timing parameters in MANGO depend on local factors. The arbitration is done locally in the routers and arbitration decisions are based on the local traffic. The latency for the transmission of a single packet through MANGO depends on the number of VCs that share this link. Throughput is inversely proportional to the VCs using the link. In contrast to TDM, there is no waiting time in the beginning of transmission, as traffic is controlled locally in the routers. This provides high flexibility and adjustment to traffic needs. Hoever, this flexibility has a high cost in area, as each VC is buffered separately in every router, and the switching is done among all the VC buffers.

Argo follows the TDM approach, i.e. relies on a global schedule to control the traffic. As opposed to local traffic control in MANGO, Argo follows a global traffic control. The global schedule in Argo allows for very simple router designs, as opposed to the high area of MANGO

routers. On the other hand, they both follow an asynchronous implementation of the routers, to deal with clock distribution issues and provide a flexible time-organization.

3.4.6. Kalray NOC

Kalray [40], [39], [61] applies rate control using flow regulation [85] to provide guarantees to connections. We present the key features below.

Key Features

The Kalray architecture contains two separate NOCs that provide different services and accomodate different types of traffic. One NOC is dedicated for data (D-NoC) and the other for control (C-NoC). The D-NoC offers GS, using rate control. The C-NoC handles short control messages without GS. Each of the two NOCs is a 2D mesh, connecting 16 clusters of cores and 16 IO subsystem nodes, i.e. consists of 32 NOC nodes. A NOC node consists of a router that is connected to a cluster of cores or an IO subsystem node through an Rx and an Tx interface. The Rx and Tx interfaces have the role of the NI. The Rx interface provides receiver services and the Tx provides trasmitter services. The routers are connected with bi-directional links.

The Kalray D-NoC provides GS by regulating the traffic flow in the network. Source-routing is used and the flow regulation is applied at the source node. The regulation is based on network calculus. The network calculus considers a number of constraints on the link bandwidth and the buffer capacities of the routers and evaluates a limit over a time window for each connection. For each connection, a regulator, the packet shaper, at the source node regulates the traffic for this connection. For every packet to be transmitted, the corresponding regulator evaluates whether the data flow in this connection has exceeded its limit within a time frame. If not, it injects the packet into the network. In this way the regulator enforces the connection limit on a packet basis over a time interval.

The routers in the Kalray D-NoC are five-ported and are connected in a 2D mesh topology. A router consists of a number of buffers placed at the output ports of the router. Each output port has one buffer per originating input port. The routers multiplex the incoming packets to the destination defined in their packet header. In the output ports a round-robin arbitration scheme is used to select which buffer uses the link.

The end-to-end connections are evaluated for all flows through a routing phase. Then constraints for each connection are defined. The constraints follow from application requirements, available link bandwidth and available buffer capacity in the routers. Using these constraints a worst-case delay for every flow can be evaluated.

The Kalray D-NoC follows a source routing scheme avoiding complex dynamic arbitration in the routers. The rate control is applied at the source eliminating this complexity from the routers. However, buffering is still required in the routers, and is a limiting factor for the flow rate and the services provided.

WCET Properties

The Kalray D-NoC is dedicated to providing GS. Therefore it is not required to support BE traffic. Since no arbitration for BE traffic is needed, the routers are expected to be less complex.

However, the virtual circuits require buffers in the outputs of the routers. The number of buffers depends on the number of input ports and not on the number of virtual circuits using this output port and link. The depth of the buffers is a design parameter considered as a constraint in the traffic flow analysis to determine the bandwidth for a connection.

The timing parameters of the Kalray NOC are based on the network calculus analysis of traffic flows. Taking into acount the link bandwidth and the buffer capacity of routers, the analysis evaluates a guaranteed bandwidth for this connection over a time window. This is controlled by setting a rate limit at the source. Based on this scheme, the initial waiting time depends on the rate limit and whether the rate limit is reached for the current time window. The thoughput depends on the rate limit over the time window. The latency of a packet transmission depends on the number of hops and some latency in the routers. The latency of a packet in the router amounts to the depth of the buffers in the worst-case.

All timing parameters of Kalray NOC are affected by the rate limit applied over a time window. The time window concept is similar to the TDM schedule period used in Argo. The rate limit in the time window resembles the number of time-slots in a TDM period assigned to a connection in Argo. They both consider the overall traffic to determine the bandwidth limits and the guarantees for a connection. However, since all traffic in Argo is precisely scheduled in all the routers, the packets do not wait within the network, and therefore there is no need for buffering in the routers.

3.4.7. Summary

This section presented a number of representative real-time NOC designs andtheir WCET properties, and compared them with Argo.

The SoCBUS and the 4S-platform NOC are representative physical circuit-switching NOCs. SoCBUS initializes circuits in a dynamic way during runtime, and if the initialization is successful, the circuit owns the resources. This implies the possibility of failure, contrary to the concept of GS. The 4S-platform NOC defines the circuits, and allocates the resources statically. They both offer high throughput and low latency to the circuits, but the circuits do not share the resources efficiently. Argo offers low latency and fixed throughput. In Argo the connections share the resources efficiently, in a time-multiplexing way. Moreover, the circuits cannot fail, as opposed to SoCBUS, as they are statically defined in the TDM schedule.

Nostrum, Æthereal and aelite all follow the TDM approach, like Argo. They offer low latency and fixed throughput to statically defined virtual circuits. They all share resources efficiently. Nostrum offers adjustment of throughput guarantees during runtime, but potentially wastes bandwidth by using circuits in looped paths, instead of source-to-end. Argo, in contrast to these TDM NOCs, uses an asynchronous implementation, offering high timing flexibility, high robustness, and considerable area gains.

The MANGO and Kalray NOCs implement non-blocking routers with rate control. They enforce a rate by regulating the traffic flow and Kalray uses network calculus to evaluate precise throughput guarantees. They both require buffering and arbitration in the routers resulting in high area cost compared to Argo. MANGO is implemented asynchronously providing robustness. The Kalray NOC operates on time windows and still faces the clock distribution and synchronization issue.

Compared to all the above NOCs, Argo offers fixed throughput guarantees with low latency. It is a robust design, that tolerates clock distribution issues. It achieves the above in a very small implementation.

3.5. Asynchronous Design and Analysis

Asynchronous pipeline circuits, similar to synchronous, are organized in pipeline stages. As described in Section 2.6, there are several ways to implement an asynchronous communication protocol. Nowick et al. [114] present an overview and comparison of high performance asynchronous pipelines. These include micropipelines [158], the Mousetrap pipeline [148], and the GasP pipeline [157]. From the comparison of the above pipelines, Nowick et al. [114] conclude that the GasP pipeline achieves the highest performance, but requires high design effort, due to its complex circuit structure that does not comply with standard EDA tools. The second best performance is offered by the Mousetrap pipeline. In addition, the Mousetrap pipeline requires minimum design effort as it uses standard cells. The Argo router uses the Mousetrap pipeline.

One of the challenges in asynchronous design, as mentioned in Section 2.6, is the lack of EDA tools to support the design process. This lack is dealt with by using standard EDA tools developed for synchronous design, and working around certain limitations. Alternatively, dedicated EDA tools for asynchronous design have been developed. Petrify [34] performs synthesis, optimization, and analysis of the control part of the asynchronous circuit, starting from a PN specification. High-level languages have also been introduced for high level description of asynchronous circuits, e.g. Tangram [76] and Balsa [45]. However, the tools and methods developed for asynchronous design address issues in certain steps of the design process, and do not yet provide an entirelly automated flow. For this reason, we implemented Argo using standard EDA tools.

For the performace analysis, models and algorithms for concurrent systems are used. Petri Nets (PNs) [109] are the most widely used model, with many subclasses to support various specifications. STGs [32] are a subclass of PN that is used in this work. Full Buffer Channel Net (FBCN) [14], [15], is another subclass of PNs used to describe asynchronous circuits. Many analysis techniques rely on PNs to analyze the performance of asynchronous designs [109], [31]. Performance analysis is based on the evaluation of the cycle time of a PN. The simplest way to evaluate the cycle time is to enumerate all cycles of the PN. Ramamoorthy et al. [132], Magott [87], and Bereel et al. [14] and [15] present algorithms for the performance analysis of timed MGs that rely on the cycle time metric. The cycle time is straightforward to evaluate and is a direct metric for the average-case performance of the system. However, the cycle time is not a measure for the worst-case performance. Maximum timing separation between events (TSE) is a a way to evaluate worst-case performance of a concurrent system [65], [91]. Hulgaard et al. [65] proposed an algorithm evaluating worst-case bounds on the timing separation of events and this is used in this thesis. In this thesis, we used STGs and FBCN models to describe Argo and TSE to perform worst-case analysis, to provide a timing evaluation and verification of Argo.

Part II.

Design and Implementation

4

Argo NOC Architecture

This Chapter presents the architecture of the Argo NOC. Argo is intended to be used in the T-CREST platform to connect the Patmos processors and implement message-passing functionality among them. The requirements for the Argo NOC are to support real-time multi-processor platforms, and to support a flexible timing organization supporting the GALS system organization. This chapter describes the design choices made to meet these requirements and the underlying ideas that lead to an efficient NOC implementation.

The content of this chapter appears in the following publications:

- "An Area-efficient Network Interface for a TDM-based Network-on-Chip", Jens Sparsø, Evangelia Kasapaki, and Martin Schoeberl [152].
- "Router Designs for an Asynchronous Time-Division-Multiplexed Network-on-Chip", with authors Evangelia Kasapaki, Jens Sparsø, Rasmus Bo Sørensen, and Kees Goossens [74].
- "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation", with authors Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø [71].
- "D3.2 - Simulation model of the self-timed NOC", T-CREST Project Deliverable [125].

4.1. Fundamental Architecture Decisions

The Argo NOC is a packet-switching NOC that implements virtual circuit switching using time-division multiplexing (TDM). The TDM scheme, also used in a number of NOCs such as Æthereal [54], aelite [59], Nostrum [96], and TTNoC [140], [120], is the basis for providing real-time guarantees. Alternative approaches for real-time NOCs, as presented in Section 3.2,

include physical circuit switching [174, 171], rate-controlled routers with arbitration [25], [61], and analytical approaches to set priorities or flow rates to provide guarantees [177], [67], [130]. Rate-controlled routers with arbitration result in a complicated router design with high buffering requirements that inflate the area and degrade performance in the routers. Physical circuit switching wastes resources, as they are dedicated to a circuit whether it is used or not. Finally, analytical approaches provide bounds based on analysis of traffic flows and communication load. They also rely on a router design with dynamic arbitration and buffering, while requiring a complicated analysis process. Thus, we chose TDM scheme as the most straightforward and efficient way to provide time guarantees.

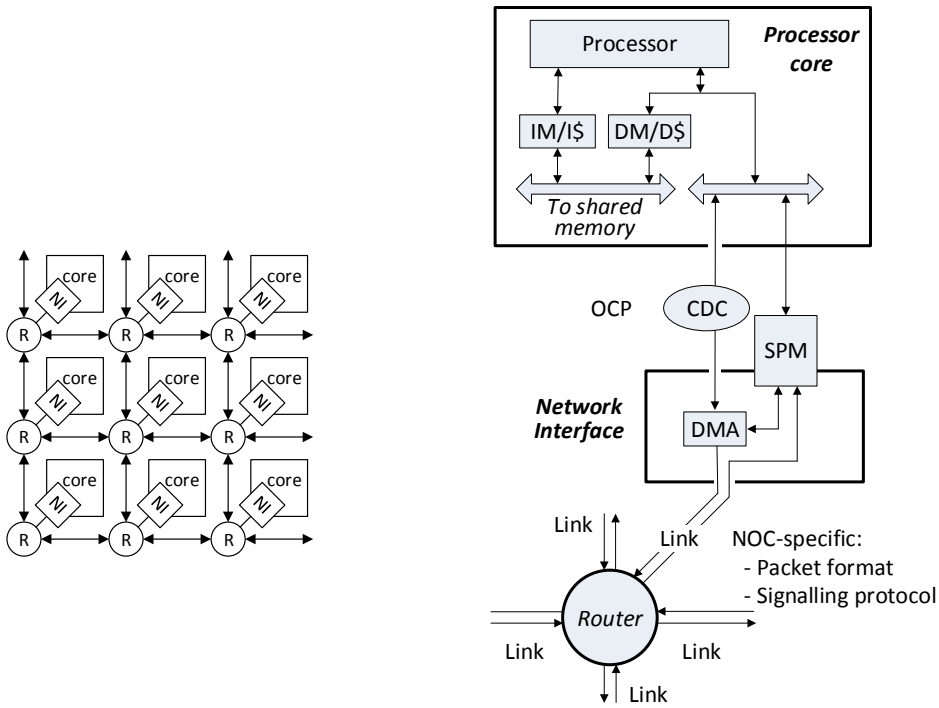
From a timing organization point of view, a network-on-chip spanning the entire chip area needs a flexible timing organization. A synchronous implementation of the NOC requires the clock to be distributed to the entire chip area. A mesochronous design requires mesochronous synchronizers added to the synchronous design, introducing a significant overhead [117], [55], [59], [169]. An asynchronous design is a better fit as it inherently offers timing flexibility. Thus, we chose an asynchronous design for the core network of routers, as a flexible and efficient design. However, mesochronous functionality is still maintained at the borders of the NOC, at the NIs, due to its straightforward relation to the TDM schedule. To support GALS organization, synchronization is required only with the processors at the endpoints of the NOC. Section 4.4 describes the timing organization of Argo and Chapter 8 provides a detailed analysis of its timing properties.

The main design principles of Argo are the statically scheduled TDM scheme and the use of an asynchronous network of routers. However, TDM is based on a global notion of time used to enforce the TDM schedule. Synchronization of all components of the NOC to a global clock is essential to maintaining the global TDM schedule. This contrasts with the flexibility of an asynchronous network, which inherently operates in a time-elastic way. Argo faces the challenge of combining the two opposing principles – the TDM global synchronicity and the asynchronous time elasticity – to offer hard real-time guarantees in an efficient way. The following section explains how Argo deals with the above requirements and challenges.

4.2. Overall Argo NOC Architecture

The Argo NOC is a packet-switching network that implements virtual circuit switching using TDM. It implements asynchronous message passing functionality among processing cores. Argo consists of a network of routers, that propagate a packet stream from one processor to another, and NIs that connect the processing cores to the network of routers. Figure 4.1(a) shows a block diagram of a multi-processor platform connected with the Argo NOC. The Argo routers have five ports and are connected in a bi-torus topology. The NIs of Argo form a bridge between the standard read/write transaction interface of the processors and the packet stream interface of the routers.

The Argo NOC implements asynchronous message passing as “push” operations, i.e. writes to remote cores. The message passing is implemented in the form of DMA-driven data transfers. As illustrated in Figure 4.1(b) the cores contain caches (instruction and data), and local explicitly managed scratchpad memories (SPMs). The local SPMs of the processing cores, in addition to their normal functionality as private memories, serve as communication memories,



(a) Multi-processor platform connected by a NOC

(b) The Argo NI

Figure 4.1.: Block diagrams of a multi-processor plarform and the Argo NI, from [71].

i.e. as source and target for the message passing. Processing cores set up the DMA controllers through the read/write transaction interface (OCP [3]) to initiate the transfer of a message. The DMAs control the transfer of messages from the private SPM of one processing core to the private SPM of another core. They organize the variable-sized messages into fixed-sized packets and forward them to the network of routers. The packets traverse the network of routers from end-point to end-point and are written into the remote SPM.

Argo uses source routing. In this form of routing, the choice of which route the packet is to follow is made at the source NI, and the route and destination address are attached to the packets. This implies that the TDM schedule and the route information for each virtual circuit is set in tables placed in the NIs. Routers do not store any routing information and therefore they are very lightweight.

The Argo NOC is based on two key ideas that lead to an efficient implementation. The first is the NI design with TDM-driven DMA controllers and the second is the use of asynchronous routers to provide time-flexibility. The two key ideas combined create a direct path from the local SPM to a remote SPM, without any arbitration, buffering, end-to-end flow control or synchronization (either mesochronous or crossing different clock-domains) along this path. The next two subsections present the two key ideas Argo is based on.

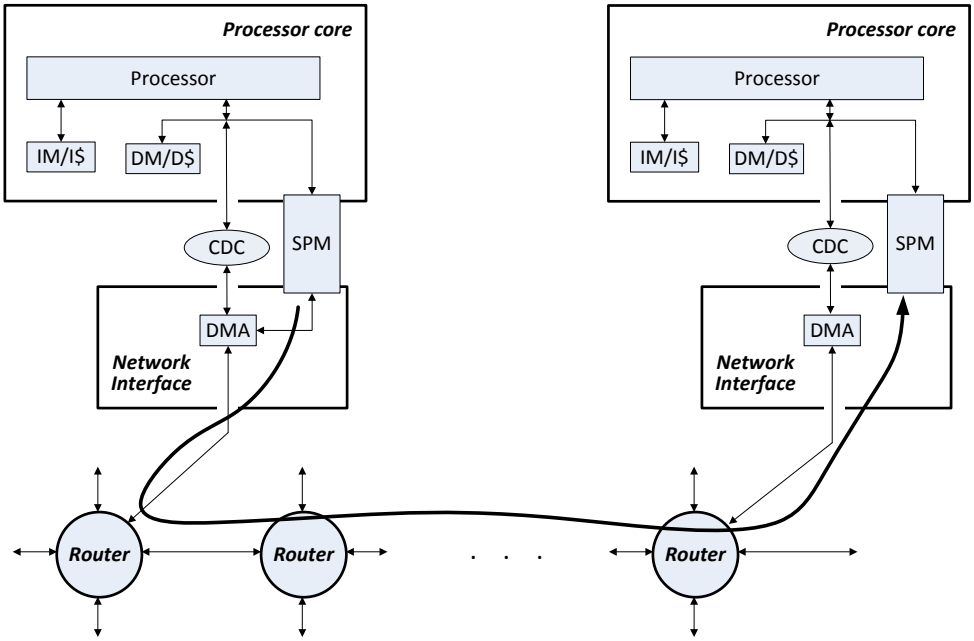


Figure 4.2.: Argo NI design and end-to-end communication datapath, from [71].

4.2.1. NIs with TDM-driven DMA Controllers

One of the core ideas of the Argo NOC is a novel NI design that integrates DMA controllers with the TDM schedule and places them in the NI. Along with the use of a dual-ported SPM for clock-domain crossing between the processor cores and the NIs, this eliminates the need for buffering, arbitration, end-to-end flow-control and clock-domain crossings for the data in the NIs.

Argo, like many typical multi-processor architectures, uses DMA controllers to implement background DMA-driven data transfers from core to core. The idea used in the Argo design is that since the DMA operation is closely related to the network, it seems natural for the DMA functionality to be part of the NI instead of the processor, as shown in Figure 4.2. Thereby, the DMA operation is integrated with the time-multiplexing of the TDM schedule and the functionality taking place in the NIs. Each NI contains a table with DMA controllers. Each DMA controller correspond to an entry in the DMA table and is bound to one outgoing channel. A channel is a one-way point-to-point connection from a source NI to a destination NI through routers with a defined path. The number of DMA controllers, i.e., outgoing channels, from an NI is configurable. During a time-slot of the TDM schedule, one DMA controller is active and injects a packet into the corresponding channel. This removes the need for arbitration of the various channels in the NIs in order to access the network. DMA controllers are assigned a number of time-slots of the TDM schedule, to satisfy the bandwidth requirements of the corresponding outgoing channel.

In addition, a dual-ported SPM implements the clock-domain crossing between the processor cores and the NIs. The SPM offers two interfaces with different clocks, one to the processor and one to the NI. In the Argo design, the dual-ported SPM is used to bridge the processor cores

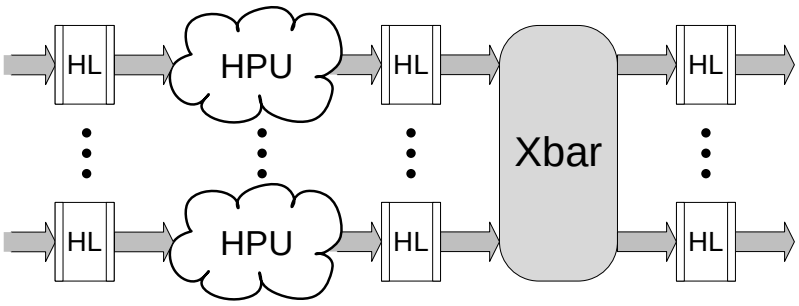


Figure 4.3.: The three-stage handshake latch router pipeline.

with the NIs for the data payload, as shown in Figure 4.2. In combination with the placement of DMA controllers in the NIs, it results in a very efficient scheme. The DMA controllers placed in the NIs can directly access the data placed in the local SPM of the processing core, eliminating the buffering requirements in the NIs. At the receiving end the data can be directly written to the SPM, eliminating the need for end-to-end flow-control. The only clock-domain crossing needed in the Argo design is in the processor and NI interface for the programming of the DMA controllers.

The above scheme entirely eliminates the need for arbitration and buffering in the NIs, for end-to-end flow control, and for clock-domain-crossing of the data payload, which are required in typical architectures. The NI design creates a direct path for the data from the source SPM to the network of routers and from the network of routers to the receiving SPM.

4.2.2. Asynchronous Network of Routers

In the network of routers, we exploit the idea that asynchronous pipelines inherently operate as FIFOs. Therefore, we combine the router functionality and the elastic synchronization in the asynchronous router pipeline.

In mesochronous networks, the way to handle clock distribution problems and skew among the synchronous routers is by using mesochronous synchronizers at the links between routers. The synchronizers are typically FIFO structures with a number of stages. As discussed in Section 3.3, they consist of five stages [116] in the typical case, a number that can be lowered to three [169, 84] in custom-made FIFOs. The synchronizers resolve skew issues up to a certain amount of skew, and need to be placed on every link, introducing additional latency and a considerable area overhead.

In asynchronous pipelines, the concept of synchronization is inherent and is implemented locally among neighboring pipeline stages with an asynchronous handshake protocol. Thus, asynchronous pipelines operate as self-synchronizing FIFOs [158]. In addition, the asynchronous pipeline implements the required router functionality in the same way as a synchronous pipeline. By implementing the routers as asynchronous pipelines we combine the router functionality with an elastic FIFO behavior and avoid the explicit synchronizers and their cost.

In an asynchronous network, the global synchronization required by the TDM scheme is implemented in a distributed way, rather than by enforcing synchronization of all components to a global clock. To explain this distributed synchronization, we look into the Argo router pipeline

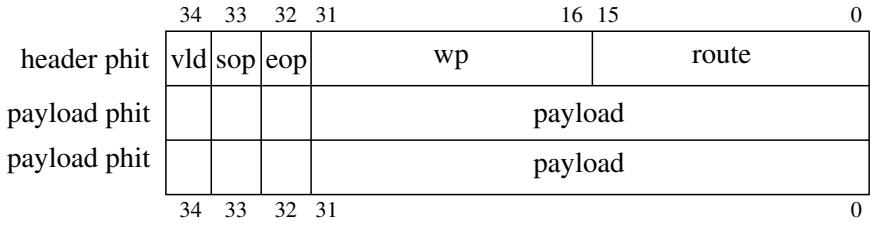


Figure 4.4.: The Argo packet format, from [71].

appearing in Figure 4.3. The Argo router was inspired by the Aelite router [59], which is also a TDM router and consists of the same pipeline stages. The Argo router is a three-stage pipeline using handshake latches in the pipeline stages instead of registers. The handshake latch consist of a normal enable latch and a latch controller implementing the asynchronous handshaking. The three stages of the pipeline are: (i) link traversal, (ii) header parsing unit (HPU), and (iii) crossbar (Xbar). Details of the design are presented in Chapter 6. The distributed synchronization is enforced by the crossbar which is implemented as a strongly indicating component [151]. As such, in every handshake cycle, it consumes five phits from the input pipelines (join functionality) and produces five phits in the output pipelines (fork functionality). In this way, it synchronizes all input pipelines, implementing join-fork (JF) functionality of the five router ports, mimicking the clock tick. This allows asynchronous pipelines to be time-elastic between JF points but synchronized at the JF point.

4.3. Packet Format

The data transmitted over every packet-switching network is organized in packets [36]. The length of the packet is specific for the individual network. A packet consists of one or more flits (flow-control digits). A flit is the basic flow-control unit, i.e. the smallest data unit that can be individually routed and on which flow control is applied. A flit consists of a number of phits (physical digits). A phit is the basic unit of the physical layer, i.e. the smallest data unit that is stored in one pipeline stage.

In Argo the size of a packet corresponds to one flit, and so the terms flit and packet coincide. The Argo packet/flit consists of three phits. Figure 4.4 shows the specific packet format of the Argo NOC. As Argo uses source-routing, the first phit is a header phit and the two following phits are payload phits. Each phit is a 32-bit data word and three control bits. The *vld* bit indicates whether the phit carries valid data (valid-phit) or not (void-phit), the *sop* bit indicates the start of the packet, and the *eop* bit indicates the end of the packet. Since fixed size packets are used, the explicit encoding of start and end of packet is not necessary; however it avoids counters in the routers and further simplifies the router design. The header phit contains the route, *route*, which the packet will follow, and the destination write pointer, *wp*. The route field is 16 bits wide. Two bits encode the output destination in every router, thus, the 16-bit route field encodes a path of at most 8 routers. The wp field is the write address for the data at the destination SPM.

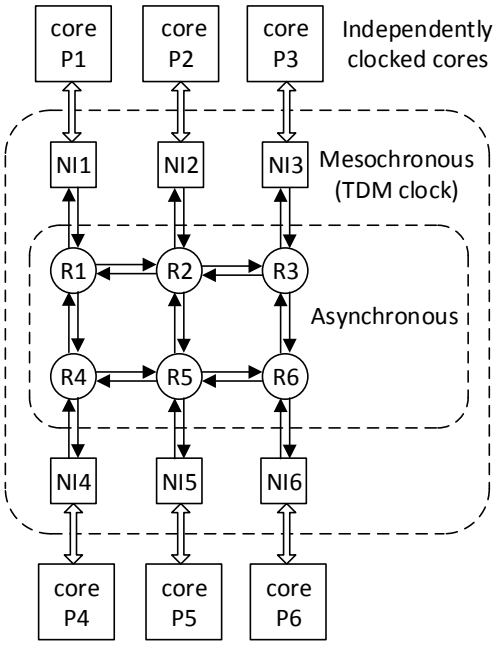


Figure 4.5.: Argo timing organization in a 2x3 multi-processor platform.

4.4. Timing Organization

The Argo NOC implements a novel timing organization, consisting of three different timing layers: (i) an asynchronous layer, (ii) a mesochronous layer, and (iii) an individually synchronous layer. The overall timing architecture of Argo is illustrated in Figure 4.5, which shows a 2x3 multi-processor platform, connected by an instance of Argo, and the different layers of timing organization. In the inner layer, six asynchronous routers form an elastic network. This network is wrapped in a layer of six mesochronous NIs. An asynchronous implementation of the NIs is also possible. However, the NI functional complexity favors a synchronous design. In addition, NIs enforce the TDM schedule and in a clocked design it is straightforward to control the frequency of the clock signal and thereby the TDM time-slotting. The outmost layer consists of six independently clocked processing cores. This organization is consistent with the GALS system organization.

The network of asynchronous routers behaves as a structure of elastic FIFOs, synchronizing in a distributed way as explained in Section 4.2.2. The global synchronization is maintained at the end-points of the time-elastic structure, i.e. at the NIs, by maintaining the data flow rate in addition to the distributed synchronization. The NIs operate mesochronously based on a common NI clock. TDM time-slotting is based on the NI clock. TDM counters, placed in the NIs and driven by the NI clock, drive the TDM schedule and enforce the NI clock rate in the time-elastic structure. In this way, Argo, from the point of view of the processor cores, is seen as a mesochronous network, able to absorb an amount of skew.

The skew present between NIs is a result of clock as well as reset signal distribution. The reset signal initializing the TDM counters may also arrive skewed at the endpoints. The effect

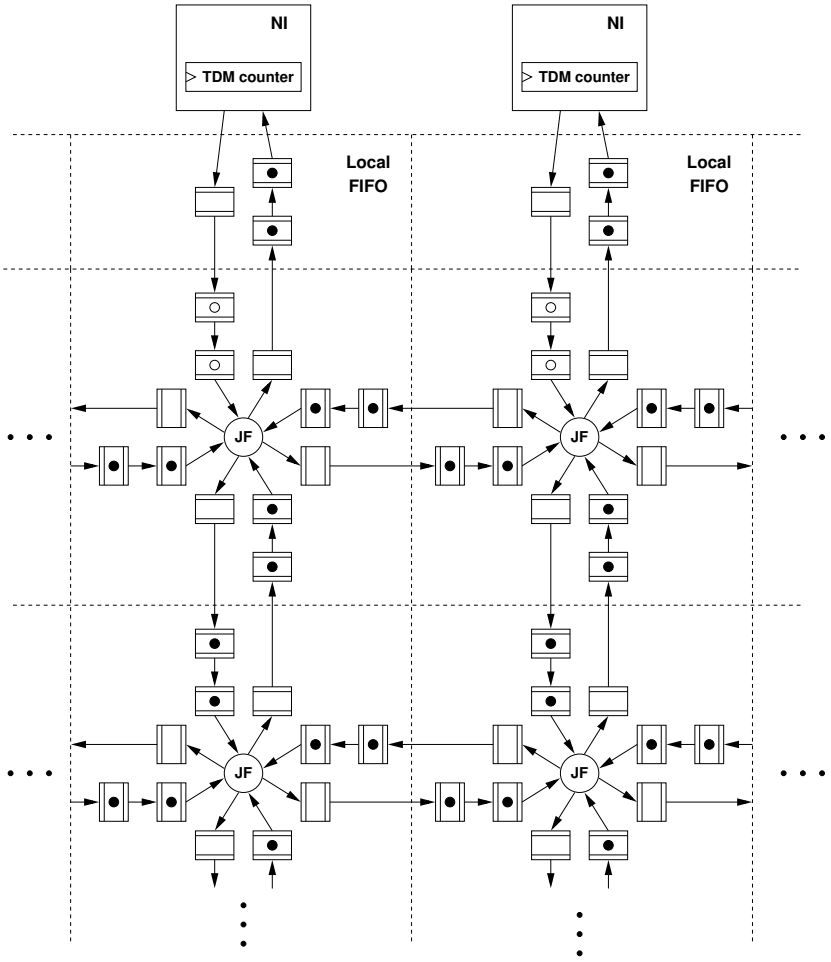


Figure 4.6.: A 2x2 Argo segment as a structure of asynchronous FIFOs, from [71].

of both, clock and reset skew, results in a phase difference in the TDM counters that can potentially be more than a clock cycle. This *TDM skew* should be absorbed by the elastic network of routers for correct operation of the NOC.

The amount of skew that can be absorbed depends on the design, the implementation technology and the operating frequency of the NI clock. The network of routers in the Argo NOC can be seen as a mesh of elastic FIFOs, connected by points of enforced synchronization (JF, NIs). An expanded view of Argo as a structure of FIFOs is shown in Figure 4.6. This representation omits the combinational logic of each stage for clarity and shows only the handshake latches and the JF points of synchronization. Between synchronization points, the asynchronous pipeline stages of the router (three-stages) offer a fixed amount of elasticity, determined by the design parameters of the pipeline. Additional elasticity can be provided by adding more pipeline stages between NIs and routers. In Argo we decided to place one additional FIFO stage in the channel from NI to router, and two FIFO stages in the channel from router to NI, the *Local FIFOs*, as shown in Figure 4.6. This decision aims at a balanced design with three pipeline stages between every synchronization point.

A design parameter that determines the timing and performance of asynchronous pipelines is the number of latch stages per token in the pipeline. A token in an asynchronous pipeline

is a data unit that can occupy one pipeline stage, i.e. a phit. The dynamic wavelength W_d of the pipeline is the number of latches per token for which maximum performance is achieved [151]. In Argo, based on implementation results, we evaluated the dynamic wavelength to be approximately 1.5, and therefore we initialize the three-stage pipeline with two tokens, i.e. void-phits, to its optimal state. The choice of three pipeline stages is further discussed in Section 6.2.2. This state is shown in Figure 4.6, where the data tokens are represented as dots in the handshake latches in the Argo structure.

During operation, global synchronization is maintained, since each NI injects one token and removes one token into/from the elastic network of routers at the clock rate of the NIs. The JF synchronization points handshake only when there are tokens in all five input pipelines, simultaneously producing five tokens in the output pipelines. This maintains the token flow in the asynchronous structure, keeping the number of tokens in the structure invariant.

To operate correctly, Argo relies on the timing assumption that the NI clock frequency is slower than the handshake capabilities of the asynchronous network of routers. On one hand this depends on the design and implementation technology, which are analyzed in Chapters 6, 5 and 7. On the other hand, the presence of skew alters the state of the asynchronous pipelines by filling or draining them. This brings them away from their optimal state, affecting their handshake capabilities. However, for a given amount of skew, there is a slower operating frequency for which this skew can be absorbed. This indicates a relationship between the operating frequency and the amount of skew that can be absorbed by the asynchronous network. This relationship is explored in Chapter 8.

4.5. Interfaces

In the Argo NOC there are three different layers of timing organization. Therefore, two types of interfaces separate the timing layers and one more operates between pairs of routers in the core network of asynchronous routers. The three interfaces have different requirements for timing and data organization.

Firstly, in the core asynchronous network, the routers are connected in a bi-torus topology. Synchronization among neighboring routers is achieved through asynchronous handshaking. The handshaking protocol is a two-phase bundled-data [151] (NRZ) protocol, as described in Sections 2.6.3 and 2.6.4. The router pipeline implements the protocol, and thus achieves an elastic timing synchronization among pipeline stages. Therefore no further synchronization mechanism, e.g. mesochronous synchronization, is required. As regards data organization, this interface considers streams of packets. The packets have the specific format for the Argo architecture shown in Figure 4.4.

Secondly, the mesochronously clocked NIs are connected to the asynchronous routers. The interface is shown in Figure 4.7. The connection of a clocked interface to a two-phase handshake protocol requires translation. In the channel going from the NI to the router, the clock signal is divided by two and is connected to the request signal of the asynchronous protocol. This generates a transition in the request line for every positive edge of the clock. The acknowledge signal from the router is ignored. Under the assumption that the NI clock period is larger than the handshake cycle of the router, it is safe to ignore the acknowledge signal. If a clock pulse initiates a handshake, the acknowledge signal completing the handshake will

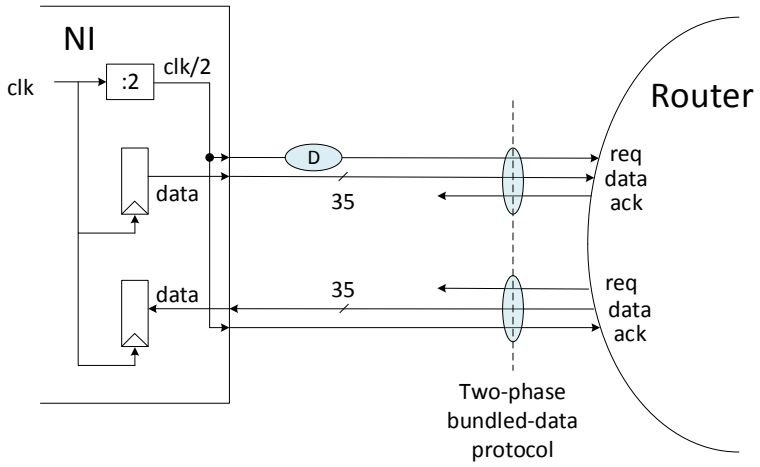


Figure 4.7.: Interface of clocked NIs to asynchronous routers.

arrive before the next clock pulse, thus no metastability will occur. A similar condition holds in the channel going from the router to the NI. The clock signal divided by two is connected to the acknowledge signal line of the asynchronous channel. This generates a transition in the acknowledge line for every clock pulse. The request signal from the router can safely be ignored.

Thirdly, in the outermost layer the independently clocked processing cores are connected to the mesochronously clocked NIs. The processing cores have the role of the master and generate read/write transactions. The cores interface with the NIs in two ways. On one hand, the local SPM implements the interface between the cores and the NIs. Both the cores and the NIs access the dual-ported SPM directly, as masters, through an OCP interface [3]. The dual-ported SPM supports two ports for two different clocks for read and write transactions for each port. Hence, it is used to implement the clock-domain crossing between the processing cores and the NI. Thus, the data can directly be accessed by the cores and the NIs, without additional synchronization cost. On the other hand, the cores access the DMA controllers to initiate a message transfer or get status information. For this interface a clock-domain crossing is required. This adds an additional delay, which is a fixed and bounded number of clock cycles, to the transaction. A clock-domain-crossing is explored in more detail in [62]. The interface for both purposes follows a subset of the OCP protocol. The complete OCP documentation can be found in [3]. The subset that is implemented in the Argo design is the OCPio, as described in Patmos Handbook [142], which supports single-word read or write transactions. The interfaces between the Argo NIs and the processing cores allow Argo to support a GALS system organization, as the cores are able to operate at their own frequency.

4.6. Initialization Process

Argo requires an initialization process. The initialization includes the resetting of the asynchronous structure to its initial state, and the configuration of the TDM schedule in the NIs.

The process is initiated by a global reset signal which resets the asynchronous pipelines and the TDM counters in the NIs.

The global reset signal sets the asynchronous router pipelines to contain two void-tokens, which is their optimal performance state, as explained in Section 4.4. These tokens are shown as black dots in Figure 4.6. In addition, the entire asynchronous structure needs to be initialized in a stable state, such that there is no handshaking until the NIs are synchronized to the same time-slot and start the operation of the TDM schedule. Therefore, the input local FIFOs, in the channels from NIs to routers, are initialized with no tokens. This initialization policy prevents the JF points at the Xbar from handshaking until the NIs start injecting tokens, and prevents the structure from entering a non-deterministic state due to reset skew.

When the global reset signal is de-asserted, the TDM counters in the NIs start free-running, i.e. without a TDM schedule yet. To bring the input local FIFOs to their optimal state as well, i.e. with two tokens in three latch stages, the NIs inject two void tokens without removing tokens from the output local FIFOs. This happens in all NIs after a small and fixed number of cycles. Figure 4.6 shows these tokens as white dots in the local FIFOs. After the injection of these two void tokens, the network enters the steady state of operation, where NIs inject one token and remove one token in every time-slot, maintaining the number of tokens in the network invariant.

At this point each processor configures the TDM schedule into its local NI. The configuration includes writing the assignment of time-slots to circuits and the route for each circuit. This is done in a fixed number of cycles for all NIs, and thus they all become synchronized in the same time-slot. The NOC is then ready for normal operation based on the TDM schedule.

4.7. TDM Scheduling

Argo implements virtual circuit switching based on the TDM scheme. As described in Section 3.2, time in TDM is divided into fixed-duration time slots. The duration of a time slot in Argo is three clock cycles, matching the number of phits in a packet. The overall traffic over the network of routers is based on a static, predefined schedule that establishes connections in the routers for each time-slot. In each time-slot, specific circuits are active, eliminating contention and the need for dynamic arbitration in the routers. Packets never wait and are immediately forwarded to the next router. The schedule describes the assignment of a number of time-slots to virtual circuits, providing different throughput guarantees to different virtual circuits.

The TDM schedule is computed before run-time based on the communication requirements of each virtual circuit and is re-computed periodically. It is static and programmed into the network during the initialization process. As Argo uses source-routing, the schedule is stored in tables in the NIs. These tables are initialized by the processor through the OCP interface during initialization. The schedule information initialized in the NIs includes the assignment of time slots to virtual circuits and the route of each virtual circuit.

Scheduling can be done on different levels of granularity (packets, flits, phits). Argo uses scheduling at the level of phits. The Argo router is a three-stage asynchronous pipeline, using enable latches in the pipeline stages. This differs from a synchronous pipeline using registers in the ability to store at most two phits instead of three. The phits stored in a router pipeline

and the packet length do not match and this creates a misalignment of packets in the routers. Thus scheduling should be done in the level of phits.

Moreover, phit-scheduling is more flexible than flit-scheduling. Aelite and Æthereal use scheduling at the level of flits. The flit size matches the pipeline depth of the router, relating the hardware components (routers) with the flit size and time slot duration. This simplifies the scheduling but requires the hardware to be a multiple of the flit size. Thus any additional hardware components, e.g. link pipelining or mesochronous synchronizers, need to have a multiple of three pipeline stages. Argo uses scheduling at the level of phits and does not have this limitation. The reason that Phit-scheduling is more flexible is that it decouples the hardware from the flit length and the time slot duration, and allows it to be optimized independently.

A phit-scheduler has been developed for the Argo NOC [149]. The scheduler was developed within the T-CREST project by Rasmus Bo Sørensen. The scheduler takes as input a topology graph defining the hardware platform and a task graph defining the task communication and the bandwidth requirements of the virtual circuits. The scheduler evaluates the bandwidth requirements and the available resources, and produces a communication schedule at the level of phits. The generated schedule guarantees that there is no collision of packets, the packets are routed on shortest paths and in-order, and bandwidth requirements are met. The schedule always satisfies the communication requirements. However, in some cases the schedule is so long that it requires a frequency greater than the maximum frequency of the design, meaning that the network resources are not sufficient to satisfy the requirements. The solution is to restructure the application or increase the resources of the network.

4.8. WCET Analysis

Argo NOC has the requirement to be time-predictable, i.e. be able to provide execution time guarantees. We chose the TDM scheme as it is a straightforward way to enable WCET analysis. In TDM all communication is scheduled, based on a pre-evaluated static schedule. This section evaluates how Argo contributes to the WCET of a message transaction.

Argo operation is based on the concept of periodic execution of a specific TDM schedule. The schedule period consists of a number of time slots assigned to virtual circuits. Argo performs DMA-driven "push" operations, and thus the transactions are block transfers. Each message consists of a number of packets. As presented in Section 2.5, the timing contribution of a NOC to the execution time of a message transfer is reflected in three metrics. For Argo, as a TDM network, the three metrics are as follows: (i) the *waiting* time (T_{wait}) until a time slot assigned to the specific virtual circuit is reached, (ii) the *transmission* ($T_{transmission}$) time, as the rate in which packets can be inserted in the network in one cycle, determined by the TDM schedule, and (iii) the *latency* of the last packet to travel through the network from one point to another ($T_{latency}$). The latency of a message transfer through the network of routers is given by the following equation:

$$T_{m.trans} = T_{wait} + T_{transmission} + T_{latency} \tag{4.1}$$

Considering a TDM schedule with a period of P time slots, a virtual circuit is assigned p time-slots within one period of the schedule. Other design parameters affecting the transaction latency are the pipeline depth of a router, D, the duration of a time slot in cycles, c, and the

packet length in phits, l. These parameters in Argo take the values $D = 3$, $c = 3$, $l = 3$. For a message transaction of n packets the timing parameters are analyzed below.

Wait time: For the transmission of the first packet through a virtual circuit, the packet needs to wait for the assigned time slot. For p time slots assigned to this virtual circuit in the period, the waiting time in the worst case is $T_{wait} = (P - p) \cdot c$ cycles, where p lies in the range $[1, P]$. The waiting time is affected by the position of the time slot. $T_{wait} = (P - p) \cdot c$ is a pessimistic bound, as the p time slots, in the typical case, will be spread out over the period of the schedule. In the Argo design it is $T_{wait} = 3 \cdot (P - p)$ cycles.

Transmission: The injection of n packets of data into the network is determined by the throughput of the network. The throughput of a virtual circuit in a schedule period of P time slots depends on the number of time slots which it owns. For a virtual circuit owning p time slots within a period, p packets can be inserted in the network in one period. Therefore, a circuit can reach the total throughput of $p/(P \cdot c)$ packets per cycle. Argo throughput is $p/(3 \cdot P)$ cycles. For a transaction of n packets, $(p \cdot n)/(3 \cdot P)$ cycles are required.

Latency: The latency of one packet to propagate through a router is the router pipeline depth, D. No waiting occurs in the routers. Therefore, for an end-to-end path of h hops, i.e. routers, the latency is $T_{latency} = h \cdot D + l = 3 \cdot h + 3$ cycles.

The latency given by Equation 4.1 applies to all TDM NOCs. Specifically in Argo, the worst-case latency for a message transaction is additionally affected by the TDM skew. The asynchronous network of routers allows the TDM schedule to drift apart among NIs. The TDM skew that the asynchronous network is able to absorb is a limited number of cycles. As analyzed in Chapter 8, this elasticity can reach two or three cycles. This number has a fixed upper limit for a specific instance of Argo and an operating frequency. For a maximum skew of s cycles the overall latency of message transaction for Argo is given by the equation:

$$T_{m.trans} = 3 \cdot (P - p) + (p \cdot n)/(3 \cdot P) + 3 \cdot h + 3 + s \tag{4.2}$$

4.9. Discussion

This chapter presented the main design principles of Argo architecture and its key ideas. It showed how Argo is time-predictable, time-elastic, able to support GALS architecture and efficient by design. It uses a TDM scheme, based on a static schedule which is pre-evaluated on the basis of the communication requirements. The operation of the TDM mechanism controls the overall traffic in the NOC, resolving the contention in the routers and providing hard throughput guarantees. A novel aspect of Argo is its timing organization and its NI design. The combination of the two creates the conditions for a very small and lightweight design as will be shown in the next chapters.

The novel NI design integrates DMA controllers and a TDM schedule, by interleaving the operation of DMA controllers based on the schedule. This flattens the layered implementation of the typical NI and allows an efficient implementation that eliminates arbitration, buffering and end-to-end flow-control. The dual-ported SPM implements clock-domain-crossing, solving the synchronization issue between processors and NIs. A clock-domain-crossing is required only for the configuration of DMA controllers. The network of asynchronous routers inherently provides synchronization. The connection between routers and NIs is safe from

metastability under the timing assumption that the frequency of the NIs is slower than the handshake capabilities of the asynchronous network. The overall design creates a direct path from local SPM to remote SPM without any buffering, end-to-end flow control or synchronization along the path.

The timing organization includes an asynchronous network of routers, wrapped in a timing boundary of mesochronous NIs. The asynchronous routers perform synchronization in a distributed way. They maintain the data flow by utilising a join-fork mechanism, synchronizing all the packets when they arrive. At the boundaries of the asynchronous network, the NIs also maintain the data flow based on the TDM schedule by inserting and removing one packet in the network in every time-slot.

The Argo NOC supports a GALS system organization. The NIs operate on a global NI clock directing the TDM scheme. The processors directly access data in their local SPM, without additional synchronization needed. The NIs also directly access the data in the local SPM from its second port, without additional synchronization. A clock domain is crossed between processors and NIs only at start-up for schedule initialization, and during normal operation for programming the DMAs. This interface separates two clock domains and requires explicit synchronization. This synchronization adds a limited number of cycles in the execution of a message transfer. This delay is added in the initialization process and during normal operation for setup and control of a DMA message transaction, but not for the data payload. Setup and control is a rare event compared to the main operation of the NI, thus the overhead is limited.

Argo is time-predictable and time-elastic. These two principles are somewhat contradictory. TDM is a way of providing latency guarantees. To do that, it requires global synchronicity. On the other hand the asynchronous network of routers allows time-elasticity between the NIs. The Argo architecture successfully maintains TDM global synchronization allowing some skew among NIs. The amount of skew absorbed is within some limits and, as shown in Section 4.8, a worst-case latency analysis is straightforward and can take possible skew into consideration.

5
Network Interface

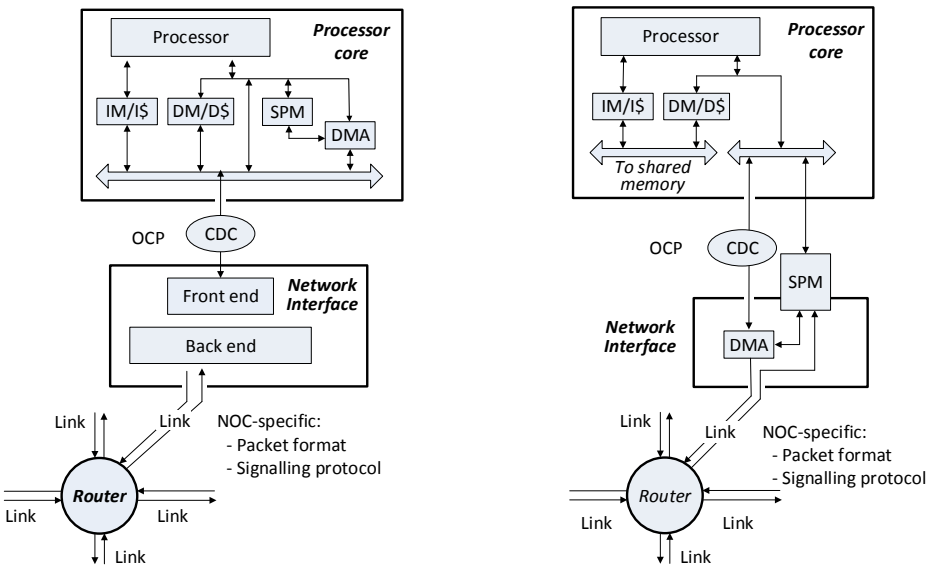
A fundamental and novel part of the Argo NOC is the network interface (NI). The novel design of the NI simplifies the overall NOC design. Through a number of design choices, it removes the need for arbitration, buffering, flow control, and clock-domain crossings for the payload. It creates a direct path from the local SPM of one processing core to the local SPM of a remote processing core. This chapter presents the NI design and explains the design choices that lead to the simple and efficient NI design and consequently the efficient NOC design. The Argo NI is evaluated through its implementation in two different technologies: ASIC and FPGA. Parameters considered in the evaluation are overall area, scalability and frequency. The Argo NI is also compared to alternative state-of-the-art NIs. The ASIC implementation involves a gate-netlist synthesized in a 65nm CMOS STMicroelectronics technology library. The FPGA implementation targets an Altera DE2-70 FPGA board with a Cyclone II EP2C70 chip.

The content of this chapter has appeared in the following publication:

- "An Area-efficient Network Interface for a TDM-based Network-on-Chip", Jens Sparsø, Evangelia Kasapaki, and Martin Schoeberl [152].

5.1. Network Interface Design

The design of the NI is based on two main ideas, as previously mentioned in Chapter 4. These are the integration of DMA controllers with the TDM schedule and the use of a dual-ported SPM for clock-domain crossing between the processor and the NI. This section presents the complete design and its functionality in detail, and explains how the basic ideas lead to an efficient design.



(a) Typical NI design

(b) The Argo NI design

Figure 5.1.: Block diagrams of typical and Argo NI designs.

5.1.1. Baseline Ideas

In a typical multi-core architecture where communication is handled over a NOC, the role of the NI is to connect the processing cores with the network of routers as seen in Figure 5.1(a). As mentioned in Section 2.4.2, typical NIs consist of a front end and a back end component. The front end interfaces the processor with a standard read-write transaction interface, specific to the processor. The back end interfaces the NOC with a packet-stream interface, specific to the network. The layered implementation of the typical NI complies with the protocol stack abstraction. On the other hand, as shown in Section 2.4.3, the end-to-end transmission of data, i.e. from the local SPM of one core to the local SPM of another core, requires data to cross several layers. The crossing of each layer requires buffering and control in the NIs to safely bridge different layers.

The essence of TDM is that it creates end-to-end virtual channels over the network, avoiding buffering, flow-control, and dynamic arbitration, throughout the network of routers. However, due to the layered implementation of typical NIs, these cannot be avoided in the NI. Thus although TDM removes the need for flow control, arbitration, and buffering along the path through the routers, there is still a need for them within the NIs. The cost implied in the NIs contrasts with the simplicity and gains in the routers resulting from the TDM-scheme. For this reason, the Argo NI reconsiders this layered approach, while maintaining the processor-specific and network-specific interfaces and implementing the required services. This is done by creating a direct data path from SPM to SPM without buffering, flow control, arbitration and synchronization of communication, resulting in a much smaller area and energy consumption. The Argo NI design is shown in Figure 5.1(b).

One of the main features of the Argo NI design is the integration of DMA controllers with the TDM schedule as a feature of the NI functionality. Since source routing is used, schedule tables are placed in the NIs and they drive the TDM operation. DMA controllers are integrated with the TDM schedule, such that each time-slot of the TDM schedule is dedicated to a DMA controller. Each DMA controller is dedicated to an outgoing channel, and thus the different end-to-end channels have access to the network in a time-multiplexed way, directed by the TDM schedule. Arbitration among the channels is handled through the TDM schedule. The time slot in Argo has a duration of three cycles, which matches the size of packets in phits. Within a time slot, a DMA controller operates by constructing one packet of data and injecting it into the network of routers, one phit per cycle. In this way the Argo NIs implement the packetization of data.

The second main feature of the Argo NI design is the dual-ported SPM and its use to directly access data from the processor and the NI. The SPM is placed between the processor and the NI, offering one port towards the processor and one port towards the NI. The dual-ported SPM serves as an interface between two clock-domains, as the data do not need to cross a clock-domain to be moved from the SPM to the network. The fact that the NIs can directly access the data in the SPM when needed eliminates the need for buffering in the NIs. It also allows the data to be written immediately to the SPM upon arriving at the NI. This eliminates the need for buffering and end-to-end flow-control following from the limited buffering capabilities.

The NIs offer an additional interface towards the processor for configuration of the TDM schedule and programming of the DMA controllers. This interface is on OCP read-write transaction interface. A DMA table contains an entry for each controller, which can be read and written by the processor. An explicit clock-domain crossing is required in this interface. However, the communication going through this interface is limited to configuration and control data. Towards the network of routers, synchronization is not required and metastability is avoided for two reasons: firstly the use of asynchronous routers, and secondly the timing assumption that the operating frequency of the NIs is lower than the handshake capabilities of the routers. The asynchronous routers exhibit time-elastic behavior and operate in a flexible manner according to the rate that is applied at their end-points, up to a maximum rate. As long as the applied rate is below their maximum rate, they will always be able to adjust to the NIs' frequency without explicit synchronization.

5.1.2. Core-to-Core Communication

The Argo NI design extends the circuits established in the packet-switching network of routers by the TDM schedule, through the NIs and until the SPMs. Thus for a message transfer in the Argo multi-core platform with the novel NI design, data travels in a direct path from SPM to SPM. The TDM schedule establishes a circuit that is active for the current time slot. Thus, the data propagates through this circuit without the need for arbitration, clock-domain-crossing, buffering, or end-to-end flow-control. The transfer of a message in the Argo architecture, as opposed to the one in a typical architecture described in Section 2.4.3, goes through the following steps, which are also illustrated in Figure 4.2:

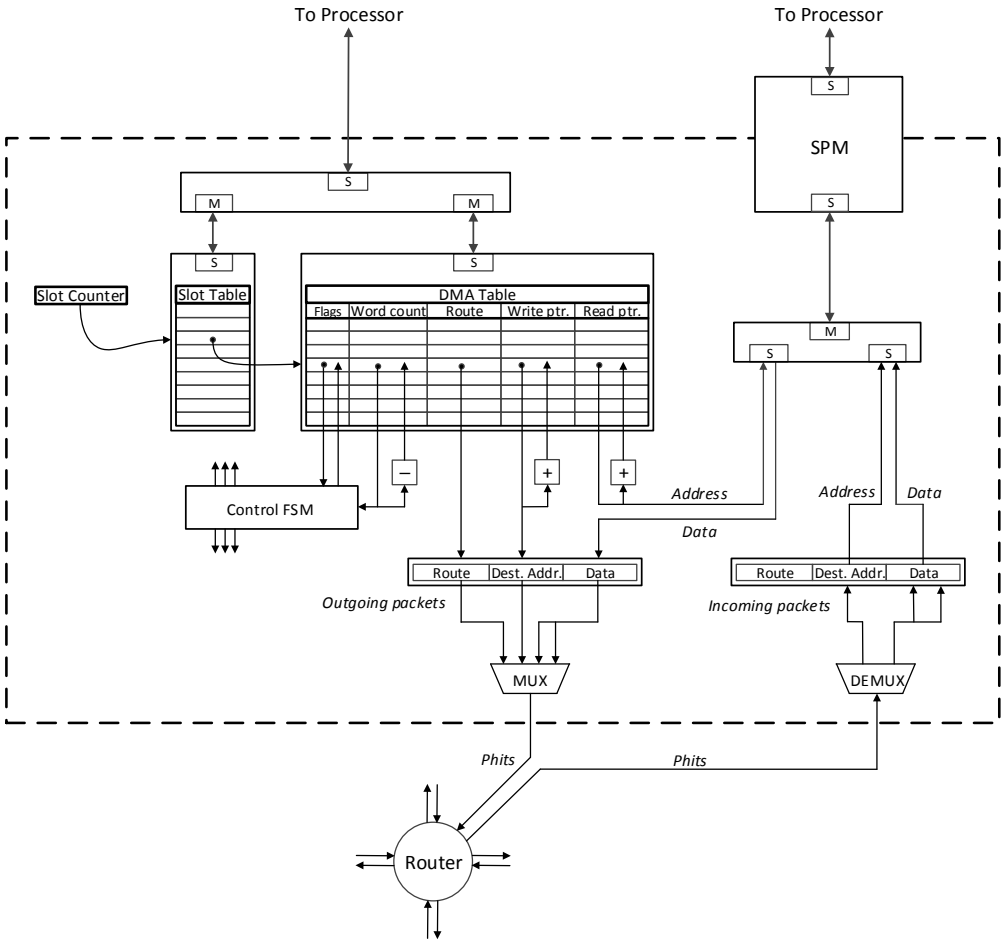


Figure 5.2.: Micro-architecture of Argo NI.

1. At the source end, during the assigned time slot, the corresponding DMA controller directly reads the data for a single packet from the SPM, and constructs a packet with the routing information.

2. During the assigned time slot the packet is injected into the network of routers. The packet traverses the network and arrives at the destination NI after a fixed number of cycles.

3. At the receiving end, the NI directly writes the data from the packet to the SPM in the address defined in the header of the packet. The complete message, which consist of a number of packets, arrives at the receiving SPM after the number of time slots which are required for the transmission of all the packets of the message.

5.1.3. Micro-Architecture and Functionality

Figure 5.2 shows a detailed block diagram of the NI. The main parts of the micro-architecture of the NI design are the *Slot Counter*, the *Slot Table* and the *DMA Table*. The NI has two types of interface: read/write transaction type interfaces towards the processor and the SPM, and packet-stream type interfaces towards the network. The read/write interfaces are either master or slave type, noted as M and S in Figure 5.2, correspondingly. The interface towards the processor is a 32-bit wide OCP interface, while the interface towards the SPM is a 64-bit wide OCP interface. Towards the network there is one incoming and one outgoing interface, both one-phit wide, i.e. 35 bits.

The *Slot Counter* is a simple counter that counts the TDM time slots and is used as a common reference point to direct the whole time-multiplexing process. It is reset concurrently (possibly with some skew) in all NIs and is clocked with the common mesochronous NI clock. The value of counter is incremented for every time slot in all NIs. Thus, the value of the Slot Counter should be the same in all NIs (with some skew) and can be used as a global reference point. In Argo the time-slot is three clock cycles.

The TDM schedule is stored in the *Slot Table*. The value of the *Slot Counter* indexes the Slot Table. Every entry in the Slot Table represents a time slot, and consists of a valid bit and an index to the *DMA Table*. The valid bit indicates whether the specific time slot is assigned to a DMA controller. If the valid bit is true then the table entry includes a pointer to the corresponding DMA controller that the time slot is assigned to.

The *DMA Table* is a table representation of the state of the DMA controllers. Each entry of the table corresponds to one DMA controller and consequently one outgoing channel. Each entry consists of two control bits (flags), a word count field, a route field, a read pointer, and a write pointer field and is 64 bits wide. The *Flags* field consists of a *valid* bit and a *done* bit. The *valid* bit indicates whether the corresponding DMA controller is active. The *done* bit indicates whether the message transfer is finished. The *Word count* field is 14 bits wide and shows the number of words to be transferred in this message transaction. The *Route* field is 16 bits, the same as in the packet format, and shows the route for this channel. The *Read ptr.* and *Write ptr.* fields are each 16 bits wide and represent the read and write addresses in the local and remote SPM from which and to which the next data are transferred, respectively. The DMA table fields of a DMA controller can be accessed as an entire entry or individually in three parts. The latter possibility is used when the processor accesses the DMA controllers through the 32-bit OCP interface. One part is for the Route field, which is initialized with the channel route information during the initialization process. A second part is for the Read ptr. and Write ptr. fields, which contain the source and destination addresses, respectively, of the next packet transfer. A third part is for the Flags and Word count fields, which are accessed to set and check the status of the controller.

A *Control FSM* controls the functionality of the NI regarding access to the table structures and access to the SPM, for reading and writing. The table structures are accessed by the NI during normal TDM operation and from the processor to set up and control the DMAs. The DMA table is accessed by the NI as an entire 64-bit entry, and by the processor in three 32-bit parts, as explained in this section, through the 32-bit OCP interface. The SPM is accessed by the NI outgoing channel for reading, and by the NI incoming channel for writing. The port to the SPM is 64 bits wide, so double words can be read/written. A TDM time slot corresponds

to the time it takes to transmit/receive a packet into/from the network of routers. As described in Section 4.3, a packet (or flit) consists of three phits, and consequently a time slot consists of three clock cycles. During the three cycles of one time slot, it is possible to access the SPM, the slot table, and the DMA table three times, which is more than enough to support the different needs.

During normal operation, the NIs send and receive packets to and from the network of routers in the following manner: The slot counter indicates the current time slot. The value of the slot counter indexes an entry in the slot table, i.e. a time slot. If the valid bit of this time slot is true, a pointer in this entry indexes an entry in the DMA table, i.e. a DMA controller. If the valid bit for that DMA controller is true, and the done bit is false, the DMA controller is active. The route and write pointer are used to construct the header phit of the packet. The read pointer is used as a read address to the local SPM. Two words of data can be read in one cycle from the SPM, and they are used to build the two payload phits of the packet. Within a time slot, one packet, i.e. one phit per cycle, is transmitted towards the router through the outgoing channel interface. For every packet that is transmitted, the read/write pointers are incremented to point to the next SPM address that contains data to be transmitted in the next time slot, and the word count is decremented to show the number of words left to complete the message transfer. At the same time, one phit is received per cycle from the router through the incoming channel interface. When a header phit is received, it is decoded. When the payload phits have arrived the data is written in the local SPM. Finally, for every time-slot the slot counter is incremented to the next time-slot value.

In the OCP interface towards the processor, the NI behaves as a slave. The processor uses the interface either to configure the TDM schedule in the slot table during initialization or to program the DMA controllers and check their status during normal operation. The slot table and DMA table are in the address space of the processor, and thus the above operations can be done as OCP write/read transactions. The NI responds by decoding the address and writing or reading the corresponding data. Transactions in which the processor accesses the entries in the DMA table cross the clock-domain boundary. This means that these transactions will suffer from the latency of the synchronization [51]. Assuming dual flip-flop synchronizers, a read transaction experiences 2 to 4 cycles of added latency. As setting up a DMA transfer is likely the be a rare event compared to DMA initiated read or write transactions, this is a small overhead. As it involves a fixed number of cycles, it does not affect the real-time properties of the Argo NOC.

5.1.4. NI for Asynchronous Network

The Argo NI can be connected either to a network of asynchronous Argo routers, or to a network of equivalent synchronous or mesochronous TDM routers. A number of clocked and asynchronous routers that can be connected to the Argo NI are presented in Chapter 6 of this thesis.

As mentioned in Section 4.7, asynchronous routers differ from the synchronous in the number of tokens, i.e. phits, they can store in their pipeline. The three-stage latch pipeline of the Argo router can hold two phits at most. This fact in combination with the packet size of three phits results in a misalignment of packets in the routers. Scheduling at the level of phits introduces the flexibility needed to deal with this, as explained in Section 4.7. The misalignment

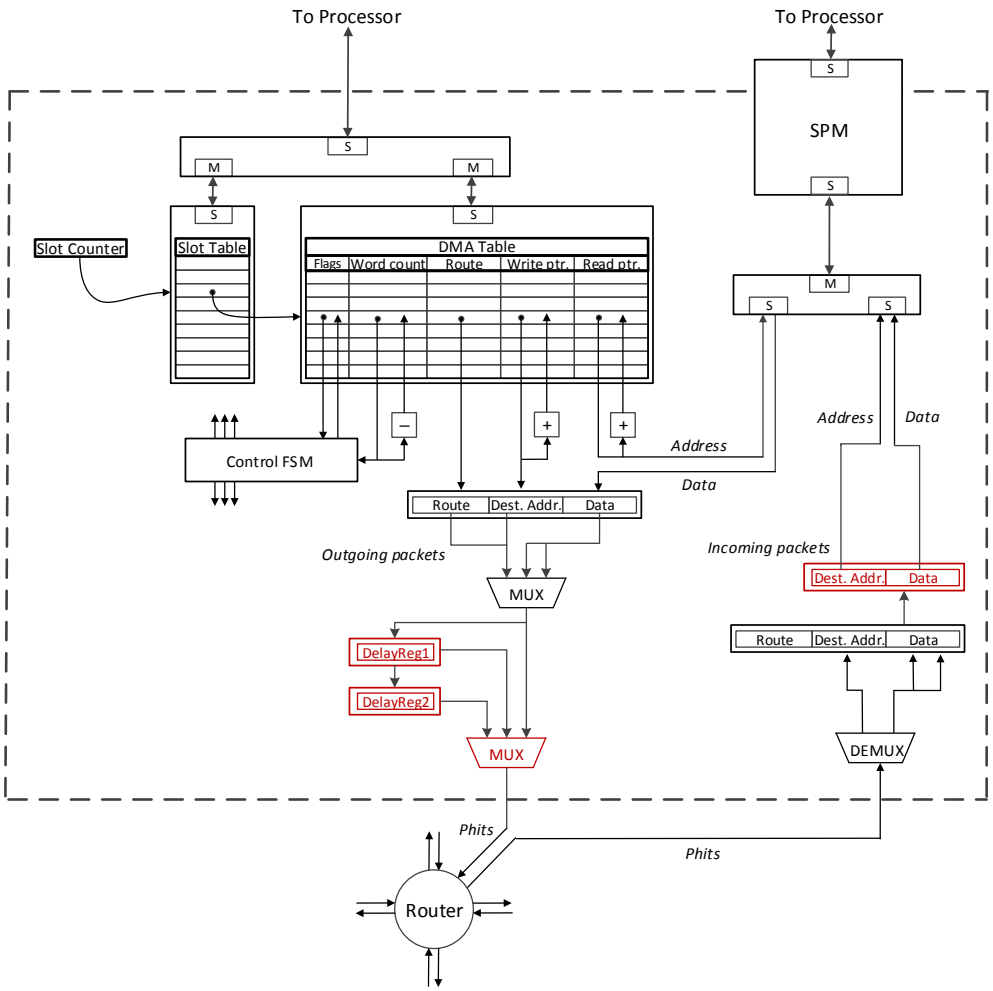


Figure 5.3.: Micro-architecture of Argo NI with three-stage asynchronous routers.

appears in the NIs as well, as the packets are misaligned within the time slot, such that the phits appear in a time slot with a delay of zero, one, or two cycles.

To handle the misaligned phits in the time slot, additional components are required in the NI micro-architecture, and they are shown in red in Figure 5.3. For the outgoing communication, a two-bit field is added in the *Slot Table* to indicate the phase difference of zero, one, or two cycles. In the outgoing channel two phit-wide buffers, *DelayReg1* and *DelayReg2*, and a multiplexer, *MUX*, are added in the path, to implement the one or two cycles of delay for the outgoing phits, and the selection among them. In the incoming channel a buffer containing the SPM destination address, *Dest. Addr,*, and the data, *Data*, is added in the path, and the write operation in the SPM is delayed. The buffer holds the write information until the entire packet arrives, even if it is delayed for one or two cycles. The extended NI of Figure 5.3 can be used for all router implementations both synchronous and asynchronous.

NI design size		NI Logic		Slot Tables			DMA Tables		
Time-Slots	DMA ctrls	LUTs	FFs	LUTs	FFs	BRAM bits	LUTs	FFs	BRAM bits
16	4	326	162	62	51	-	175	323	-
	8	337	162	72	68	-	378	579	-
	16	341	162	82	85	-	34	3	1024
32	4	326	163	111	99	-	175	323	-
	8	339	163	-	-	128	378	579	-
	32	346	163	-	-	192	34	3	2048
64	4	328	164	-	-	192	175	323	-
	8	340	164	-	-	256	378	579	-
	64	351	164	-	-	448	34	3	4096

Table 5.1.: Area figures for a selection of NI implementations when synthesized for an Altera EP2C70 FPGA.

5.2. Implementation Results

To evaluate the NI design, and assess the effects of the schedule tables in the NIs and the scaling of the NOC dimensions, we implemented different instances of the NI. We developed the design as a parametrized VHDL description and implemented a variety of configurations. All the implemented instances include the NI logic as presented in this chapter and the slot and DMA tables. The SPM is not included in the NI implementation, as it is considered the private memory of the processor. The implementation was done in two different technologies, FPGA and ASIC. This section presents the results of the implementation.

5.2.1. FPGA Implementation

For the FPGA design we used an Altera DE2-70 FPGA board with a Cyclone II EP2C70 chip and Altera Quartus for the implementation.

Table 5.1 shows the area figures for a selection of NI implementations. The implementation results refer to the NI design of Figure 5.2, as a clocked design is preferred for the FPGA implementation, without the SPM. The different configurations consider a slot period of 16 to 64 time slots, and a number of DMA controllers between 4 and 64. The instances implemented were chosen, considering an all-to-all communication pattern with one uni-directional channel towards every processor. In such a pattern of communication, and considering that there is a one-to-one correspondence of DMA controllers and outgoing channels, the number of DMA controllers directly represents the dimension of the NOC. In this way, 4 DMA controllers are sufficient for a 2x2 NOC, 8 DMA controllers for a 3x3 NOC, 16 DMA controllers for a 4x4 NOC, and 64 DMA controllers for an 8x8 NOC. The length of the slot period represents the traffic load of communication as it is evaluated by the scheduler. The slot period is an indirect measure of the NOC dimension, as bigger NOC dimensions produce larger amounts of traffic, and require longer slot periods to satisfy the requirements.

#time-slots	#DMA ctrls	NI logic	Slot table	DMA table	Total Area
8	4	6131 μm^2	795 μm^2	5466 μm^2	12632 μm^2
23	16	6655 μm^2	4215 μm^2	19565 μm^2	30395 μm^2

Table 5.2.: Area figures for two instances of the Argo NI in ASIC implementation.

Table 5.1 lists the breakdown of the area resources into the NI logic excluding the tables, the slot table, and the DMA table for the different configurations in four-input look-up tables (LUTs), flip-flops (FFs), and block RAM (BRAM) bits. The results show that the size of the NI logic stays constant for the different NI dimensions. Most of the area is due to the storage requirements of the DMA and slot tables.

The number of time slots reflects the size of the schedule as evaluated by the scheduler, i.e. the traffic load in the NOC. As expected, the cost of the slot table is the number of slots multiplied by the number of bits in an entry. In the Argo NOC we consider all-to-all communication, so each NI contains one DMA controller per processor. Therefore the size of the DMA table directly relates to the size of the NOC. As expected, the cost of the DMA table is the number of DMA controllers multiplied by the number of bits (64) in an entry.

Overall, Table 5.1 shows that the cost of the tables is small and it scales well with the size of the NOC (DMA table) and the communication requirements (slot table).

5.2.2. ASIC Implementation

For the ASIC design flow, the NI was synthesized in a 65 nm CMOS standard cell technology library from STMicroelectronics. We simulated the designs in ModelSim and synthesized them in the Synopsys Design Compiler. To compare with alternative designs from the literature, we also synthesized an instance in an 90 nm STMicroelectronics CMOS technology library. The ASIC implementation results presented in this section refer to the NI design of Figure 5.3 without the SPM.

Table 5.2 shows the area resources of the implementation of two instances of the NI, one for a 2x2 and one for a 4x4 NOC. The table shows the area breakdown of the NIs into the NI logic excluding the tables, the slot table and the DMA table, together with the total area. As in the FPGA implementation, the ASIC results show that the NI logic excluding the table size is minimally affected by the size of the network. The DMA table size scales linearly with the number of outgoing channels from that NI, i.e. with the dimensions of the NOC. The slot table scales with the schedule period. The frequency achieved by both of the two instances is approximately 1 GHz, as it is not affected by the different sizes of the tables.

A NOC with similar functionality to Argo (TDM) is the aelite NOC [59]. The implementation costs for a specific aelite NOC instance are discussed in [59, Sect. 8.1]. This NOC instance comprises 6 routers and 11 NIs and supports 45 bi-directional channels. The TDM schedule period is 24 slots. The aelite instance was synthesized in 90 nm technology. FIFO buffers in the NIs related to the channel endpoints account for 85% of the area of the entire NOC, using flip-flop based FIFO buffers. According to the figures reported in [59], we calculate the average area of one of the 11 NIs to be 0.49 mm^2 when using flip-flop based FIFOs, 0.22 mm^2 when using SRAM based FIFOs, and 0.13 mm^2 when using custom FIFOs. We implemented

an instance of our NI in 90 nm technology for comparison. This NI instance configuration is intended for a 3x3 NOC, comprising 9 routers and 9 NIs, and supports 8 outgoing channels from each NI, i.e. 72 uni-directional channels for the entire NOC. This Argo NI instance, including DMA controllers that are not part of aelite NI, has an area of 0.024 mm^2, 5.4 times smaller than the aelite instance.

5.3. Discussion

An NI design provides a bridge between a read/write transaction interface towards the processor (front end) and a NOC-specific interface towards the network of routers (back end). Thus the front end of an NI design implements a standard protocol while the back ends of different NI designs can be as different as the routers of a NOC. The features of the NOC and the communication requirements have an impact on the NI design. Therefore, different NIs implement different features and are therefore difficult to compare. Moreover, few NI designs and implementations are presented in the literature, compared to the number of published router designs. Exceptions that address the design of NIs are [138], [131] and [59], which are further explained in Section 2.4.2.

To make a generalized comparison, area measures for typical-size NIs are reported [138] to range from 7 to 50 kgates – a gate being a minimum size two-input NAND. A typical instance of the original Æthereal NOC, supporting both GS and BE traffic, is 0.25 mm^2 in a 0.130 µm CMOS technology [131], or 21 kgates [138]. The Argo NI corresponds to 5.5 kgates, considering the area of a minimum drive-strength 2-input NAND cell to be 4.4 μm^2. This number shows that the Argo NI is 3.8 times smaller than Æthereal. As mentioned in Section 5.2.2, the Argo NI is also 5.4 times smaller than an aelite NI instance [59].

An interesting perspective on the area measures reported in literature is given by a comparison with the size of a synthesizable 32-bit processor. The implementation of the original MIPS R2000 CPU (excluding caches) used 110,000 transistors or 27.5 kgates. In a 90 nm technology the size of a mips32-m14kc processor is 0.2-0.5 mm^2 [98]. These figures are of the same magnitude as the published NI designs, and consequently a typical NI is as large as a 32-bit CPU.

Overall, the area figures reported for each NI design are not directly comparable, as each NI implements different features. The Argo NI includes DMA controllers and schedule tables which are not normally a part of NI design. On the other hand, other NI designs may implement additional features that are not part of the Argo NOC. All in all, the comparisons with NIs designed for TDM NOCs as aelite and Æthereal show an area improvement of 4 to 5 times.

The increased area of typical NI designs comes from their layered implementation. The NIs are designed with the protocol layers in mind. This separation is intended for easy composability of components and services. The crossing between layers inflicts a hardware cost on the NI design. The Argo NI design flattens this layered implementation of the NI for the data, as it allows direct access to the core local memory, i.e. SPMs, from NI elements, i.e. DMA tables, based on the NOC time-multiplexing operation, i.e. the TDM schedule. This results in high gains in area. However, Argo NI still maintains composability in the processor interface for configuration, as it implements a standard protocol (OCP) for the front end. The back end is NOC-specific for all NOCs and needs to be designed for the individual NOC features.

6
Router Architecture

A basic structural and fundamental part of the Argo NOC is the router. The Argo router was inspired by the aelite TDM router [59] and an initial attempt at an asynchronous version of it [48]. However, previous work did not reach an implementation or a complete exploration of its behavior. This chapter presents the Argo router design along with alternative asynchronous designs and equivalent clocked versions for comparison. All of the designs are based on the same pipeline but use different ways to synchronize between pipeline stages. The synchronization may rely on a global clock, on mesochronous synchronizers or on asynchronous handshaking. In addition, this chapter presents the implementation of these designs, compares the results, and explains the reasons that led to the specific choices for the Argo router. The implementation involves gate-netlist synthesis in a 65nm CMOS STMicroelectronics technology library, as well as layout of the routers. The content of this chapter has been presented in the following publications:

- "Router Designs for an Asynchronous Time-Division-Multiplexed Network-on-Chip", with authors Evangelia Kasapaki, Jens Sparsø, Rasmus Bo Sørensen, and Kees Goossens [74].

- "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation", with authors Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø [71].

6.1. Basic Router pipeline

All the routers described in this chapter use the same basic pipeline. This section presents the basic pipeline and the combinational logic of the stages which it comprises.

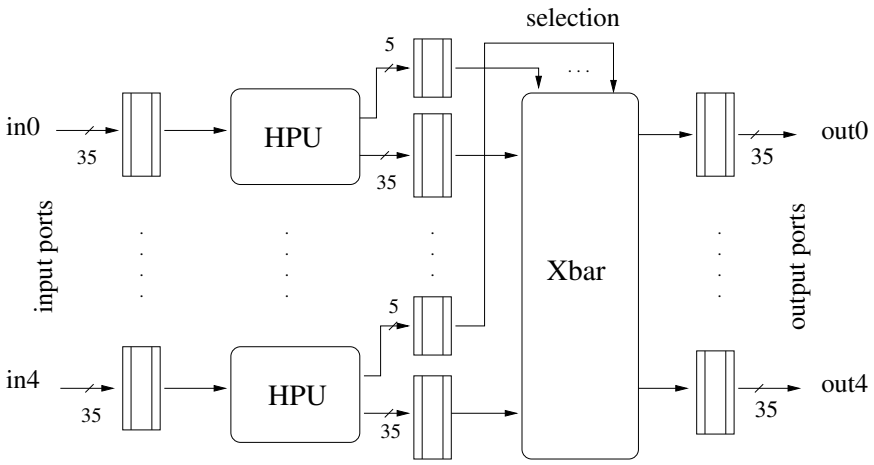


Figure 6.1.: Basic Argo router pipeline.

The basic router has five input and five output ports, which allows the construction of mesh and bi-torus network topologies. The router consist of three pipeline stages, each of which is one phit (i.e. 35 bits) wide. The pipeline stages are: (i) the link traversal stage, (ii) the header parsing unit (HPU) stage, and (iii) the crossbar switch (Xbar) stage. A block diagram of the pipeline is shown in Figure 6.1.

Since source routing is used, the router contains no routing information. Instead, the header phit of every packet contains the routing information of the packet. The HPU decodes the header phit and the corresponding output port selection is forwarded to the Xbar to enable the appropriate connections. The router has no information about the connections enabled or the length of the packet. The decoding is done for the header phit and is kept until the traversal of the last phit. Thus no routing tables or any other information is kept in the router, apart from the connections enabled for the propagation of the current packets.

According to the TDM scheme, specific connections are enabled in every time slot in every router. This eliminates contention in the routers and removes the need for arbitration. The packets therefore traverse the router without waiting for arbitration, and consequently no buffering is required in the router and the latency for a packet to traverse the router is the time for the packet to propagate through the three-stage pipeline. All the above shows that the router is a very lightweight design. The main functionality of the router is divided between the two stages of the HPU and the Xbar. The link traversal unit is used to pipeline the link.

6.2. Asynchronous Argo Router

The proposed design for the Argo router is an asynchronous design that uses the basic pipeline described in Section 6.1. Instead of clocked registers, the Argo router uses handshake latches to separate the pipeline stages. A handshake latch consists mainly of a normal enable latch and a latch controller. The latch controller implements the synchronization of communication between neighboring stages to control the propagation of data from one stage to the next, using an asynchronous handshake protocol. The protocol used in the Argo router is a two-phase

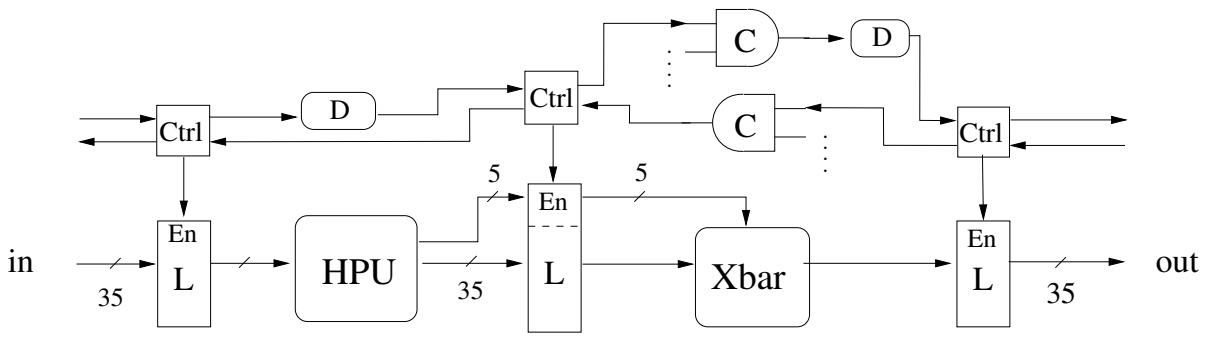


Figure 6.2.: Argo router pipeline showing with the elements of asynchronous communication.

(NRZ) handshake protocol with bundled-data encoding [151] as described in Sections 2.6.3 and 2.6.4.

As explained in Section 2.6.3 the two-phase handshake protocol requires a request and an acknowledge signal. The handshake controller of a pipeline stage synchronizes the communication between the next and the previous pipeline stage controllers. As explained in Section 2.6.4, the bundled-data encoding requires the request signal to be asserted only after the data are available. This means that the request signal should arrive at the following controller in the pipeline after the data from the current combinational logic stage is available. Therefore, delay elements need to be added in the request lines, matching the delay of the combinational logic. The delay values are determined after implementation of the logic and measurement of their delay. An additional margin is added for delay fluctuations and process variations. The process is described in more detail in Section 6.5. The complete block diagram of the Argo router with the elements implementing the asynchronous communication is shown in Figure 6.2. Latches *L* are normal latches and they are enabled by the handshake controllers *Ctrl*. Delay elements *D* match the combinational logic delay of each stage. Finally, C-elements *C* implement synchronization of all input and output pipelines in the Xbar stage.

6.2.1. Architecture and Functionality

As shown in Figure 6.2, the Argo router consists of a number of components implementing the datapath and control of the router. The micro-architecture and the functionality of these components are presented in this section.

Handshake controller

The handshake latch used in every pipeline stage in the Argo router appears in Figure 6.3. It consists of a handshake controller, a gating block, and data enable latches. The controller is the Mousetrap controller [148]. This controller was chosen because it is performance efficient and easy to implement. It is fast, as it consists only of a latch and a simple logic gate (XNOR), as shown in Figure 6.3. Furthermore, it uses no special asynchronous cells like C-elements, reducing the needs for C-elements in the router in only the two used in the Xbar stage. This choice of this controller is supported by implementation results comparing a range of controllers. The implementation results are presented in Section 6.6.

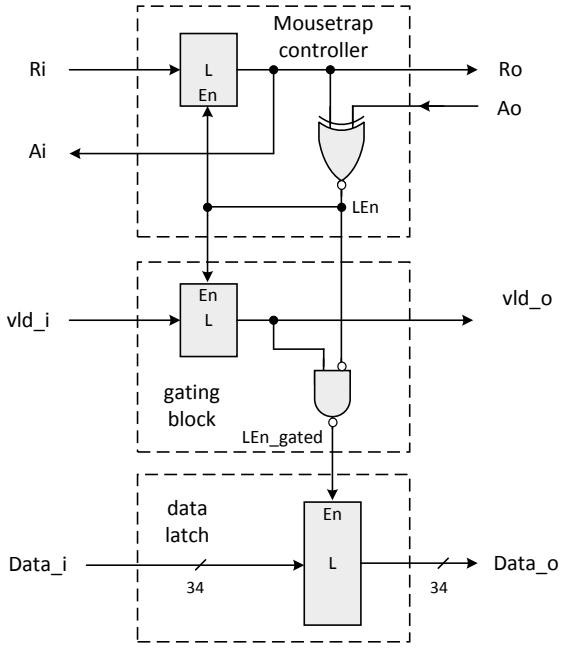


Figure 6.3.: Argo handshake latch, from [71].

The functionality of the controller during a handshake cycle is as follows: Initially, all the handshake signals (request and acknowledge) are initialized to '0', the latch enable signal has the value '1', and the latches are normally transparent. When a request arrives, the transition propagates through the controller latch and is propagated as a request to the next stage and as an acknowledge to the previous. Subsequently, the output of the XNOR-gate, i.e. the enable signal, turns to '0', setting the latches to the opaque state. When the acknowledge from the next stage signifies its reception of the data, the output of the XNOR gate changes to '1', and the latches become transparent again. A following transition on the request line initiates a new handshake cycle.

Gating

The Argo router implements a "gating" scheme, to save energy consumption when traffic is idle, i.e. in the case of void phits. It resembles clock gating and is applied locally on the latch enable signal that controls the data latches of the current pipeline stage. The phits traversing a pipeline stage may be either valid phits (containing data) or void phits (empty time slots). The *vld* control bit in the header phit of the packet (Figure 4.4) indicates whether a valid phit or a void phit is propagated. In the case of a void phit, the latch enable signal that controls the data latches is gated out. This sets the data latches transparent which is their normal state, propagating 0's. The gating block lies between the controller and the enable latch, as shown in Figure 6.3. The gating consists of an one-bit latch and a simple logic NAND gate. The latch holds the *vld* control bit and is already part of the design. The NAND gate is also used as a buffer, required to drive the enable signal for the data latch. Following from the above, the

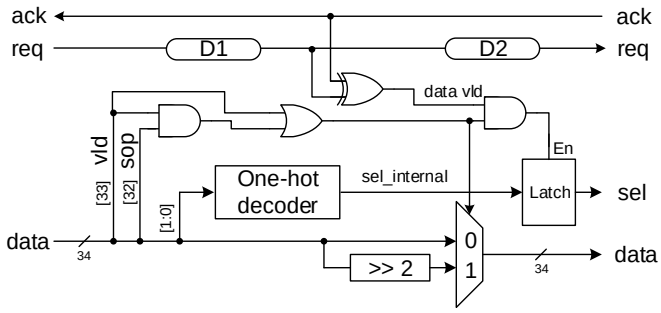


Figure 6.4.: Combinational logic of HPU stage.

Bit-code	Destination
00	north
01	east
10	south
11	west

Table 6.1.: Output port destination encoding.

cost of the gating block appears to be minimal, as it uses components that are used in the latch controller in any case.

HPU

The combinational logic of the HPU stage is shown in Figure 6.4. The HPU logic detects the arrival of a valid header phit from *vld* and *sop* bits of the header phit. If the header phit is valid, the HPU decodes the route field of the header. Two bits of the route field encode the destination port of one hop. The last two bits of the route define the output destination in the current router. A decoder, shown in Figure 6.4, extracts the last two bits of the route and generates an output port selection in a five-bit one-hot encoding. The selection is stored in a latch. At the same time the HPU shifts the header phit two positions to align the header for the next router along the path. The output port selection is also pipelined along with the data and forwarded to the Xbar.

The selection for the appropriate output port is loaded into the latch when a valid header phit traverses the HPU, indicated by the *sop* bit. It remains unchanged until the last phit of the packet has propagated through the HPU, indicated by the *eop*. The enable signal of the selection latch is also controlled by the handshake signals of this stage. The latch cannot be loaded unless the input data are valid within the handshake cycle. The data are valid when a request from the previous stage is signaled and sufficient time has passed for the selection to be evaluated. Thus a delay element in the request line matching the selection decoding is used in the HPU. A second delay element in the request line matches the delay of the selection propagating through the latch.

Table 6.1 shows the two bits encoding for the corresponding output port destination used in the Argo router. The directions are absolute and not relative to the input port. The local output port, leading to the NI, does not have a specific encoding but uses the code of the input port.

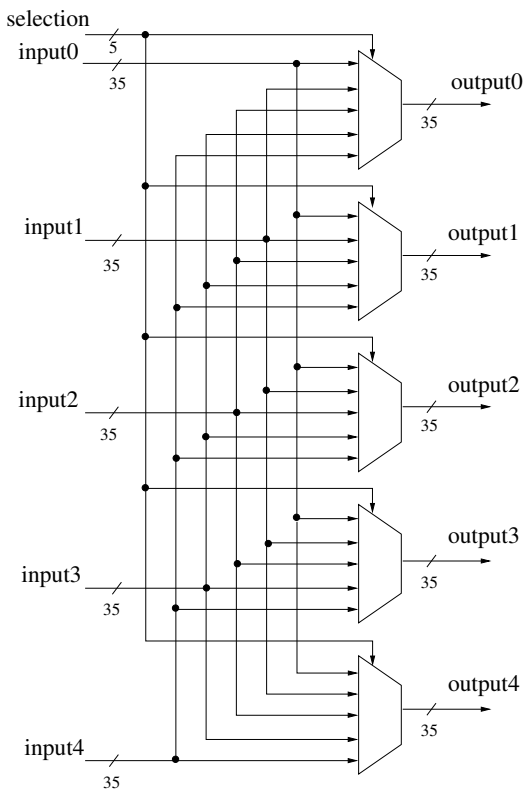


Figure 6.5.: Combinational logic of the Xbar stage.

Argo does not allow forwarding of phits to the same destination port as their source input port. Therefore, the code of the source port is used for encoding the local output port as the final destination of the packet.

Xbar

The Xbar is a simple crossbar switch from five inputs to five outputs. It consists of five five-to-one multiplexers. For every input port the 35-bit data along with the 5-bit one-hot selection is pipelined from the HPU unit to the Xbar as shown in Figure 6.1. The 5-bit selection determines the destination output port. The pipelined selection locks the Xbar in the selected state until the arrival and propagation of the last phit of the packet through the Xbar. If an output is not selected by any of the input ports a void phit is propagated to this output. A block diagram of the Xbar is shown in Figure 6.5.

Synchronization in the routers is achieved in a distributed way. A fundamental concept of the Argo router design is the strongly indicating [151] implementation of the Xbar stage. Besides switching all the input ports to the selected output ports, the Xbar stage synchronizes all input and output communication. This is done by two C-elements implementing join-fork functionality. The C-element is described in Section 2.6.5. A five-input C-element joins all the request signals arriving in the Xbar from the input ports, and forks them to all the output ports. Similarly, a five-input C-element joins all the acknowledge signals arriving in the Xbar from the output ports, and forks them to all the input ports. In every handshake cycle, the

Xbar consumes one phit from every input pipeline (join), and produces a phit on every output pipeline (fork). This join-fork mechanism resembles the ticking of the clock and maintains the data flow in the Xbar, and it is the foundation for enforcing synchronization in a distributed way.

6.2.2. Initialization and Performance

The performance of an asynchronous pipeline is determined by its structural parameters, i.e. the number of stages, the number of data tokens in those stages, and the forward and reverse latency, L_f and L_r, respectively [151]. Maximum performance is achieved when the number of handshake latches per token matches the dynamic wavelength of the circuit. For a two-phase handshake protocol, the handshake period is given by $P = L_f + L_r$ and the dynamic wavelength by $W_d = P/L_f$, [151]. For the Argo router we used gate delays from the implementation and calculated the dynamic wavelength of the pipeline to approximately $W_d \approx 1.5$ latches per token. Thus we decided to initialize the three-stage router pipeline with two tokens. Alternative options that would satisfy the dynamic wavelength would be three tokens in five stages or four tokens in six stages, as only an integer number of latches and tokens can be used. These options would increase the hardware cost of the router. The option with two tokens in three stages was chosen as the most efficient one in terms of hardware and performance.

6.3. Alternative Asynchronous Routers

In the process of developing the Argo router design, we implemented a number of asynchronous routers with the same functionality, to explore and compare alternative options. These options use different handshake protocols (two-phase, four-phase) and data encodings (bundled-data, delay insensitive) and explore the use of the gating scheme. Overall, the designs developed in connection with this thesis are: a four-phase bundled-data design without gating ("4ph-bd"), a two-phase bundled-data design without gating ("2ph-bd"), a two-phase bundled-data with gating ("2ph-bd-g"), and a two-phase bundled-data design with “gating” that uses delay-insensitive Level-Encoded Dual-Rail (LEDR) links ("2ph-LEDR-g"). In comparison with the above designs, the Argo router is a two-phase bundled-data design with gating, but the two-phase protocol is implemented with a different handshake controller. The designs and the reasons for selecting them are presented in the following subsections.

6.3.1. 4ph-bd

The 4ph-bd design implements a four-phase handshake protocol (RZ) with bundled-data encoding and without the gating mechanism. The controller used in this design is a simple four-phase bundled-data handshake controller, shown in Figure 6.6. Initially, all signals are set to ‘0’ and the enable signal *Lt* keeps the data latches in the opaque state. A *Req_in* rise transition changes signals *Req_out* and *Lt* to ‘1’ and the latches to transparent, so the data from the previous stages can be loaded into the latches. An *Ack_out* rise transition signals the reception of the data but does not change the output of the C-element or the state of the latches. A

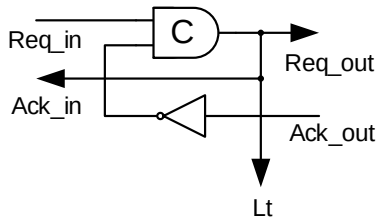


Figure 6.6.: Handshake controller for four-phase protocol, from [74].

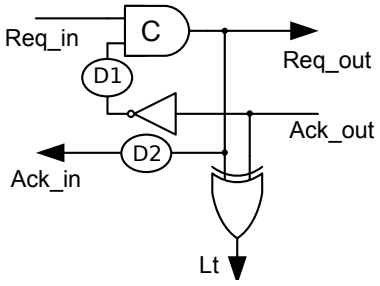


Figure 6.7.: Handshake controller for two-phase protocol, from [74].

falling transition of *Req_in* changes *Req_out* and *Lt* to '0' and sets the latches to opaque again, keeping the latched data stable for the next pipeline stage.

The reason for exploring a four-phase protocol is that it requires simpler hardware for its implementation than a two-phase protocol. On the other hand, the four-phase protocol potentially exhibits worse performance than the two-phase protocol. The reason for this is that a complete four-phase handshake cycle involves four signal transitions, and thus has a longer cycle and higher energy consumption. These assumptions are verified by the implementation shown in Section 6.6.

6.3.2. 2ph-bd

The 2ph-bd design implements a two-phase bundled-data handshake protocol (NRZ), without the gating mechanism. It implements the same communication protocol with the Argo router but uses a different handshake controller. The controller is the reduced complexity two-phase micro-pipeline latch controller [160] shown in Figure 6.7. Compared to the four-phase controller shown in Figure 6.6 it has an additional XNOR gate to generate the latch enable signal *Lt*. Initially, all the signals are set to '0', apart from the enable signal *Lt*, which is '1', and which holds the data latches in the transparent state. A *Req_in* rise transition changes signals *Req_out* and *Lt* to '0' and the latches to opaque, ensuring that the data for the next stage are stable. An *Ack_out* rise transition signals the completion of the handshake cycle and changes the *Lt* signal to '1' and the latches to transparent again.

As shown in the Figures 6.6 and 6.7 the two-phase controller has an additional XNOR gate. However, the handshake cycle is shorter, as it involves two signal transitions instead of four.

Furthermore, this controller needs to fulfil two timing assumptions for safe operation. Firstly, the data latches need to be transparent for sufficient time for the data to be latched, from the time *Ack_out* arrives and until a new *Req_in* initiates a new handshake cycle. This timing requirement condition is satisfied by use of the delay element *D1* in Figure 6.7. Secondly, the data latches need to be opaque for sufficient time for the data to be stable for the next stage, before the acknowledge from the next stage changes their state to transparent. This timing requirement is satisfied by use of the delay element *D2* in Figure 6.7. These delays may affect the handshake cycle. Implementation results are presented in Section 6.6.

6.3.3. 2ph-bd-g

The 2ph-bd design implements a two-phase bundled-data handshake protocol (NRZ), with the addition of the gating mechanism. It uses the same controller as the 2ph-bd design. This design was developed for direct comparison with 2ph-bd, to evaluate the effect of gating.

6.3.4. 2ph-bd/LEDR-g

The 2ph-bd/LEDR-g design implements a two-phase bundled-data handshake protocol (NRZ) with the gating mechanism. In addition, it uses delay-insensitive links. To be delay-insensitive, the links use Level-Encoded Dual-Rail (LEDR) encoding [41]. As explained in Section 2.6.4, LEDR uses two lines (rails) for every signal, to represent the true value and the parity value. The communication is done by alternating between an *odd* and an *even* phase. In the *odd* phase the two rails carry the real value and the parity value. In the *even* phase both rails carry the real value. This implies an area and energy consumption overhead arising from the encoding of each signal with two lines. Nevertheless, a delay insensitive encoding has the advantage of operating correctly regardless of the delays involved. This is a good match for the links, since they can be of arbitrary length. However, the computational functionality within the router calls for a simpler encoding like bundled-data. This design therefore implements the links with LEDR encoding and the router pipeline with bundled-data encoding.

The use of two encodings requires a translation in the input and output ports. At the input ports in the first stage of the router pipeline, a converter translates LEDR to bundled-data encoding. Figure 6.8 illustrates the block diagram of the LEDR to bundled-data converter circuit. The *Type Detection* circuit detects whether the arriving phit is valid and whether it is in its odd or even phase. In the case of a valid phit, the select element (*Sel*) raises a signal at the True or the False port if the phit is in its odd or even phase, respectively. The *Completion Detection* circuit uses a 34-bit C-element to detect the arrival of all data bits at the input port of the router. The *Token Merge* circuit detects the arrival of data, either in their even or in their odd phase and generates a transition in the request signal to the latch controller.

Figure 6.9 illustrates the block diagram of the bundled-data to LEDR converter circuit, used at the output ports of the router at the last pipeline stage. It is a modified version of a previously published converter [66], to allow the gating mechanism to handle different types of phits (valid or void). It differs from the handshake latch of the previous designs in the use of flip-flops in the last pipeline stage of the router instead of enable latches. Double the number of flip-flops are required in order to generate the dual-rail encoding. The flip-flops are enabled with the pulse from the latch enable signal generated by the controller. The *Toggle* element toggles

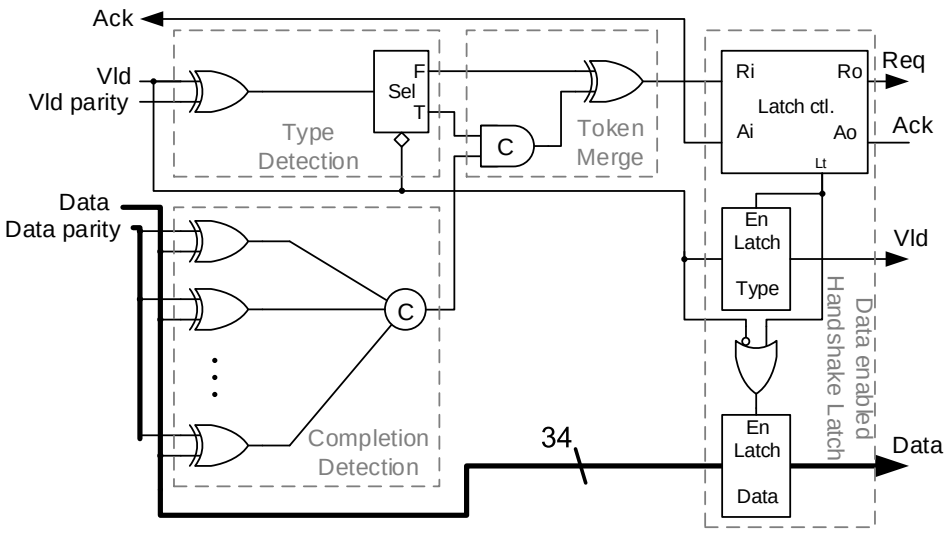


Figure 6.8.: Circuit converting LEDR to bundled-data encoding, from [74].

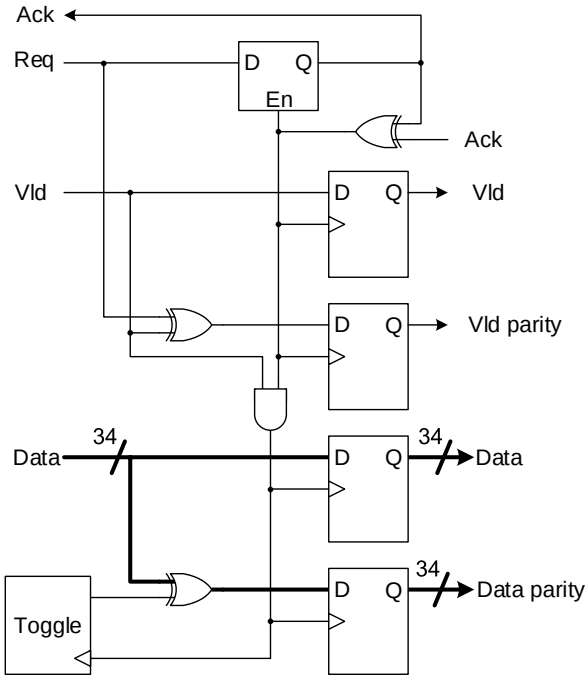


Figure 6.9.: Circuit converting bundled-data to LEDR encoding, from [74].

between the even and the odd phases of the LEDR encoding, such that it generates the true or the parity value of data in the parity rails.

6.4. Mesochronous Router

The typical approach to mesochronous design is to connect synchronous building blocks with mesochronous synchronizers. The mesochronous version of the TDM router presented here follows the same approach. Synchronous routers are used and mesochronous synchronizers connect the synchronous routers, such that a mesochronous synchronizer is placed in every link. The mesochronous functionality introduces a cost that amounts to one synchronizer per link. This cost can be significant, considering that the routers have five ports, i.e. five synchronizers per router.

The synchronous router resembles the Aelite router design [59], since it follows the same pipeline stages. The pipeline of the synchronous router is as shown in Figure 6.1. As a clocked design, it uses registers in the pipeline stages. Every stage is 35 bits wide and 5-bit registers are used in the HPU stage to hold the output selection. The delay elements and the C-elements used for synchronization in the crossbar are not required in the synchronous version.

The mesochronous synchronizers, as explained in Section 3.3, use FIFOs with a number of stages. The synchronizer used in the mesochronous version of the Argo router is a bi-synchronous FIFO [117]. This FIFO implements a full-detector, using a token ring. In particular, the synchronizer implemented in this work is the mesochronous adaptation of the bi-synchronous FIFO [117], with five stages. This FIFO provides flow control signals and can sustain a throughput of one data item per cycle for a clock skew of plus/minus one clock period. Optimized versions of FIFOs (e.g. full-custom FIFOs [169]) may reduce the cost of synchronization. The FIFO design presented in [84] includes only three register stages, since the flow control signals are not required for a mesochronous system. We therefore estimate that it reduces the cost of FIFOs by half. However, the overhead for the mesochronous synchronization is still present.

6.5. Implementation

All the router designs presented in this chapter were implemented, evaluated and compared. The implementation involves description of the designs in VHDL, synthesis in a gate-netlist, and layout. Since the focus is mainly on the asynchronous router designs, the implementation was done in an ASIC design flow. We used a 65nm CMOS standard cell technology library from STMicroelectronics, and conventional EDA tools for the design flow. We simulated the designs in ModelSim, synthesized them in Synopsys Design Compiler, and performed the circuit layout in Cadence SOC Encounter. We performed power analysis in Synopsys Prime Time according to the switching activity on a number of testcases.

The asynchronous design process is not as straightforward as the synchronous one, and is complicated by the lack of tools for asynchronous design. We decided to use conventional EDA tools in this work, and some special features have to be taken into account for the asynchronous designs. A characteristic of asynchronous circuits is the combinational loops in the circuit. In this work we chose to manually disable the combinational loops, rather than allow the design tools to disable them at random points. The reason is to provide more precise path delay evaluations, which are needed by the optimization algorithms in the logic synthesis process [95]. The delay of a standard cell is a function of its fan-out load. If paths are disabled at

certain points, the output load at these points is not considered in the path delay evaluation, leading to poor optimizations. By manually selecting the disable points we aim at minimizing the effect of disabling the paths in the loop.

Furthermore, asynchronous circuits often include special cells, such as the C-elements [158] described in Section 2.3. They can either be implemented with standard cells or be custom made. We chose to do the implementation with standard CMOS library cells. The implementation of the C-element is an SR latch based implementation. We described the behavior in VHDL and let the tool derive the gate-netlist implementation. The result is an NOR gate implementation of the SR latch with some additional gates. This implementation includes feedback loops which were also cut manually.

In the synchronous design process, the combinational part of the design is optimized according to the clock period. The designer sets a target clock period and the synthesis tool optimizes the combinational logic, such that the logic of each stage fits within the time-limits of one clock period. In the asynchronous design process there is no global clock to set a target period. However, combinational logic is still organized in pipeline stages. The process we followed in this work is to imitate a clock signal and use it as a target for optimization. Therefore, we specify the enable signals of the data latches as clock signals. We associate the pipeline stages with two non-overlapping clocks in an alternating way, and use these clock periods as optimization targets. In this way, the tool optimizes the combinational part of the circuit while considering the setup and hold timing constraints of the latches.

Another challenge comes from the timing assumptions that need to hold for the correct operation of the circuit. These involve delay elements whose values need to be determined. For this purpose the designer repeats a process of setting values, synthesizing and performing timing analysis. In our designs the delay elements that need to be determined are the delays in the request lines, matching the combinational logic of each stage (i.e. the HPU and crossbar), and the wire delays, i.e. the link pipeline stage. They are estimated according to the circuit implementation delays with a 20% additional timing margin. The 20% margin is also used in other asynchronous implementations and is considered a safe margin for covering delay fluctuations [83]. Additionally, some delay assumptions exist in the controllers of the 2ph-bd and 2ph-bd-g designs. The corresponding delay elements were set according to the timing assumptions described in Section 6.3.2.

6.6. Results

Table 6.2 lists the results of the implementation of all the router designs presented in this chapter. The second column reports the cell area of the designs in μm^2. The areas of the Argo, the 4ph-bd, the 2ph-bd, and the 2ph-bd-g routers are similar. The 2ph-bd and the 2ph-bd-g designs are slightly bigger than the 4ph-bd due to the additional XOR gate and the delay elements in the controller. The gating does not contribute additionally to the area, as it uses elements that are required in the non-gated designs as well. The 2ph-bd is slightly (less than 1%) bigger than the 2ph-bd-g as a result of heuristics applied by the synthesis tool. The Argo router is slightly bigger than the the 4ph-bd, the 2ph-bd, and the 2ph-bd-g design due to the latch used in the controller. All the above designs are smaller than the synchronous router. The size of 2ph-bd/LEDR-g is twice the size of Argo, due to the cost of delay insensitive links,

Router Designs	Cell Area	Post-synthesis Freq.	Post-synthesis Energy / cycle	Post-layout Freq.	Post-layout Energy / cycle
	μm^2	MHz	pJ	MHz	pJ
Mesochronous	24239	1111	4.32	724	7.87
Synchronous router	8026				
FIFO [116]	16213				
FIFO [169, 84]	(8106)				
4ph-bd	7401	833	7.91	701	8.20
2ph-bd	7594	998	7.92	711	8.03
2ph-bd-g	7536	900		593	
link util. 0 %			1.64		2.11
link util. 70 %			6.51		7.23
link util. 100 %			8.77		9.66
2ph-bd/LEDR-g	12578	862		645	
link util. 0 %			3.82		3.80
link util. 70 %			8.31		9.95
link util. 100 %			10.02		12.00
Argo	7715	1126		746	
link util. 0 %			1.28		2.17
link util. 70 %			6.45		6.54
link util. 100 %			8.24		8.39

Table 6.2.: Results for the router designs implementations.

e.g., the double wires and the translation circuits. The size of the mesochronous router is three times the size of Argo router, due to the cost of FIFOs. The FIFOs' cost depends on their implementation. The FIFOs implemented in this work [116] have an area which is twice that of the synchronous router. With a different implementation [169], [84], we estimate the size of the FIFOs to be equal to the size of the synchronous router.

The third and fifth columns present the operational frequency for the post-synthesis and the post-layout routers, respectively. We consider frequency of an asynchronous circuit the handshake frequency, which is not evaluated by the EDA tools. As asynchronous designs their frequency depends not only on the design but also on the data and operating conditions. We evaluated the frequency by simulating the post-synthesis and post-layout netlists in an environment of eager producers and consumers. The designs operate on similar frequencies with small differences. In the post-synthesis results Argo is slightly faster than the rest. Among the asynchronous designs, the 4ph-bd is the slowest as expected from the four-phase handshake protocol. The gating scheme and the delay insensitive links also affect the handshake cycle, as shown for the 2ph-bd-g and 2ph-bd/LEDR designs. For all designs, the frequency degrades in the post-layout stage by approximately 15-35% due to wireload effects. The Argo router is still slightly faster than the rest.

The fourth and sixth columns respectivey show the post-synthesis and post-layout energy consumption per (clock/handshake) cycle. The energy consumption was derived from a sim-

ulation of the switching activity of a testcase. The testcase considers 70% link utilization. For the designs using the gating mechanism, different percentages of utilization (0, 70, 100) were considered, to assess the effect of gating. The results show that the asynchronous designs without the gating scheme (4ph-bd, 2ph-bd) have a higher energy consumption compared to the mesochronous design, due to the additional switching activity for handshaking. However, the gated designs (2ph-bd-g, Argo) show energy gains depending on the utilization of links. The 2ph-bd/LEDR-g shows a slightly higher energy consumption, due to the dual-rail encoding.

6.7. Discussion

The router presented in this chapter combines the router functionality with inherent synchronization capabilities. The results presented show that the area and frequency of the Argo router is comparable to a synchronous router, offering elastic FIFO behavior in the same design. This elastic behavior can be compared to a mesochronous router. However, the overhead of the synchronization FIFOs required in the mesochronous design is significant. This overhead amounts to additional area that can be three times that of the synchronous or the Argo router. The Argo router can be used to replace the synchronous router and the FIFOs in its five ports, offering the same, and possibly more, elasticity in a design that is three times smaller, with similar speed, and with similar energy on the same link utilization. The energy consumption of the Argo router can be further improved at low link utilization with the gating scheme. The FIFO overhead can be reduced, but even with optimized FIFOs the overhead is considerable.

The proposed Argo router is more efficient than other asynchronous designs. It is faster due to the two-phase protocol and the simple implementation of the controller with only standard CMOS library cells. It is slightly larger than the other asynchronous designs, due to its latch-based controller. However, the Argo router is half the size of the 2ph-bd/LEDR-g design, due to the delay insensitive links of the latter. The delay insensitive links offer a trade-off of area and robust design. Furthermore, the Argo router consumes slightly less energy, also due to the simple standard cell implementation of the controller.

Another reflection upon the design and implementation process of the presented designs is related to the effort required. The asynchronous routers have local timing assumptions that need to hold for correct operation. They need to be defined at design time and verified in every step of the process. This requires a design cycle of implementation, timing analysis, and adjustment of timing constraints until timing closure is achieved. In this regard the Argo router was shown to be easier to implement than the alternative asynchronous designs. The implementation is straightforward, as the handshake controllers do not have additional timing constraints that need to be adjusted by the designer, as in the 2ph-bd design, or special asynchronous cells (C-elements). The asynchronous elements that still exist in the Argo router are the delay elements following the bundled-data encoding and the C-elements in the crossbar that are required for the distributed synchronization.

An aspect of the asynchronous design process that was touched on in this chapter is the delay element margin required in the bundled-data encoding for a safe but optimized design. A small additional margin is optimal but not safe as it may not cover high delay fluctuations. A bigger additional margin is safe but it adds delay in the the handshake cycle. The amount of

margin added is not explicitly studied in the literature. However, the amount used in this thesis project agrees with the numbers mentioned in the literature.

7
Implementation Results

This Chapter presents implementation results of the entire NOC in ASIC and FPGA technologies. The Argo NOC is an NxN NOC connected in a bi-torus topology. It consists of N^2 NIs as presented in Chapter 5, and N^2 asynchronous Argo routers as presented in Chapter 6. Alternatively, a synchronous version of Argo is also possible. Such a version uses synchronous routers with the same pipeline as the asynchronous Argo router. This chapter lists the possible timing organizations of a platform, using either the asynchronous Argo or the synchronous equivalent. It describes what each version comprises and the simulation environment used to verify both versions. It also presents the different implementation scenarios explored, both for the proposed asynchronous Argo NOC and for the synchronous equivalent, and results for each scenario. The ASIC implementations involve synthesis in a 65 nm CMOS technology library of STMicroelectronics, and layout. The results are in terms of area, maximum frequency and energy consumption, as well as verification of correct operation. The FPGA implementation results are in a Xilinx ML605 and on an Altera DE2-115 FPGA boards, and involve area resources utilization, frequency, and verification of correct operation.

The content of this chapter has appeared in the following publications:

- "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation", with authors Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø [71]

- "Synthesis and Layout of an Asynchronous Network-on-Chip using Standard EDA Tools", with authors Christoph MÏller, Evangelia Kasapaki, Rasmus Bo Sørensen, and Jens Sparsø [106]

- "D3.4 - Report documenting the hardware implementation of the self-timed NOC", T-CREST Project Deliverable [126]

- "D3.5 - Report on Impact of Asynchronicity on Predictability of the NOC", T-CREST Project Deliverable [127]
- "D3.8 - Integration report of the full system implemented in an FPGA", T-CREST Project Deliverable [129].

7.1. Platform Timing Organizations

The proposed timing organization of Argo is an asynchronous implementation of the routers, mesochronous NIs and individually clocked processors, presented in Section 4.4. This is a globally-synchronous locally-asynchronous (GALS) implementation and is the most flexible and robust to timing variability. A range of simpler timing organizations are also possible, using either the asynchronous or synchronous versions of the routers. We implemented the Argo in different timing organizations and integrated it in a number of multi-processor platforms with the following organizations:

1. A GALS organization where each processor operates with its own independent clock, all the NIs are clocked by the same clock, possibly with some bounded skew (mesochronous), and the routers are implemented as asynchronous circuits. This allows independent frequency scaling in the processors and the use of different processors with different clock frequencies. In addition, it offers tolerance towards clock skew at the global level.
2. A globally multi-synchronous implementation. Each processing core uses the same clock as the NI to which it is attached, and the routers are implemented as asynchronous circuits. This implementation tolerates a bounded amount of skew between processor-NI pairs.
3. An implementation where all the processors and NIs operate synchronously and the routers are implemented as asynchronous circuits. This implementation is relevant for testing purposes.
4. A globally-synchronous implementation where the same global clock is used in all the processors as well as in all the routers and NIs. This implementation offers no flexibility but it is relevant for FPGA prototyping, for software development, and perhaps for implementation of small systems

7.2. Argo NOC Implementation

The above timing organizations use the asynchronous Argo router or the equivalent synchronous router and the clocked Argo NI.

A 2x3 instance of the proposed asynchronous Argo NOC is shown in Figure 7.1. As we see in the figure, Argo can be regarded as a mesh of synchronization points, i.e. NIs and JF points, connected by asynchronous FIFO pipelines. These points enforce synchronization of the operation either with a clock (at the NIs), or synchronization of all input pipelines (at JF points). The asynchronous FIFOs consist of the router pipeline stages. The design also

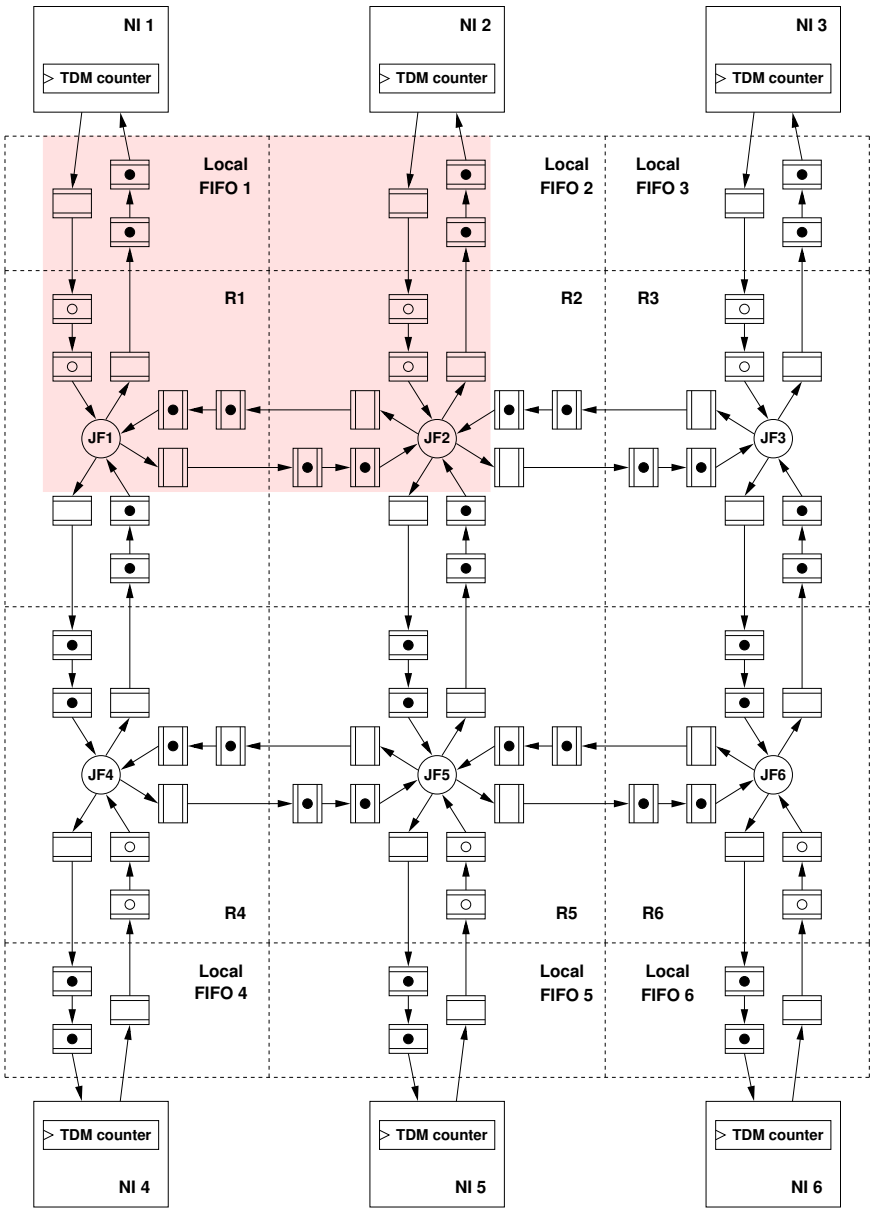


Figure 7.1.: A 2x3 Argo NOC viewed as a structure of FIFOs, [73].

includes a number of local FIFO stages, *Local FIFOs*, connecting the NIs and routers. In particular, the proposed Argo NOC design and the implemented instances include one FIFO stage in the channel from NI to router and two FIFO stages in the channel from router to NI. This design choice creates a balanced design with three stages of pipeline between any pair of synchronization points, i.e. JFs or NIs. The choice of three stages in the router pipeline initialized with two tokens aims for maximum performance and minimum hardware resources, as discussed in Section 6.2.2. For the same reason we maintain the three-stage pipeline between NIs and JF points in the router. Alternatives that maintain maximum performance are five stages with three tokens or six stages with four tokens. These options would increase the elastic capabilities of the NOC at the cost of a hardware increase.

The equivalent synchronous version consists of routers with a three-stage register pipeline, the same as the asynchronous Argo router. The design does not offer any flexibility and it does not include additional FIFO stages in the connection between NIs and routers. However, it is useful for testing purposes, as it is simpler to test in the environment of an entire platform. It is also easier for FPGA prototyping and for software development purposes.

7.3. Argo Simulation

To simulate the behavior of the Argo design, we developed a parametrized RTL-level VHDL description of the Argo NOC. It is a tile-based description of NOC nodes, where each node consists of an NI and a router, in its synchronous or asynchronous version. Additionally, we developed a behavioral VHDL description of a processor and an SPM. The dimensions of the NOC are configured as parameters in a VHDL package. Parameters that can be configured are the dimension N of the network, the length of the slot period, the number of DMA controllers etc. For the simulation we used ModelSim.

The contents of the slot table and the DMA table are programmed by the processor. Each entry in the slot table and the DMA table is mapped into the processor address map. The processor configures the slot table and programs the DMA controllers through the OCP interface as a series of OCP writes to a specified address in the address space. The address space mapping is defined in a VHDL package description. The slot table corresponds to a continuous address space as a block-aligned array of words. The route field of the DMA controllers is also placed in a separate contiguous block-aligned array of words. The rest of the configuration of the DMA controllers is placed in a different two-word block-aligned array. The slot table and the route field of the DMAs are accessed only during initialization, while the rest of the DMAs are accessed in normal operation.

For testing the design, we developed a testbench environment. This environment includes behavioral descriptions of the processors and SPMs. The SPMs are described as dual-ported RAM memories. The processors are described as test vector generators that feed test vectors into to the NI design and the SPM. We used ModelSim and a number of testcases to verify the correct operation of both versions during different steps of the design process, such as post-synthesis and post-layout.

7.4. Implementation Scenarios

Both the asynchronous and the synchronous versions of the Argo NOC have been implemented and have been integrated into bigger platforms, e.g., the T-CREST platform in various implementation scenarios:

1. An ASIC implementation of the Argo NOC using the asynchronous routers, synthesized in a 65nm CMOS standard-cell library. This implementation was verified in post-synthesis and post-layout simulation. The results of this implementation are reported in Section 7.5.

2. An FPGA implementation of the Argo NOC using the asynchronous routers, implemented and tested in a Xilinx ML605 FPGA board. This version was verified by post-layout simulation in ModelSim using the technology libraries of the board. Using this implementation, we verified the asynchronous Argo NOC design in an FPGA and we demonstrated its ability to tolerate skew. The results of this implementation are reported in Section 7.6.

3. An FPGA implementation of the Argo NOC using the synchronous routers, implemented and tested on an Altera DE2-115 FPGA board. This version was integrated into an open-source platform including the Argo NOC and a number of Patmos processors. This platform was used for testing purposes. The results of this implementation are shown in Section 7.7.

4. An FPGA implementation of the Argo NOC using the synchronous routers, implemented and tested in a Xilinx ML605 FPGA board. This version was integrated into the final T-CREST platform including the Argo NOC, a number of Patmos processors, a proprietary memory controller, and the bluetree memory-NOC developed within T-CREST project. Implementation results of this platform are shown in Section 7.8.

7.5. Argo ASIC Implementation

This section describes in detail the implementation of the asynchronous Argo NOC in ASIC flow (item 1 in Section 7.4). Implementation results of the components of the NOC, i.e. NIs, routers, are presented in Sections 5.2 and 6.6, respectively, for individual evaluation. In this section, results are presented for two complete Argo NOC instances, a 2x2 and a 4x4 instance. We have synthesized the integrated Argo NOC in a 65nm standard cell CMOS technology library from STMicroelectronics and produced a layout. We verified the correct functionality of the designs in post-synthesis and post-layout stages of the design process by simulation. This section reports results in terms of area, maximum frequency and energy consumption. The tools we used in the design process are ModelSim for simulation, Synopsys Design Compiler for synthesis, Cadence SOC Encounter for layout, and Synopsys Prime Time for power analysis.

7.5.1. Synthesis

Initially, we synthesized a 2x2 instance of Argo in a 65nm gate-netlist. This instance includes 4 routers, 4 NIs, and 3 FIFO stages between NIs and JF points, as explained in Section 7.2. Each NI includes 4 DMA controllers and uses a schedule with 8 time-slots. The NOC has a total area of 81388 μm^2. This gate-netlist was tested for correct operation with the test environment presented in Section 7.3. As an asynchronous network the frequency was evaluated throuhg post-synthesis simulation, which shows that it reaches a maximum frequency of 929 MHz. Moreover, we tested the 2x2 gate-netlist under skew between NIs. The testcase enforced skew between two NIs and the simulation showed that the netlist can absorb 0.94 cycles of skew when running at maximum frequency and 2.85 cycles of skew when running at 500MHz. The skew tolerance will be further discussed in Chapter 8.

	Routers	NIs	Links	FIFOs	total
Synthesis Cell Area					
total (μm^2)	120931	486321	13578	12738	633568
per node (μm^2)	7558	30395	849	796	39598
relative (%)	19.09	76.76	2.14	2.01	100
Layout Cell Area					
total (μm^2)	127446	537397	43289	12613	720745
per node (μm^2)	7965	33587	2706	788	45047
relative (%)	17.68	74.56	6.01	1.75	100
Layout energy					
total (pJ/cyc)	96.40	430.80	363.20	13.41	903.81
per node (pJ/cyc)	6.03	26.93	22.70	0.84	56.49
relative (%)	10.67	47.66	40.19	1.48	100

Table 7.1.: Synthesis and post layout area breakdown and post layout energy distribution for the 4x4 Argo.

To evaluate bigger structures and further explore the Argo NOC design we implemented a 4x4 instance. The results of this implementation were obtained with the help of Christoph Müller. This instance of Argo includes 16 routers, 16 NIs and 3 local FIFO stages between NIs and JF points. In addition, it includes one pipeline stage in the links to pipeline the link delay. The NIs include 16 DMA controllers and a schedule of 23 time-slots, as it was generated by the scheduler for an all-to-all communication pattern with equal bandwidth requirements.

The first section of Table 7.1 shows cell area results for the synthesized netlist and a breakdown into routers, NIs, links, and local FIFOs. The total cell area of the post synthesis netlist is 633568 μm^2. As Table 7.1 shows, the contribution to the cell area from the routers is very small, 19%, confirming the expectation of very lightweight routers from the TDM scheme. The NIs contribute 77% in the total area, a figure that is due to the tables in the NIs. The local FIFO pipeline stages contribute minimally, approximately 2%. Links also contribute 2% from buffers required to drive the links.

7.5.2. Layout

To take into account the wireload effects of long wires after placement and routing, we produced a layout of the 4x4 instance of Argo. This was done with the help of Christoph Müller. The layout is a tile-based layout that comprises 16 tiles. Each tile includes one Argo router and one NI. Empty space in the tile is dedicated to one processor and SPM memories. Figure 7.2 shows the layout of the 4x4 instance. Each tile occupies an area of 1.5mm x 1.5mm, of which the tile itself is defined to cover 1.9 mm^2, which is considered sufficient to account for realistic link lengths, and the rest of the space is available for link channels. Each link is pipelined with one stage. The bi-torus topology of the NOC has been folded to even out link lengths. The tiles are placed with a flipping scheme to reduce congestion in the link channels.

The second section of Table 7.1 shows cell area for the post layout netlist. The total area of the post layout netlist is 720745 μm^2. The difference of the post layout area compared to the

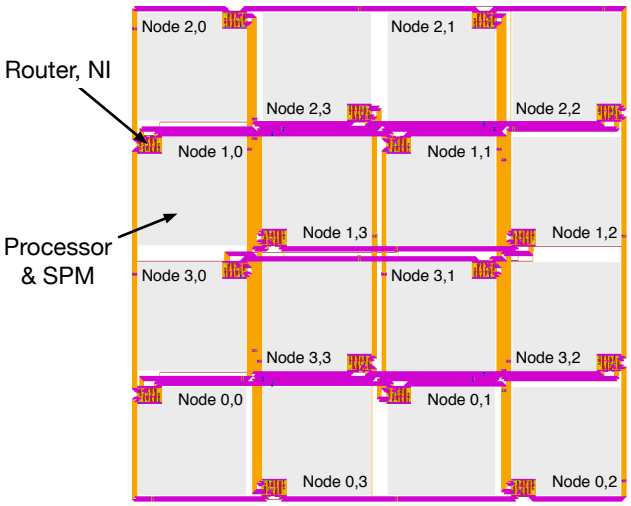


Figure 7.2.: Layout of the 4x4 Argo NOC with a single pipeline latch added to each link, from [106].

post synthesis cell area (633568 μm^2) is due to buffers introduced during layout from the tool to drive the links and for the clock tree. The same trends in cell area appear for the post layout netlist as for the post synthesis netlist. The routers are very small contributing approximately 18% in the total area. The NIs contribute 75%, the local FIFOs 2% and the links 6%

The last section of Table 7.1 shows the energy consumption per cycle. This is based on the switching activity of a simulation testcase and is independent of the operating frequency. The testcase produces link activity of 22% with 78% of the links being idle on average. As shown in the table, a large contribution to the energy comes from the NIs (48%), and an equally large contribution from the links (40%). A much smaller contribution comes from the routers (11%), due to the gating mechanism. The local FIFOs contribute 1.5%.

The maximum frequency of the post layout 4x4 Argo instance was measured through simulation to be 450 Mhz. In principle, the maximum frequency is defined by the maximum handshake frequency of the asynchronous routers, which was measured to be 746 MHz, as shown in Table 6.2. The performance reduction is due to changes in environment conditions, as the results of Table 6.2 were derived for a single router connected to an environment of eager producers and consumers. The measured frequency for the 4x4 NOC, considers a realistic environment of routers and NIs and has a larger delay on the links.

7.6. Argo FPGA Implementation

A 2x2 instance of Argo with the asynchronous Argo routers was implemented in FPGA technology (item 2 in Section 7.4). The results of this section were obtained with the help of Ioannis Kotleas. The FPGA board used was a Xilinx ML605 board and Xilinx ISE was used for synthesis and layout.

For the FPGA implementation of the asynchronous Argo router, the asynchronous components need special care. This applies to the C-elements in the crossbar stage of the router and

	Self-timed NOC				Synchronous
	2x2 NOC	NI	FIFOs	Router	Router
Number of Registers	5,440	672	108	580	545
used as Flip Flops	2,688	672	0	0	545
used as Latches	2,752	0	108	580	0
Number of LUTs	3,985	438	19	538	423
used as logic	3,985	438	19	538	423
used as Memory	0	0	0	0	0

Table 7.2.: Resource utilization of the 2x2 asynchronous Argo NOC on the Xilinx ML605 FPGA board.

	3x3 platform	SPM	Patmos	NI	Router
Number of registers	43599	4	4107	277	565
Number of LUTs	48799	9	4765	277	301
Block RAMs	407	8	35	4	0

Table 7.3.: Resource utilization of the 3x3 open-source integrated platform using the synchronous NOC on the Altera DE2-115 FPGA board.

the delay elements. The C-elements were implemented explicitly as a LUT function and the delay elements as an array of LUTs. To set the right delay for the delay elements, an iterative process is required for implementing and adjusting the delay elements. More details can be found in [128].

Table 7.2 presents the resource usage of the 2x2 Argo instance implemented in the Xilinx ML605 FPGA board. The second column shows the area resources for the entire 2x2 asynchronous Argo NOC. Columns three to five show the breakdown of one node into NI, FIFOs and the asynchronous Argo router. The last column shows the area resource of a single synchronous router that has been implemented on the specific FPGA for comparison.

As expected, the asynchronous router is a latch-based design and the synchronous router is a flip-flop based design. As shown in the table, the area resources for an asynchronous router are slightly bigger but comparable to the synchronous one. This is mainly due to implementation of the delay elements. The table also shows that the NI tables are implemented as flip-flops, as their size is small.

The operation of the implemented netlist was verified by post place and route simulation in ModelSim using the technology libraries of the board, and on the FPGA board. Although clock skew cannot be introduced in the FPGA board, we introduced reset skew by delaying the reset signal of specific NIs by one clock cycle. Simulation and running on the FPGA board verified that the implemented design operates correctly under skew.

7.7. Open-Source Platform with the Synchronous NOC

In the context of the T-CREST project, an open-source version of the platform was developed (item 3 in Section 7.4). The platform is a 3x3 instance and consists of 9 Patmos processors,

	3x3 platform	SPM	Patmos	NI	Router
Number of registers	58714	110	4830	1627	565
Number of LUTs	65327	33	6153	810	429
Block RAMs	310	8	26	0	0

Table 7.4.: Resource utilization of the 3x3 final T-CREST platform using the synchronous NOC on the Xilinx ML605 FPGA board.

the Argo NOC with synchronous routers, an off-chip memory, and a memory controller. This platform was implemented in the Altera DE2-115 FPGA board. The implementation of this platform is a combined contribution of the T-CREST group.

The resource utilization of the platform is shown in Table 7.3. The second column shows the area resources of the entire 3x3 platform. Columns three to six show a breakdown of one node of the platform into the SPM, Patmos processor, NI, and the synchronous version of the router. The tables in the NI are implemented as block RAM memories instead of registers. The numbers in the table show the area efficiency of the NOC. It can be seen that for a single node of the platform 89% of the LUTs of the node are used for the Patmos processor, while 5% are used for the NI and 6% for the router. Considering that NIs presented in the literature have a similar area to processors, as shown in Section 5.3, the contribution of the Argo NOC to the entire platform is very small.

This platform runs at a frequency of 80 MHz and the critical path is not in the NOC.

7.8. T-CREST Platform with the Synchronous NOC

The final T-CREST platform, which is used for evaluation by the industry partners, (item 4 in Section 7.4) consists of a number of Patmos processors, connected by the Argo NOC to synchronous routers. Access to shared memory is provided through the memory tree NOC and is controlled by the memory controller, both developed within the T-CREST project. The platform is implemented and runs on the Xilinx ML605 FPGA board. The implementation of this platform is a combined contribution of the T-CREST group.

Resource utilization results of a 3x3 implementation of the final T-CREST integrated platform are shown in Table 7.4. The second column shows the area resources of the entire 3x3 platform. Columns three to six show a breakdown of one node of the platform into the SPM, Patmos processor, NI, and the synchronous version of the router. The numbers are in the same range as the ones shown in Table 7.3 for the open-source platform presented in Section 7.7. The resources of the T-CREST platform shown in Table 7.4 are slightly bigger than the ones of the open-source platform shown in Table 7.3 due to the different memory controller and memory tree they use. The difference in the NI register resources is because the tables in the NIs are implemented as registers in the T-CREST Xilinx platform instead of block RAM memories as in open-source Altera platform.

This platform runs on a frequency of 75 MHz and the critical path is not in the Argo NOC.

7.9. Use of Argo NOC

In the context of the T-CREST project and to facilitate the use of the Argo NOC, a message passing library has been developed by Rasmus Bo Sørensen. It offers the application programmer a way to handle communication over the Argo NOC, while abstracting the hardware details. It is a C library and provides functions such as *"send"*, *"receive"*, and *"acknowledge"*, for sending, receiving, and acknowledging the reception of a message, respectively. The functions take as arguments the message size, the pointer address in the local SPM and the pointer address in the receiving SPM.

To provide guidelines on the use of Argo NOC we developed the Argo Programming Guide [72]. This describes how an open-source T-CREST platform can be implemented in an FPGA and how the Argo NOC can be used by an application programmer to implement message passing functionality. The guide also provides examples of simple parallel application with the use of the message passing library. The T-CREST platform was used and evaluated by industry partners within the T-CREST project on a number of avionics and railway applications.

Accessing the Source Code

The source code or the Argo NOC is provided through the T-CREST GitHub repository. The Argo repository includes the Argo VHDL source files, the test environment, and scripts for the simulation and the implementation, both in ASIC and FPGA, of Argo. The repository can be accessed via the `git` link:

`https://github.com/t-crest/argo`

7.10. Discussion

As shown in Chapter 5 the Argo NI design is extremely small. Alternative NIs in the literature are as big as a processor and NI designs for similar TDM NOCs were shown to be four to five times bigger than Argo. Similarly, as shown in Chapter 6, the Argo router is very small, as follows from the TDM scheme, and efficient. Equivalent mesochronous TDM router designs are two to three times bigger than the Argo router. Overall, the entire NOC, as shown in this chapter, results in a very efficient implementation in terms of area.

Comparisons among NOCs are not easy to reach due to the different logical dimensions, features and requirements implemented by the NOCs. Argo NOC is a virtual circuit switching TDM NOC that uses source routing. Aelite is a TDM NOC that follows the same principles and is comparable to Argo. They both offer hard real-time guarantees via the TDM scheme. They also operate mesochronously at their end-points (NIs), yet Argo offers bigger amounts of elasticity as will be shown in Chapter 8. Implementation results for aelite [59] present an instance that includes 6 routers, 11 NIs and 45 bi-directional channels. This instance was implemented in ASIC design flow, in 65nm technology. The area of this instance after synthesis was calculated to be 2.5 mm^2. As shown in Section 7.5, a 4x4 instance of Argo synthesized in 65nm technology has an area of 0.63 mm^2. This Argo instance includes 16 routers, 16 NIs, 3 FIFO stages between NIs and JF points, and 240 uni-directional channels. The two architectures are not directly comparable. For example, Argo contains the DMAs in the NIs

while aelite contains them in the processors. In addition, the logical dimensions are not exactly the same and the communication load differs. However, the Argo NOC is roughly four times smaller for slightly bigger dimensions. We can conclude that in general Argo is at least four times smaller for instances of equal logical dimensions. In terms of performance, the post synthesis maximum frequency achieved by the two architectures is similar.

The Argo NOC has been used in the T-CREST multi-processor platform in a number of implementations. The T-CREST project produced various versions of the platform with synchronous and asynchronous versions of the Argo NOC. The T-CREST platform has been implemented in instances of up to 4x4 nodes. Software support with a message passing library was also generated within the T-CREST project group, and application programming documentation is also provided. The platform was used and evaluated in a number of industrial applications from the industrial partners of the T-CREST project and was shown to be easy to use.

Part III.

Elasticity Analysis

8
Theoretical Analysis

The timing organization of Argo has been briefly discussed in Section 4.4. This chapter analyzes in detail the timing organization and behavior of the Argo design. Furthermore, it explores the elastic behavior of the asynchronous network of routers and the parameters that affect it. Along these lines it presents a qualitative analysis of the elastic behavior of Argo. Material from this chapter has been published in:

- "Argo: A Time-Elastic Time-Division-Multiplexed NOC Using Asynchronous Routers", with authors Evangelia Kasapaki and Jens Sparsø [73]
- "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation", with authors Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø [71]

8.1. Argo Timing Organization

As explained in Section 4.4, Argo is organized in three timing layers. As Figure 8.1 shows, in the core of Argo a network of asynchronous routers operates in an elastic, self-timed way. At the endpoints of this elastic structure, the mesochronous NIs operate at a common rate, with some bounded phase difference, i.e., skew. The NIs enforce the TDM schedule and the NI clock is the point of reference for global synchronization. The overall system supports a GALS organization; the processors connected at the endpoints of Argo may operate at independent clock rates.

The enforcement of global synchronization for TDM relies on the use of TDM counters in the NIs. All TDM counters are reset with a global reset signal and incremented by a global NI clock signal. Thus the *TDM skew* is defined as the skew among the TDM counters. The TDM skew is a combination of the skew of two signals, the reset and the clock, and may

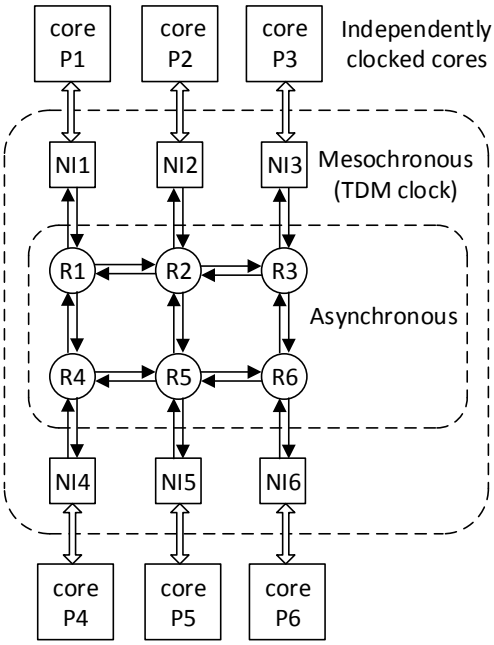


Figure 8.1.: The Argo timing organization in three timing layers, from [73].

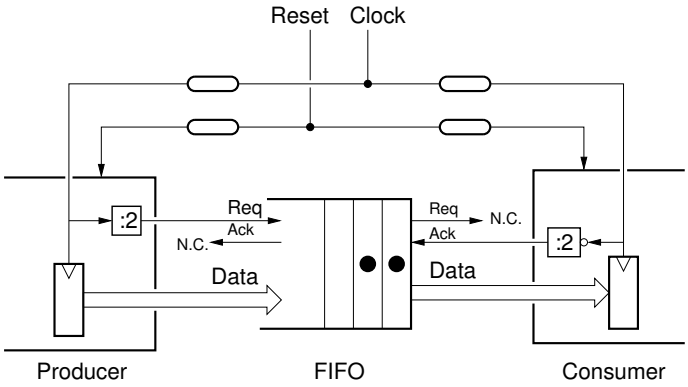


Figure 8.2.: Asynchronous ripple FIFO connected to a clocked producer and a clocked consumer [71].

result in a phase difference in the TDM schedule of more than one clock cycle. For the global synchronization, and thereby for the TDM scheduling to operate correctly, the elastic network of routers should be able to absorb the TDM skew that is present. A goal of this work is to explore the capabilities of the Argo NOC to absorb the TDM skew. In the following we simply use "skew" to refer to the TDM skew.

Asynchronous pipelines inherently operate as ripple FIFOs. Figure 8.2 shows a FIFO connected to a clocked producer and a clocked consumer. This is a simplified view of the elastic network connected to clocked NIs as producers and consumers that enforce the data flow and its rate. Argo relies on the timing assumption that the asynchronous routers are capable of handshaking faster than the NI clock frequency. The producer's clock generates a request sig-

nal towards the FIFO and a data token to be stored in the FIFO. If there is space available in the FIFO for a token and the handshake cycle is faster than the NI clock period, the acknowledge signal from the FIFO can be safely ignored without the possibility of metastability. The consumer's clock generates an acknowledge signal to the FIFO and consumes a data token from the FIFO. If there are data tokens in the FIFO and the handshake cycle is faster than the NI clock period, the request signal from the FIFO can also be safely ignored. Thus, with a half-filled FIFO, there is available free space in the input of the FIFO for tokens from the producer, and available data tokens in the output of the FIFO, for the consumer. The above half-filled FIFO connected to a producer and consumer with a slower clock rate than the handshaking capabilities of the FIFO is safe from metastability. In the same way, a system with half-filled asynchronous router pipelines connected to NIs with a slower clock rate than the router handshake capabilities is safe.

The network of asynchronous routers maintains the data token flow, such that at its endpoints the token flow complies with the TDM schedule. This is assured by two principles: Firstly, the propagation of data tokens between neighboring stages is controlled locally with asynchronous handshaking, and secondly, the global synchronization of different data flows is controlled in a distributed way. The distributed synchronization is done by the join-fork (JF) functionality of the crossbar. Between JF points, the router pipeline behaves as an elastic FIFO.

8.1.1. Argo as a Structure of FIFOs

To analyze Argo and evaluate its elastic properties, it can be viewed and modeled as a mesh of elastic FIFOs. Ignoring the combinational logic of the pipeline stages of the routers and considering only the latches, the routers can be viewed as an asynchronous three-stage FIFO. The Argo design of Figure 8.1 can be represented in an abstract way as illustrated in Figure 8.3, which shows a 2x3 instance of Argo consisting of six routers and six NIs. The structure of a router with the three pipeline stages and the JF point of synchronization at the crossbar can be recognized in the figure. The figure also shows the additional FIFO stages, i.e. *Local FIFOs*, placed between the NIs and the routers. In the proposed Argo design we chose one pipeline stage for the input channel towards the network of routers, and two pipeline stages for the output channel from the network of routers, as explained in Section 6.2.2. The black dots show the two tokens which each router pipeline is initialized with. The white dots in the NI to router channels are the void tokens injected by the NIs after reset, to initialize the operation of the network of routers. The initialization process has been described in Section 4.6.

The focus of this work was also to evaluate the elasticity of Argo and its performance, taking into consideration the Argo design and implementation presented in previous chapters. The elasticity is in other words the amount of TDM skew which the elastic network of routers is able to absorb. As the next section shows, the skew tolerance of a FIFO is related to the clock frequency applied at its endpoints. The goal is to evaluate the relation between skew and operating frequency.

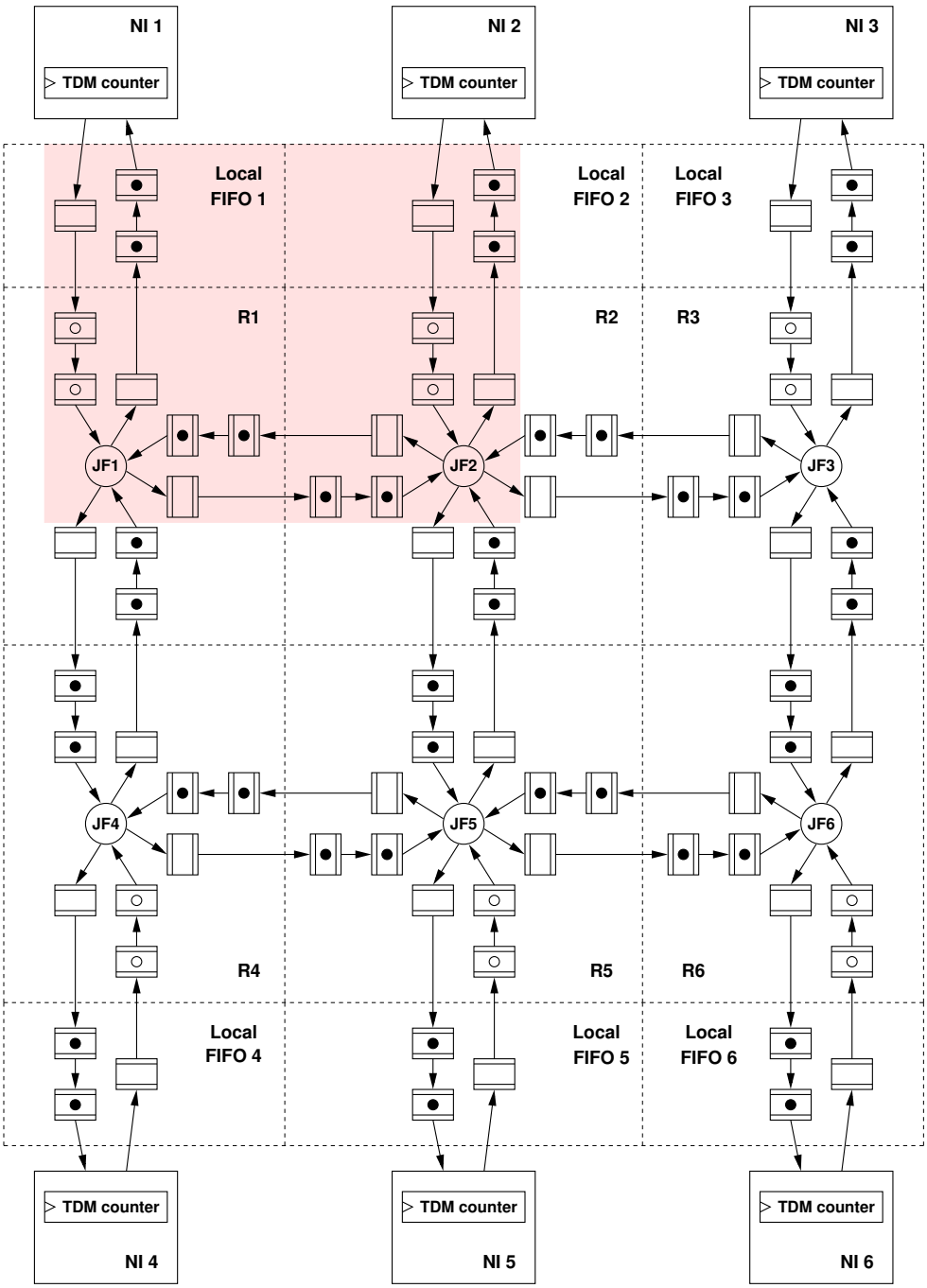


Figure 8.3.: A 2x3 Argo NOC as a structure of FIFOs, from [73].

8.2. Qualitative Analysis

This section presents a qualitative analysis based on theory, in order to build up an understanding of the elastic behavior of Argo and evaluate its properties and limits.

8.2.1. Performance and Elasticity of Asynchronous FIFOs

The behavior of an asynchronous pipeline is elastic and resembles a self-synchronizing FIFO like the one shown in Figure 8.2. The parameters affecting the performance of the pipeline are the design, the initialization of the design, the implementation technology, the rate applied at its endpoints by the producer and consumer, and the potential presence of skew. This section reviews these parameters and how they relate to the Argo pipeline.

The design refers to the number of pipeline stages, the handshake protocol among stages, etc. The Argo design has been presented in detail in the previous chapters. The asynchronous Argo router pipeline, as seen in Chapter 6, consists of three pipeline stages. JF points in the crossbar synchronize the router pipelines, allowing a pipeline of three stages between every pair of JF points to be time-elastic, resembling an elastic three-stage FIFO. Additional FIFO stages can be added between NIs and routers to improve the performance of the FIFO and allow more space for elasticity. This is a trade-off between hardware resources and elasticity.

Another parameter affecting the performance is the implementation technology, which determines the gate delays of the circuit and therefore the handshake cycle delays. Timing parameters that determine the cycle timing of an asynchronous handshake are the forward and reverse latency of the pipeline, L_f and L_r, respectively [151]. These parameters determine the inherent minimum handshake cycle of the FIFO. For an implementation of the Argo router in 65nm CMOS technology, these parameters are estimated from gate delays to be $L_f \simeq 420\,ps$ and $L_r \simeq 190\,ps$. For a two-phase handshake protocol the minimum handshake period for this implementation is $P = L_f + L_r = 610\,ps$.

The state of the FIFO refers to the number of tokens and bubbles in the FIFO; the initialisation state is the state which the FIFO is initialized with. The pipeline reaches maximum performance when the number of latch stages per token matches the dynamic wavelength of the design [151], $W_d = P/L_f \simeq 1.45$. The Argo router is therefore initialized with two tokens and one bubble for three stages to satisfy the dynamic wavelength, while keeping the hardware cost low.

The clock rate applied at the endpoints of the FIFO determines the data flow rate. If the applied rate is slower than the inherent handshake rate of the FIFO, the FIFO will adjust to this rate. If the applied rate is faster, the FIFO will fail to satisfy this rate. This leads to the timing assumption for correct operation of Argo, that the inherent handshake rate of the asynchronous pipeline is faster than the NI clock frequency. Thus, the FIFO operates correctly for a range of periods between the inherent handshake period P and the applied clock period. The presence of skew between the producer and the consumer fills or drains the FIFO compared to its initial state, causing a slowdown. This reduces the range of periods in which the FIFO operates correctly.

Figure 8.4 shows the relation between the performance of an asynchronous FIFO, i.e. the handshake cycle, and the skew, i.e. the state of the FIFO. The x-axis represents the state of the FIFO, in other words the number of FIFO stages per token. A FIFO operating at its optimal performace is in a state where the number of FIFO stages per token matches the dynamic wavelength W_d. The y-axis represents the handshake cycle time of the asynchronous FIFO. In the state where the FIFO matches its dynamic wavelength, W_d, the FIFO operates at its minimum handshake period, T_{min}. Any change away from that state causes the handshake cycle time to increase, as the graph in Figure 8.4 indicates. The presence of skew creates a phase

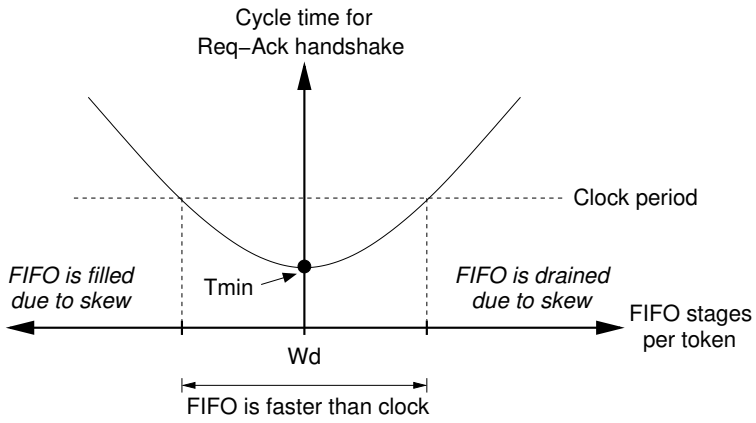


Figure 8.4.: Relation of skew and handshake cycle.

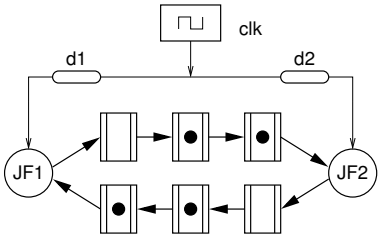
difference between the producer and the consumer connected at the endpoints of the FIFO. If the producer lags behind, the FIFO is drained, changing its initial state. This introduces a slowdown, as the number of tokens per latch stage no longer matches the dynamic wavelength. If the consumer lags behind, the FIFO is filled. This also introduces a slowdown, as the FIFO state moves away form the optimal state of matching the dynamic wavelength. Eventually, more skew leading to further slowdown of the FIFO leads to failure, as the FIFO becomes slower than the clock frequency. The horizontal dotted line in Figure 8.4 shows the operating clock period of the producers/consumers. The FIFO operates correctly below this limit, and fails above it. This line also bounds the region where the timing assumption for the Argo NOC is satisfied.

8.2.2. Performance and Elasticity of Asynchronous FIFO Rings

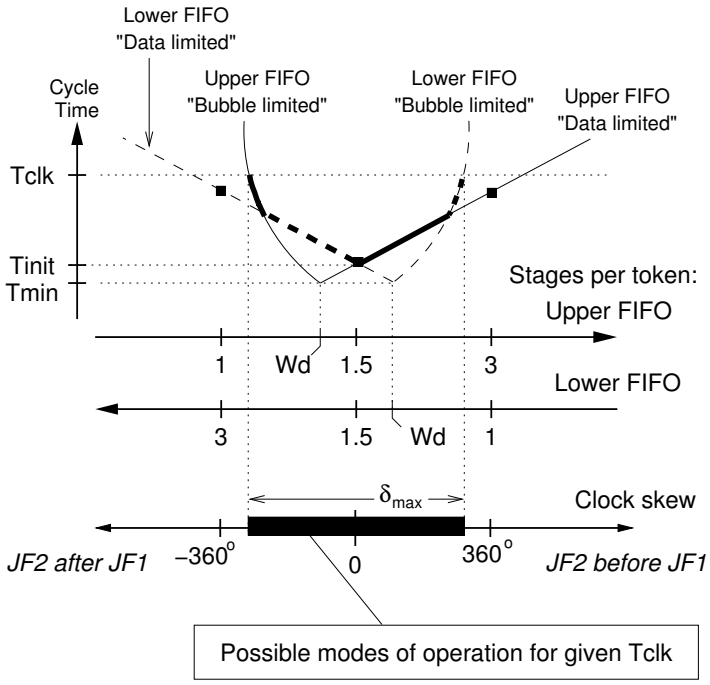
The abstract view of Argo as a structure of FIFOs includes several repeated ring structures. They are all balanced due to the balanced pipeline depth throughout the design, i.e. three-stage pipelines. Thus the FIFO ring that appears in Figure 8.5(a) can be recognized in many places in the Argo structure in Figure 8.3. This section gives an analysis of the timing and elastic properties of this FIFO ring as a smaller component of Argo in a first step towards providing an analysis of the bigger picture.

The ring shown in Figure 8.5(a) consists of two identical FIFOs. These represent the three-stage router pipeline, initialized with two tokens and one bubble and connecting two JF points, i.e. two points of synchronization. To evaluate the elastic properties of the FIFO ring, we enforce a fixed rate at the two JF points by connecting them to a clock. By skewing the clock by delays d1 and d2 for the JF1 and JF2 points, respectively, we explore the behavior of this ring under skew.

Looking at the upper FIFO alone, the situation is similar to the one described in Section 8.2.1. The synchronization points can be seen as a producer, JF1, and a consumer, JF2, operating at the same rate but possibly with some skew. If the skew causes the consumer to lag behind the producer ($d1 = 0, d2 > 0$), the FIFO will be filled more than its initial state. If the skew causes the producer to lag behind the consumer ($d1 > 0, d2 = 0$), the FIFO will be drained compared to its initial state. In both cases there will be a slowdown in the speed of



(a) Ring with JF points connected by FIFOs.



(b) Analysis of the above ring under skew between the JF points.

Figure 8.5.: Two neighboring join-fork points connected by three-stage ripple FIFOs forming a ring and analysis of the system behavior under some amount of skew, from [73].

the FIFO. For the lower FIFO the behavior is analogous, with JF2 as a producer and JF1 as a consumer.

Figure 8.5(b) illustrates the performance of the two FIFOs and the combined performance of the ring. The two x-axes show the FIFO stages per token for the two opposite FIFOs of the ring and the y-axis shows the handshake period of the ring. The continuous graph shows the performance of the upper FIFO. The dashed graph shows the performance of the lower FIFO. The two FIFOs are identical except for their direction of operation, and their performance graphs are therefore vertically mirrored copies. The two graphs are aligned according to the FIFOs' initial state and for the case with no skew between JF1 and JF2. The 360° clock skew point corresponds to the FIFO containing one token per stage, i.e. being completely filled. The bold section of the two graphs is the combined performance of the ring formed by the two FIFOs. Optimal performance (T_{min}) is reached when the FIFO stages match the dynamic wavelength W_d, which for Argo is computed to be 1.45. Initializing the FIFO ring to the closest

integer number of latches (3) which will give this dynamic wavelength, leads to an inherent cycle time of T_{init} for the combined FIFO ring. The presence of skew changes the combined performance in one or the other direction, filling one FIFO and draining the other. One of the two effects dominates and therefore defines the performance. T_{clk} is the clock period applied at the endpoints. While the inherent cycle time of the FIFO ring is smaller than T_{clk}, the circuit operates correctly. If the skew increases beyond a certain threshold, one of the FIFOs (or perhaps both) will fail to cope with the rate dictated by the clock.

8.2.3. Skew Limits

The elasticity of Argo is translated into the question of how much skew can be absorbed by Argo, if it is still to operate correctly. Exact answers can be given for specific scenarios. An interesting question to answer is what are the skew bounds within which Argo is guaranteed to operate correctly. The aim of this section is to estimate worst-case bounds on the skew among any two endpoints, within which the circuit operates correctly.

To define worst-case bounds, we focus and analyze a subcircuit of Argo that is representative of the entire design. This subcircuit is illustrated by the shaded area in Figure 8.3. We denote the maximum skew between two JF points δ_{JF}. We provided an analysis for the relationship between skew and performance in Section 8.2.2. A similar analysis can be made for the skew between an NI and a JF point. We denote the maximum skew between an NI and its nearest JF point δ_{NI}. The subcircuit in the shaded area of Figure 8.3 consists of three ring segments. Therefore, the maximum possible skew that can be present between two neighboring NIs is:

$$\delta \leq 2\,\delta_{NI} + \delta_{JF} \tag{8.1}$$

For NIs that are further apart, the path between them contains more FIFO ring segments, i.e. more room for skew, and thus the maximum possible skew can potentially be higher. However, we are interested in the worst-case skew between any two NIs. That would involve the shortest path, i.e. the skew between two neighboring NIs. In the worst case, the ring segment between JF points may contribute negatively to the skew, as it is synchronized with the other neighboring JF points that it is connected to. Thus it can contribute δ_{JF} in the opposite direction. This results in the following worst case bound on the maximum skew between any pair of NIs in Argo:

$$\delta_{NOC} \leq 2\,\delta_{NI} - \delta_{JF} \tag{8.2}$$

This is a pessimistic but safe bound. Further analysis is required to evaluate the contribution of the ring segment connecting two JF points. A model analysis of the ring structure and of bigger structures is presented in Chapter 9.

8.3. Discussion

This chapter presented an analysis of the elastic timing behavior of Argo. Argo is based on a novel timing organization than is time-predictable at its endpoints (NIs) and elastic in its core (routers). This chapter analyzed the performance within this novel timing organization and

explored the limits of the elastic behavior in relation to the overall performance. We defined the problem of elasticity analysis and approached it in a bottom-up fashion.

Elasticity is defined as the maximum skew that can be absorbed by the core of Argo structure and is related to the frequency at its endpoints. Argo consists of specific repeated elastic structures, FIFO pipelines and FIFO rings. The performance of FIFOs and rings was examined and illustrated in graphs relating skew to frequency both for the general case and for the Argo structure. The skew limits were explored for structures that appear in the Argo NOC in relation to the skew of the elastic segments of the structure. This exploration produced equations defining the skew limits of Argo.

Theoretical analysis contributed to the intuitive understanding of the problem. A qualitative assessment of simple structures gave the trends and derived limits for the elasticity. Applying the theory to the Argo structure models verified the intuition. Simulation of the actual circuit also verified the analysis. More detailed analysis will be presented in the next chapter.

9
Model-based Analysis

Chapter 8 defined elasticity and presented a qualitative analysis of elasticity in the Argo NOC. This chapter quantifies the elasticity in Argo. The quantitative analysis uses timed graph models to describe the elastic behavior of Argo and an algorithim for the timing separation of events to analyze the timing behavior of Argo. Two different models are used, one on a coarse and one on a fine level of granularity.

Material from this chapter has been published in:

- "The Argo NOC: Combining TDM and GALS", with authors Evangelia Kasapaki and Jens Sparsø [1]
- "Argo: A Time-Elastic Time-Division-Multiplexed NOC Using Asynchronous Routers", with authors Evangelia Kasapaki and Jens Sparsø [73]
- "Argo: A Real-Time Network-on-Chip Architecture with an Efficient GALS Implementation", with authors Evangelia Kasapaki, Martin Schoeberl, Rasmus Bo Sørensen, Christoph Müller, Kees Goossens, and Jens Sparsø [71]

In Section 9.3, some new material from preliminary analysis is included that has not yet been published.

9.1. Elasticity Analysis Approach

The previous chapter derived worst-case bounds for the skew between any pair of NIs. This section aims at determining safe skew values for δ_{JF}, δ_{NI} and δ_{NOC}, and at determining the relationships among them and with respect to the period. For this reason, we use two timed graph models to describe the behavior and timing of the design and a timing analysis algorithm to analyze the models. In particular, the two models describe the behavior of the design, one

in a fine-grained way, aimed at defining skew values, and the other in a coarse-grained way, aimed at defining the relationships between them. One of the models is an extended version of the signal transition graph (STG) model [32], that defines the behavior of the design as a set of events and timing dependencies among them. The second model is a version of the full-buffer channel net (FBCN) model [14] that defines the behavior of the design based on its timing parameters, forward and reverse latency. They are both subclasses of the PetriNet models described in Section 2.6.6. We use timing separation of events (TSE) analysis [65] to evaluate worst-case performance as explained in Section 3.5. For the TSE analysis we use an automated TSE EDA tool [70] based on the algorithm of Hulgaard et al. [65] to evaluate the worst-case handshake time at the endpoints of the asynchronous structure, i.e. the NIs, and to verify that it is within the clock cycle time.

9.1.1. TSE Algorithm

TSE analysis evaluates the maximum timing separation between events in concurrent systems, like asynchronous circuits. In this work we use the TSE algorithm of Hulgaard et al. [65]. This algorithm considers a concurrent system described in a timed graph representation. Based on this model it evaluates a mathematical worst-case bound on the maximum timing separation between events.

TSE analysis [65] is applied to *process graphs (PGs)*. PGs extend Event-Rule systems [28] and Timed-STGs [20]. They are defined as follows:

Definition 9.1.1. a PG is a graph model $G = (E, R)$ where:

- $E = \{e_1, e_2, ...e_m\}$ is a finite set of events, the nodes of the graph,
- $R = \{r_1, r_2, ...r_m\}$ is a finite set of rules, the arcs of the graph,
- each arc is labeled with a delay range $[d_{min}, d_{max}]$ and an integer ε,
- the graph contains a unique event *root* that defines the initial state of the system.

The behavior of a concurrent system is determined by timing dependencies among events. Such systems have an initial state that represents the state of the system after reset. The execution of a concurrent system starts from its initial state, and displays varying behavior, until it reaches a point of steady operation. After this point, the system balances and displays steady and repeating timing behavior. The key idea of the TSE algorithm is to unfold the execution of the repetitive system starting from its initial behavior until the system reaches a steady state of operation. When the steady state is reached, the timing behavior of the system is maintained and repeated, and so further unfolding is not required. The TSE algorithm uses functional decomposition and applies concepts of (max,+) algebra [35] to analyze the infinite execution of the system and evaluate when the steady state is reached. The maximum time separation between a source and a target event appears when the source event happens as early as possible, and the target event as late as possible. This value is stable when the system reaches the steady state of operation.

The decision to use this algorithm is motivated by the fact that it evaluates worst-case performance and it also considers the initial behavior of the design, from reset to steady state of

operation. This is in contrast to algorithms used in the performance analysis of asynchronous designs as they focus on the steady state of operation [91]. This ignores the initial behavior of the system in the evaluation of the worst-case bound. The initial behavior is essential for verification of correct operation as it includes worst-case timing conditions occurring during the initial behavior.

In this thesis we use an automated TSE analysis tool [70] to verify the correct operation of Argo in the presence of skew. Argo is based on the assumption that the asynchronous routers are able to handshake faster than the NI clock frequency. If the worst-case timing separation between handshake signals is always within the NI clock period, the initial assumption is true and the correct operation of the design is verified. In addition, as described in Section 8.2.2, the operating clock period and the skew which a design is able to absorb are related. By changing the skew values, we determine the minimum period for which the design operates correctly and define the relationship between the skew and clock period.

In the following sections we describe the two models, STG and FBCN, used in this work to model Argo, and the specific models for various Argo parts or instances. For each model we explain how the TSE analysis is applied to the model and its results.

9.2. STG Model Analysis of Argo

This section focuses on the STG model and analysis of Argo. The STG model specifies the handshake signals of the control circuit and their behavior. It is a fine-grained description of the control part and gives the timing behavior of the control signals based on gate delays of the asynchronous pipeline. Because it includes information on gate delays, it takes more detailed information about the signal transitions into consideration.

9.2.1. STG Specification

STGs are an interpreted type of PNs [32]. In STGs the transitions are interpreted as signal transition events, i.e. rising/falling transitions, of control signals.

The PGs have similar semantic interpretations to Timed-STGs [20], where transitions are associated with timing constraints. The nodes of the graph represent transitions in the STG and events in the PG. The arcs that connect nodes represent places and rules, respectively. The initial marking of the STG is corresponds to the root event in a PG. They both include the concept of timing dependencies as timing constraints on the edges that lead to transition firings in the STG or on the rules of the PG. They also follow the same semantics for firing/occurrence of an transition/event [109], [20], [65].

In the analysis presented here, we use the STG specification of the system, annotated with timing constraints (Timed STGs) on the input arcs of transitions. For the TSE analysis we use the Timed STG models of Argo and extend them with the root event to indicate the initial behavior.

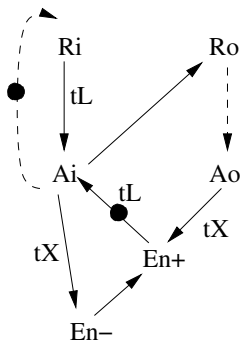


Figure 9.1.: Timed STG of the Mousetrap latch controller, from [73].

9.2.2. Argo STG Model

The STG description of Argo deals with its control part. For the subcircuit in the shaded area of Figure 8.3, we model the structure and timing behavior of the latch controllers, the JF points and the delay elements. The latch controllers are the basic building block of the design and the foundation of its timing behavior. As Argo uses the Mousetrap controllers, the analysis is based on the Mousetrap STG model. However, the analysis presented here can easily be adapted to another controller by replacing the STG model of the controller.

The STG model that describes the operation of the Mousetrap latch controller is shown in Figure 9.1. The nodes Ri, Ai and Ro, Ao represent transition events in the input and output handshake channels of the controller, respectively. $En+$ and $En-$ represent rising and falling transition events on the enable signal of the latch, putting the latch into its transparent and its opaque state respectively. The edges in the graph represent timing dependencies between signal events and they are associated with delays of the corresponding circuit components. In relation to the circuit of the latch controller in Figure 6.3, we can understand the delays noted on the edges of Figure 9.1 as follows: t_L represents the propagation delay of the latch and t_X the delay of the XNOR gate. Edges with no annotation refer to zero-delay dependencies due to concurrent occurrence of the events. For example, edge $Ai \rightarrow Ro$ indicates that events Ai and Ro are produced by the same signal. Dotted edges represent timing dependencies that model the environment.

The STG model of the latch controller can be used as a building block to build more complicated structures in a tile-based way. Figure 9.2 shows a three-stage pipeline of latch controllers connected to a clocked producer NI and a clocked consumer NI. The timing annotations on the dependencies are omitted for clarity. The figure also illustrates the interfaces of the NI and the pipeline, including the translation of the clock pulse to the asynchronous two-phase protocol. A positive transition of the producer NI clk generates a transition (positive/negative) of the Ri of the leftmost pipeline stage. Similarly, a positive transition of the consumer NI clk generates a transition (positive/negative) of the Ao of the rightmost pipeline stage. Possible delay elements placed between pipeline stages to match the combinational logic delay are modeled as timing annotations on the corresponding edges of the graph, i.e. t_D on $Ro \rightarrow Ri$. The clock is modeled as a repeating event (positive pulse of the clock) with the period of the clock, P. Skew is modeled as a delay dependency of the producer and consumer clk from the main clk, $d1$ and $d2$.

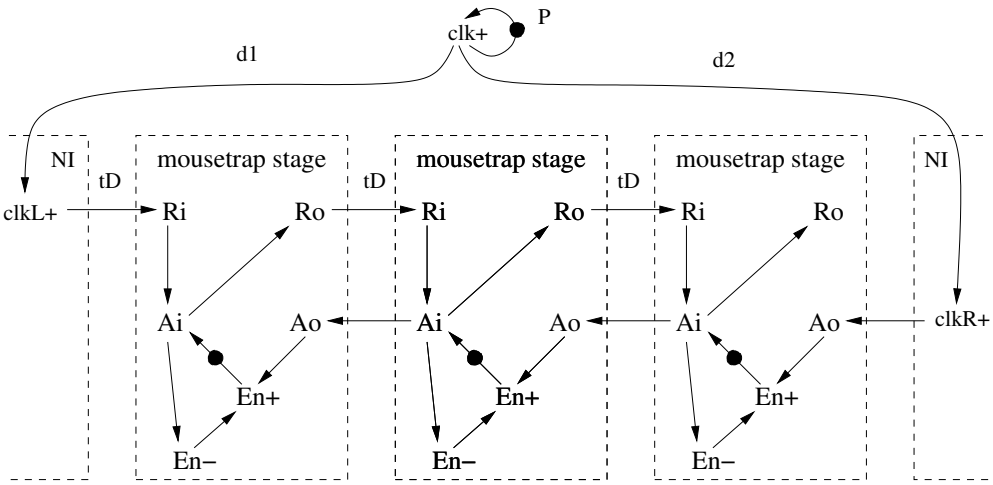


Figure 9.2.: Timed STG of a three-stage Mousetrap pipeline connected with clocked NIs.

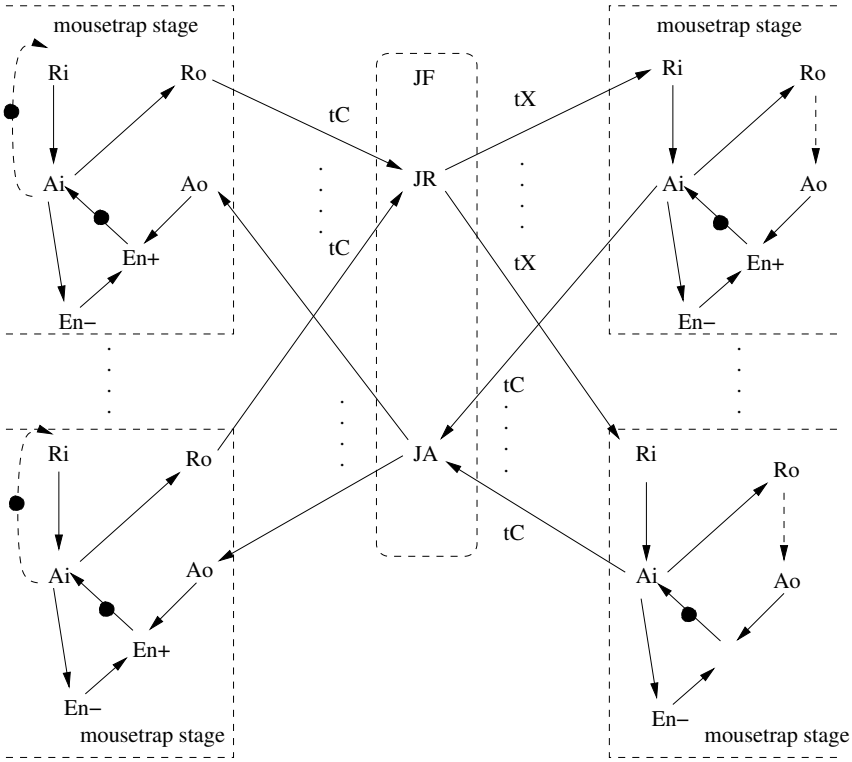


Figure 9.3.: Signal Transition Graph of JF synchronization point synchronizing a number of latch pipelines.

Figure 9.3 shows how the STG models the synchronization join-fork point in the crossbar. The *Ro* events from the input pipeline are joined in the *JR* event. The $Ro \rightarrow JR$ dependencies represent this join and they are annotated with the delay of the the five-input C-element, t_C, that implements the join. The dependencies $JR \rightarrow Ri$ represent the fork of the joined request signal to the output pipelines and are annotated with the propagation delay of the crossbar, t_X.

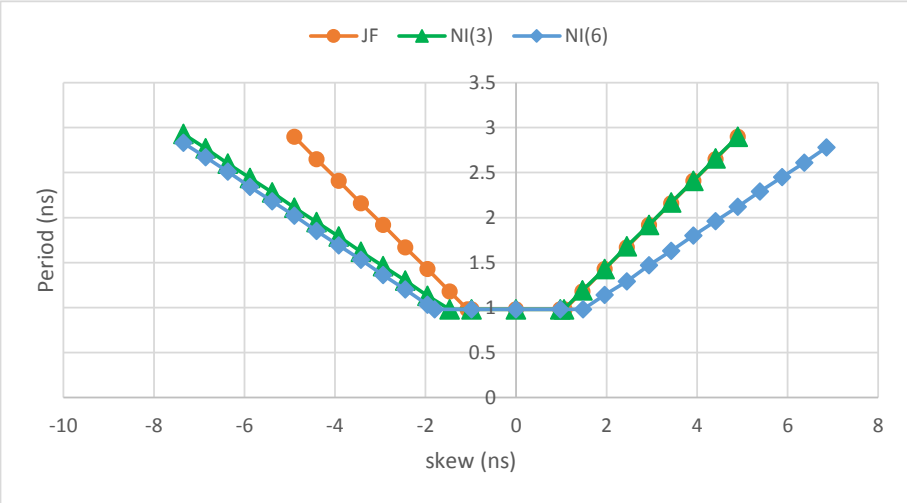


Figure 9.4.: Skew analysis of a JF-JF ring segment (with 3 FIFO stages) and of NI-JF ring segments with 3 and 6 FIFO stages respectively, from [73].

Similarly, the *Ai* events from the output pipelines are joined in the *JA* event with the delay of a five-input C-element, t_C, and forked to the *Ao* events of the input pipelines.

9.2.3. TSE Analysis Results

This section presents results from the TSE analysis for a number of STG models. The results show the relationship between the skew that the models tolerate and the NI clock period. The TSE analysis performed in this work assumes delay timing parameters that were obtained from an implementation of the NOC. The delays are derived by simulating a gate-netlist synthesized in a 65 nm CMOS standard cell technology library. The TSE analysis uses constant delay values. Ranges of values are also possible, but their use implies an unrealistic behavior of the design, where consecutive occurrences of an event alternate between minimum delay and maximum delay values.

The graphs in Figure 9.4 show the relationship between the tolerated skew and the period of the NI clock for three segments of the design. Graph JF shows the skew-period relationship for a FIFO ring connecting two JF points, like the one shown in Figure 8.5. The FIFOs include three stages of pipeline, resembling the Argo routers. Graph $NI(3)$ shows the skew-period relation for a FIFO ring connecting a JF point and an NI, considering a FIFO depth of three stages. Graph $NI(6)$ shows the skew-period relation for a FIFO ring connecting a JF point and an NI, considering a FIFO depth of six stages. As expected, at slower periods the model absorbs more skew. The graphs of Figure 9.4 follow the same pattern as the graph presented in Figure 8.5(b). The comparison of the graphs $NI(3)$ and $NI(6)$ shows that models with more FIFO stages absorb more skew.

The graphs in Figure 9.5 show the relationship between the tolerated skew and the period of the NI clock for two extended sections of the design, from an NI to a neighboring NI. Graph $NOC(3)$ shows an NI to NI segment, considering a FIFO depth of three stages between NI and

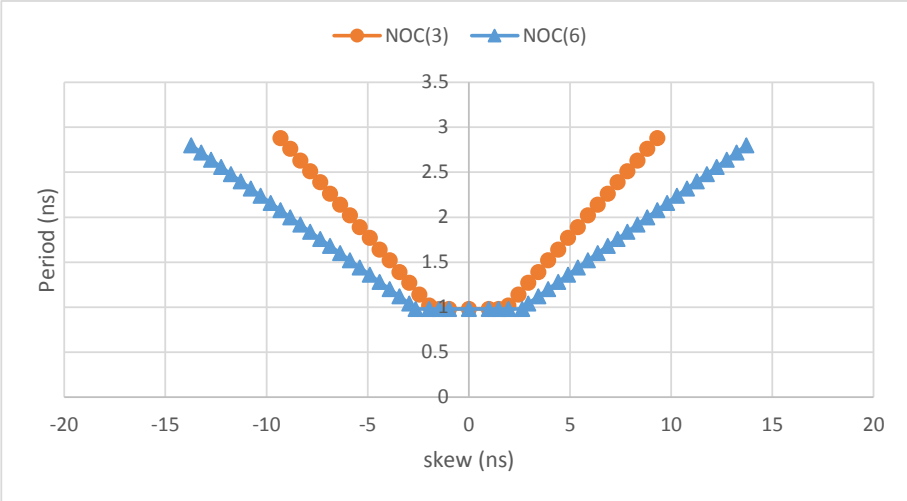


Figure 9.5.: Skew analysis of a complete NI-NI segment, using NI-JF ring segments with 3 and with 6 FIFO stages, from [73].

Points in design space (NI clock, NI-JF)	STG analysis			Bound	Simulation
	δ_{JF}	δ_{NI}	δ_{NOC}	Eqn. 8.2	2×2 NOC
1 GHz 3 stages	1.09	1.08	1.84	1.07	
1 GHz 6 stages	1.09	1.51	2.68	1.93	
500 MHz 3 stages	1.55	1.54	2.91	1.53	2.85
500 MHz 6 stages	1.55	2.26	4.72	2.97	

Table 9.1.: Skew tolerance in cycles for different circuit structures and different operating conditions.

JF points. Graph $NOC(6)$ shows an NI to NI segment, considering a FIFO depth of six stages between NI and JF points. The same trends appear as in the graphs of Figure 9.4.

At the maximum frequency 1 GHz, the segment connecting NIs with three FIFO stages between NI and JF point is able to absorb up to 1.8 ns (1.8 cycles) of skew. The NI to NI segment with six FIFO stages between NI and JF points is able to absorb 2.6 ns (2.6 cycles). At a slower frequency 500 MHz, the two structures, with three and six stages of FIFO depth, are able to absorb 5.8 ns (2.9 cycles) and 9.4 ns (4.7 cycles) respectively.

As expected from the performance analysis of ring structures in Section 8.2.2, a slower system (longer period) is capable of absorbing more skew. However, even at the maximum frequency the structures presented here are able to absorb some skew. This is illustrated in all the graphs presented in Figures 9.4 and 9.5, which have a flat bottom within a certain interval, and a positive linear slope outside this interval. They resemble the graphs shown in Figure 8.5(b), with the difference that the latter refers to clock cycle units (a relative relation) instead of time units. Furthermore, the graphs show that by increasing the number of FIFO stages between the NI and JF points, the skew tolerance increases. In other words, deeper FIFO structures can tolerate more skew, verifying our expectation.

Table 9.1 shows the maximum skew for four different points in the design space, as obtained from three different methods. Each row in the table represents a design point. The design points refer to two designs operating at two different frequencies. The two designs are respectively with three and six FIFO stages between NI and JF points. The operating frequencies are the maximum frequency of 1 GHz and a frequency of 500 Mhz. The methods for deriving the skew values are: (i) TSE analysis on the STG models of different structures, (ii) the safe bound from Equation 8.2, and (iii) simulation from an actual implementation of the design. Columns δ_{JF}, δ_{NI} and δ_{NOC} report results obtained from the TSE analysis of the corresponding models as presented in Section 9.2.2. Column "Bound Eqn. 8.2" shows results for safe bounds of skew obtained from Equation 8.2. The last column shows results obtained from simulation of a synthesized 4x4 Argo NOC in a 65 nm CMOS standard cell technology library. The implemented design includes 4 NIs, 4 routers and three-stage FIFOs between NIs and JF points.

The TSE analysis in Table 9.1 shows that even at the maximum frequency the structure between two NIs can absorb up to 1.84 cycles. This number increases to 2.68 cycles when three more stages are added to the FIFO structure connecting the NIs and JF points. The skew tolerance also increases when the operating frequency drops. At 500MHz the skew tolerance reaches 2.91 cycles with three FIFO stages among NIs and JF points and 4.72 cycles with six FIFO stages.

In Table 9.1, a comparison between the bound calculated from Equation 8.2 and the TSE analysis results for δ_{NOC}, shows that Equation 8.2 indeed provides a safe bound for the skew absorbed between NIs. The TSE analysis results for δ_{NOC} indicate that the maximum skew absorbed between NIs is less or equal to $2 \times \delta_{NI}$. Depending on the operating frequency, the negative term to be added to $2 \times \delta_{NI}$ may range from 0 to δ_{JF}. Thus, the JF ring segment may contribute negatively to a bigger or smaller extent, but overall it does not exceed δ_{JF}. This confirms that Equation 8.2 is a safe bound. This bound may be overly pessimistic in some cases. However, it still gives a considerable amount of skew that the structure is able to absorb.

The last column of Table 9.1 reports the skew that was absorbed from an actual circuit implementation. The implementation is for a single design point and serves as a comparison to the results obtained from the analysis. The implemented instance of Argo is a complete 4x4 structure, not entirely modeled by the STGs analyzed. However, it shows that the real circuit is able to absorb 2.85 cycles of skew. This figure is very close to the δ_{NOC} value found by the TSE analysis of the models. Furthermore, it also confirms that Equation 8.2 gives a pessimistic but safe bound.

9.3. FBCN Model Analysis of Argo

This work includes a preliminary analysis based on a second type of timed-graph model, FBCN. This model provides an abstraction of the detailed circuit signals. It considers handshake channels between pipeline stages, and the timing behavior is based on the timing parameters, i.e. the forward and reverse latency of the pipeline, L_f and L_r, respectively. Since the model is more coarse-grained, it ignores detailed signal information from the design, but allows analysis of larger structures. It also allows the exploration of the timing behavior and behavioral effects that would be difficult to study in a more detailed structure. Thus this anal-

ysis serves mainly to provide insight into the behavior of the system rather than evaluating specific values of skew and period.

9.3.1. FBCN Specification

FBCN models, like STGs, are a type of timed marked graphs, and are defined as follows [14].

Definition 9.3.1. A FCBN N is a PetriNet $N = (P \cup \bar{P}, T, F, m_0)$ which satisfies the following properties:

- place symmetry, i.e. $|P| = |\bar{P}|$,
- channel connectivity, i.e. for every $p \in P$, there exist (t_i, p) and $(p, t_j) \in F$ and $p \in \bar{P}$ such that (t_j, p) and $(p, t_i) \in F$
- single token channels, i.e. for every $p \in P$, $m_0(p) + m_0(\bar{p}) = 1$.

The intuitive understanding of this model is that pipeline stages are connected with handshake channels. Each pipeline stage is modeled as a transition, and each channel is modeled as two places and four arcs connecting two transitions in an opposite way. Places p represent the forward dependencies, and the delays assigned to them, $d(p)$, represent the forward latencies L_f. The corresponding places $\bar{p}$ represent the backward dependencies, and the delays assigned to them, $d(\bar{p})$, represent the reverse latencies L_r.

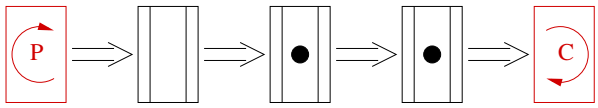
The semantics and the firing conditions are similar to the STG model. A transition *fires* when a token is propagated through the specific pipeline stage. For a transition to fire, the preconditions need to be satisfied, i.e. a request from the previous stage and an acknowledge from the following stage are needed, as modeled by the channel arcs. The cycle time of a channel represents the local handshake period, as given by $L_f + L_r$ [151]. The maximum TSE of two consecutive firings of the same transition indicate the maximum handshake period. The same analysis can be applied as in Section 9.2.3.

9.3.2. Argo FBCN model

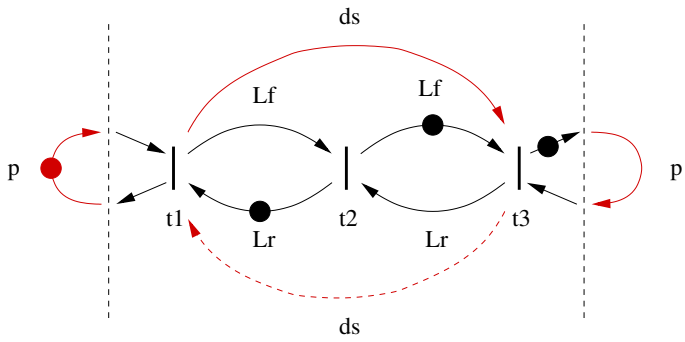
In this section we use the FBCN model to construct structures for Argo similar to those in Section 9.2.2 .

A three-stage FIFO connected to clocked producers and consumers and its FBCN model is shown in Figure 9.6. The black parts of the circuit and the model represent the FIFO, while the red parts represent the clocked interface. For simplification, and since each place has one input and one output arc, the places are omitted in this and the following figures, as for STGs. The input and output arcs are replaced by one that connects the corresponding transitions. The delays and tokens of the places are assigned to the corresponding arcs. The forward and reverse latency, L_f and L_r, are annotated in the figure, as well as the initial marking of tokens. The FIFO is initialized with two tokens in three stages, as shown by the the marking in the FBCN model of Figure 9.6.

To model the interface of the asynchronous pipeline to a clocked producer and consumer, we extend the model with the red segments shown in Figure 9.6. The FCBN considers channel-oriented modeling and does not consider clocks in its specification. Based on the concept



(a) Circuit block diagram.



(b) FBCN model

Figure 9.6.: Three-stage FIFO connected to a clocked producer and consumer.

of the clock and the specifics of the model we model the clock as shown in Figure 9.6(b) in red, based on the following considerations. The concept of the clock is a pulse that repeats with a certain period with no other pre-condition. In the model of Figure 9.6(b) the clocked producer and consumer are separated by the dotted lines and the clock is represented as a loop arc with delay p, the period of the clock. The pulse of the clock is dependent on the firing of the last stage transition, i.e. the presence of a token at the output of transition t_3, in contrast to the lack of precondition for the clock pulse. However, this modeling is consistent for the following reasons: In the initial marking the corresponding arc is initialized to contain a token, and thus the model is consistent upon initialization. Moreover, Argo has the timing assumption that the asynchronous pipeline handshakes faster than the clock frequency, and consequently there will always be a token in this arc. The case in which the design fails to adhere to this timing assumption is detected through TSE analysis of the clock cycle, as the timing separation between two clock pulses exceeds the period of the clock. This condition is studied in the TSE analysis of this section to verify the correct operation of the design.

Figure 9.6(b) also illustrates the skew between the two clocked interfaces, i.e. the producer and the consumer. The skew is modeled as an arc between the endpoint transitions and is annotated with the phase difference, d_s. The direction of the arc shows the direction of the skew. In Figure 9.6, the solid line represents the case where the consumer lags behind the producer by d_s, while the dotted line represents the case where the producer lags behind the consumer by d_s. Depending on the case, the graph model includes one of the two arcs.

Larger structures can be easily constructed in a tile-based way. To achieve further understanding, we study the behavior of the FIFO rings in the Argo structure. Figure 9.7 illustrates the model of a FIFO ring like the one shown in Figure 8.5(a). Figure 9.7 shows the pipeline stages of the FIFO ring and a producer and a consumer clocked interfaces joined at opposite sides of the ring to enforce a clock rate of period p. The skew between the two clocked interfaces is modeled by the two red arcs t_1 to t_4 and the reverse, with delay d_s. As in Figure 9.6,

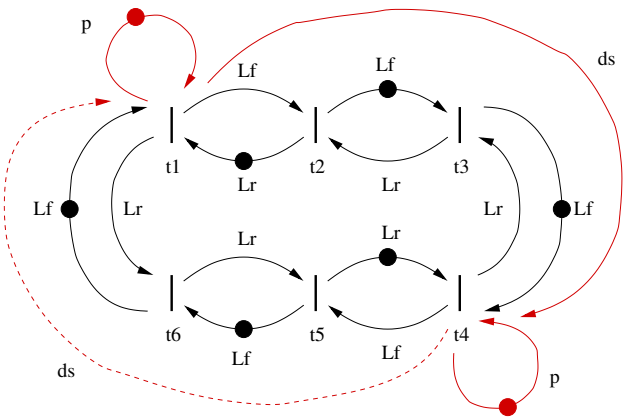


Figure 9.7.: Extended FBCN model of a six-stage FIFO ring connected to clocked interfaces.

depending on whether the producer or the consumer lags behind, the graph model includes one of the two arcs.

9.3.3. Performance Analysis

Some observations on the models shown in Figures 9.6 and 9.7 are presented here. We assume uniform forward latencies L_f and uniform reverse latencies L_r for all the FIFO stages. The models are symmetric in terms of the forward and reverse path dependencies and the cycles in the graph. Consequently, we can assume $L_f > L_r$ without loss of generality. According to these assumptions and parameters we derive equations relating the handshake period to the skew and explore the effect of a clocked environment on the performance of the systems.

The timing behavior in these models is determined by the critical cycle and the cycle time of the graph. The cycle time in timed graph models has been used in research as a metric for timing and performance analysis purposes as explained in Section 3.5. As explained in Section 2.6.6, a cycle is a sequence of transitions $t_1, t_2, ..., t_1$ connected by arcs. The cycle weight is the sum of delays of the arcs $d(c)$ divided by the sum of the tokens placed in the cycle $t(c)$, $d(c)/t(c)$. The critical cycle(s) are the ones with the highest weight and that weight is the cycle time of the graph. The cycle time is straightforward to evaluate and is a direct metric for the average-case performance of the system. It also affects the worst-case timing separation between events, but it is not sufficient in the general case for evaluating the worst-case timing separation.

FIFO

In the FIFO model of Figure 9.6, the critical cycles C_{cr} of the FIFO, ignoring the clock and skew parameters are (t_1, t_2, t_1) and $(t_1, t_2, t_3, t_2, t_1)$ with weight $W(C_{cr}) = L_f + L_r$. The critical cycle is a metric of the inherent performance capabilities of the FIFO. With the addition of clock and skew arcs, more cycles are created in the model, affecting the handshake period of the model. If the clock period is larger than $W(C_{cr})$, the model operates at the clock period. Otherwise, the handshake at the endpoints of the FIFO fails to follow the clock rate. With the assumption the clock period is larger than the $W(C_{cr})$, we now explore the effect of skew.

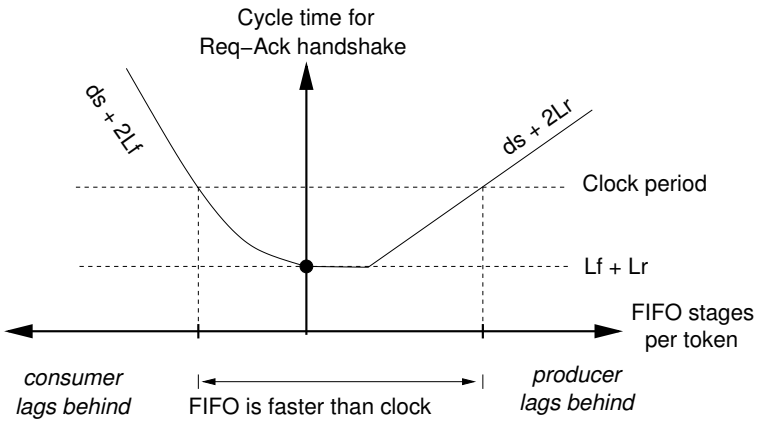


Figure 9.8.: Performance graph based on the FBCN model of the three stage FIFO.

In the presence of skew with delay d_s, more cycles are created and further affect the timing. If the consumer lags behind the producer, cycle (t_1, t_3, t_2, t_1) with weight $d_s + 2L_r$ is created. Therefore, the handshake cycle time of the FIFO is determined by:

$$P = max\{L_f + L_r, d_s + 2L_r\} \tag{9.1}$$

If the producer lags behind the consumer, cycle (t_3, t_1, t_2, t_3) with weight $d_s + 2L_f$ is created, and the handshake cycle time of the FIFO is determined by

$$P = max\{L_f + L_r, d_s + 2L_f\} = d_s + 2L_f \tag{9.2}$$

under the assumption $L_f > L_r$.

The above analysis conforms with the theoretical analysis presented in Section 8.2.1, and Figure 9.8 presents the performance graph of the three-stage FIFO model based on this analysis. The flat part of the right graph represents the part where $L_f + L_r > d_s + 2L_r$. The slope of the left graph is steeper since $L_f > L_r$. In the opposite case where $L_f < L_r$, the left and right graphs would be mirrored about the Y-axis.

FIFO ring

In the FIFO ring model of Figure 9.7, we identify a number of possible critical cycles C_{cr}, ignoring the clock and skew parameters. Those are the cycle (t_1, t_2, t_1) and all the local cycles with weight $L_f + L_r$, the cycle $(t_1, t_2, t_3, t_4, t_5, t_6, t_1)$ with weight $6L_f/4 = 1.5L_f$, and the cycle $(t_1, t_6, t_5, t_4, t_3, t_2, t_1)$ with weight $6L_r/2 = 3L_r$. The weight of the critical cycle C_{cr} is

$$W(C_{cr}) = max\{L_f + L_r,\ 1.5L_f,\ 3L_r\} \tag{9.3}$$

As in the case of the FIFO model, in the presence of skew with delay d_s, more cycles are created and further affect the timing. If the consumer lags behind the producer, the cycle $(t_1, t_4, t_5, t_6, t_1)$ is introduced with weight $(d_s + 3L_f)/2$ and the cycle $(t_1, t_4, t_3, t_2, t_1)$ with weight $(d_s + 3L_r)/1$. Therefore, the handshake cycle time of the ring is determined by

$$P = max\{L_f + L_r,\ 1.5L_f + 0.5d_s,\ 3L_r + d_s\} \tag{9.4}$$

If the producer lags behind the consumer the model is symmetric, and thus the handshake cycle time of the ring is the same. The performance graph of the ring resembles the one shown in Figure 8.5(b) and is symmetric about the Y-axis.

9.3.4. Oscillating behavior

An interesting behavior appears in some ring models. This behavior has also been observed in [6] in the example of [6, Figure 5]. In this ring example with specific delay parameters, the timing separation between consecutive occurrences of event a oscillates between two values, 4 and 8. This gives a worst-case timing separation of 8, indicating that the occurrences of event a may be 8 time units apart in the worst case.

The same behavior is observed in the ring shown in Figure 9.7. Assume a ring with $L_f = 3$, $L_r = 1$, without a clock enforcing a rate. The critical cycle weight from Equation 9.3 is $W(C_{cr}) = 4.5$. The value $W(C_{cr})$ indicates the average-case performance of the ring, while the maximum timing separation for consecutive occurrences of the same transition represents the worst-case handshake cycle. The TSE analysis applied to the ring evaluates the worst-case handshake cycle of t_1 to 5, oscillating between values 4 and 5. This indicates that for safe operation the clock period to be applied at the ring should be greater than or equal to 5. However, by applying a clock with a period corresponding to the average performance period given by the $W(C_{cr})$, the ring balances at the average performance period. The worst-case handshake cycle is now 4.5 as evaluated by TSE analysis, without oscillations. This indicates that the elastic behavior of a self-timed ring may result in fluctuating handshake cycles, leading to large deviations from the average case performance and large worst-case handshake cycles. Nevertheless, a ring constrained by a clock period equal to $W(C_{cr})$ eliminates the fluctuations and results in a lower worst-case handshake period.

This observation opens up the possibility that an average case performance analysis can be enough for evaluating the rate that should be applied at the endpoints of the elastic ring. Average case analysis amounts to traversing the graph models and enumerating the cycles in them and their weights, as was done in Section 9.3.3. It does not require unfolding the execution of the model or complex mathematical analysis to evaluate the maximum timing separation over infinite occurrences. Therefore, it is a low complexity analysis. This is promising, as it allows bigger structures and more precise models to be analyzed and a simple way to evaluate the maximum clock rate that can be applied at the endpoints of the elastic asynchronous network.

9.3.5. TSE Analysis Results

To study the effects of bigger structures we performed TSE analysis on larger FBCN models. The purpose of this is to explore the relative effects on bigger structures, and therefore we use uniform delay stages. The forward and reverse latencies are based on the implementation of the Argo pipeline. For the delay elements we consider uniform delays in every stage. In the implementation, $t_L = 120$ ps is the propagation delay of the latch, $t_X = 70$ ps is the propagation delay of the XNOR gate. We consider $t_D = 300$ ps to be a realistic delay for the delay elements to match the combinational logic stages. Thus, $L_f = 420$ ps and $L_r = 190$ ps.

Ring

With these parameters and from Equation 9.3 we get $W(C_{cr}) = 630$ ps. From Equation 9.4 the average handshake cycle is:

$$P_{avg} = max\{610,\ 630 + 0.5d_s,\ 570 + d_s\} \tag{9.5}$$

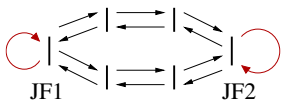
By running TSE analysis on this ring model with no skew ($d_s = 0$) and without a constraining clock, the worst-case handshake cycle time is evaluated as 650 ps, while it oscillates between the values 610 ps and 650 ps. By setting a clock on the opposite transitions t_1 and t_4 with period $P_{avg} = 630$ ps, the worst-case handshake period of the ring found from the TSE analysis is constrained to 630 ps. In the presence of skew, TSE analysis yields the same results as Equation 9.5 and these are presented in Figure 9.9(c). Overall, we conclude that by constraining an elastic ring with its average-case rate, the worst-case handshake period which the ring exhibits at its endpoint does not exceed the average case.

NOC

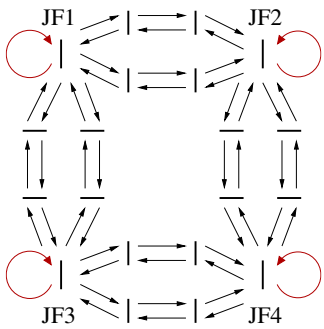
The analysis of larger models provides better understanding of the Argo NOC structure. We built a number of models, which are shown in Figures 9.9(a-b) and 9.10(a-b). Since the models consider uniform delay for all pipeline stages, and to simplify the models we consider FBCN ring models both the FIFO ring connecting two JF points and the NI to JF points. The red arcs in the models define the clocked interfaces, while the arcs defining the skew are omitted from the figures. The graphs in Figures 9.9(c) and 9.10(c) illustrate for the different models the relationship between the skew and the handshake period for which this skew can be absorbed safely.

Figure 9.9(a) is a simplified representation of the ring of Figure 9.7 and is used as a building block for the rest of the models. Figure 9.9(b) models a 2x2 mesh consisting of four JF points connected by four ring segments. In Figure 9.9(c), graph JF shows the relationship between skew and period between points JF1 and JF2 of the model in Figure 9.9(a). Graph $JF4-n$ shows the relationship between skew and period between two neighboring JF points in the 2x2 mesh of the model in Figure 9.9(b). If we consider JF1 as a reference point then graph $JF4-n$ refers to the skew between JF1 and JF2 while JF3 and JF4 are not skewed with reference to JF1. This case is symmetric to the skew between JF1 and JF3 while JF2 and JF4 are not skewed with reference to JF1. Graph $JF4-c$ shows the relationship between skew and period between two JF points in opposite corners of the 2x2 mesh of the model in Figure 9.9(b). If we consider JF1 as a reference point then graph $JF4-c$ refers to the skew between JF1 and JF3. All the graphs show the same trends and the expected behavior as explained in the performance analysis in Section 9.3.3. Graph $JF4-n$ coincides with graph JF, revealing that two neighboring JF points, connected by a ring segment in a mesh of four, absorb the same skew as a single ring. Graph $JF4-c$ shows that opposite corner JF points, connected by two ring segments in a mesh of four, absorb double the skew of a single ring.

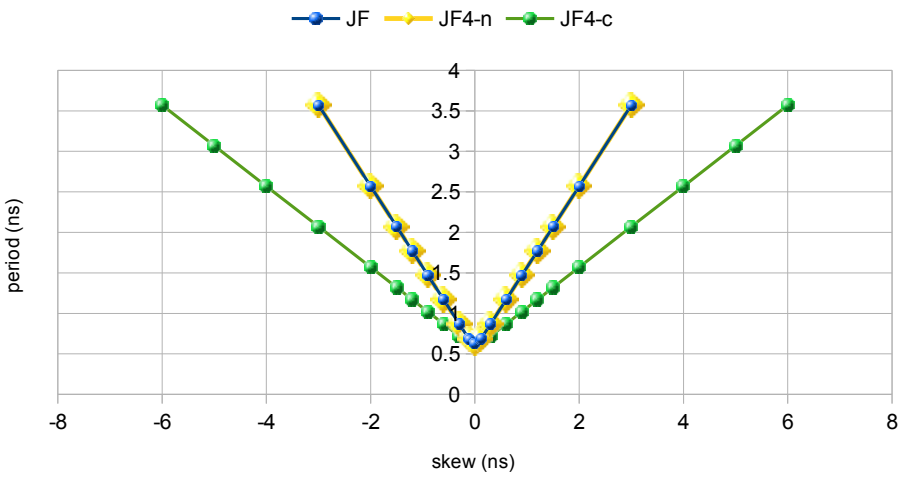
Figure 9.10(a) models two NIs connected by three ring segments and two JF points. Figure 9.10(b) represents a 2x2 mesh consisting of four NIs and four JF points. In Figure 9.10(c), graph JF repeats the graph of Figure 9.9 for comparison. Graph NI illustrates the relationship between skew and period between points NI1 and NI2 of the model in Figure 9.10(a).



(a) FBCN ring model of two JF points



(b) FBCN model of a 2x2 mesh of four JF points

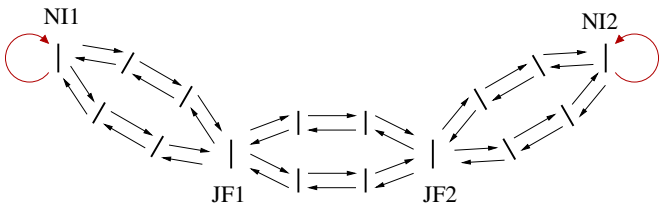


(c) Graphs relating skew and period

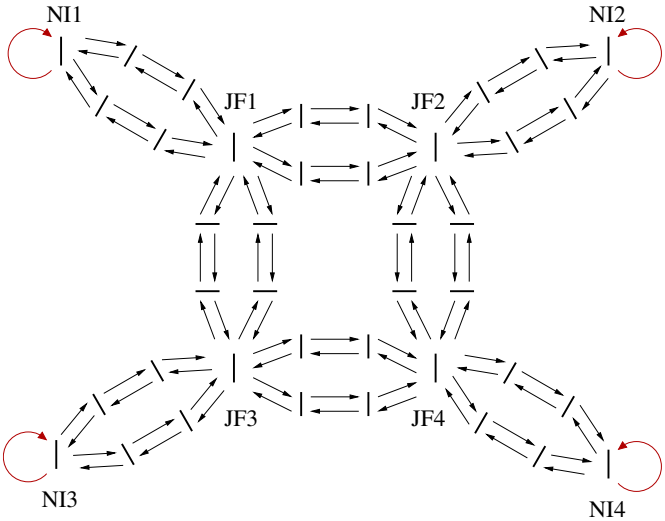
Figure 9.9.: FBCN ring models connecting two and four JF points, along with the graphs showing the relation of skew and period in these models.

Graph $NI4-n$ illustrates the relationship between skew and period between two neighboring NI points in the 2x2 mesh of the model in Figure 9.10(b). If we consider NI1 as a reference point then graph $NI4-n$ refers to the skew between NI1 and NI2 while NI3 and NI4 are not skewed with reference to NI1. This case is symmetric to the skew between NI1 and NI3 while NI2 and NI4 are not skewed with reference to NI1. Graph $NI4-c$ illustrates the relationship between skew and period between two NI points in opposite corners in the 2x2 mesh of the model in Figure 9.10(b). If we consider NI1 as a reference point then graph $NI4-c$ refers to the skew between NI1 and NI3. Graph NI compared to JF reveals that the segment between two NIs, consisting of three rings absorbs three times the skew absorbed by a single ring. Graph $NI4-n$ coincides with graph NI illustrating that two NIs connected by 3 ring segments absorb the same skew regardless the rest of the structure connected to JF points. Graph $NI4-c$ shows that NIs connected by four ring segments absorb four times the skew that a single ring absorbs.

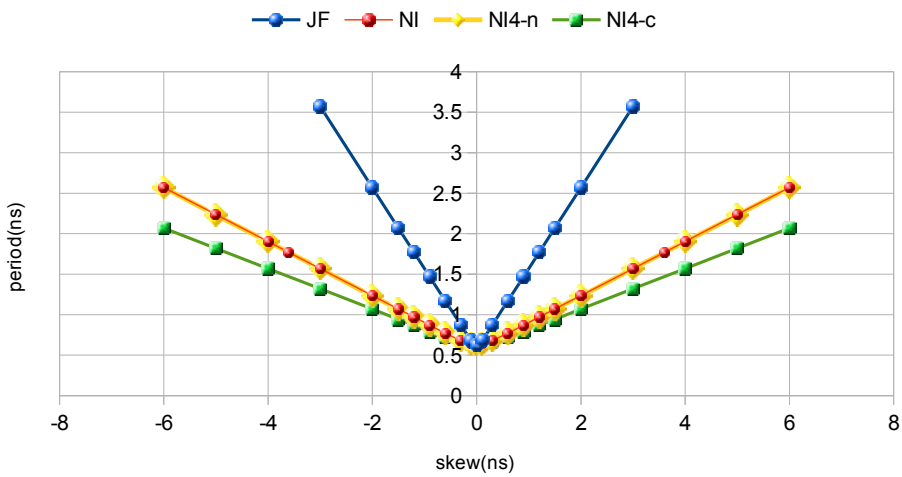
Overall, the above results are not to be taken as an absolute analysis of Argo, as the models are abstract and hide some signal details, and the stages are modeled uniformly. However, a



(a) FBCN model of two NIs and two JF points



(b) FBCN model of a 2x2 mesh of four NIs and four JF points



(c) Graphs relating skew and period

Figure 9.10.: FBCN ring models connecting two and four NIs and JF points, along with the graphs showing the relation of skew and period in these models.

relative comparison contributes to the understanding of the Argo NOC structure and the timing effects that take place. In particular, there are two main points that come out of the analysis. Firstly, the skew that can be absorbed between two points is the sum of the skew absorbed by the ring segments in the shortest path that connects the two points. Secondly, a bigger structure of JF points does not affect the skew absorbed between two neighboring JF points. Thus the

skew absorbed safely between any pair of NIs in a 2x2 mesh is the skew absorbed by the three ring segments in the path that connects the NIs.

9.4. Discussion

This chapter presented a quantitative analysis on the elasticity of Argo. Argo is an elastic network of asynchronous routers wrapped in a clocked environment of mesochronous NIs. This organization requires some analysis of how this elastic structure operates in a clocked environment. The elasticity of Argo was explored through timing models of Argo which were constructed in a bottom-up fashion and analyzed with a timing analysis algorithm tool, to produce results quantifying the relationship between skew and performance.

Analysis of the more detailed STG model, with real timing parameters from the implementation, provided a quantitative evaluation of the elasticity of Argo. The graphs presented showed that Argo is able to absorb roughly 2-3 cycles of skew, depending on the clock frequency. The elasticity is reduced at higher frequencies as expected, and can be increased by adding FIFO stages between NIs and routers. Simulation of the actual circuit confirmed the analysis of this model as well as the theoretical skew bound equations.

Analysis of the more abstract FBCN model revealed an interesting behavior appearing in self-timed rings. With specific timing parameters, the timing at certain points of the ring might oscillate between two values. The timing at these points is determined by cyclic paths alternating between best-case and worst-case timing. The safe timing in this case is determined by the worst case. However, when constrained by a clock with a period matching their average-case performance, they maintain an average-case timing, eliminating the fluctuations between a best-case and a worst-case timing. This leads to the conclusion that a clock can constrain the rings to operate safely with average case timing. With this observation we can further conclude that the choice of clocked NIs driving the asynchronous elastic structure is an efficient design choice. Another aspect of this is that the exploration of Argo elasticity is reduced to the exploration of the average-case performance. This is an considerably easier task, as it does not involve evaluation over infinite executions of the system.

Further exploration of the elasticity can move in the direction of more precise modeling and of modeling larger and more complete structures. The analysis of the models presented in this chapter relies on fixed timing constraints. Further analysis could include ranges of delays for the timing constraints of models and evaluation through TSE analysis. This modeling is more flexible but not realistic for the TSE analysis used, since this then considers immediate changes in timing from as early as possible to as late as possible or *vice versa*, a behavior not common in digital circuits. A more accurate modeling of the circuit could consider stochastic models, assuming probability distributions for the timing constraints. The analysis presented here is a step into a field open for exploration.

Part IV.

Conclusions

10
Conclusions

This chapter summarizes and concludes the work of this thesis. It reviews the initial goals and how they were met. It lists the contributions made in this thesis and discusses the topics raised during the completion of this work. Finally, it reports open issues and interesting ideas revealed by this work that require further investigation.

10.1. Overview

The initial goals for Argo were for it to be: (i) time-predictable, such that it provides guarantees for real-time requirements, (ii) time-elastic, such that it is tolerant to signal distribution issues and variability effects, and (iii) able to be implemented efficiently. This section reviews how Argo NOC met these initial goals.

Argo is *time-predictable* through TDM. In the Argo design, the TDM scheme is enforced by the NIs, which are clocked components. The NI frequency directs the TDM mechanism. Thus the throughput of a message which traverses the NOC follows directly from the TDM schedule. In this thesis the timing parameters of Argo, i.e.the latency and throughput, have been analyzed, and the WCET that Argo contributes to a message transaction has been evaluated. The end-to-end latency which Argo contributes to a message transaction is also affected by the TDM skew between the NIs. In this thesis, it was shown that the skew tolerance of Argo has fixed limits and contributes a fixed worst-case number of cycles to the latency. Overall, this thesis showed that Argo offers WCET guarantees in an easy and straightforward way.

Argo is *time-elastic*, as it is a network of asynchronous routers. The asynchronous network can be seen as a mesh of FIFOs and exhibits elastic behavior. This elastic behavior is able to absorb skew among the mesochronous NIs. This skew originates from clock and reset skew and appears as shifts in the TDM schedule. In this thesis the limits of this elasticity have been explored, and models of the Argo NOC have been studied, leading to the derivation of

equations for safe bounds within which Argo operates safely. The analysis of the models showed that the asynchronous Argo network is able to tolerate two to three cycles of skew depending on the frequency of the NIs.

Argo has an *efficient implementation*. This is based on the efficient design of its components, i.e. the NIs and the routers, and the combination of the two. The main idea leading to an efficient design is to eliminate the overhead in the end-to-end path of data communication, following from timing synchronization, flow control, and the buffering that this incurs. This design creates direct paths from the local memory of one core to a local memory of another core, thus eliminating synchronization, flow control, and buffering overhead. The implementation of the overall Argo design is at least four times smaller than alternative TDM-based NOC designs. Moreover, it is power efficient, saving power when there is no traffic scheduled.

10.2. Contributions

This thesis contributed in a number of areas. This section reviews the contributions in detail and summarizes interesting points that were raised in relation to each topic.

Router

The router proposed in this thesis is a very efficient design. TDM routers are by design very lightweight routers. In addition, the Argo router combines the router functionality with elastic FIFO behavior. As shown by the implementation and comparison in Chapter 6, the Argo router is two to three times smaller than a mesochronous TDM router with the same functionality. The overhead of the mesochronous router comes from the mesochronous FIFOs implementing synchronization. Alternative custom designs of mesochronous synchronization in the literature still introduce an overhead of at least the size of the router. Moreover, the Argo router saves energy and reaches a similar speed to the equivalent mesochronous router. The energy savings follow, on one hand, from eliminating the energy consumed by the FIFOs, and on the other hand, from gating off the unnecessary switching activity occurring in the asynchronous pipeline while there is no traffic scheduled.

NI

The Argo NI design is a novel design that integrates the DMA functionality and TDM schedule. DMA controllers are placed in the NIs and their operation is time-multiplexed according to the TDM schedule. In addition, dual-ported SPM memories implement the clock-domain-crossing interface between processors and NIs. Moreover, the direct access to the SPM at the receiving end eliminates the requirement for buffering and end-to-end flow control. This design flattens the layered implementation of the NI, avoiding the overhead and the cost of crossing different layers, and leading to a very efficient NI design. A direct comparison of NIs from the literature is difficult, as each NI implements different features, depending on the processor and the router designs that it bridges. Section 5.3 presents a discussion on a comparison with other NI designs. As the reported results show, the Argo NI is approximately five times smaller than other NI designs of equivalent NOCs.

NOC

The combination of the routers and NIs in the overall Argo NOC builds end-to-end paths from a local SPM to a remote SPM without any buffering, synchronization, or end-to-end flow control. This results in an efficient implementation which is three to four times smaller for roughly equal dimensions of similar TDM NOC architectures, as explained in Section 7.10. A direct comparison is difficult, since each architecture implements different features. As shown by the implementation, Argo is at least four times smaller than a comparable instance of aelite, which is a very lightweight TDM NOC.

The Argo NOC follows a novel timing organization. The NIs are clocked designs operating on a common mesochronous clock. The core network of routers is asynchronous operating in an elastic way. Correct operation relies on the timing assumption that the NI clock is slower than the handshake cycle of the asynchronous network. As long as this assumption holds, there are no issues of metastability in the NOC and there is no need for explicit synchronization, as the elastic NOC handshakes faster than the clock period.

Elasticity

This work explored the elasticity of Argo, building an intuitive understanding of its elastic behavior, and a more specific model-based analysis of Argo. A bottom-up theoretical analysis contributed in defining and understanding the problem and exploring its bounds. The more precise modeling of Argo as a timed-STG was used to evaluate the limits of its elasticity. The skew tolerance of Argo was evaluated by TSE analysis, which gives worst-case bounds estimations, and found to be to be two to three cycles of the NI clock. The skew tolerance can be increased by adding more FIFO stages connecting the NIs and the routers. We performed TSE analysis in this work and the results agree with the theoretical analysis and were verified by simulation of an implemented instance of Argo.

The more abstract FBCN model gave some insight into the behavior of Argo. As shown by the worst-case TSE analysis of the model, the timing of ring structures oscillates between worst-case and best-case timing. Under these conditions the timing is limited by the worst case. Applying an average-case rate at certain points of the ring eliminates the oscillations and constrains the timing of the ring to its average case. While the timing at those points is fixed, the structure between them is allowed to be flexible. This indicates that a clock can constrain a ring to its average-case rate, not allowing it to reach the worst case. This shows that average-case analysis is sufficient to verify the safe operation of the network. This is an important observation, as an average case analysis is far simpler than worst-case analysis, and moreover, it allows the clocking of the NOC to be done at the average instead of the worst-case rate.

10.3. Discussion

The work of this thesis raised some points for discussion in certain areas of the design, implementation, and analysis of the Argo NOC. These points are mentioned and discussed throughout the thesis. This section aims at summarizing interesting discussion points and presenting an overall view.

The design of Argo revisits the overall NOC architecture, to provide an overall solution rather than providing solutions to certain problems individually. At first sight, Argo seems to combine two contradictory principles: time predictability and time elasticity. However, each principle operates within certain limits without contradicting the other. The elasticity provided by the asynchronous network of routers absorbs skew caused by signal distribution issues up to a certain number of cycles. Taking this limit into account, the time-predictability is maintained at the level of the NIs.

For the above combination to be valid, the data flow should be maintained within the NOC. The Argo NOC employs a distributed method of synchronization. The asynchronous network of routers is time-elastic but maintains the data flow of different connections, due to the strongly indicating implementation of the crossbar. The crossbar performs join-fork functionality resembling the clock ticks. From this distributed local ticking, a global synchronization emerges. At the endpoints of the asynchronous network, the NIs operate at a specific frequency. Based on this frequency, they inject and remove one packet in every time slot. So at the interfaces of the NOC towards the processors, the data flow is maintained. Thus, the combination of the above two principles, i.e. local join-fork and end-point flow rate, ensures the data flow and maintenance of the TDM schedule at the NIs.

The NIs of the Argo NOC reconsider the traditional layered design of the NIs. The purpose of the layered design is to encapsulate and distinguish the services of various protocol layers and implement clear interfaces between layers and components. The Argo NI flattens this design. Yet it maintains a standard interface to the processor (OCP) and allows the implementation of services of higher protocol stack layers. The front and back end functionality of the NI can still be recognized in the Argo NI. The NI uses common resources, i.e. DMA tables, slot tables and SPM, accessed both by the processor and the packet-switching network. The key idea is the time multiplexing of the accesses to these common resources. This extends the TDM scheme from the packet-switching network of routers to include also the functionality of the NIs. In this way Argo achieves very lightweight NIs and routers.

Argo has a lightweight implementation both in the routers and in the NIs. The hardware architecture implements all the necessary functionality to support communication among processors. The NOC encapsulates this functionality and offers the processors a view of a mesochronous TDM NOC. The hardware does not offer complex services that are required in higher levels of the protocol stack. However, it provides all the functionality required to implement these features in software. Argo follows the approach of retaining a low cost hardware implementation and providing complex functionality in software when needed. Argo has been used in the T-CREST platform, and libraries supporting the abstraction of communication services were built over the NOC. The entire platform has been used in a number of industrial applications.

Another aspect of Argo is the design of routers as asynchronous circuits. Argo takes advantage of the inherent time flexibility of asynchronous circuits that offers a natural solution to clock and reset distribution problems. The asynchronous design process is a challenge as it requires more effort from the designer. Within the area of asynchronous design the design options offer trade-offs between design effort and efficiency. The Mousetrap controller was shown to be a good balance between design effort and performance.

The design of Argo avoids metastability at the interface between clocked NIs and asynchronous routers, as it relies in the timing assumption that the NI clock frequency is lower

than the router handshake capabilities. However, the asynchronous pipeline is elastic and may exhibit fluctuations in its timing. To verify the safe operation of the elastic network, we therefore evaluated its worst-case timing over infinite executions with a TSE analysis. However, analysis on elastic rings has shown that, depending on their timing parameters, their timing behavior may oscillate between their worst- and best-case performance. Constraining them with a clock limits these oscillations and eliminates them when a clock period equal to or larger than their average case handshake cycle time is applied. This leads to the conclusion that the average-case cycle is sufficient to maintain the elastic structure in safe operation. Due to this observation, the use of clocked NIs wrapping around the elastic network of routers seems to be a good design decision.

10.4. Future Work

The work described in this thesis has contributed to the design and implementation of an efficient, time-predictable, and time-elastic NOC, and provided an analysis of its elasticity. In addition, it has revealed some new aspects for further investigation.

At the architectural level of the platform, the Argo NOC supports message passing as write operations to remote cores. This scheme is sufficient to implement all-to-all communication. The support of read operations from remote cores would be an extension to the NOC for more flexible communication. In addition, Argo currently implements statically scheduled TDM. End-to-end connections are defined statically and are assigned with different bandwidths. A step towards more flexible communication would be the reconfiguration of the TDM schedule, with different end-to-end connections. This would permit various modes of operation in the NOC, and mode changes depending on the requirements.

A feature that would benefit the operation of the entire platform would be the implementation of interrupts, on the completion of a message transfer. Currently, the processor has the responsibility of polling the NI structure to check the status of a message transfer. While this respects the hard real-time nature of the platform, the triggering of an interrupt may lead to a more optimal mechanism. The handling of the interrupt should be in accordance with the real-time requirements.

At the platform level, heterogeneous platform organizations would be another approach to increase flexibility and create space for performance improvement of the entire platform. Initial work on integrating hardware accelerators with the Argo NOC has been done within the T-CREST project [123].

With respect to hardware platform optimization, the Argo NOC is already an efficient design. Multiple design options were explored on the way to the final design. The performance of the asynchronous pipelines is based on analysis of the design parameters, e.g. the dynamic wavelength, and on the implementation technology of the components, e.g. gate delays. Additional effort could be put into evaluation of alternative options along the same lines (latches/tokens ratio, fine-tuning of delay elements, etc.). Further exploration to optimize the hardware platform could focus on optimizing the link delay. In particular, in the post-layout implementation the long links between routers incur a high delay. This delay is dependent on link length and affects the performance of the entire NOC. Link pipelining has been used in Argo, but it is an area that can be further explored.

With regard to the usability of the NOC, software libraries for message passing functionality have been implemented in the context of the T-CREST project. An area for further exploration could be the development of several software layers to support additional functionality. Services that have been developed within the T-CREST project to support the application development include high level flow control, double buffering, acknowledgments, and synchronization primitives. Alternative programming models and scheduling algorithms may be areas worth further exploration.

With respect to the elasticity analysis, a qualitative theoretical and a quantitative analysis provided insight and estimation of the worst-case limits for the elasticity in Argo. The analysis was based both on abstract and more detailed models. There is space of further investigation in the modeling of Argo structures. Bigger structures and ways of more precise modeling can be explored. The models use timing constraints of fixed values. The TSE analysis offers the option of delay ranges in the timing constraints. As the delay ranges have uniform probability, this was considered an unrealistic option for digital circuits and was not explored in this work. Other options such as stochastic models with different probability distributions offer a promise of more realistic modeling.

Furthermore, the elasticity analysis revealed some interesting aspects of the timing which would benefit from further exploration. Rings exhibit oscillating behavior, which is eliminated by constraining them with average-rate clocks. This indicates that the worst-case timing will not exceed the average-case at the endpoints of the structure, that is constrained by an average-case clock rate. This indicates that an average-case analysis of the rings is equivalent to the worst-case analysis. Average-case analysis is less complex and allows for more precise modeling and bigger structures to be analyzed. This is a promising indication that requires further investigation.

10.5. Conclusion

Overall, the work described in this thesis resulted in the design and implementation of the Argo NOC. Argo uses statically scheduled TDM and supports a GALS system organization. It offers time-predictability in a straightforward way with the TDM schedule, and inherent time-elasticity using asynchronous routers. Its novel timing organization eliminates signal distribution issues and the need for additional synchronization. We implemented the Argo NOC in various technologies, proving the functionality of the NOC, its time-elastic behavior, and its low cost implementation. Additionally, we analyzed the elasticity of Argo and its limits. Model-based analysis gave insight into the elastic nature of Argo, estimated the skew tolerance in clock cycles for specific Argo instances, and defined the parameters characterizing it. Finally, the analysis revealed interesting aspects of the timing behavior of Argo and opened the way for further exploration of this behavior.

Bibliography

[1] The argo noc: Combining tdm and gals.

[2] S. Abbaspour, F. Brandner, and M. Schoeberl. A time-predictable stack cache. In *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

[3] Accellera Systems Initiative. Open Core Protocol specification, release 3.0. `http://www.accellera.org/downloads/standards/ocp/ocp_3.0/`, 2013.

[4] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. Zeferino. SPIN: a scalable, packet switched, on-chip micro-network. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 70–73 suppl., 2003.

[5] B. Akesson, K. Goossens, and M. Ringhofer. Predator: a predictable SDRAM memory controller. In *CODES+ISSS '07: Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, pages 251–256, New York, NY, USA, 2007. ACM.

[6] T. Amon, H. Hulgaard, S. Burns, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. In *Computer Design: VLSI in Computers and Processors, 1993. ICCD '93. Proceedings., 1993 IEEE International Conference on*, pages 166–173, Oct 1993.

[7] ARM Ltd. AMBA Advanced eXtensible Interface (AXI) Protocol, 2011.

[8] Arteris the Network-on-Chip company. A comparison of network-on-chip and busses. Technical report, Arteris the Network-on-Chip company, 2005.

[9] G. Ascia, V. Catania, M. Palesi, and D. Patti. Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. *Computers, IEEE Transactions on*, 57(6):809–820, June 2008.

[10] J. Bainbridge and S. Furber. Chain: A delay-insensitive chip area interconnect. *IEEE Micro*, 22(5):16–23, Sept./Oct. 2002.

[11] M. Bakhouya, S. Suboh, J. Gaber, and T. El-Ghazawi. Analytical modeling and evaluation of on-chip interconnects using network calculus. In *Proc. ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 74–79, May 2009.

[12] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *Software Technologies for Embedded and Ubiquitous Systems*, pages 35–46. Springer, 2011.

[13] R. Bastos, G. Sicard, F. Kastensmidt, M. Renaudin, and R. Reis. Evaluating transient-fault effects on traditional C-element's implementations. In *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, pages 35–40, July 2010.

[14] P. Beerel, A. Lines, M. Davies, and N.-H. Kim. Slack matching asynchronous designs. In *Asynchronous Circuits and Systems, 2006. 12th IEEE International Symposium on*, pages 11 pp.–194, March 2006.

[15] P. A. Beerel, R. O. Ozdag, and M. Ferretti. *A designer's guide to asynchronous VLSI*. Cambridge University Press, 2010.

[16] E. Beigne, F. Clermidy, P. Vivet, A. Clouard, and M. Renaudin. An asynchronous NOC architecture providing low latency service and its multi-level design framework. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 54 – 63, 2005.

[17] E. Beigne and P. Vivet. Design of on-chip and off-chip interfaces for a GALS NoC architecture. In *Asynchronous Circuits and Systems, 2006. 12th IEEE International Symposium on*, pages 10 pp.–183, March 2006.

[18] L. Benini and G. De Micheli. Powering networks on chips: Energy-efficient and reliable interconnect design for SoCs. In *Proceedings of the 14th International Symposium on Systems Synthesis*, ISSS '01, pages 33–38, New York, NY, USA, 2001. ACM.

[19] L. Benini and G. D. Micheli. Networks on chips: A new SoC paradigm. *Computer*, 35(1):70–78, Jan. 2002.

[20] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux. Comparison of different semantics for time Petri nets. In *Automated Technology for Verification and Analysis*, pages 293–307. Springer, 2005.

[21] D. Bertozzi and L. Benini. Xpipes: a network-on-chip architecture for gigascale systems-on-chip. *Circuits and Systems Magazine, IEEE*, 4(2):18–31, 2004.

[22] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, 38(1):1–51, 2006.

[23] T. Bjerregaard, S. Mahadevan, R. G. Olsen, and J. Sparsø. An OCP compliant network adapter for GALS-based SoC design using the MANGO network-on-chip. In *Proceedings of the International Symposium on System-on-Chip (SoC'05)*, pages 171–174. IEEE, Nov. 2005.

[24] T. Bjerregaard and J. Sparsø. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proc. Design Automation and Test in Europe (DATE)*, pages 1226–1231. IEEE Computer Society Press, 2005.

[25] T. Bjerregaard and J. Sparsø. A scheduling discipline for latency and bandwidth guarantees in asynchronous network-on-chip. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 34–43. IEEE Computer Society Press, 2005.

[26] T. Bjerregaard and J. Sparsø. Implementation of guaranteed services in the MANGO clockless network-on-chip. *IEE Proceedings: Computing and Digital Techniques*, 153(4):217–229, 2006.

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

[28] S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, Computer Science Department, California Institute of Technology, 1991. Caltech-CS-TR-91-01.

[29] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, volume 3, page 10190. Citeseer, 2003.

[30] T. Chelcea and S. Nowick. Low-latency asynchronous FIFO's using token rings. In *Advanced Research in Asynchronous Circuits and Systems, 2000. (ASYNC 2000) Proceedings. Sixth International Symposium on*, pages 210–220, 2000.

[31] A. M. Cheng. *Real-time systems: scheduling, analysis, and verification*. John Wiley & Sons, 2003.

[32] T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1987.

[33] M. Coppola, M. D. Grammatikakis, R. Locatelli, G. Maruccia, and L. Pieralisi. *Design of cost-efficient interconnect processing units: Spidergon STNoC*. CRC press, 2008.

[34] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on information and Systems*, 80(3):315–325, 1997.

[35] R. Cuninghame-Green. Minimax algebra and applications. *Fuzzy Sets and Systems*, 41(3):251–267, 1991.

[36] W. Dally and B. Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684–689, 2001.

[37] W. J. Dally and J. W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[38] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Elsevier Science Publishers, 2003.

[39] B. de Dinechin, P. G. de Massas, G. Lager, C. Léger, B. Orgogozo, J. Reybert, and T. Strudel. A distributed run-time environment for the kalray mppa®-256 integrated manycore processor. *Procedia Computer Science*, 18:1654–1663, 2013.

[40] B. de Dinechin, D. van Amstel, M. Poulhies, and G. Lager. Time-critical computing on a single-chip massively parallel processor. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6, March 2014.

[41] M. Dean, T. Williams, and D. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In C. H. Séquin, editor, *Advanced Research in VLSI: Proceedings of the 1991 UC Santa Cruz Conference*, pages 55–70. MIT Press, 1991.

[42] P. Degasperi, S. Hepp, W. Puffitsch, and M. Schoeberl. A method cache for Patmos. In *Proceedings of the 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2014)*, Reno, Nevada, USA, June 2014. IEEE.

[43] R. Dobkin, V. Vishnyakov, E. Friedman, and R.Ginosar. An asynchronous router for multiple service levels networks on chip. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 44–53. IEEE Computer Society Press, 2005.

[44] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks*. Morgan Kaufmann Publishers, 2003.

[45] D. Edwards and A. Bardsley. Balsa: An asynchronous hardware synthesis language. *The Computer Journal*, 45(1):12–18, 2002.

[46] O. Elissati, E. Yahya, S. Rieubon, and L. Fesquet. Optimizing and comparing CMOS implementations of the C-element in 65nm technology: self-timed ring case. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization, and Simulation*, pages 137–149. Springer, 2011.

[47] F. Feliciian and S. Furber. An asynchronous on-chip network router with quality-of-service (QoS) support. In *SOC Conference, 2004. Proceedings. IEEE International*, pages 274–277, Sept. 2004.

[48] T. Felicijan, D. Dielissen, and K. Goossens. Asynchronous TDMA networks on chip. Technical report, Philips Research Eindhoven, Jan. 2004. Technical Note 2004/00801.

[49] S. B. Furber, J. D. Garside, P. Riocreux, S. Temple, P. Day, J. Liu, and N. C. Paver. AMULET2e: An asynchronous embedded controller. *Proceedings of the IEEE*, 87(2):243–256, Feb. 1999.

[50] J. Garside and N. C. Audsley. Investigating shared memory tree prefetching within multimedia NoC architectures. In *Memory Architecture and Organisation Workshop*, 2013.

[51] R. Ginosar. Metastability and synchronizers: A tutorial. *IEEE Design & Test of Computers*, 28(5):23–35, 2011.

[52] M. D. Gomony, B. Akesson, and K. Goossens. Architecture and optimal configuration of a real-time multi-channel memory controller. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1307–1312, 2013.

[53] K. Goossens, A. Azevedo, K. Chandrasekar, M. D. Gomony, S. Goossens, M. Koedam, Y. Li, D. Mirzoyan, A. Molnos, A. Beyranvand Nejad, A. Nelson, and S. Sinha. Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. *ACM SIGBED Review*, 10(3):23–34, Oct. 2013.

[54] K. Goossens, J. Dielissen, and A. Rădulescu. The Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test of Computers*, 22(5):414–421, Sept-Oct 2005.

[55] K. Goossens and A. Hansson. The Æthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proc. ACM/IEEE Design Automation Conference (DAC)*, pages 306 –311, June 2010.

[56] S. Goossens, B. Akesson, M. Koedam, A. Beyranvand Nejad, A. Nelson, and K. Goossens. The CompSOC design flow for virtual execution platforms. In *Proceedings of the 10th FPGAworld Conference*, pages 7:1–7:6, New York, NY, USA, Sept. 2013. ACM.

[57] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '00, pages 250–256, New York, NY, USA, 2000. ACM.

[58] J. Gustafsson, B. Lisper, M. Schordan, C. Ferdinand, M. Jersak, and G. Bernat. ALL-TIMES – a European project on integrating timing technology. In T. Margaria and B. Steffen, editors, *Proc. Third International Symposium on Leveraging Applications of Formal Methods (ISOLA'08)*, pages 445–459. Springer, Oct. 2008.

[59] A. Hansson and K. Goossens. *On-chip interconnect with aelite / Composable and predictable systems*. Springer, 2011.

[60] A. Hansson, M. Subburaman, and K. Goossens. Aelite: a flit-synchronous network on chip with composable and predictable services. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 250–255, 2009.

[61] M. Harrand and Y. Durand. Network on chip with quality of service, Dec. 31 2013. US Patent 8,619,622.

[62] M. Herlev, C. K. Poulsen, and J. Sparso. Open core protocol (ocp) clock domain crossing interfaces. In *NORCHIP, 2014*, pages 1–6. IEEE, 2014.

[63] J. Hu and R. Marculescu. DyAD: Smart routing for networks-on-chip. In *Proceedings of the 41st Annual Design Automation Conference*, DAC '04, pages 260–263, New York, NY, USA, 2004. ACM.

[64] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 1, pages 234–239 Vol.1, Feb 2004.

[65] H. Hulgaard, S. Burns, T. Amon, and G. Borriello. An algorithm for exact bounds on the time separation of events in concurrent systems. *Computers, IEEE Transactions on*, 44(11):1306–1317, 1995.

[66] M. Imai and T. Yoneda. Improving dependability and performance of fully asynchronous on-chip networks. In *Proc. Intl. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 65–76. IEEE Computer Society Press, apr 2011.

[67] L. S. Indrusiak. End-to-end schedulability tests for multiprocessor embedded systems based on networks-on-chip with priority-preemptive arbitration. *Journal of Systems Architecture*, 60(7):553–561, 2014.

[68] A. Jerraya and W. Wolf. *Multiprocessor systems-on-chips*. Elsevier, 2004.

[69] F. Karim, A. Nguyen, and S. Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, 2002.

[70] E. Kasapaki. An EDA tool for the timing analysis, optimization and timing validation of asynchronous circuits. Master's thesis, Computer Science Department, University of Crete, Greece, Heraklion, Crete, Greece, Apr. 2008.

[71] E. Kasapaki, M. Schoeberl, R. Sorensen, C. Muller, K. Goossens, and J. Sparso. Argo: A real-time network-on-chip architecture with an efficient gals implementation. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2015. to appear.

[72] E. Kasapaki and R. B. Sørensen. Argo programming guide.

[73] E. Kasapaki and J. Sparsø. Argo: A time-elastic time-division-multiplexed NOC using asynchronous routers. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 45–52. IEEE Computer Society Press, May 2014.

[74] E. Kasapaki, J. Sparsø, R. Sorensen, and K. Goossens. Router designs for an asynchronous time-division-multiplexed network-on-chip. In *Proc. of Euromicro Conference on Digital System Design (DSD)*, pages 319–326, Sept. 2013.

[75] N. Kavaldjiev, G. Smit, P. Jansen, and P. Wolkotte. A virtual channel network-on-chip for GT and BE traffic. In *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, volume 00, page 6 pp., march 2006.

[76] J. Kessels and A. Peeters. The Tangram framework.

[77] H. Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[78] M. Krstić, E. Grass, F. K. Gürkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Des. Test*, 24(5):430–441, Sept. 2007.

[79] E. Lakis and M. Schoeberl. An SDRAM controller for real-time systems. In *Proceedings of the 9th Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2013.

[80] J.-Y. Le Boudec. Application of network calculus to guaranteed service networks. *Information Theory, IEEE Transactions on*, 44(3):1087–1096, May 1998.

[81] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest. Spatial division multiplexing: A novel approach for guaranteed throughput on NoCs. In *Proceedings of the 3rd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '05, pages 81–86, New York, NY, USA, 2005. ACM.

[82] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[83] J. Liu, S. M. Nowick, and M. Seokl. Soft MOUSETRAP: a bundled-data asynchronous pipeline scheme tolerant to random variations at ultra-low supply voltages. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 1–7, 2013.

[84] I. Loi, F. Angiolini, and L. Benini. Developing mesochronous synchronizers to enable 3D NoCs. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 1414–1419, 2008.

[85] Z. Lu, M. Millberg, A. Jantsch, A. Bruce, P. van der Wolf, and T. Henriksson. Flow regulation for on-chip communication. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 578–581, April 2009.

[86] D. Ludovici, A. Strano, D. Bertozzi, and G. N. Gaydadjiev. Mesochronous NoC technology for power-efficient GALS MPSoCs. In *Intl. Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip (INA-OCMC)*, pages 27–30. Association for Computing Machinery, 2011.

[87] J. Magott. Performance evaluation of concurrent systems using Petri nets. *Information Processing Letters*, 18(1):7–13, 1984.

[88] A. J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.

[89] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. V. Cummings, and T.-K. Lee. The design of an asynchronous MIPS R3000. In *Proceedings of the 17th Conference on Advanced Research in VLSI*, pages 164–181, 1997.

[90] P. McGee, M. Agyekum, M. Mohamed, and S. Nowick. A level-encoded transition signaling protocol for high-throughput asynchronous global communication. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 116–127, 2008.

[91] P. B. McGee and S. M. Nowick. An efficient algorithm for time separation of events in concurrent systems. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '07, pages 180–187, Piscataway, NJ, USA, 2007. IEEE Press.

[92] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual channels in networks on chip: Implementation and evaluation on Hermes NoC. In *Integrated Circuits and Systems Design, 18th Symposium on*, pages 178–183, Sept 2005.

[93] B. Mesgarzadeh, C. Svensson, and A. Alvandpour. A new mesochronous clocking scheme for synchronization in SoC. In *Proc. Int'l. Symp. Circuits and Systems*, pages II.605–II.608, 2004.

[94] D. Messerschmitt. Synchronization in digital system design. *Selected Areas in Communications, IEEE Journal on*, 8(8):1404–1419, Oct 1990.

[95] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1st edition, 1994.

[96] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 890–895. IEEE Computer Society Press, 2004.

[97] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The nostrum backbone-a communication protocol stack for networks on chip. In *VLSI Design, 2004. Proceedings. 17th International Conference on*, pages 693 – 696, 2004.

[98] MIPS Technologies Inc. MIPS32 M14Kc Core.

[99] I. Miro-Panades, F. Clermidy, P. Vivet, and A. Greiner. Physical implementation of the DSPIN network-on-chip in the FAUST architecture. In *Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '08, pages 139–148, Washington, DC, USA, 2008. IEEE Computer Society.

[100] M. Moadeli, A. Shahrabi, W. Vanderbauwhede, and M. Ould-Khaoua. An analytical performance model for the Spidergon NoC. In *Advanced Information Networking and Applications, 2007. AINA '07. 21st International Conference on*, pages 1014–1021, May 2007.

[101] A. Molnos, J. A. Ambrose, A. Nelson, R. Stefan, S. Cotofana, and K. Goossens. A composable, energy-managed, real-time MPSOC platform. In *Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on*, pages 870–876. IEEE, 2010.

[102] A. Molnos and K. Goossens. Conservative dynamic energy management for real-time dataflow applications mapped on multiple processors. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD'09. 12th Euromicro Conference on*, pages 409–418. IEEE, 2009.

[103] F. G. Moraes, A. Mello, L. Möller, L. Ost, and N. L. V. Calazans. A low area overhead packet-switched network on chip: Architecture and prototyping. In *VLSI-SOC*, pages 318–323, 2003.

[104] M. Moreira, B. Oliveira, F. Moraes, and N. Calazans. Impact of C-elements in asynchronous circuits. In *Quality Electronic Design (ISQED), 2012 13th International Symposium on*, pages 437–343, March 2012.

[105] F. Mu and C. Svensson. Self-tested self-synchronization circuit for mesochronous clocking. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 48(2):129–140, Feb 2001.

[106] C. Muller, E. Kasapaki, R. Sorensen, and J. Sparso. Synthesis and layout of an asynchronous network-on-chip using standard EDA tools. In *NORCHIP, 2014*, pages 1–6, Oct 2014.

[107] D. E. Muller. Asynchronous logics and application to information processing. In H. Aiken and W. F. Main, editors, *Proc. Symp. on Application of Switching Theory in Space Technology*, pages 289–297. Stanford University Press, 1963.

[108] D. E. Muller and W. S. Bartky. A theory of asynchronous circuits. In *Proceedings of an International Symposium on the Theory of Switching, Cambridge, April 1957, Part I*, pages 204–243. Harvard University Press, 1959. The annals of the computation laboratory of Harvard University, Volume XXIX.

[109] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, Apr 1989.

[110] C. Neeb, M. J. Thul, and N. Wehn. Network-on-chip-centric approach to interleaving in high throughput channel decoders. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 1766–1769. IEEE, 2005.

[111] L. S. Nielsen, C. Niessen, J. Sparso, and K. Van Berkel. Low-power operation using self-timed circuits and adaptive scaling of the supply voltage. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):391–397, 1994.

[112] N. Nikitin and J. Cortadella. A performance analytical model for network-on-chip with constant service time routers. In *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pages 571–578, New York, NY, USA, 2009. ACM.

[113] S. Nowick, K. Yun, P. Beerel, and A. Dooply. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Advanced Research in Asynchronous Circuits and Systems, 1997. Proceedings., Third International Symposium on*, pages 210–223, Apr 1997.

[114] S. M. Nowick and M. Singh. High-performance asynchronous pipelines: an overview. *Design & Test of Computers, IEEE*, 28(5):8–22, 2011.

[115] U. Ogras, P. Bogdan, and R. Marculescu. An analytical approach for network-on-chip performance analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 29(12):2001–2013, Dec 2010.

[116] I. M. Panades and A. Greiner. Bi-synchronous FIFO for synchronous circuit communication well suited for network-on-chip in GALS architectures. In *Proc. IEEE/ACM Intl. Symposium on Networks-on-Chip (NOCS)*, pages 83–92, 2007.

[117] I. M. Panades, A. Greiner, and A. Sheibanyrad. A low cost network-on-chip with guaranteed service well suited to the GALS approach. In *1st International Conference on Nano-Networks (Nano-Net)*, pages 1–5, 2006.

[118] P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 5, pages V–217–V–220 vol.5, May 2003.

[119] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *Computers, IEEE Transactions on*, 54(8):1025–1040, 2005.

[120] C. Paukovits and H. Kopetz. Concepts of switching in the time-triggered network-on-chip. In *Proc. IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 120–129, 2008.

[121] N. C. Paver, P. Day, C. Farnsworth, D. L. Jackson, W. A. Lien, and J. Liu. A low-power, low-noise configurable self-timed DSP. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 32–42, 1998.

[122] L.-S. Peh and W. J. Dally. A delay model for router microarchitectures. *Micro, IEEE*, 21(1):26–34, 2001.

[123] L. Pezzarossa. Hardware accelerators in network-on-chip based multi-core platforms. Master's thesis, Technical University of Denmark, Dept. of Applied Mathematics and Computer Science, 2014.

[124] C. Pitter and M. Schoeberl. A real-time Java chip-multiprocessor. *ACM Trans. Embed. Comput. Syst.*, 10(1):9:1–34, 2010.

[125] T.-C. Project. D3.2 - Simulation model of the self-timed NOC. `http://www.t-crest.org/page/results`, 2012.

[126] T.-C. Project. D3.4 - Report documenting the hardware implementation of the self-timed NOC. `http://www.t-crest.org/page/results`, 2013.

[127] T.-C. Project. D3.5 - Report on impact of asynchronicity on predictability of the NOC. `http://www.t-crest.org/page/results`, 2013.

[128] T.-C. Project. D3.6 - FPGA implementation of self-timed NOC. `http://www.t-crest.org/page/results`, 2014.

[129] T.-C. Project. D3.8 - Integration report of the full system implemented in an FPGA. `http://www.t-crest.org/page/results`, 2014.

[130] Y. Qian, Z. Lu, and Q. Dou. QoS scheduling for NoCs: Strict priority queueing versus weighted round robin. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 52–59, Oct. 2010.

[131] Radulescu, Dielissen, Pestana, Gangwal, Rijpkema, Wielage, and 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):4–17, 2005.

[132] C. Ramamoorthy and G. S. Ho. Performance evaluation of asynchronous concurrent systems using petri nets. *Software Engineering, IEEE Transactions on*, (5):440–449, 1980.

[133] K. Ramamritham and J. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, Jan 1994.

[134] K. Ramamritham, J. Stankovic, and P.-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. *Parallel and Distributed Systems, IEEE Transactions on*, 1(2):184–194, Apr 1990.

[135] Rijpkema, Goossens, Radulescu, Dielissen, van Meerbergen, Wielage, and Waterlander. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *Computers and Digital Techniques, IEE Proceedings-*, 150(5):294, 2003.

[136] S. Rusu. Clock distribution for high performance designs. In *Interconnect-Centric Design for Advanced SoC and NoC*, pages 125–152. Springer, 2005.

[137] E. Salminen, A. Kulmala, and T. D. Hamalainen. Survey of network-on-chip proposals. *white paper, OCP-IP*, pages 1–13, 2008.

[138] S. Saponara, L. Fanucci, and M. Coppola. Design and coverage-driven verification of a novel network-interface IP macrocell for network-on-chip interconnects. *Microprocessors and Microsystems*, 35(6):579–592, 2011.

[139] S. Sathe, D. Wiklund, and D. Liu. Design of a guaranteed throughput router for on-chip networks. In *System-on-Chip, 2004. Proceedings. 2004 International Symposium on*, pages 25 – 28, nov. 2004.

[140] M. Schoeberl. A time-triggered network-on-chip. In *International Conference on Field-Programmable Logic and its Applications (FPL 2007)*, pages 377–382, Amsterdam, Netherlands, Aug. 2007.

[141] M. Schoeberl. A Java processor architecture for embedded real-time systems. *Journal of Systems Architecture*, 54/1–2:265–286, 2008.

[142] M. Schoeberl, F. Brandner, S. Hepp, W. Puffitsch, and D. Prokesch. Patmos reference handbook.

[143] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C. W. Probst, S. Karlsson, and T. Thorn. Towards a time-predictable dual-issue microprocessor: The Patmos approach. In *First Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems (PPES 2011)*, pages 11–20, Grenoble, France, March 2011.

[144] M. Shams, J. Ebergen, and M. Elmasry. Modeling and comparing CMOS implementations of the C-element. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(4):563–567, Dec 1998.

[145] A. Siebenborn, O. Bringmann, and W. Rosenstiel. Communication analysis for network-on-chip design. In *PARELEC*, pages 315–320, 2004.

[146] D. Sigüenza-Tortosa, T. Ahonen, and J. Nurmi. Issues in the development of a practical NoC: the Proteo concept. *Integration, the VLSI Journal*, 38(1):95–105, 2004.

[147] D. Sigüuenza-Tortosa and J. Nurmi. From buses to networks. In *Interconnect-Centric Design for Advanced SoC and NoC*, pages 231–251. Springer, 2005.

[148] M. Singh and S. Nowick. MOUSETRAP: High-speed transition-signaling asynchronous pipelines. *IEEE Transactions on VLSI Systems*, 15(6):684–698, 2007.

[149] R. B. Sørensen, J. Sparsø, M. R. Pedersen, and J. Højgaard. A metaheuristic scheduler for time division multiplexed network-on-chip. In *Software Technologies for Future Embedded and Ubiquitous Systems (SEUS), 2014*. IEEE, 2014.

[150] J. Sparsø. Networks-on-chip for real-time multi-processor systems-on-chip. In *Proc. International Conference on Application of Concurrency to System Design (ACSD)*, pages 1–5, June 2012.

[151] J. Sparsø and S. Furber, editors. *Principles of asynchronous circuit design – A systems perspective*. Kluwer Academic Publishers, 2001.

[152] J. Sparsø, E. Kasapaki, and M. Schoeberl. An area-efficient network interface for a TDM-based network-on-chip. In *Proc. Design Automation and Test in Europe (DATE)*, pages 1044–1047, Mar. 2013.

[153] J. Sparsø and J. Staunstrup. Design and performance analysis of delay-insensitive multi-ring structures. Technical report, Department of Computer Science, Technical University of Denmark, Dec. 1992. pages 1-25.

[154] J. Stankovic and K. Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2(4):247–254, 1990.

[155] J. A. Stankovic. Real-time and embedded systems. *ACM Comput. Surv.*, 28(1):205–208, Mar. 1996.

[156] J. A. Stankovic. *Deadline scheduling for real-time systems: EDF and related algorithms*. Springer Science & Business Media, 1998.

[157] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Asynchronus Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on*, pages 46–53. IEEE, 2001.

[158] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[159] R. Systems. Rapitime white paper. Technical report, 2008.

[160] G. S. Taylor and G. M. Blair. Reduced complexity two-phase micropipeline latch controller. *IEEE Journal of Solid State Circuits - Institute of Elect and Electr Engineers*, 33(10):1590–1593, 1998.

[161] The International Technology Roadmap for Semiconductors. ITRS 2011 Edition – Design, 2011.

[162] L. Thiele and R. Wilhelm. Design for timing predictability. *Real-Time Systems*, 28(2-3):157–177, 2004.

[163] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P. Zaykov, Z. Petrov, B. Boddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Casse, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quinones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. parMERASA – multi-core execution of parallelised hard real-time applications supporting analysability. In *2013 Euromicro Conference on Digital System Design (DSD)*, pages 363–370, Sept 2013.

[164] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, and J. Wolf. MERASA: Multi-core execution of hard real-time applications supporting analysability. *Micro, IEEE*, 30(5):66–75, 2010.

[165] C. H. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Schalij. Asynchronous circuits for low power: a DCC error corrector. *IEEE Design & Test*, 11(2):22–32, 1994.

[166] K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, F. Schalij, and R. van de Viel. A single-rail re-implementation of a DCC error detector using a generic standard-cell library. In *2nd Working Conference on Asynchronous Design Methodologies, London, May 30-31, 1995*, pages 72–79, 1995.

[167] T. Verhoeff. Delay-insensitive codes—an overview. *Distributed computing*, 3(1):1–8, 1988.

[168] J. Wang. *Timed Petri nets: Theory and application*, volume 9. American Mathematical Soc., 1998.

[169] P. Wielage, J. Marinissen, M. Altheimer, and C. Wouters. Design and DfT of a high-speed area-efficient embedded asynchronous FIFO. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 853–858, 2007.

[170] D. Wiklund. Mesochronous clocking and communication in on-chip networks. In *Proc. Swedish System-on-Chip Conf*, 2003.

[171] D. Wiklund and D. Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *Proc. IEEE International Parallel and Distributed Processing Symposium, IPDPS 2003*, page 78a. IEEE Computer Society, 2003.

[172] P. Wolkotte, G. Smit, and J. Becker. Energy efficient NoC for best effort communication. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 197–202, Aug 2005.

[173] P. Wolkotte, G. Smit, N. Kavaldjiev, J. Becker, and J. Becker. Energy model of networks-on-chip and a bus. In *System-on-Chip, 2005. Proceedings. 2005 International Symposium on*, pages 82 –85, Nov. 2005.

[174] P. T. Wolkotte, G. Smit, G. Rauwerda, and L. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proc. 19th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2005*, page 155a, Apr. 2005.

[175] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10):1374–1396, 1995.

[176] L.-R. Zheng and H. Tenhunen. Wires as interconnects. In *Interconnect-Centric Design for Advanced SoC and NoC*, pages 25–54. Springer, 2005.

[177] S. Zheng, A. Burns, and L. S. Indrusiak. Schedulability analysis for real time on-chip communication with wormhole switching. *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, pages 1–22, May 2010.