Design and Performance Optimization of Asynchronous Networks-on-Chip

Weiwei Jiang

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2018

ABSTRACT

Design and Performance Optimization of Asynchronous Networks-on-Chip

Weiwei Jiang

As digital systems continue to grow in complexity, the design of conventional synchronous systems is facing unprecedented challenges. The number of transistors on individual chips is already in the multi-billion range, and a greatly increasing number of components are being integrated onto a single chip. As a consequence, modern digital designs are under strong time-to-market pressure, and there is a critical need for composable design approaches for large complex systems.

In the past two decades, networks-on-chip (NoCs) have been a highly active research area. In a NoC-based system, functional blocks are first designed individually and may run at different clock rates. These modules are then connected through a structured network for on-chip global communication. However, due to the rigidity of centrally-clocked NoCs, there have been bottlenecks of system scalability, energy and performance, which cannot be easily solved with synchronous approaches. As a result, there has been significant recent interest in combing the notion of asynchrony with NoC designs. Since the NoC approach inherently separates the communication infrastructure, and its timing, from computational elements, it is a natural match for an asynchronous paradigm. Asynchronous NoCs, therefore, enable a modular and extensible system composition for an object-orient design style.

The thesis aims to significantly advance the state-of-art and viability of asynchronous and globally-asynchronous locally-synchronous (GALS) networks-on-chip, to enable high-performance and low-energy systems. The proposed asynchronous NoCs are nearly entirely based on standard cells, which eases their integration into industrial design flows. The contributions are instantiated in three different directions.

First, practical acceleration techniques are proposed for optimizing the system latency, in order

to break through the latency bottleneck in the memory interfaces of many on-chip parallel processors. Novel asynchronous network protocols are proposed, along with concrete NoC designs. A new concept, called monitoring network, is introduced. Monitoring networks are lightweight shadow networks used for fast-forwarding anticipated traffic information, ahead of the actual packet traffic. The routers are therefore allowed to initiate and perform arbitration and channel allocation in advance. The technique is successfully applied to two topologies which belong to two different categories a variant mesh-of-trees (MoT) structure and a 2D-mesh topology. Considerable and stable latency improvements are observed across a wide range of traffic patterns, along with moderate throughput gains.

Second, for the first time, a high-performance and low-power asynchronous NoC router is compared directly to a leading commercial synchronous counterpart in an advanced industrial technology. The asynchronous router design shows significant performance improvements, as well as area and power savings. The proposed asynchronous router integrates several advanced techniques, including a low-latency circular FIFO for buffer design, and a novel end-to-end credit-based virtual channel (VC) flow control. In addition, a semi-automated design flow is created, which uses portions of a standard synchronous tool flow.

Finally, a high-performance multi-resource asynchronous arbiter design is developed. This small but important component can be directly used in existing asynchronous NoCs for performance optimization. In addition, this stand-alone design promises use in opening up new NoC directions, as well as for general use in parallel systems. In the proposed arbiter design, the allocation of a resource to a client is divided into several steps. Multiple successive client-resource pairs can be selected rapidly in pipelined sequence, and the completion of the assignments can overlap in parallel.

In sum, the thesis provides a set of advanced design solutions for performance optimization of asynchronous and GALS networks-on-chip. These solutions are at different levels, from network protocols, down to router- and component-level optimizations, which can be directly applied to existing basic asynchronous NoC designs to provide a leap in performance improvement.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank many people who have helped me throughout the time I have spent in Columbia. They have made this dissertation possible and successful.

First and foremost, I would like to express my sincerest gratitude to my advisor, Steven Nowick. During my entire PhD life, Steve is my greatest supporter. I really appreciate his patient guidance and the amount of time he spent with me for each research meeting. I still remember that there was more than one time when we worked together until early morning and successfully made a submission in time. He continually teaches me how to define problems, create systematic solutions, and build communication skills. Outside research, Steve is always kind and encouraging. We have a very good personal relationship. He has also spent much time in helping me with my job search. Without Steve, I could never become the researcher I am right now.

I would like to thank other members of my dissertation committee – Luca Carloni, Simha Sethumadhavan, Montek Singh and Gennette Gill – for their time, valuable feedback and insightful comments. In particular, I have enjoyed classes given by Luca and Simha at Columbia. They taught me important background and useful skills for my research. Gennette, a former postdoc researcher of Steve, also overlapped with me in Columbia. She, from time to time, gave interesting innovations and inspired my research.

To other faculty members of the Columbia University – Stephen Edwards and Martha Kim – I enjoyed interacting with you during group seminars.

I would like to thank Greg Sadowski and Wayne Burleson for giving me an opportunity to work at AMD Research for a 6-month internship. Many thanks to Greg's guidance and encouragement during my internship. The outcome of the internship was a great success. We jointly issued a US patent application. Our continued collaboration after my internship led to a paper publication in the industrial session of DATE-17, a top-level conference. Also, many thanks to Wayne, who volunteered to give the presentation in the conference.

Then, I would like to thank my colleagues in the Computer Systems Lab, as well as my friends. They made my daily working life enjoyable, and gave me a great amount of support through bad times: Kshitij Bhardwaj, Marco Cannizzaro, Yu Chen, Nicola Concer, Emilio G. Cota, Georgios Faldamis, Davide Giri, Cheoljoo Jeong, Jihye Kwon, Geoffray Lacourba, Hung-Yi Liu, Paolo Mantovani, Peggy McGee, Gabriele Miorandi, Michele Petracca, Luca Piccolboni, Baolin Shao, Christos Vezyrtzis and Young Jin Yoon. There are many more people who I cannot include in this list. I wish all of you good luck in the future of your life.

Finally, I would like to give thanks to my parents, Shaoming Jiang and Qin Xu. You have raised me and helped me build good attributes. Although you know almost nothing about my research, during my entire PhD period, you encouraged me when I was under pressure, and gave me very good advice and kept me on the ground during the good times. You also kept visiting me every year to make my working life a joy. Thank you is simply insufficient. I would never have made it without you.

Several grants have made this research possible: NSF Grant No. CCF-1527796, NSF Grant No. CCF-1219013, NSF Grant No. CCF-0964606, NSF Grant No. CCF-0811504, and the TA scholarship from Computer Science Department, Columbia University.

This thesis is dedicated to:

My parents – Shaoming Jiang and Qin Xu.

Chapter 1

Introduction

Digital systems continue to grow in complexity. As transistor scaling continues, nowadays, the number of transistors on individual chips is already in the multi-billion range. Meanwhile, more components, such as processing cores, accelerators and memories arrays, are integrated onto a single chip, as many-core architectures are targeted [17, 186, 200, 241]. As highlighted in the International Technology Roadmap for Semiconductors (ITRS), conventional digital design approaches are facing unprecedented challenges. These include dealing with the impact of increased variability, power and thermal bottlenecks, high fault rates, aging and scalability issues [166]. Also, success will rely on the designers' ability to conceive large-scale electronic engines under strong time-to-market pressure [18, 49].

As an alternative paradigm to address most of the design challenges faced by a conventional synchronous, i.e. centralized clock, approach, asynchronous design – or the use of a hybrid mix of asynchronous and synchronous components – has received continuous growth of interest over the last two decades. Asynchronous systems naturally support modular and extensible system composition for an 'object-oriented' design style, on-demand operation without extensive instrumented power management, and variability-tolerant design [166]. Asynchronous approaches have been studied and successfully applied to a number of high-performance processors, as well as low-power embedded consumer electronics. In addition, it is generally believed that some form of asynchrony will inevitably be required to enable novel computing paradigms, such as quantum cellular automata, nanomagnetics and self-assembled molecular electronics [167].

Design challenges for asynchronous circuits still remain, however, in a wide range of topics,

including synthesis (both logic- and high-level design and optimization), testing, performance and timing analysis, and formal verification [167], due to the unique features of asynchronous systems. Although the historical lack of commercial asynchronous CAD tools is gradually been remedied, at the current standing, only a few companies have developed custom in-house tools [13, 65, 227], which are typically specialized for particular design styles, and are not generally available to other researchers and designers [167]. Also, occasionally, when individual asynchronous components are implemented within a complex clocked system, speed benefits of asynchrony can be lost due to the synchronization needed at mixed-timing interfaces [167].

The use of networks-on-chip (NoC's) is another recent thread for resolving the increasing complexity in digital systems [26, 138]. NoC's borrow ideas from general computer networks, and provide structural communication infrastructures. In an NoC-based system, functional blocks are first designed individually and may run at different clock rates. These modules are then connected through a structured network for on-chip global communication [18, 49]. NoC's entirely separate computation from communication, hence they naturally support modular system composition and plug-and-play assembly of functional units and cores. Research on networks-on-chip is highly active since its invention. NoC's are very cost efficient, since they find a balanced point between network performance and the level of network resource sharing. Also, NoC's have been demonstrated to be able to handle a variety of real-world issues, such as congestion, fault tolerance and real-time constraints.

In recent years, there has been significant interest to combine the notion of asynchrony with NoC designs. Since the NoC approach inherently separates the communication infrastructure, and its timing, from computational elements, it is a natural match for an asynchronous paradigm. The increasing role of asynchrony in system communication is also highlighted in the recent ITRS report.

This thesis aims to significantly advance the state-of-art for high-performance asynchronous NoC's. The contributions include several aspects. First, practical acceleration techniques are proposed to focus on optimizing the network latency, while the throughput is still largely maintained or even improved. Latency bottlenecks have been identified in many NoC's, both synchronous and asynchronous, especially for the memory interfaces of on-chip multiprocessors. In this thesis, novel asynchronous network protocols, along with concrete NoC designs, are proposed to break

through the latency wall. The solutions improve the network performance considerably with only small overhead. Second, an advanced asynchronous NoC router is instantiated and validated using a leading industrial technology library. And for the first time, an asynchronous router is compared to a commercial energy-efficient synchronous NoC baseline in an advanced technology. Finally, we focus on a key component in an NoC – the arbiter. Arbiters are critical for NoC performance, and sometimes the bottleneck for the router as well as the entire network. The proposed efficient and scalable design can be directly applied in existing asynchronous networks, and promises its future use in opening up new NoC directions.

This introduction chapter first reviews the background for two different aspects – *(i)* the notion and potential benefits of asynchronous designs; and *(ii)* the concept of networks-on-chip and its advantage over traditional interconnects. It then highlights the contributions and organization of the thesis.

1.1 Asynchronous Design: an Alternative Paradigm

The section starts with a discussion of trends and challenges with traditional synchronous approaches. Then, the notion of asynchronous design, along with its advantages and challenges, is introduced. Finally, we present an overview of recent advances in the asynchronous world.

1.1.1 Trends and Challenges in Synchronous Design

Synchronous design, as shown in Fig. 1.1, is the most commonly-used style in digital systems. The system contains a collection of functional components which communicate using a global clock. The components only exchange information at clock ticks: computations are complete and new data must be ready before the arrival of the next clock tick.

The key benefit of a centralized clock is that it provides the designer with a discrete-time representation of the system, which enables a firm understanding of time within the design. Specification, implementation and verification are greatly simplified when the outputs only matter at the end of every cycle. However, as the complexity of an individual chip increases, the large-scale and high-density digital systems are posing significant challenges to the existing paradigm for globally synchronous design.

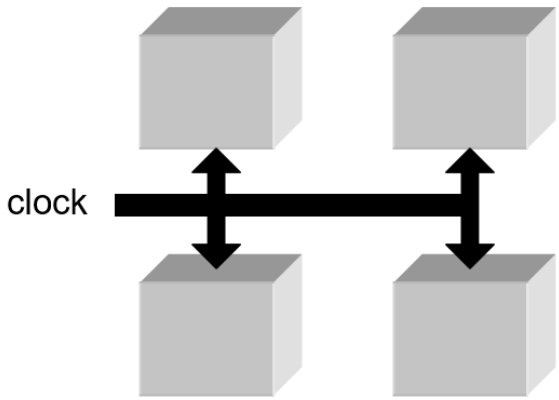


Figure 1.1: A synchronous system

Clock distribution and clock power dissipation. As chips grow in complexity, the task of distributing a single clock across an entire chip becomes harder. The global clock now must reach a greater number of components, and physical design challenges become severe. Also, as the target clock rate increases, the tolerable margin for clock skew decreases. On the other hand, about 1/4 to 1/3 of a synchronous chip's power may be burned by the clock distribution itself [207, 210]. Power consumption is now typically the key limitation for a system to achieve high operation speed, as well as to prolong battery lifetimes in portable devices.

Worst-case performance. In a synchronous design, all components operate at the same pace, and the clock rate is typically limited by the slowest component. Also, synchronous designs cannot easily exploit variable data-dependent completion times, thereby being bound to worst-case performance.

Design reuse and scalability. The synchronous approach becomes more difficult for design reuse. Each component in a new product may need re-design if the new product imposes a different clock rate from the old generation. However, as the system scales, there is a trend towards greater modularity and reusability of hardware components. Design reuse is becoming critical to keep a reasonable design cycle.

Interfacing with arbitrary environments. Frequently, a system environment is asynchronous. The external devices such as memories and other peripheral equipment run at different speeds. Synchronous designs need synchronizers to be added on the boundary of two clock domains, and therefore extra performance, area and power overhead are required.

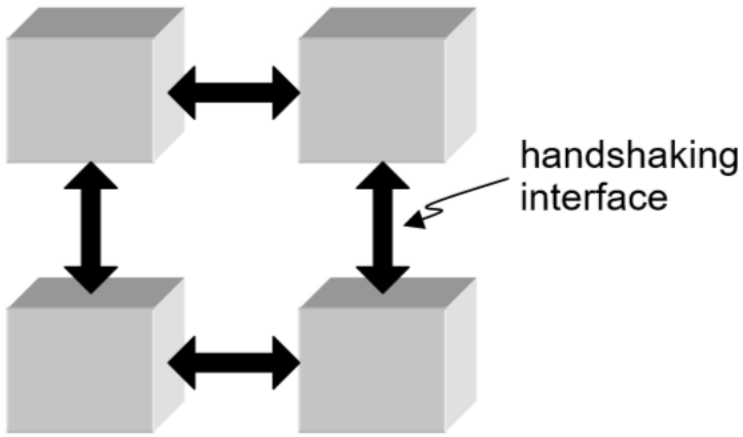


Figure 1.2: A asynchronous system with distributed control

1.1.2 Introduction to Asynchronous Design: Advantages and Challenges

In order to resolve the challenges faced by conventional synchronous designs, an alternative approach, named asynchronous or clockless design, has been targeted. A simplified model of an asynchronous system is illustrated in Fig. 1.2, which has no global clock. Components communicate through local handshaking channels, and can operate at different speeds.

While asynchronous design directly addresses the current challenges within the conventional clocked design and offers fundamental benefits, it still has not been fully accepted by mainstream industry due to its own unique challenges.

Several key advantages of asynchronous approach are first presented:

Lower power. Asynchronous circuits may save power consumption in terms of two aspects. First, the clock distribution circuitry is entirely eliminated as the global clock is no longer needed. Second, asynchronous components are only activated and consume power when stimulated by the arrival of new inputs. In contrast, synchronous designs have switching activity throughout the entire chip on every clock cycle, even when part of the design is idle and does no useful work. Although modern synchronous designs are able to selectively shut down the clock for inactive modules using clock-gating techniques, they are only partially effective and require extra overhead and design efforts. In particular, coarse-grain clock-gating can only turn on or shut down the entire clock network. The approach is not so useful when there are activities in only a portion of the circuits controlled by a single clock tree [23]. On the other hand, modern fine-grain clock-gating can selectively en-

able or disable individual flip-flops. However, as the clock-gating controls are added close to the leaves of the clock distribution tree, the technique largely does not yield power saving on the clock network. Also, extra control logic at leaves can pose additional challenges in clock tree balancing and physical layout [23, 101]. For asynchronous design, however, these power benefits are obtained automatically [229, 230], since inactive components do not consume any dynamic power.

Higher performance. Asynchronous systems can potentially obtain better performance than synchronous counterparts because they are not limited to worst-case timing assumptions. Components notify their environment when the operation is completed through local handshaking communication. Several early asynchronous designs take advantage of average-case performance, including speculative completion for high-performance dynamic adders [168], and an asynchronous implementation of the IA32 instruction-length decoder in Intel's RAPPID project [212]. Recently, average-case performance was also explored in NoC applications. In asynchronous NoC's, while the header flit needs to set up the path for the entire packet and incurs longer delay, the body and tail flits are processed much faster [80, 82, 97]. The overall performance can be determined by a combination of header, body and tail flits. In contrast, in synchronous NoC's, every flit typically advances at the same pace, which can limit performance.

Better scalability and modularity. Asynchronous components are self-contained. Smaller modules can be simply aggregated to build complex large systems. Also, asynchronous components can be modularly replaced without having impact to the correctness of the entire system. The ease of large-scale system integration was already shown in several recent industrial examples (STMicroelectronics' P2012 [17], IBM's TrueNorth [144] and the SpiNNaker neuromorphic chips [74, 76]), all of which used asynchronous or globally-asynchronous and locally-synchronous (GALS) approaches. In contrast, any small changes in a synchronous system can lead to a potential re-work for clock distribution [229].

Lower electromagnetic interference (EMI) and robustness. A synchronous design generates noise on its power-supply lines and emits electromagnetic radiation at the clock frequency and its higher harmonics. In digital-analog mixed-signal designs, these electromagnetic emissions can cause analog receivers to malfunction. Solutions such as shielding are costly. Asynchronous circuits, in contrast, have much smoother radiation spectra, and also much lower radiation power, thereby making them more compatible with sensitive analog circuitry [75]. A

good examples is the asynchronous 80C51 microcontroller fabricated by Philips Semiconductors. The chip was able to achieve lower EMI noise emissions so that the microcontroller could operate harmoniously with the radio-frequency (RF) data link, without the use of shielding [166, 230]. An enhanced version of the asynchronous microcontroller (SmartMX) is now used in more than 75 countries, for biometric passports and IDs [166]. In fact, asynchronous designs are overall more robustness to voltage, temperature and process variation. The approach has also been explored to handle extreme environments, such as for space missions [198].

While asynchronous approach exhibits large attractions due to the above advantages, the design style is not widely used because it has more challenging design requirements. A few of these challenges are now presented:

Hazard-free design requirements. Hazards are usually not a problem for clocked designs, as long as signals are stabilized before the arrival of each clock tick. On the other hand, in asynchronous style designs, every transition on certain critical wires can potentially matter. The fundamental challenge of asynchronous design is to develop optimization techniques at each level of the classic synthesis flow, from logic optimization to technology mapping, which simultaneously guarantee hazard-freedom [167].

Testability. The state of an synchronous system is typically determined after each clock cycle. Most testing is done through a 'single-stepped' approach: the design is paused every cycle, and the internal states of the circuits are extracted and compared with the expected results. In contrast, asynchronous systems cannot be easily slowed down or paused, because of the lack of a centralized control that clocking can provide. Also, an asynchronous operation or communication can complete for an arbitrary long time, instead of a given number of cycles. This non-determinism adds testing complexity. In addition, hazard-free requirements pose a special challenge to testability. The testing tools need to ensure the absence of hazards, besides the functional correctness. Finally, asynchronous designs use distinct storage elements from synchronous circuits and require different testing approaches. Many asynchronous datapath use level-sensitive latches for storage instead of edge-triggered flip-flops; asynchronous controllers typically store state on combinational feedback wires rather than using flip-flops [167].

'Computer-aided design' (CAD) tools. Automated CAD tools are important for the design, optimization and verification of large digital systems. Without having proper CAD tools, it is ex-

tremely difficult to obtain a good and correct design within a given time period. It is always challenging to develop asynchronous CAD tools which resolve the balance between two completing needs: *(i)* to provide compatibility with existing synchronous languages and CAD tool flows, and *(ii)* to design specification languages that best capture the fine-grain concurrency, distributed synchronization, and underlying clockless paradigm of asynchronous systems [167]. Also, at current standing, although automated CAD flows have been developed and used at several companies [13, 65, 227], these custom in-house tools are specialized for particular design styles, and are not generally available to other researchers and designers.

1.1.3 Asynchronous Design: Overview of Recent Success

Asynchronous design is not new [166]. The early years, from the 1950's to the early 1970's, included the development of classical theory (Huffman [223], Unger [223], McCluskey, Muller [155]), as well as use of asynchronous design in a number of leading commercial processors (Iliac I/II, Atlas, MU-5) and graphic systems (LDS-1). The middle years, from the mid 1970's to early 1980's, were largely an era of reduced activity, due to the advent of the synchronous VLSI era. The mid 1980's to late 1990's, represented a revival of asynchronous design, with the beginning of modern methodologies for asynchronous controllers [14, 43, 70, 163, 220], pipelines [72, 129, 169, 165, 244], initial CAD tools, initial commercial low-power consumer products (Philips Semiconductors [231]) and high-performance interconnection networks (Myricom). The modern era, starting from the early 2000's, includes a surge of activity, with modernization of design approaches, CAD tools and optimization techniques, migration into on-chip interconnection networks (Platform 2012 from STMicroelectronics [17]), several industrial update update at leading companies (IBM's FIR Filter [205], Intel's RAPPID project [212]) as well as startups (Achronix [217], Fulcrum [55]), and applications to emerging technologies (sub-/near-threshold circuits [135], sensor networks [66], energy harvesting [40], CT-DSP [36, 37, 38, 234, 235], neuromorphic computers [74, 144]).

In this sub-section, we will not fully cover all the activities in the entire history. Instead, a set of interesting and recent topics are selected and presented.

Asynchronous synthesis and optimization problems have been systematically addressed. In terms of logic synthesis, hazard-free logic minimization techniques were proposed for both two-

level [71, 163] and multi-level optimizations [123, 164]. Two alternative widely-used approaches for the specification and synthesis of hazard-free asynchronous controllers, burst-mode [162] and Petri-net based [44], were also proposed. In addition, an alternative approach, called *NULL Convention Logic (NCL)* was introduced by Karl Fant, targeted for the unified synthesis of both control and datapath [132]. For high-level synthesis, in an early asynchronous approach called *ACK* [102], systems are specified at a procedural level using VHDL with an add-on package of asynchronous channel abstractions, and the compiler maps the system to distributed asynchronous control and datapath. Resource sharing [7, 91] and scheduling [92] problems were also targeted.

A number of asynchronous specifcation languages and tool flows have also been implemented, including the influential *Caltech Synthesis Method* from Alain Martin's group [139], Philips' *Tangram* Compiler [225], along with its variant, Haste, and another enhanced public-domain version Balsa [12], *NCL-based* approaches [132, 152, 187], and synthesis tools which target optimizing pipelined systems [13, 45, 83] (including Proteus [13], which was developed at Fulcrum Microsystems).

Testing of asynchronous circuits provides both challenges and opportunities which are distinct from the testing of clocked circuits [166]. An advantage of some asynchronous circuits is that they exhibit the useful property of *self-checking*, entering a deadlock state when subjected to certain stuck-at faults. Fault diagnosis can take advantage of this additional failure mode [53, 176]. Also, full-scan approaches were proposed for Sutherland's micropipelines, in which the pipeline latches and their controllers were modified to introduce a clocked scan mode of operation [113, 173]. A similar full-scan approach was commercially used at Philips Semiconductors and was integrated into their Tangram design flow. In addition, as the overhead can be unacceptably high when fine-grained high-speed pipelines are used, several partial-scan [118] or non-scan [199] approaches were also developed. Finally, a novel approach to testing delay faults – in particular, timing constraint violations – in asynchronous pipelines has also been proposed [81].

A number of asynchronous processors have been built throughout the history of asynchronous design. Several leading proessors from the 1950's to 1970's used asynchronous circuits extensively, including ILLIAC I/II (University of Illinois), the Atlas and MU-5 (University of Manchester), the DDM-1 dataflow machine (AI Davis/Burroughs) [56], and the designs from the seminal Macromodules project [41]. The first modern single-chip asynchronous microprocessor was de-

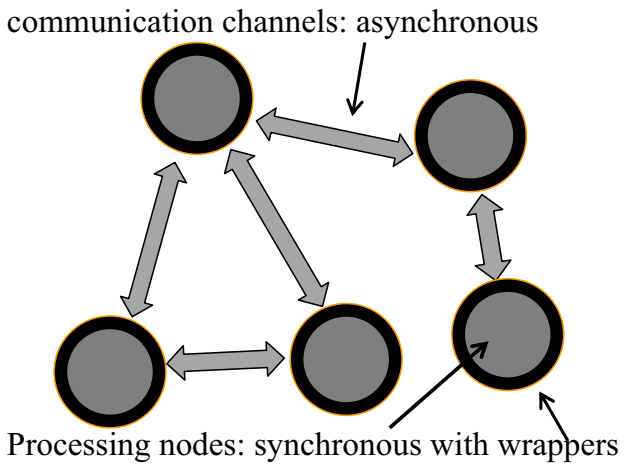


Figure 1.3: A GALS system

signed by Martin's group at Caltech in 1988 [142]. The 16-bit RISC processor was developed as a proof-of-concept to demonstrate the CHP complication approach, and the speed and robustness of QDI circuits. Another influential processor was the Amulet 1, which was developed in 1993 by Furber's group, as an asynchronous ARM using micropipelines. These two projects set a foundation for two decades of technical advances in all aspects of architecture, including pipeline circuits, cache and memory design, speculation, exception handling, and on-chip networks [166]. These first forays were quickly followed by more advanced designs, such as TITAC-1/2 [215], Amulet2e, Amulet3i [78], MiniMIPS [142]. Other important milestones, using novel architectures include the counterflow pipeline processor at Sun Microsystems Laboratories [209], an out-of-order architecture featuring precise exceptions at University of Utah [188], a super-pipelined multimedia processor at Sharp [219] and a low-power sensor-network processor from Cornell [66].

An alternative of constructing fully-asynchronous systems is called a globally-asynchronous locally-synchronous (GALS) approach, which was also first proposed in this period [34, 193]. In a GALS system, synchronous components, such as computational cores, memories and I/O units, are integrated through asynchronous communication. The communication fabric can be either distributed asynchronous channels, as an example shown in Fig. 1.3, or a centralized communication network. The latter structure effectively uses an asynchronous network-on-chip, which will be the core of the thesis and introduced in Section 1.2. GALS approach allows design reuse of synchronous functional blocks, and combines them with flexible asynchronous interconnect. The elimination of

global clocking provides a highly-scalable, low-power and robust mechanism for assembling complex systems [120, 166, 216].

Commercial applications and industrial experiments. Asynchronous techniques has been successfully migrated into commercial products by leading companies. Also, a number of industrial researches have shown the success of asynchronous design, though they are not translated to actual products.

At IBM research, a project was undertaken jointly with Columbia University to design a mixed synchronous-asynchronous implementation of a finite impulse response (FIR) filter in 2000 [205]. The development was used for real channels of modern disk drives. The implementation exhibits 50% of reduction in worst-case latency and 15% higher throughput over IBM's leading commercial clocked version in the same technology.

Intel led an experimental project, called RAPPID, in the mid 1990's, in which an asynchronous implementation of the IA32 instruction-length decoder was designed and fabricated [212]. The design significantly outperforms their commercial synchronous version: 3 times higher throughput, half the latency and half the power consumption with similar area cost.

In the late 1990's through early 2000's, Philips Semiconductors (now NXP) achieved much commercial success with its asynchronous 80C51 microcontroller [231]. The design exploited the advantage of asynchrony, and achieves 3-4$\times$ lower power than their synchronous version. In addition, it shows much lower electromagnetic interference (EMI) noise emission, so that the controller could operate along with the RF data link, with no use of shielding. The design was integrated into a wide range of consumer electronics, such as pagers, cell phones, smart cards and digital IDs, etc. [70, 163, 231]. More than 700 million copies were sold during that period.

Achronix Semiconductor, another example of asynchronous commercialization, invented the Speedster 22i family of FPGAs in the mid 2000's [217]. These high-performance FPGA chips were claimed to be the fastest FPGA's at release, which could operate at 1.5 GHz under 22nm technology. The design applies asynchronous fine-grain bit-level pipelines, which is the key to achieve fast operation.

Asynchronous design has also been used as a foundation for large-scale inter-processor communication, including the Torus routing chip [48], FLEETzero at Sun Microsystems Laboratories [42] and the terabit-rate commercial crossbar switches of Intel/Fulcrum [55]. Also, several recent neu-

romorphic processors – IBM's TrueNorth [144] and SpiNNaker [74], etc. – all of which use fully-asynchronous interconnection networks to integrate massively-parallel architectures with thousands of processing elements. (Detailed descriptions for some of these asynchronous NoC-based designs are presented in Section 1.2.4.)

Other emerging areas. One of the intriguing directions is the development of continuous-time digital signal processing (CT-DSP) processors [166], where input samples are generated depending on the actual rate of change of the input's waveform. The first general-purpose continuous-time DSP was proposed by Vezyrtzis et al. [235]. Unlike synchronous DSPs, it maintains its frequency response intact over varying sample rates and can support multiple input formats without any internal mode change,. Also, the approach eliminates all aliasing, and demonstrates a signal-to-error ratio for certain inputs which exceeds that of clocked systems.

Asynchronous designs are also explored to handle extreme environments, such as for space missions. An asynchronous 8-bit data transfer system is designed using NCL and the high-temperature SOI (HTSOI) process. The processor is proved to be fully operational over a 400 °C temperature range (from -175 °C to +225 °C), with good resilience to single event transients (SET's) [198].

1.2 Networks-on-Chip Introduction

Networks-on-chip is the other theme of the thesis. The concept was invented around 2000 [18, 49, 94], with several concrete NoC examples proposed around the same year [90, 238].

Nowadays, on-chip communication technology has become the limiting factor to achieve high performance and low power consumption for the digital systems, and needs to be considered as a first-class issue [18]. The reason is two-fold. First, modern chips can simply consist of tens or hundreds of computational cores built with billions of transistors, divided into multiple timing domains [207]. Success will rely on not only providing correct functional blocks, but also reliable operation of the interaction between these components. Second, whereas computational units, storage arrays and other functional blocks greatly benefit from transistor scaling, the delay and energy consumption for global communication does not scale down [95, 214].

Networks-on-chip borrow ideas from general computer networks, and provide structural communication infrastructures [18, 46, 138]. Much of the progress in this field is stimulated by the de-

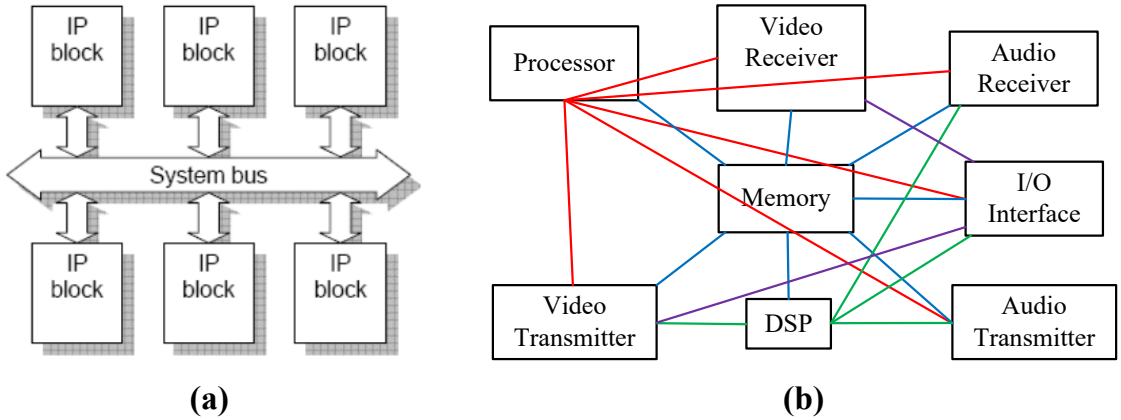


Figure 1.4: Conventional on-chip interconnects: (a) Shared bus; (b) Point-to-point (P2P)

signer's ability to complete large-scale digital system designs under strong time-to-market pressure in a cost-effective way [18]. Since NoC's separate the communication fabric from the computational units, which addresses the need of design modularity and allows computational modules to be designed in a plug-and-play fashion, the designers thus does not need to focus on the communication technology. The notion of a network-on-chip provides integrated solutions to a wide range of design problems, including high-performance computation, telecommunications, multimedia, and consumer electronics domains.

1.2.1 Conventional On-Chip Interconnects

Before the notion of NoC's was introduced, bus-based and point-to-point (P2P) interconnections, illustrated in Fig. 1.4, were widely used for on-chip communication. However, these traditional communication schemes have major limitations in scalability, and are no longer suitable for many modern digital systems.

In a shared bus structure, shown in Fig. 1.4(a), a centralized bus is used, and all functional blocks are attached to the bus. A transmission begins with a request to obtain the permission for utilizing the bus. After the use of bus is arbitrated and granted, data is transferred. Other transmissions are blocked until the current transmission is completed, and the bus is released.

Bus-based architectures are straightforward and simple to construct, but not scalable in terms of performance and power consumption for modern large-scale systems. First, the structure has

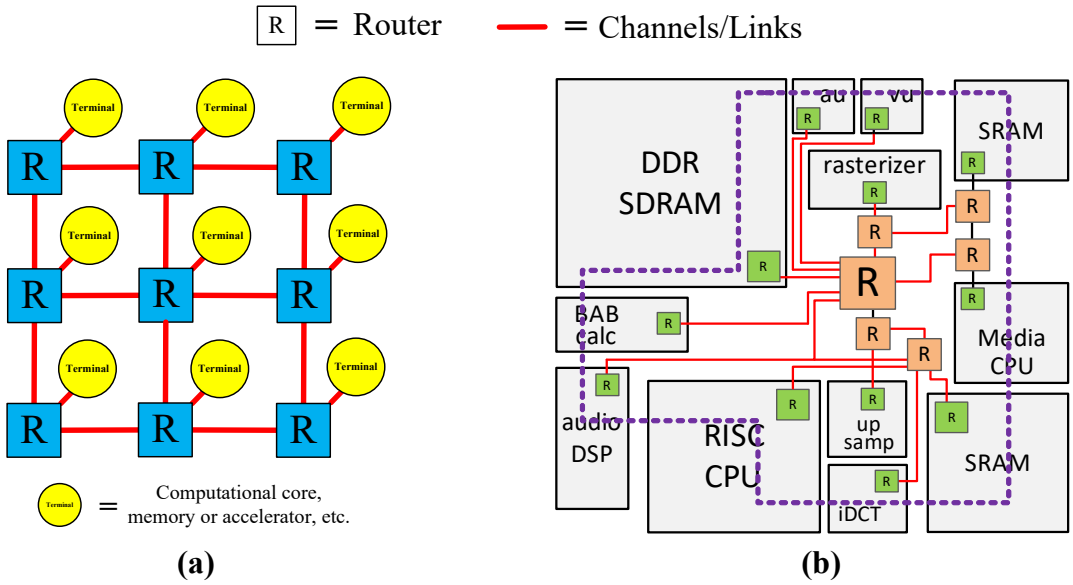


Figure 1.5: NoC examples: (a) A regular 2D-mesh NoC for a chip multiprocessor; (b) An irregular network for a multimedia SoC

almost no capability for parallel data transmission, which leads to unacceptable performance degradation as well as an extremely low efficiency for the use of the communication fabric. Also, since each transmission is effectively a broadcast on the bus and to all units, a large amount of power is consumed and wasted when the number of terminals increases.

Another conventional interconnect is point-to-point (P2P) architecture, shown in Fig. 1.4(b). A dedicated channel is added for each pair of functional units, wherever needed. While the structure usually has high performance, the number of channel links grows quadratically with the number of functional blocks, and becomes unacceptable for large networks. Also, the routing of the channel wires becomes extremely difficult when the system scales.

1.2.2 Network-on-Chip: Potential Benefits

Networks-on-chip, which borrow ideas from general computing networks, use separate infrastructure for on-chip communication between functional blocks [18, 46, 138]. However, they differ from general wide area networks in their smaller scale, and higher performance, as well as quite different chip-level cost metrics [18]. Fig. 1.5 shows two NoC examples. They belong to two different categories. The network on the left uses a regular mesh-based topology, in or-

der to connect a group of homogeneous functional units; On the other hand, the network on the right is an irregular NoC used in a multimedia system to connect heterogeneous IP blocks [21, 182].

A network-on-chip is composed of nodes and channels. Nodes, as shown in Fig. 1.5, are also called routers, which are used to direct the traffic in the network. Channels, also called links, connect the nodes. Data are transmitted between nodes through a channel, using a communication protocol. During a full transmission, the source functional block first communicates the attached router to initiate the data transmission. The data is then packetized by the source router and injected into the network. Then, packets are sent from a router to the next router, through a particular route in the network, until they arrive at the destination. Finally, packets are decoded and the data is sent to the receiver terminal.

Networks-on-chip have many benefits and provide a promising solution for modern digital system design. Several key advantages of NoC's are the following:

Improved scalability. While conventional interconnects will suffer unacceptable performance degradation and area/energy inefficiency with the increasing number of functional blocks they connect, networks-on-chip can easily scale to connect a large number of homogeneous or heterogeneous components, because NoC's are largely built in a structured manner. When the size of a system increases, NoC's are able to maintain stable performance with only moderate increases in area and power cost for the communication fabric.

Better performance and energy efficiency. The two conventional interconnect structures represent two extremes – a system bus is a fully shared infrastructure for all terminal blocks, while P2P channels are fully-dedicated for each source-sink pair. A NoC structure, however, finds a sweet point between the two extremes. Unlike bus-based structures, NoC's allow parallel data flows; on the other hand, unlike P2P interconnects, NoC's share resources between multiple transmissions. Therefore, networks-on-chip provide relatively high performance with small to moderate area and power cost – a much better performance and energy efficiency than traditional interconnects.

Design reuse. Since NoC's entirely separate communication infrastructure from computational units, the processing elements can be verified and optimized individually and employed in different platforms by means of a plug-and-play design style [21]. The use of pre-verified design blocks largely increases productivity, and time-to-market therefore can be kept as low as possible.

Design flexibility. As shown in Fig. 1.5, NoC-based systems provide the designer with a degree of freedom to instantiate different structures, and therefore are suitable for a wide range of digital designs. Networks can be tuned to have different topologies and routing strategies to accommodate different applications. To support this flexibility, many NoC design automation techniques have been developed, where routers and links are synthesized and connected together in an automated design flow. Design flexibility and network reconfigurability will be the key in providing plug-and-play component use.

1.2.3 Why the Asynchronous Paradigm Fits NoC Architecture

Many networks-on-chip are built using synchronous techniques, which are divide into two major categories: *(i)* fully synchronous NoC's and *(ii)* multi-synchronous NoC's. However, these synchronous NoC's are facing unprecedented performance bottleneck and design challenges.

A fully synchronous NoC has a single clock distributed across the entire communication network, which may run at a different clock rate from the functional IP blocks. It is a simple solution but somewhat a compromise to support a system with multiple clock domains. As the network needs to span across the chip, clock distribution is still very challenging. According to recent research, these NoC's consume up to 30% of the chip's power budget due to clock distribution, otherwise complex clock-gating techniques have to be applied [171]. Also, the NoC cannot run at a very high speed, usually 2-3$\times$ slower than the functional IP blocks because of the power budget. The speed constraints considerably limit the performance of the communication and the NoC may become the bottleneck of the entire system.

A multi-synchronous NoC uses a different approach. Each router uses the same clock as the terminal block it attaches to, or a dedicated and possibly faster clock [197]. Each router therefore belongs to a different timing domain and typically runs at different speeds. Data transmissions, therefore, require clock-domain crossing (CDC) at each hop, resulting in a significant performance degradation. The approach is limited to systems with small to medium sizes, otherwise the clock frequencies of the neighboring routers are forced to have close relationships.[1]

[1]Typically, a mesochronous approach is used. Neighboring routers run at the same frequency with undefined phase skews. CDC's between mesochronous blocks are much simpler, which can achieve much lower latency than that between two heterogeneous asynchronous blocks.

In order to overcome the limitations of the above synchronous techniques, an alternative approach is proposed: an asynchronous NoC is used to connect the synchronous components. The asynchronous paradigm is actually a natural match for NoC structures. Asynchronous interconnect eliminates the need for global clock management across a large network, while the synchronization is limited at only the source and the sink node. Effectively, the sync-to-async synchronization at the source node is trivial, since asynchronous NoC's can accept data at any time. Therefore, the async-to-sync synchronization at the destination node is the only complication. In fact, the asynchronous network, together with the synchronous functional blocks form an example of *globally-asynchronous locally-synchronous (GALS)* system [120, 216]. The entire communication network in a GALS system is asynchronous, and conversions to and from synchronous occurs only at the destination and source nodes.

Overall, the asynchronous NoC area is a promising arena where the integrative benefits of asynchronous design are making important inroads. This area is the focus of the thesis.

1.2.4 Synchronous and Asynchronous NoC's: Recent Advances and Future Trends

Since the invention of the concept of network-on-chip, significant progress has been made for both synchronous and asynchronous NoC's. Networks-on-chip have been incorporated in a number of academic and experimental design projects, as well as commercial applications.

Synchronous NoC Overview

Synchronous NoC's are still the mainstream design approach for on-chip communication. A large number of NoC designs have been proposed and a number of them have been fabricated [191]. These designs cover a wide range of topologies, including mesh [87, 98, 127, 146], ring [111], and tree structures [90, 115]. Different flow-control mechanisms, including circuit- [87, 111, 127] or packet-switching [87, 90, 98, 111, 146, 211], and different routing algorithms, from deterministic [98, 111, 127, 211] to adaptive routing [90, 98, 146], have been explored and applied. Also, several NoC's support guaranteed service (GS) and multiple service levels [87, 111, 127, 146], while others propose advanced power management techniques [28, 30, 54, 186].

Design flexibility and network reconfigurability are the key to provide plug-and-play component use. Network libraries are created to include all network components, such as routers and

channels. After designers describe communication requirements and constraints at high level of abstraction, the most suitable network-on-chip will be synthesized, which assembles proper library components [177, 190, 192, 211].

Network simulators are used to estimate and evaluate NoC performance and power consumption without having a concrete gate-level design. These auxiliary tools are important for NoC structural decisions at an early stage of the design flow, in order to minimize the design cost and the design cycle. Also, they are used for a more accurate NoC evaluation at mid- to late-stages. NoC simulators are not only proposed for conventional synchronous designs [3, 110, 237], but also for NoC's with extended wireless channels [33].

As size, complexity and integration density are making NoC's increasingly vulnerable, fault tolerance is another important topic [184]. Solutions for building reliable NoC's are provided at different levels. First, at a higher level, hardware redundancy can be provided. A backup spare copy can replace the original piece of hardware when a hardware failure is detected. At the level of network routing, adaptive routing allows data to be routed around the router with failure [181]; stochastic communication, on the other hand, transmits multiple copies of data and hopes at least one of them will be received [178]. Finally, at a lower level, error detection/correction coding schemes are used to detect or correct errors in each packet transmission [22].

Effectively, as of today, virtually all large-scale multicore processor chips are designed with NoC's [145]. As an early example, IBM's CELL multiprocessor used a ring structure to connect a main processor (PPE) and eight coprocessors (SPEs) [117]. The ring structure was recently extended and used in their POWER series multi-threaded chips [207]. On the other hand, 2D-mesh networks are also widely used for tile-based structures, such as in the TRIPS processor [88], as well as in Intel's Polaris 80-core Teraflops processor [232].

Asynchronous NoC Overview

While synchronous NoC's are powerful enough to support most of large-scale digital designs nowadays, asynchronous NoC's are likely to become a more suitable approach for larger parallel systems in the near future.

There has been a surge of research on asynchronous interconnect, due to its potential merits over conventional synchronous networks. A number of asynchronous and GALS NoC designs have been

proposed in the past decade. They cover a wide range of research goals, forming a solid foundation of the asynchronous NoC field. Chain is an early approach, using delay-insensitive (DI) codes on the channel [8]. The interconnect mitigates crosstalk on the channel links, and was successfully applied to an ARM-based smart-card chip. Several designs to support QoS have been proposed [27, 62, 63, 189]. They combine guaranteed service (GS) with best effort (BE) traffic, and enable multiple service levels. Fine-grain power management through dynamic voltage and frequency scaling (DVFS) has been proposed in a GALS NoC structure [16]. Reliability and fault tolerance was also targeted at the different network levels [99, 100]. Automated design flows are also being developed, leveraging commercial synchronous CAD tools, which use directives to meet asynchronous timing without introduction of control hazards [80, 147, 221]. A recent asynchronous NoC applies time-division-multiplexing (TDM), which traditionally requires a common time reference and was usually implemented synchronously [112]. The design also has interesting elastic timing properties that allow it to tolerate significant skew between network interfaces. Multicasting capability has also been addressed, which can support many emerging NoC technologies [24].

Performance and power benefits. Performance and power benefits of asynchronous NoC's over their synchronous counterparts have been demonstrated in both high-performance shared-memory chip multiprocessors [97] and Ethernet switch chips [133].

As a recent example, STMicroelectronics' Platform 2012 (P2012) included a fully-asynchronous network-on-chip [17]. implementing an advanced highly-customizable accelerator-based GALS system. The asynchronous NoC is the key to facilitate its fine-grain power, reliability and variability management. The first prototype chip integrates 4 clusters, each with 16 synchronous processors, and delivered 80 GOPS performance with only 2W power consumption. Its performance efficiency, i.e. performance per unit area and unit power, is much better than several recent Quadro and Nvidia commercial GPU's.

Another example is a recent hybrid packet/circuit-switched NoC from Intel Lab [35]. The NoC can operate in two modes: a normal synchronous packet-switched mode and a fast *'source-synchronous'* circuit-switched mode. The latter mode is effectively implemented by asynchronous operations. The chip was fabricated using advanced 22 nm tri-gate CMOS technology. The *'source-synchronous'* mode (which is actually a fully-asynchronous mode) shows a 2.7$\times$ increase in throughput and a 93% reduction in end-to-end latency, compared to its normal synchronous mode.

The performance and power benefits of asynchronous NoC's will be again demonstrated in this thesis, through a head-to-head comparison between an asynchronous router vs. a synchronous commercial counterpart in advanced technology (see Chapter 6).

Neuromorphic computers. An interesting emerging domain where asynchronous and GALS NoC's play an important role, is in building neuromorphic chips [166]. These brain-inspired architectures involve billions of computational units, and each has closely-coupled local memory. In a neuromorphic system, neurons (or clusters of neurons) are mapped to processing elements or cores, and synapses are mapped to communication channels. The communication is event-driven and high-fanout. An asynchronous NoC architecture, therefore, is a natural fit for neuromorphic computing, with its scalability, ease-of-integration and distributed communication.

As a recent example, IBM's TrueNorth is a 5.4-billion transistor neuromorphic chip, the largest chip when it appeared [5, 144]. The chip uses a GALS architecture: the design integrates 4096 synchronous neurosynaptic cores, modeling 1 million neurons and 256 million synapses; the cores are then integrated by fully-asynchronous on-chip communication. The power consumption of the chip is extremely low: only 63 milliwatts when processing a 400x240 video input at 30 frames/second.

Another example for neuromorphic computer is SpiNNaker – *Spiking Neural Network Architecture* [74, 76]. The chip consists of up to 1 million ARM9 cores with 7T bytes RAM, which is also GALS. The cores and memories are synchronous components, connected by an asynchronous communication fabric. The computer shows comparable performance to modern PCs in several applications with ultra-low energy consumption.

Commercial applications. Asynchronous NoC's have already been applied to commercial products. One example is Intel's recent FM5000/6000 series Ethernet switch chips, which are based on its acquisition of the asynchronous startup Fulcrum Microsystems in 2011 [55]. The products support industry-leading 40 gigabit Ethernet, include a fully-asynchronous high-speed crossbar switch that provides a maximum of 640 Gbps bandwidth, 400 ns cut-through latency and high energy efficiency.

Future Directions

Higher-dimension chips. One of trends for NoC structure is the growth from 2D topologies to 2.5D and 3D [1, 61, 174]. A 3D chip is a stack of multiple device layers with direct vertical interconnects tunneling between layers [51, 154]. Compared to conventional 2D networks, stacked

3D interconnects have a lower network diameter,[1] resulting in potentially higher performance. So far, recent research in this area also has shown that 3D networks can achieve higher packing density, reduce the total wiring cost, and are more immune to noise [50, 109, 174, 222, 236]. The 3D structure, therefore, becomes a promising architecture for future many-core parallel systems.

Photonic networks-on-chip. On-chip photonics have been proposed as radical low-power and low-latency alternatives to the conventional wire-based NoC's. Fundamentally, optical techniques allow concurrent multiplexing of the channel links. Multiple data streams using different wavelengths can transmit data simultaneously on a single link, called a waveguide [20, 137, 195]. The wavelength-division-multiplexing (WDM) technique effectively enables ideally an all-to-all P2P interconnect topology without large overhead which is usually associated with wired NoC's. In photonic NoC's, transmission does not need to wait for arbitration to access network resources. Additionally, the communication is completed at the speed of light. However, many challenges, such as manufacturing cost and yield, temperature sensitivity of photonic devices, design complexity and testing issues, need to be overcome before silicon-photonic integration can be widely used [20].

Wireless networks-on-chip. Wireless communication channels can be added on top of a conventional wired network to create low-latency and high-throughput single-hop communication between two nodes that are physically far apart [60, 158]. By adding these virtual long links, the average hop between two routers is considerably reduced, resulting in potentially much higher performance. While millimeter-wave antennas can be manufactured using current manufacturing techniques, several challenges, such as transceiver design issues and robustness of noisy wireless channels, still need to be resolved before wireless interconnect technologies can be widely adopted [60, 158].

New applications: artificial intelligence and autonomous driving. Networks-on-chip also have shown importance roles in emerging applications, such as AI and autopilot. AI processors typically require massively parallel computing and include high-performance accelerators, while self-driving applications must satisfy various timing constraints (e.g. for emergency braking control). NoC's are considered to be the best solution because of their flexibility to achieve high performance and accommodate various constraints.

A number of AI processors have been proposed in the past few years since significant research

[1] The network's diameter is the maximum number of hops that data can be transmitted.

progress has been made in AI field recently [39, 114, 116]. All of them include an array of processors and accelerators connected through a NoC. On the other hand, Mobileye (acquired by Intel) and NVIDIA recently announced their autopilot platforms, both of which are also NoC-based systems [6, 124, 145].

1.3 Research Focus

The thesis aims to significantly advance the state-of-art and prove the viability of asynchronous and GALS networks-on-chip, for the use of high-performance and low-energy systems. The contributions are instantiated in three different directions.

First, practical acceleration techniques are proposed to focus on optimizing the system latency, while the throughput is still largely maintained or even improved, in order to break through the latency bottleneck in the memory interfaces of many on-chip parallel processors. While these novel acceleration technologies lift the network performance considerably, the designs are very cost efficient in terms of both area and power.

Second, an advanced high-performance and low-power asynchronous router is instantiated and validated using a leading industrial technology library. And for the first time, we compare an asynchronous router design to a commercial energy-efficient synchronous NoC baseline in an advanced technology. In addition, a semi-automated flow was used to develop the router design. The work was completed in AMD Research, which shows great interest for further research and investments in this area.

Finally, a high-performance multi-resource asynchronous arbiter design is developed. This small but important component can be directly used in existing asynchronous NoC's for performance optimization. In addition, the well-structured stand-alone design promises its future use in opening up new NoC directions.

This section only focuses on the first contribution, presenting the challenges and goals of network latency acceleration. All the three contributions will be presented in details later in Section 1.4.

1.3.1 An Overview of NoC Acceleration

Network performance is drawing an increasing attention for recent NoC designs. Both latency and throughput are primary metrics for performance evaluation. For on-chip multiprocessors, other

high-performance parallel computing systems, and applications with hard real time constraints, network latency is often more important. These networks typically have small to medium traffic load, with occasionally bursty traffic. On the other hand, throughput is critical for many GPU's and multimedia-based systems – large amount of data needs to be transmitted at a stable speed.

In particular, system latency draws rising attentions for memory (or cache) interfaces for shared-memory parallel processors, and has been identified as a critical limitation [89, 185]. In practice, the communication network between cores and cache typically has light traffic, as low as only 5-10% of network saturation, and the key bottleneck is transport time.

Recently, for high-performance NoC's, increasing interest has been raised in using transition-signaling (i.e. 2-phase) single-rail bundled data protocol on inter-router channel communication [79, 80, 97, 165]. (The detailed explanation of the protocol is presented in Section 2.1.) The transition-signaling only contains a single round-trip communication per data transaction, which can in principle double the channel throughput, compared to a 4-phase protocol. Simultaneously, the single-rail data encoding, compared to most delay insensitive codes, can significantly increase the coding efficiency, resulting in a much smaller area and a much better energy efficiency. It also allows design reuse of synchronous components in asynchronous implementation. Therefore, in this thesis, we select transition-signaling single-rail bundled data protocol for all the NoC designs.

1.3.2 Research Challenges for NoC Acceleration

There are two inevitable overheads when we use a network-on-chip: *arbitration overhead* and *transmission overhead*. The arbitration overhead is the time spent on resources acquisition, i.e. routers and channels, before they can be used for data transmission. The acquisition occurs with the arrival of the header flit, and the acquired resources are held for the rest of flits in the packet. The acquisition is usually completed on hop basis. Therefore, data needs to be stalled at each router on the transmission path, waits for resource allocation for the next hop, and cannot advance seamlessly. On the other hand, the transmission overhead is the actual time for advancing the data continuously through the routers and links along the path, assuming no arbitration overhead. Transmission overhead always exists, unless each pair of source and sink is directly connected through a dedicated link.

It is always challenging to target both *arbitration overhead* and *transmission overhead* in a

single NoC solution. A smaller arbitration overhead usually leads to a more intelligent NoC design, and a more complicated router, while a smaller transmission overhead is in favor of a simpler and faster router design. One of the key focus areas of this thesis is to address both of the above two overheads for network acceleration, and optimize the network latency.

1.3.3 Synchronous NoC Acceleration: Existing Approaches and Bottlenecks

A large amount of research has been pursued for synchronous NoC acceleration. The approaches largely fall into two categories: *(i)* router speed-up techniques [93, 156, 157, 172] and *(ii)* router bypassing techniques [122, 131, 150, 218]. While these techniques are efficient and have been widely used for many existing synchronous networks, each of them has its own shortcomings.

Router speed-up techniques only target the *transmission overhead*, while *arbitration overhead* is not touched at all. In these techniques, speculation is used to allow parallel operations, which belonged to two different pipeline stages and were only able to complete serially, to be performed in a single clock cycle. Therefore, the pipeline depth is decreased; the latency of data processing in the router is therefore reduced. (More details on synchronous router pipeline optimization are presented in Section 3.4.4.) Most fast router designs, however, result in a more complex router with moderate to large area and power overhead due to speculation. Also, the clock rate is slower, which partially cancels the benefit of having a shorter pipeline [156, 157, 172].

Router bypassing techniques provide additional dedicated high-speed routes which bypass intermediate routers, therefore resulting in significant performance improvements. Unlike router speed-up techniques, the approach decreases both *arbitration overhead* and *transmission overhead*. A bypassing route typically only eliminates arbitration for a short distance every time (e.g. 3-4 hops in [122]), and accelerates the transmission speed for the segment. Also, moderate to high hardware overhead is required to construct bypassing routes: additional VCs are used in [122, 131], while an extra circuit-switched network plane is added in [157, 172], which almost doubles the network area in the latter case. In addition, the non-bypass traffic will be slowed down, if it loses the arbitration to access the bypassing route, as well as due to a more sophisticated router design.

1.3.4 Asynchronous NoC Acceleration: A Missing Research Area

There has been only limited work for performance optimization of asynchronous NoC's. Since resource arbitration and data transmission have to be carried out in continuous time without being

aligned to clock cycles, there is no easy way to extend a synchronous acceleration technique to asynchronous. Such extension, if possible, requires almost a complete re-design for network protocol, as well as for the router itself. Performance optimization for asynchronous NoC's thus becomes an understudied area. One of the key targets for this thesis is to close this gap. We propose several novel high-performance asynchronous NoC designs, and open new research directions for the future.

1.4 Contribution of the Thesis

In sum, this thesis presents a set of design solutions for the advancement of asynchronous on-chip interconnection networks. The contributions are instantiated in three aspects, including: *(i)* latency acceleration techniques to break through the latency wall in many high-performance CMP's; *(ii)* an industrial implementation of an advanced asynchronous router in a leading industrial library, and its comparison to a commercial synchronous router counterpart; and *(iii)* an asynchronous multi-resource arbiter design for direct use in existing asynchronous NoC's or opening new NoC directions for the future.

These three contributions are presented below in detail:

- ***Latency acceleration techniques for asynchronous NoC's***

  A novel asynchronous network protocol is proposed, along with concrete asynchronous network designs. We introduce a new concept: the use of a *monitoring network*. Monitoring networks are lightweight shadow networks used for fast-forwarding anticipated traffic information. The routers are notified with incoming traffic in the near future, and therefore allowed to initiate and perform arbitration and channel allocation in advance. While in a non-optimized network, each header packet has to request the arbiter and win the arbitration before it can be transmitted on the channel, the proposed solution entirely conceals the arbitration overhead. To support this approach, entirely new communication protocols and router node designs are proposed.

  The new asynchronous NoC's only have relatively small overhead. Although *'early arbitration'* techniques have already been proposed in several synchronous designs [2, 119,

121], those solutions are inadequate: Most require major hardware resource allocation – multiple extra VCs, hybrid networks with additional network planes, or wide control or monitoring channels. In contrast, our proposed NoC's only contain lightweight monitoring networks with small channel widths, without creating an extra network plane; nor extra VCs are needed. In addition, considerable and stable latency improvement are observed in all kinds of traffic patterns, with moderate throughput gains.

The technique is successfully applied to two topologies – a *mesh-of-trees (MoT) variant* structure and a *2D-mesh* structure. They are two important and quite distinct design points, for different application domains. The MoT topology is largely used for memory interfaces of on-chip multiprocessors (CMP's), to connect between cores and caches (L1, L2, or last-level caches). As these networks typically handle low to moderate traffic density, latency is the key of our focus, and throughput is only moderately important. On the other hand, 2D-mesh is heavily used in a wide range of different applications, from consumer electronics to high-performance parallel computers. Latency is important for all these domains, while throughput is also critical, since there are many applications (e.g. multimedia-based applications) that are throughput dominated. Our goal is therefore two-fold: optimization for latency while maintaining, or improving, throughput.

The ideas of the two NoC's are quite similar, the *'early arbitration'* protocols are, however, fundemantally different for the two topologies.

(i) Accelerated NoC for a mesh-of-trees (MoT) variant topology. In the MoT network, a coarse-grain monitoring synchronization protocol is used; All related routers on the paths are informed with the anticipated traffic information in rapid sequence. The protocol is simple and well suited for the topology, where the level of congestion and traffic interference is low.

(ii) Accelerated NoC for a 2D-mesh topology. On the other hand, in the 2D-mesh network, a finer-grain router-by-router early arbitration is forced. While the coarse-grain synchronization protocol is suitable for the MoT structure, it does not work well for 2D-mesh. This is because there are much more traffic interference in mesh-based topologies: pre-allocation of routers too early for one data stream can block other data streams from using this node for a large amount of time, resulting in an unacceptable performance degradation. Hence, more tightly-

coupled synchronization of the monitoring token and the data flow is required.

- ***An industrial instantiation for a high-performance and low-power asynchronous router***

  The work presents a real industrial instantiation of an advanced high-performance and low-power asynchronous router in a leading commercial technology library. This work was carried out in collaboration with AMD Research. The proposed router integrates several advanced router design techniques, including a low-latency circular FIFO for buffer design, and a novel end-to-end credit-based virtual channel (VC) flow control. This asynchronous router is then validated and compared to an AMD commercial synchronous router using a realistic 14nm FinFET library. This is the first such comparison, to the best of our knowledge, for an asynchronous router vs. an industrial synchronous baseline using an advanced technology library. Unlike other baselines for research purposes, the synchronous design is used in recent high-end AMD processors and graphic products. The results are thus more persuasive and closer to the reality.

  In addition, a semi-automated design flow is created for the asynchronous router design. In particular, industrial tools are used for place-and-route (P&R) and design validation. These tools leverages from a standard synchronous design flow and therefore open real future opportunities for industrial asynchronous NoC designs.

  In summary, asynchronous NoC's have been shown to be a very good solution for future industrial products. Also, AMD has shown interests for further research in this area.

- ***A pipelined high-throughput asynchronous multi-resource arbiter***

  Arbiter is critical for NoC performance, and sometimes the bottleneck for the router as well as the entire network. A new fine-grain pipelined asynchronous multi-resource arbiter is proposed. Multi-resource arbitration is used in existing asynchronous NoC's for VC allocation. Also, it serves as a key component for a potential future NoC direction – spatial-division multiplexing (SDM) NoC's. A SDM NoC divides inter-router links into identical sub-channels for concurrent data transmission. SDM technique is proved to be an alternative strategy to VCs for mitigating head-of-line blocking, in order for high network throughput.

  The new arbiter design aims for high throughput while retains a low latency. The allocation of a resource to a client is divided into several steps. Multiple successive client-resource pairs

can be selected rapidly in sequence, and the completion of the assignments can overlap in parallel. Besides the NoC applications, it can also be treated as a great stand-alone design for future large-scale parallel systems.

The research for this dissertation led to a patent application [107], and to several publications [67, 104, 105, 106].

1.5 Organization of the Thesis

This thesis is organized as follows.

Chapters 2 and 3 provide background for the thesis. Chapter 2 introduces basic concepts of asynchronous design, including different types of handshaking, data encoding schemes, special asynchronous components and several asynchronous pipelines. Chapter 3 covers fundamentals of networks-on-chip. This includes a variety of network topologies, routing and flow control methods, as well as typical synchronous router architectures and operations.

Chapters 4, 5, 6 and 7 form the core of this thesis. Chapters 4 and 5 present a novel network protocol for end-to-end latency acceleration using monitoring techniques. This new protocol is applied to a Mesh-of-Trees variant topology [67] and the 2D-mesh network [105], respectively, in these two chapters. Chapter 6 presents a real industrial instantiation for a high-performance and low-power asynchronous NoC router, using a partial automated design flow [104, 107]. The router shows dominating results in all key metrics, compared to a leading commercial synchronous counterpart. This indicates that industry has shown strong interests in developing asynchronous NoC's, and it is potentially a viable solution for future large-scale digital systems. In Chapter 7, a high-throughput asynchronous multi-resource arbiter is proposed [106], which optimizes a core component for existing and potentially future asynchronous NoC routers. It is also a good stand-alone design for future parallel systems.

Finally, concluding remarks and future research directions are presented in Chapter 8.

Chapter 2

Background: Asynchronous Design Basics

Since all the proposed networks-on-chip in the thesis are designed in asynchronous style, a review of basic background material for asynchronous designs is now given. For more details on asynchronous foundations, see [165, 166, 167].

The chapter begins with the two most fundamental decisions for asynchronous designs: the selection for the type of handshaking protocol between adjacent stages (Section 2.1) and the selection of the way that data is encoded (Section 2.2). Then, more advanced asynchronous background is introduced, including several special but widely-used asynchronous elements and components (Section 2.3), as well as asynchronous pipelines (Section 2.4). These are requirements for understanding the more technical sections later on.

2.1 Handshaking Protocols: Control Signaling

Components of asynchronous circuits communicate with each other based on local handshaking interfaces. A typical point-to-point communication is illustrated in Fig. 2.1. The structure contains a *sender*, a *receiver*, and a *handshaking channel*. The channel typically has two control signals – a *request (REQ)* and an *acknowledgment (ACK)*. The role of REQ is to indicate when the data is valid and ready to be sent by the sender, while ACK indicates when the receiver has successfully received the data and ready to accept another one. Alternatively, the *REQ* may indicate a request for data,

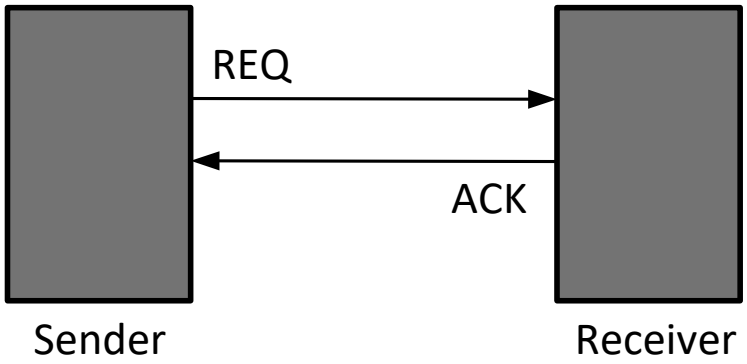


Figure 2.1: An asynchronous point-to-point communication channel

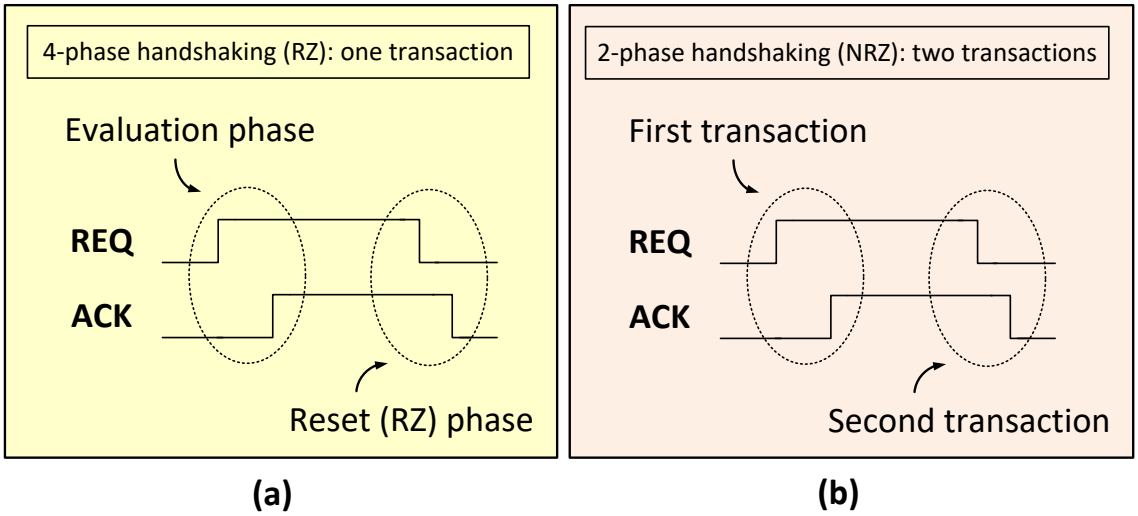


Figure 2.2: Asynchronous handshaking protocols:
(a) four-phase (RZ); (b) two-phase (NRZ)

and the *ACK* indicate a confirming response.

The two most common handshaking protocols are shown in Fig. 2.2: *(i)* a *four-phase protocol (return-to-zero (RZ))*, and *(ii)* a *two-phase protocol (non-return-to-zero (NRZ))*, also known as *transition-signaling* [25, 31, 159, 193, 213]. In a *four-phase protocol*, each transaction contains two round-trip communications between the *sender* and the *receiver*. In contrast, the two-phase protocol only uses a single round-trip communication per transaction.

Other communication protocols include *pulse mode* [179, 180] and *single-track handshak-*

ing [228]. These non-standard protocols attempt to combine the efficiency of two-phase signaling (i.e. no *RZ* phase) with the benefits of four-phase signaling (i.e. level-based implementation). However, they have complexities in timing requirements, implementation, and signal integrity, and are used in only a few applications, hence they are outside of the current research scope.

2.1.1 Four-Phase Protocol

A single transaction of a *four-phase handshaking protocol* is shown in Fig. 2.2(a). Both *REQ* and *ACK* are initially at zero. The protocol is composed of two operating phases (but four wire transitions): an *evaluation phase* and a *reset phase*, with two events in each. During the *evaluation phase*, the sender initiates the operation by asserting *REQ* high, indicating the data is ready; the receiver then asserts *ACK* high to the sender, indicating the data is received and no longer needed. Next, in the *reset phase*, the *REQ* and *ACK* are deasserted in turn.

In a four-phase protocol, the state of channel always returns to a unique value after each transaction, and therefore usually leads to a simpler hardware design. However, a major penalty of having a *reset phase* is a decreased throughput performance due to two communication round-trips per transaction.

2.1.2 Two-Phase Protocol

The *two-phase handshaking protocol* is illustrated in Fig. 2.2(b). Each transaction requires a single round-trip with only two events. The *REQ* first toggles to initiate the transaction; the *ACK* then toggles to indicate the data is received and completes the transaction. After each transaction, the signals do not necessarily return to zero. A *two-phase handshaking protocol* is also known as *transition-signaling*, because the actual value of the handshaking signals are not important. Instead, each transition event of the signals is important.

Since there is no *reset phase*, the two-phase protocol can obtain much higher throughput performance, and can be more favorable to high-performance systems. However, in practice, a number of asynchronous designs supporting a two-phase protocol have larger overhead in complexity and delay. In contrast, the proposed asynchronous NoC routers in this thesis are carefully engineered to be quite lightweight, based on Mousetrap pipelining (see below), and results confirm the benefits of this approach.

2.2 Data Encoding Schemes

After deciding a handshaking communication protocol, data needs to be added for most realistic transmissions. Because there is no synchronizing clock, and only 'on-demand' operation, data encodings are needed which can directly indicate the validity of new data. Data typically replaces the *req* wire in the asynchronous channel. The two most commonly used data encoding schemes are *(i) delay-insensitive (DI) codes* and *(ii) single-rail bundled data* [57].

2.2.1 Delay-Insensitive Codes

Delay-insensitive data encoding allows data bits to be sent in any order, and can tolerate any skew of bit arrival during a data transmission [166, 167, 233].

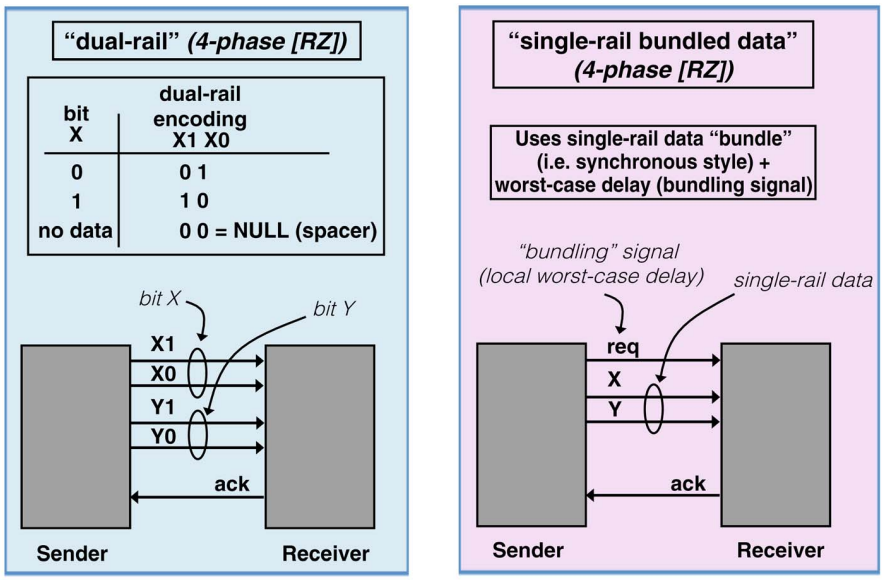
Typically, four-phase communication protocol is used along with DI codes. Initially, all the data wires are initialized to zero, and during a transaction, a certain set of data bits will be asserted high in any order. After the entire codeword is received by the receiver, in turn, the *ACK* is asserted high. Finally, all the data bits are reset low, followed by the *ACK* going low [4].

DI codes have the following property. The receiver needs to identify a valid codeword without ambiguity, regardless of the transmission time and relative skew of the distinct bits. Therefore, every valid code in the codeword space cannot be a subset of any other codeword. (Such codes are also called 'unordered codes' in the fault-tolerant community.) Several examples of DI codes are introduced below.

Dual-Rail Code

A Dual-rail code encodes each bit with two wires, as shown in Fig. 2.3(a) [9, 57, 166, 233, 239]. The combinations 01 and 10 are used to communicate 0 and 1 value respectively. The value 00 is a spacer, indicating the absence of valid data. The value 11 is not used and is invalid in the protocol.

Dual-rail encoding is simple but expensive. 2n wires are needed to communicate an n-bit value, resulting in a severe reduction in coding efficiency. Nonetheless, dual-rail codes have been widely and effectively used in asynchronous digital systems, including some commercial ones [55, 144, 166, 167, 198, 217, 240].



(a) (b)

Figure 2.3: Asynchronous data encoding schemes:
(a) dual-rail; (b) single-rail bundled data

1-of-N and M-of-N Codes

Both *1-of-N* and *M-of-N* codes are extensions of dual-rail codes. The 1-of-N code is also called *one-hot* code [136]. The code contains N data wires, and in each transmission, only one wire will be asserted. Therefore, N wires can encode N possible values. For large N, the 1-of-N code has an extremely low coding efficiency, while it is very simple to implement. Therefore, it can be used for small codeword spaces. However, 1-of-4 codes are widely used, with the same coding efficiency as dual-rail, but less switching activities. An *M-of-N* encoding scheme also represents each codeword using N wires. However, exactly M wires are asserted in each data transmission [73]. As a result, *M-of-N* codes have much improved coding efficiency, and have been used for several applications [74, 76]; however, the encoding and decoding logic can be quite complex.

Transition-Signaling DI codes

Conventionally, DI codes are typically combined with a four-phase RZ communication protocol. However, in principle, any DI code can also use a two-phase (NRZ) transition-signaling scheme. An assertion of '1' on a wire in the four-phase protocol is replaced by a 'toggle' on the same wire

in the two-phase protocol. Transition-signaling DI codes inherit the pros and cons of two-phase communication. They halve the number of wire transitions on the channel and achieve higher throughput, compared to the corresponding four-phase DI codes. However, they may induce more complex hardware design.

Examples of transition-signaling DI codes are *level-encoded dual-rail (LEDR)* [58] and *level-encoded transition-signaling (LETS)* codes [143]. These codes combine the benefits of both four-phase and two-phase handshaking protocols. As in a four-phase protocol, data can be directly extracted from a subsection of codewords. On the other hand, as in a two-phase protocol, there is no *RZ* phase. Using *LEDR* as an example, it uses two wires to represent a single data bit. If the new data value is different from the previous one, the first wire toggles, otherwise the second wire toggles. Therefore, at any time, the actual data value is represented by the level of first wire. Also, the arrival of a new data is indicated by a transition on exactly one of the wires. Details of these codes are not presented, since they are beyond the current research scope.

2.2.2 Single-Rail Bundled Data

Another alternative encoding approach is *single-rail bundled data*, as shown in Fig. 2.3(b) [57, 165, 166, 167, 213]. The data channel uses a standard synchronous-style single-rail data encoding with an extra bundling request, which indicates the data is valid. Unlike DI codes, single-rail data encoding requires a simple one-sided timing contraint: the bundling request *must* arrive at the receiver after all data bits are stable and valid.

The bundled data scheme allows arbitrary glitches on data channel, as long as the data is stabilized before *REQ* is transmitted. Both four-phase [230] and two-phase [201, 204, 213] communication protocols are widely used to combine with single-rail data encoding. Using the two-phase operation as an example, a data transaction begins with changes on all data bits, if any. There is no particular transition order imposed on the data wires. Also, arbitrary glitches are allowed. After all data bits are stabilized, the *REQ* is toggled, indicating the data is valid. Then, the *ACK* toggles, indicating the data is received and completes the transaction.

There are several advantages for single-rail bundled data scheme. First, since arbitrary data glitches are allowed, the scheme facilitates simple reuse of synchronous functional blocks. Second, it provides a much better coding efficiency, compared to any widely-used DI codes. For example,

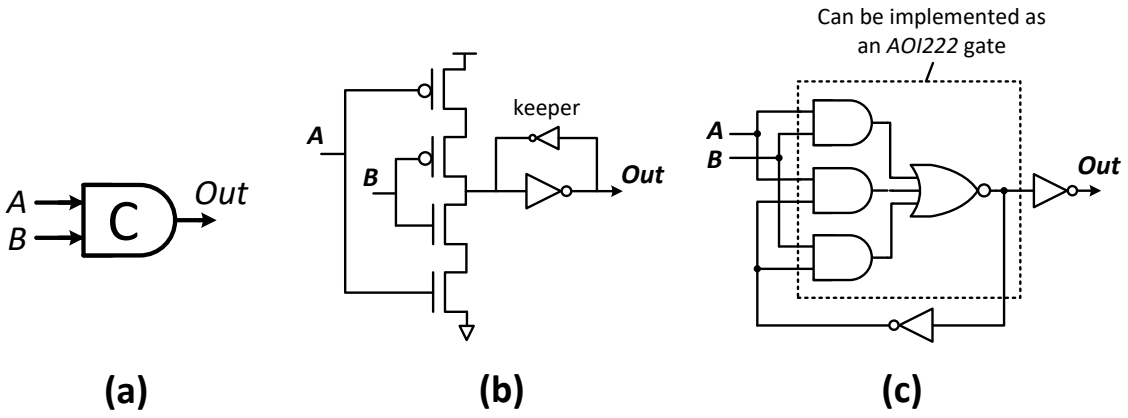


Figure 2.4: A two-input C-element: (a) symbol; (b) implementation directly using transistors; (c) implementation using standard cells

compared to dual-rail code, single-rail encoding only needs half number of wires to encode the same number of values. Finally, although a local one-sided timing constraint is forced with respect to the arrival time of bundling request for each stage, the stages can be highly unbalanced and optimized individually. This is unlike any synchronous clocked design.

In this thesis, we select single-rail bundled data encoding, combined with 2-phase (i.e. transition-signaling) protocol for all the NoC designs, in order to obtain high performance with low cost.

2.3 Special Asynchronous Elements and Components

Several special asynchronous components are introduced in this section. These elements are rarely used in synchronous designs, but widely-used in many asynchronous systems. They are preliminaries to understand the following research.

2.3.1 C-Element and Asymmetric C-Element

A C-element, also called a *Muller C-element* [153, 155, 166, 167, 213], is a standard asynchronous state-holding component. The element produces a high value output when all its inputs are high, and a low value when all its inputs are low. If the inputs are not all high or all low, the C-element holds its output.

Fig. 2.4 shows the symbol of a 2-input C-element and two possible implementations. The design

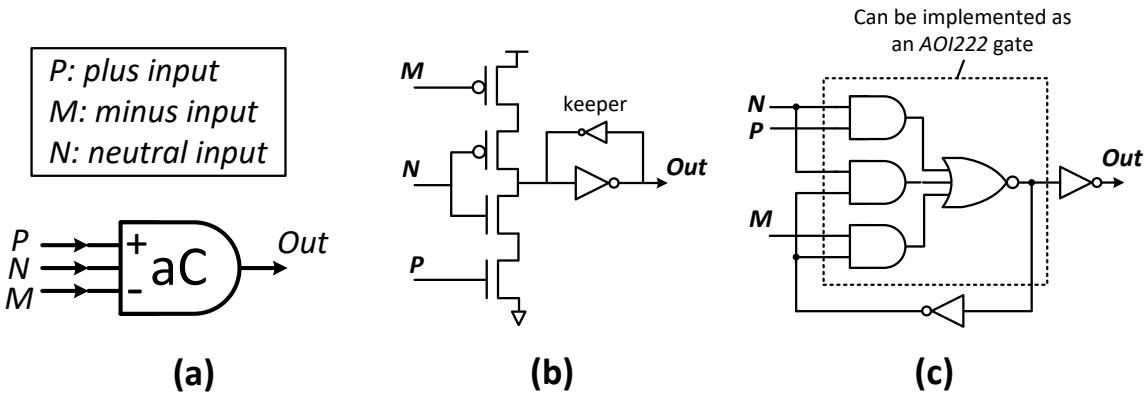


Figure 2.5: An example asymmetric C-element: (a) symbol; (b) implementation directly using transistors; (c) implementation using standard cells

on the left, shown in Fig. 2.4(b), is a simple and direct implementation using several transistors. The design on the right, shown in Fig. 2.4(c), is built completely based on standard cells, as a small state machine. While the former implementation is typically used in most of asynchronous designs due to better performance and less power consumption, the standard-cell based design is easier to be integrated into an automated design flow without the need of creating a new C-element cell. In fact, the standard-cell based implementation was used in Tangram, an early commercial asynchronous design flow by Philips Semiconductors [225]. Both implementations can be easily generalized to more than two inputs. However, a high fan-in C-element is usually decomposed into multiple logic levels, in order to maintain reasonable electrical behavior.

Asymmetric C-elements (aC) are extended C-elements, which allow inputs to only affect transitions in one of the directions [15, 72]. Inputs of an aC are divided into three categories: *(i)* inputs that affect up-transition only, marked as '+' symbol; *(ii)* inputs that affect down-transition only, marked as '-' symbol; *(iii)* neutral inputs that affect both transitions. The output goes from 0 to 1 when all '+' inputs and neutral inputs are high; the output goes from 1 to 0 when all '-' inputs and neutral inputs are low; otherwise the aC holds its output value.

Fig. 2.5 shows the symbol and implementations of a simple aC, with one '+' input, one '-' input and one neutral input. Similarly, a transistor based and a standard-cell based design are shown. More complicated aCs are typically decomposed into multiple logic levels and require careful custom design.

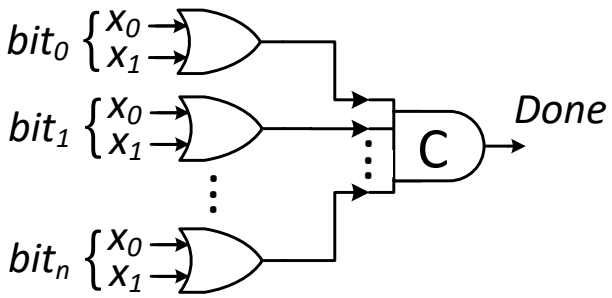


Figure 2.6: A dual-rail completion detector

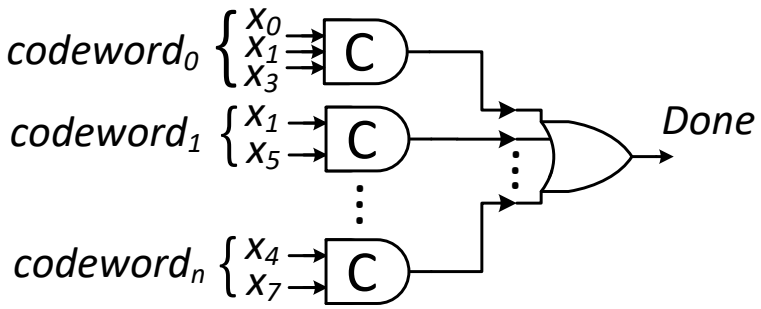


Figure 2.7: An example of DIMS completion detector

2.3.2 Completion Detectors

Unlike in synchronous designs, where the arrival of data is indicated by a global clock, completion detectors (CD) are required in asynchronous systems, to detect the arrival of a valid delay-insensitive codeword. Completion detectors are divided into two classes: CDs for return-to-zero DI codes [4, 175] and CDs for non-return-to-zero DI codes [32, 58, 143].

Fig. 2.6 shows a CD for dual-rail code, the simplest return-to-zero code [140, 227, 229]. The design is composed of an array of OR2 gates feeding into a C-element. Each OR2 gate detects the arrival of a single bit in the code. Therefore, the output of the C-element is asserted high when all bits become valid, and deasserted low when all bits are reset.

A more general return-to-zero CD design style is the *delay-insensitive minterm synthesis (DIMS)* approach [52, 68]. An example is shown in Fig. 2.7. The design consists of a bank of C-elements feeding into an OR gate. In the approach, each C-element represents a distinct valid code (i.e. a minterm) in the codeword space. When a valid code arrives, exactly one of the C-elements is acti-

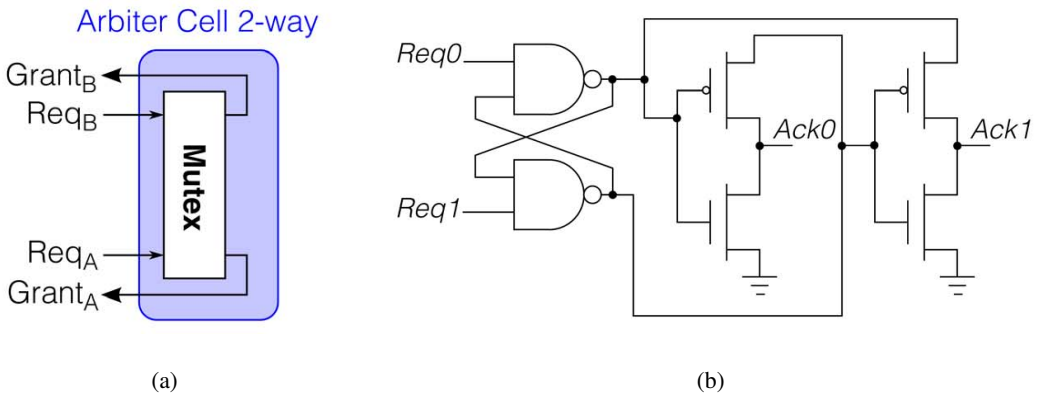


Figure 2.8: Mutual-exclusion element (Mutex): (a) block diagram; (b) implementation

vated. However, this particular design style has exponential growth with respect to code size, and thus has very high area and power cost.

An advanced CD design approach for 2-of-N codes was proposed in [32]. The detector architecture is a simple binary tree, composed of basic cells. The design is much more scalable than the DIMS approach, as the area cost only grows logarithmically with respect to N. Also, the structure can be used for both RZ and NRZ codes.

2.3.3 Mutual-Exclusion Element and Asynchronous Arbiters

Arbitration is critical in all asynchronous systems. An arbitration involves the resolution of two or more competing clients requesting a shared resource. Arbitration is typically simple in synchronous designs: in each clock cycle, requests arrived in previous cycles are examined and one of them is selected to be the winner. However, arbitration is much more challenging in asynchronous designs: requests can arrive in continuous time, possibly only a few picoseconds apart, and resolution must be completed cleanly and safely, unaligned to any clock cycles [166].

Mutual-Exclusion Element

A *mutual-exclusion element* (*mutex*) is a basic component to resolve two-way arbitration. The structure and a basic implementation of a mutex is shown in Fig. 2.8, due to Seitz [183, 189]. This analog unit has two input requests and two output grants (i.e. acknowledgments), operating under a four-phase RZ protocol.

The design is composed of a pair of cross-coupled NAND2 gates on the left (i.e. a SR latch) and

a filter on the right (consists of a cross-coupled structure using four transistors). If only one input is asserted high, it is granted without contention. If both requests are asserted within a short period, the SR latch on the left is put into a metastable state with undefined output. The filter on the right keeps both outputs low until the metastability is resolved. Therefore, only one of the requests is granted first, the second request will be granted until the first channel releases the request. Theoretically, the resolution of metastability can take arbitrarily long time. However, in practice, only when two requests arrive extremely closely (e.g., $<$1 ps), the arbiter may result in a relatively long delay.

Asynchronous Arbiters

Components that resolve N input requests are called N-way arbiters. A simple N-way arbiter can be implemented using a tree structure composed of mutexes. However, this naive implementation has very poor performance. The latencies in response to different input channels also vary a lot when the tree is not balanced. (It is natural for a non-power-of-two arbiter in this implementation.) Many different designs and optimizations have been thus proposed [62, 63, 80, 108, 148, 151, 160, 206, 226, 242], which target better performance, robustness, scalability and fairness. In particular, a new arbiter design approach is introduced in [148], and applied to a family of arbiters with different numbers of input requests (practically up to 16). The design targets an overall high performance, as well as an increased robustness and impartiality across all inputs. The proposed arbiters use flattened and rebalanced tree structures, which decrease the number of tree cells from each leaf to the root (for performance), and simultaneously, allows an equal number of tree cells from each leaf to root (for fairness).

Here we introduce two small but recently proposed arbiter examples: a balanced 3-way arbiter and a high-performance 4-way arbiter. They are fundamental components used in my NoC research.

Fig. 2.9 shows the balanced 3-way arbiter solution [148]. The design contains three mutexes connected in a ring-like structure. Each of the mutexes resolves a distinct pair of competition (a-b, b-c or a-c). When a request arrives, before it is granted, it must pass through two mutexes (i.e win two arbitrations over the other two inputs) in series. Although the design has a larger latency than a basic 3-way tree arbiter in some cases, it provides high latency equalization between all inputs.

When all inputs arrive almost simultaneously, the arbiter may deadlock with each request winning only the first stage of arbitration. Therefore, a *deadlock detector* is added to resolve this case.

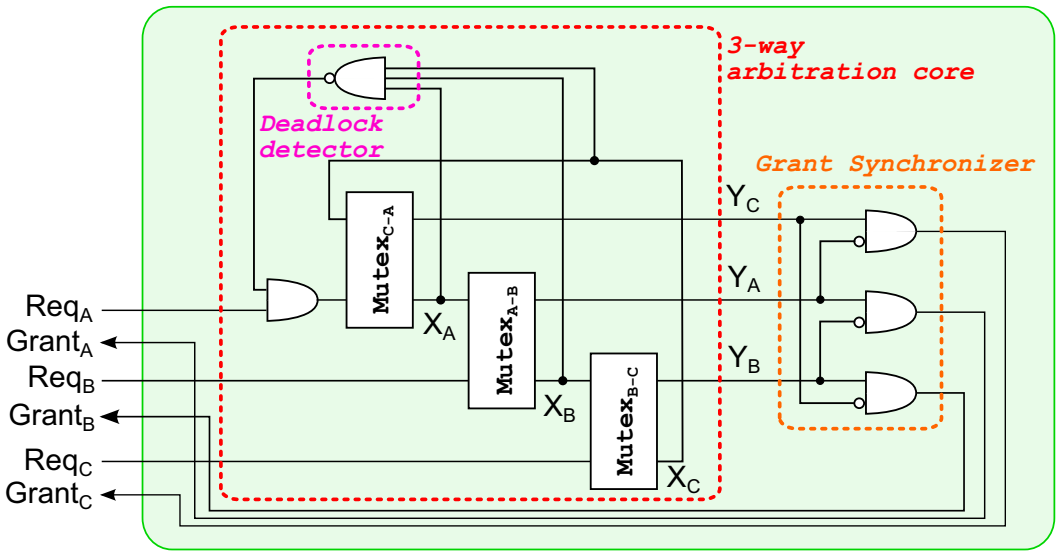


Figure 2.9: A balanced 3-way arbiter

It kills Req_A and allows Req_C to win the arbitration. Since this scenario is an extremely rare case, the addition of deadlock detector has minimal impact on overall fairness and performance results.

Fig. 2.10 illustrates a high-performance 4-way arbiter [80, 148]. Unlike the conventional tree arbiter, in which 2 levels of arbitrations are performed in series for any request being granted, the critical path of the solution only has 1 mutex element.

As shown in the figure, the design contains three mutexes. The left mutex arbitrates between requests 0 and 1. Similarly, the right mutex arbitrates between requests 2 and 3. The central mutex, however, arbitrates between requests from the left and right pairs. When a request arrives, it activates the left (or right) and the central mutexes simultaneously. Both arbitrations are resolved in parallel, and therefore results in an improved performance.

2.4 Asynchronous Pipelines

Pipelining is widely used for boosting throughput performance in high-speed digital systems. Modern general-purpose processors, graphic units, as well as application-specific digital circuits, such as multimedia and DSP processors, are all pipelined. So are the routers in most high-performance NoC's. A typical pipeline implementation divides complex functional blocks into smaller blocks. Multiple data tokens are inserted into the pipeline back-to-back, and they are processed in different

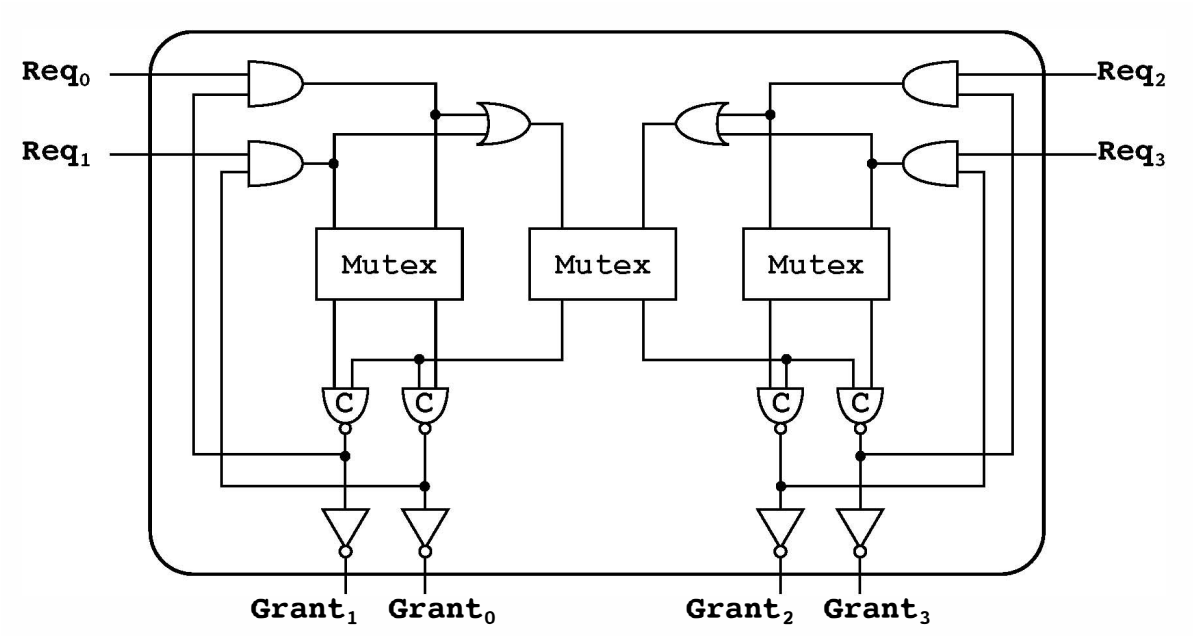


Figure 2.10: A high-performance 4-way arbiter

sub-steps concurrently.

Although the idea of parallel processing for multiple input tokens is the same, the implementation approaches for synchronous vs. asynchronous pipelines are different [165]. In synchronous pipelines, registers are inserted to separate sub-functional blocks and a global clock is simultaneously applied to all registers. However, in asynchronous pipelines, neighboring stages communicate using local handshaking protocols, with unaligned timing of different stages.

This section reviews several leading asynchronous pipeline implementations, which are preliminaries to understand my research. Section 2.4.1 reviews the *Mousetrap* pipeline [201, 204], a well-known and widely-used high-performance static asynchronous pipeline. Sections 2.4.2 and 2.4.3 introduce two dynamic pipelines: Williams' PS0 style [240] and the high-capacity style [202]. The former has a simple design, while the latter improves the data capacity and targets high performance.

2.4.1 Mousetrap Pipeline

Mousetrap was a high-performance asynchronous static pipeline developed at Columbia University. Fig. 2.11 shows a basic 3-stage Mousetrap pipeline. The pipeline uses a two-phase handshaking

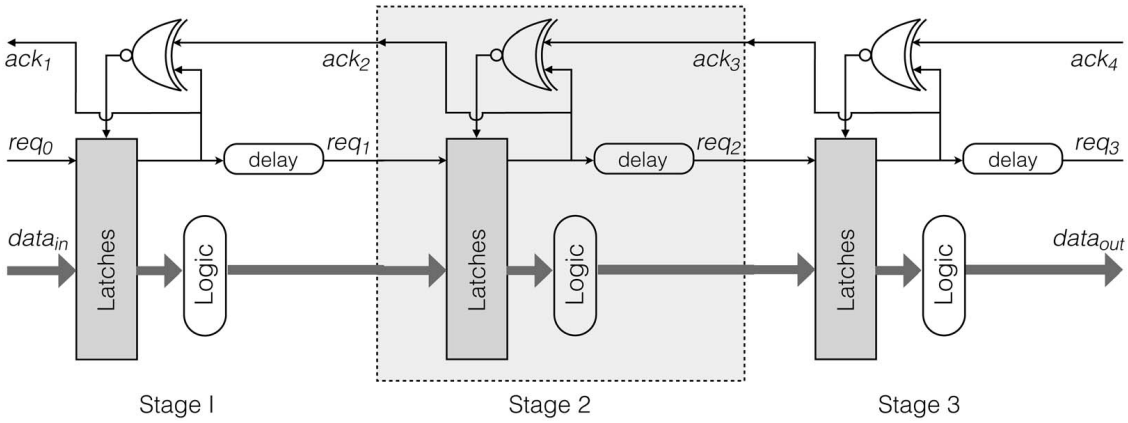


Figure 2.11: Mousetrap pipeline

protocol and single-rail bundled data encoding. A delay element is added to match the worst-case delay of the functional block for each stage.

The pipeline uses a so-called *capture-pass* protocol. Unlike the synchronous pipelines, all the latches are normally transparent. Overall, the entire pipeline forms a combinational flow-through path. Locally, when a data token passes through the latches and enters the corresponding stage, the latches are immediately closed to safely *capture*, i.e. protect, the data. Only after the successor stage receives and captures this data token, the latches of the current stage are made open again and allow new data to *pass* through.

While *capture-pass* protocol was first proposed and used in *Sutherland's micropipeline* [213], *Mousetrap* has much simpler control circuits and data latches. This leads to a much smaller area and a much higher performance. In particular, the control of each stage is simply a XNOR gate, and the data register is a single bank of level-sensitive D-latches. Both of them are available in standard cell libraries.

A quick sketch of the operation is presented now. Initially, all the data latches are transparent, and all req_i and ack_i are at 0. As new data arrives at stage *i* from left, with the corresponding bundling signal req_{i-1} toggles, the data immediately passes through the latch and enters the stage. Three events occurs concurrently: *(i)* the stage's XNOR toggles from 1 to 0, and captures the data by closing the latches; *(ii)* an acknowledgment is sent to the left through toggling the ack_i; *(iii)* the data is processed by the functional block and then sent to the next stage. Subsequently, when the

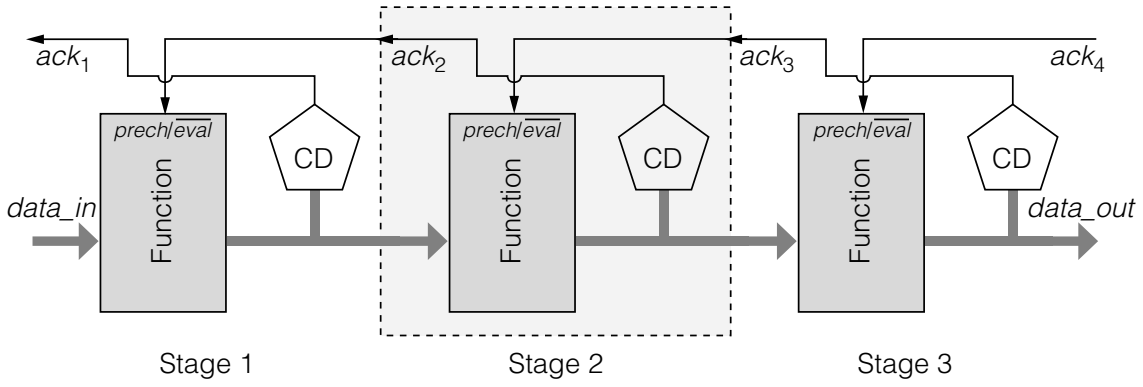


Figure 2.12: Williams' PS0 pipeline

next stage *i+1* has received the data and an acknowledgment is placed on ack_{i+1}, stage i's XNOR toggles back to 1, opening the latches again and ready to accept the next new data. This completes the cycle.

2.4.2 Williams' PS0 Pipeline

Dynamic logic datapaths are commonly used for many high-performance digital systems. By removing the pull-up logic, dynamic circuits have smaller area and power costs, and smaller switching capacity for higher performance. However, dynamic logic is typically only used for speed-critical components, since it requires greater design and validation efforts, and is less immune to noise.

Interestingly, dynamic logic is a particularly good match for asynchronous pipeline designs. Local handshaking removes the need of designing a complex multiphase global synchronous clock. Also, a DI datapath is immune to noises induced by delay variations on different data bits. Furthermore, a key feature for most of asynchronous pipelines is that they are latchless. Asynchronous pipelines can exploit implicit latching property of dynamic logic and avoid explicit data registers. However, a synchronous implementation achieving the same latchless feature requires complex multiphase global clock design. With the above advantages, asynchronous dynamic pipelines have been used in several industrial products [55, 165, 217].

Fig. 2.12 shows a basic structure of a PS0 pipeline [240], which is the foundation of much subsequent research. A four-phase (RZ) communication protocol is used, with dual-rail data encoding. Each stage has a functional block and a completion detector. As shown in the figure, there are no

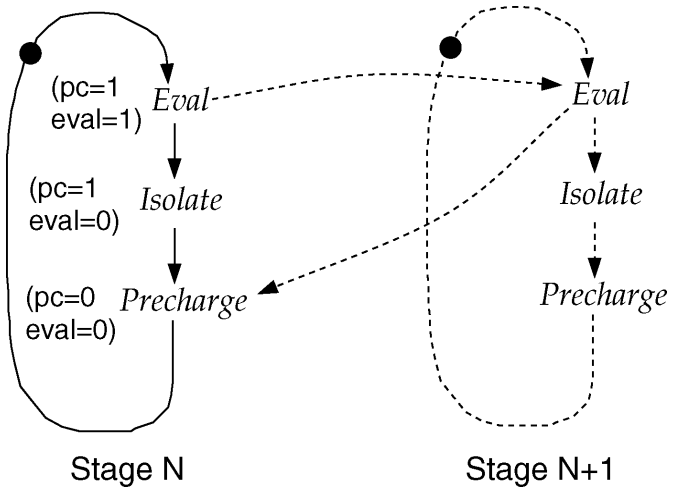


Figure 2.13: High-Capacity (HC) pipeline: operational protocol

explict data registers to separate neighboring stages. Each functional block uses dynamic logic and alternates between two phases: an *evaluate* phase and a *precharge* phase. In the *evaluate* phase, the incoming data is evaluated and sent to the next stage; in the *reset* phase, the previously evaluated data is cleared, and the output of the functional block is forced to be all 0s. In particular, when a stage is in *precharge*, it has a blocking property to separate consecutive data tokens. The CD identifies when the stage completes data evaluation, or when its output are all reset to 0s.

The operational protocol for PS0 pipeline is very simple. Overall, as a new data appears at the input of the pipeline, the functional blocks evaluate the data one after the other, from left to right, without any other synchronization. The fact of no explict latches results in low latency for each data processing. Locally, the pipeline follows two basic rules: *(i)* a stage is precharged when the successor stage completes evaluation; *(ii)* a stage is enabled to evaluate when the successor stage completes precharge. The protocol ensures a stage holds a stable data output as long as the next stage is still using the data (i.e. until the successor stage places an acknowledge on *Prech/ Eval*. Therefore, two successive data tokens are always separated by a spacer, i.e. a precharging stage.

2.4.3 High-Capacity Dynamic Pipeline

A series of advanced dynamic pipelines are proposed based on *Williams' PS0 pipeline*, targeting higher throughput [64, 69, 77, 134, 202, 203]. We focus on a so-called *High-Capacity (HC)*

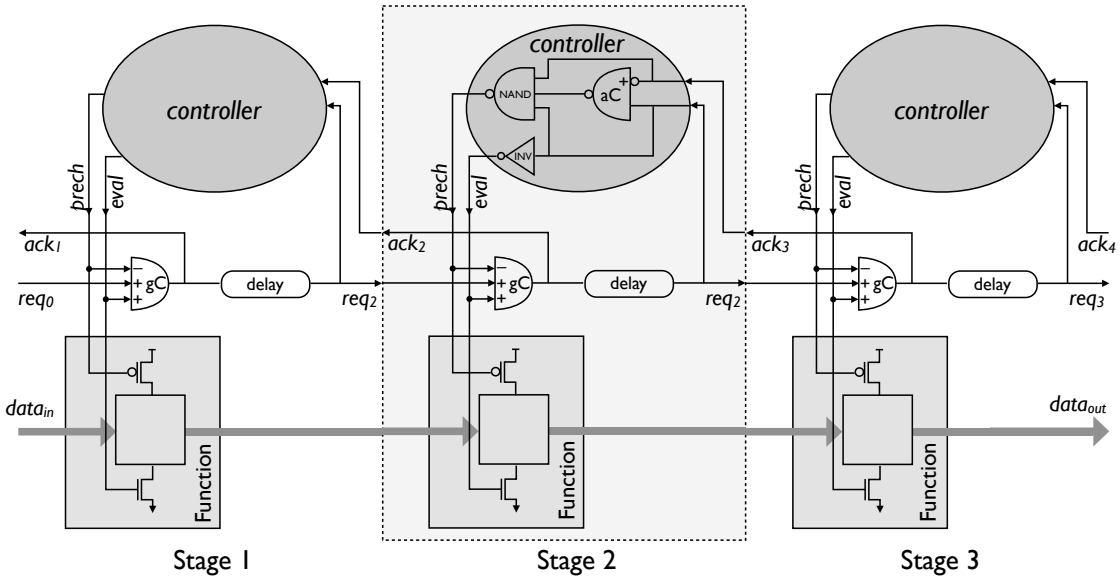


Figure 2.14: High-capacity (HC) pipeline: implementation

style [202].

One of the drawbacks for PS0 pipeline is that it only provides 50% storage capacity. Back-to-back data tokens must be separated by an empty stage. Therefore, this leads to a considerable area overhead: the number of pipeline stages are doubled for the same data capacity. This becomes a disaster for applications where buffering capacity is important. In contrast, the *HC* pipeline allows 100% storage. Each stage can hold a distinct data item under congestion.

The *HC* pipeline uses a four-phase communication protocol with single-rail bundled data. Similar to Mousetrap pipeline, a matched delay is inserted to each stage for matching worst-case delay through the corresponding funtional block.

The pipeline protocol is fundamentally different from PS0. Each stage cycles through three phases, as shown in Fig. 2.13, while the implementation of the pipeline is shown in Fig. 2.14. This is realized by having two decoupled controls for dynamic gates: *Precharge* and *Eval*. In *evaluate* phase, *Eval* is asserted high and *Precharge* is deasserted high. The stage is ready to process incoming new data item; In *precharge* phase, *Precharge* is asserted low and *Eval* is deasserted low. The stage is reset and its output are forced to be all 0s. In *isolate* phase, both controls are deasserted. The output of the stage holds the value and immune from all inputs. Pull-up and pull-down are simultaneously inactive and the gate is neither in evaluate nor precharge. This is the key innovation

of the design.

The operation of the pipeline is simple. Initially, all stages are in *evaluate* phase, and ready for new data tokens. As soon as a new data enters the stage and is evaluated, the stage is *isolated* to protect the data. In parallel, an acknowledge is sent to the left, and allows new data arrival. Once the successor stage evaluates the data and an acknowledge is received from right, the current stage is enabled to complete an entire new cycle, without any further synchronization – *precharge*, *evaluate* a new data item, and enters *isolate*.

As a consequence, only one backward synchronization between a pair of neighboring stages is required in each cycle: the current stage has to stay *isolated* until the successor stage completes *evaluate*. This is unlike the PS0 protocol, where two backward synchronizations are forced, by the two operational rules listed above, respectively. The decreased number of synchronization points allows a more loosely-decoupled protocol and therefore increased throughput.

Chapter 3

Background: Network-on-Chip Basics

An overview of basic concepts for networks-on-chip is now given. The chapter begins with network topology, the static arrangement of channels and nodes in the network (Section 3.1). Next, basic routing concepts (Section 3.2) and widely-used flow control methods (Section 3.3) are then introduced. They describe how the data are transmitted in the network. Finally, the structures of typical synchronous routers are presented, along with their operation (Section 3.4). Since most concepts of synchronous routers are also inherited by asynchronous counterparts, the section will help the readers to understand the asynchronous networks-on-chip and router designs in the following chapters.

3.1 Network Topology

A network-on-chip is always composed of router nodes and channels (i.e. links). Network topology defines the arrangement of nodes and how they are connected by the network links. Selection of a network topology is usually the first step in designing a network-on-chip. Routing strategies and flow control mechanism are highly dependent on the topology.

In this section, we first introduce common ways for topology classification. Then several topology examples are introduced. In particular, my research focuses on two topologies – the variant mesh-of-trees topology (used for NoC's in Chapter 4) and the 2D mesh-based topology (used for NoC's in Chapter 5 and 6). Other widely-used topologies can be found in [46, 170].

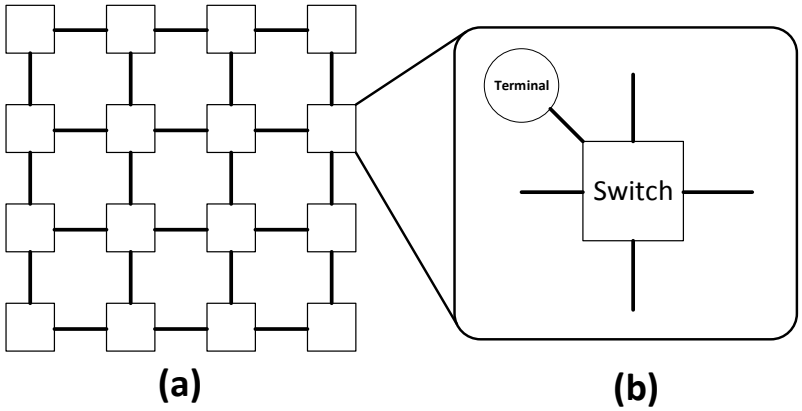


Figure 3.1: 4×4 2D-mesh: (a) Network topology; (b) node structure

3.1.1 Topology Classification

Direct and Indirect Networks

A router node can serve two functionalities. It is a *terminal node* if it acts as source and sink for data transmission; it is a *switching node* if it is an intermediate node that only forwards data.

In a direct network, every router node is both a *terminal* and a *switch*. Data is forwarded *directly* between terminal nodes. On the other hand, in an indirect network, a router can be either a *terminal* or a *switch*, but cannot serve both functions. Indirect networks transmit data *indirectly* through intermediate switches between each source-sink pair [18]. Many irregular networks are neither direct nor indirect.

Regular and Irregular Networks

Regular network topologies arrange nodes and links in a disciplined way following regular graph structures, such as rings, meshes and trees, etc. In contrast, irregular networks do not.

A regular network is often used if it needs to handle a wide range of traffic patterns when the bandwidth requirements are unknown and unpredictable. An irregular network, however, is usually carefully designed for certain types of applications, e.g. video coding or gaming. An irregular network is called a custom topology. In such NoC's, network traffic distributions are largely known in advance and particular network topologies can be tuned for the applications [21].

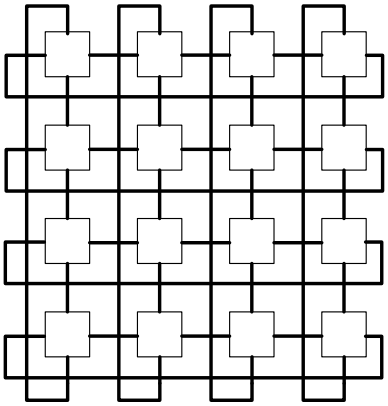


Figure 3.2: 4×4 2D torus topology

3.1.2 Network Topology Examples

2D Mesh-Based and 2D Torus Topologies

A *2D mesh-based* topology is a direct network, where each router node in the network is connected to a functional block, as shown in Fig. 3.1. A functional block can be a processing unit, a memory bank, an accelerator etc. Each router typically has 5 ports, connecting to its four neighbor nodes – west, east, north and south – and the local functional block.

A *2D torus network* is another direct network, as shown in Fig. 3.2. It is almost the same as 2D-mesh topology, with extra added links. The routers on the edge for the same row/column are now connected with wrap-around channels.

Both 2D mesh-based and torus networks are simple, and they are attractive for several reasons. First, the regular physical arrangements are well matched for connecting homogeneous functional blocks, such as cores in a parallel computer. Also, the links between neighboring routers are short. Logically minimal paths are almost always physically minimal as well. This allows networks to exploit physical locality in communication [46]. What is more, mesh-based topologies have good path diversity. Typically, multiple minimal paths exist for each source-destination pair. Non-minimal paths can also be used in advanced routing algorithms.

One disadvantage for these two networks is that they have larger hop count than most of tree-based topologies. This leads to higher latency. However, note that an increment in hop count is required for path diversity [46].

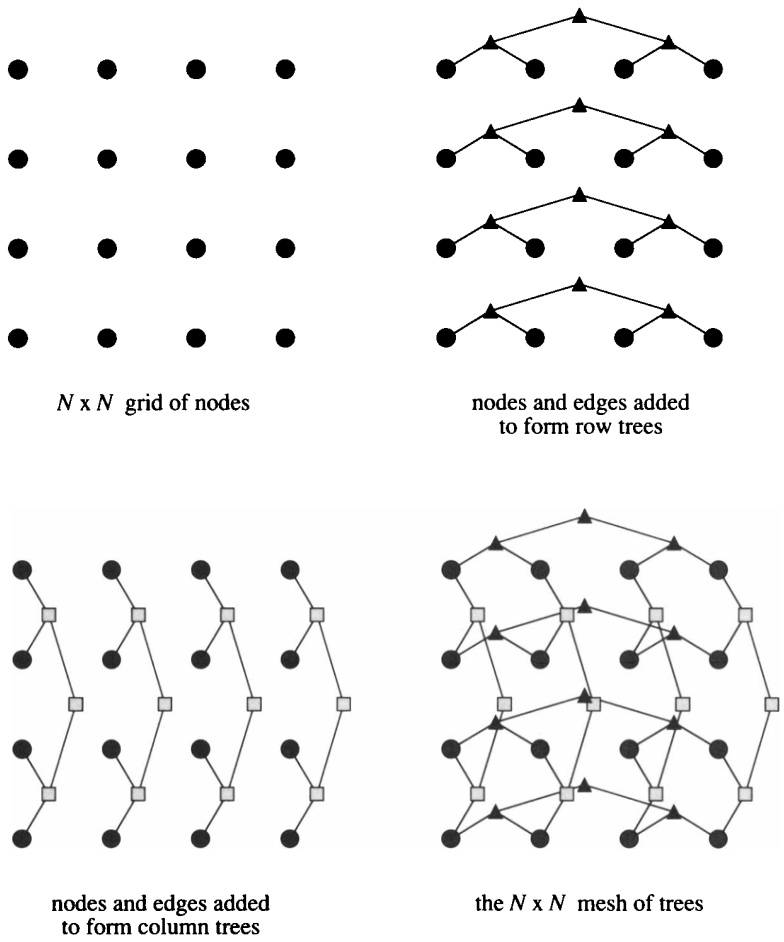


Figure 3.3: A basic 4×4 mesh-of-trees (MoT) network

Basic and Variant Mesh-of-Trees Topologies

The construction of a *basic 4×4 mesh-of-trees (MoT) network* is shown in Fig. 3.3 [125]. The network is constructed from a 4×4 grid of nodes, by adding binary trees in each row and column. The topology is an indirect network. The leaves of binary trees (i.e. black circles) serve as sources and sinks, while the remaining nodes (i.e. triangles and squares) are intermediate switches.

A simple minimal routing is usually used. If the source and the destination are inside the same row/column, only the corresponding row/column binary tree is used for routing. Otherwise, if the source and the destination are not in the same row or column, data is first sent from the source to an intermediate node, which is at the same column (row) of the destination, through a row (column)

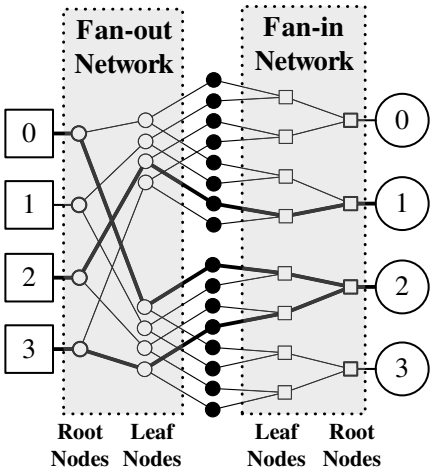


Figure 3.4: A variant 4×4 mesh-of-trees (MoT) network

binary tree, then from that intermediate switch to the destination node through a column (row) binary tree. It is simple to identify that there are two optimal minimal path in this case.

My research, however, focuses on a *variant mesh-of-tree (MoT) topology* [10, 11], instead of the basic one, as shown in Fig. 3.4. While the figures look quite different, the variant topology actually has the same structure as the basic one, but with the roots of binary trees being sources and sinks. Compared to the traditional MoT network, the variant topology forces a simpler deterministic routing algorithm and has much less traffic contention, which is, therefore, more suitable for building high-performance NoC's.

The variant MoT network consists of a fan-out network and a fan-in network. In more detail, a section of the network is shown in Fig. 3.5. It consists of a binary fan-out tree composed of *routing* nodes, which emanates from the source root node and terminates in mid-network with leaf nodes; a symmetric binary fan-in tree composed of *arbitration* nodes, which emanates from the destination root node and also terminates in mid-network with leaf nodes. Leaf nodes of the two sub-networks are directly connected.

Network routing for the variant MoT network is deterministic. Fig. 3.5 highlights a single flit traversal through the network, from a source fan-out root to a destination fan-in root node. Each source-destination pair is associated with a unique path through the network. This feature significantly minimizes contention, since distinct fan-out sources are attached to different fan-out trees, and distinct fan-in destinations are attached to different fan-in trees. This separation guarantees that,

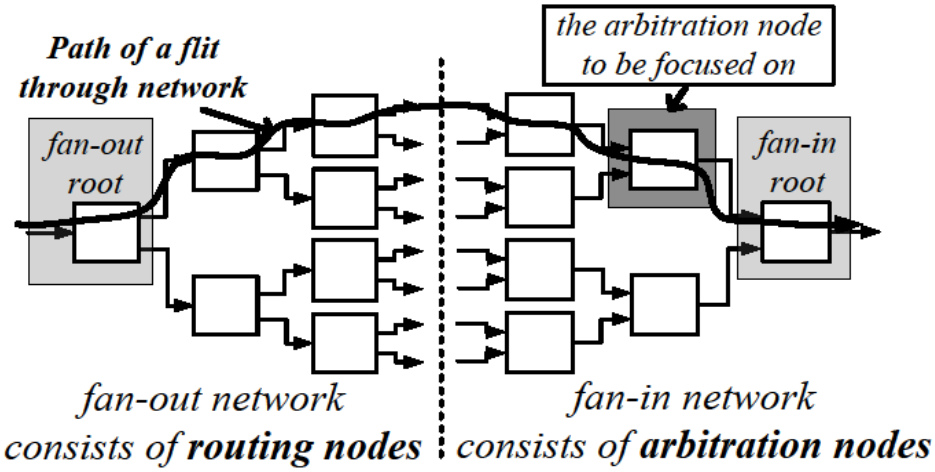


Figure 3.5: A variant 4×4 mesh-of-trees (MoT) network: a fan-out tree + a fan-in tree

unless the traffic is extremely unbalanced, data between different sources and destinations will not interfere [82].

3.2 Routing Basics

Once a network topology is determined, routing is the next logical step. Routing involves selecting a path from the source node to the destination.

A well-designed routing algorithm is critical, and can push the network topology to its ideal performance. It may have the following characteristics. First, a good routing algorithm attempts to keep path lengths as short as possible. This reduces the number of hops and the overall latency of data transmission. Many algorithms only use minimal paths, or in most cases. Second, the network traffic is well balanced across the channels. There is no congestion area or hot spot most of the time. This leads to a very good network throughput. Finally, the routing algorithm needs to be simple, which is either independent to network state, or only monitors simple parameters in the network. This is closely related to the router complexity.

3.2.1 Classification of Routing Algorithms

Deterministic Routing

Deterministic routing algorithms are simplest and most inexpensive to implement. In such routing

schemes, each source-sink pair only has a single pre-determined path. Data is always sent over exactly the same route.

Besides its simplicity and low cost, deterministic routing also has several other advantages. Such routing can always be used for any topologies. Other routing schemes, i.e. *oblivious* and *adaptive* routing (see below), may require path diversity in the network, which may not even exist. Also, deterministic routing is widely used especially for custom topologies, where designing randomized or adaptive algorithms is much more difficult [46].

However, deterministic routing cannot adapt to different traffic patterns on the fly, and therefore can result in large load imbalances in the network. It may create hot spots and congestion areas, and sometimes can severely pull the network performance from what it can maximumly achieve.

A well-known example of deterministic routing is *dimension-order* routing in mesh-based and torus networks [46]. Data first travels in one of the dimensions to the row or column of the destination node, and then is delivered to the destination in the other dimension. It is obvious that *dimension-order* routing only uses minimal paths. This routing algorithm is also called *X-Y* or *Y-X* routing, depending on which dimension data travels first.

Oblivious Routing

Compared to deterministic routing, *oblivious routing* can use multiple paths between each source-sink pair. However, the selection of paths is random, regardless of the network state. Most oblivious routing algorithms are still simple and easy to implement, since after data is injected into the network, the routing path is determined. This is the same as deterministic routing. Such a routing scheme, however, provides path diversity in a limited way and somewhat leverages the traffic load imbalances.

Two examples for oblivious routing are now introduced.

O1TRUN (*orthogonal one-turn*) routing, is a simple algorithm for 2D-mesh topology. Data randomly selects either X-Y or Y-X path with equal probability when it is injected into the network [194]. The routing guarantees data to turn at most once during its network traversal. O1TURN routing is proved to provide nearly optimal worst-case throughput for all-to-all random traffic [194].

Valiant's randomized routing is a more complicated oblivious routing example for 2D-mesh or torus topology [46, 224]. When data is sent from a source to a destination, it is first sent from

the source to a randomly chosen intermediate node, and then sent from that intermediate node to the destination. Usually dimension-order routing is used for both of the two phases. This algorithm turns any original traffic pattern to an all-to-all random traffic, and therefore results in better network load balance [224].

Adaptive Routing

Adaptive routing is a more advanced routing strategy. Such a routing scheme uses network state information and determines the routing path on the fly during data transmission. A good adaptive routing algorithm typically outperform a deterministic or oblivious routing algorithm, since they route data around congestion areas in the network, which leads to a much better load balance. However, these algorithms are more expensive and difficult to implement.

Most adaptive routing algorithms only monitor local network states, for example, whether 1-hop or 2-hop neighbors are congested. This nature of locality can lead to a delay in responding to changes in traffic patterns. There are many existing adaptive routing algorithms [46, 84, 98, 130, 245]. A survey of these algorithms is beyond the current research scope.

3.2.2 Encoding Routing Information

A *packet* is the basic unit in data transmission. However, a packet is usually further divided into smaller units, called *flits*. Each packet contains a *header flit*, followed by a serial of *body flits*, and ends with a *tail flit* [46, 49]. Routing information of a packet is always stored in its header flit. There are two primary ways to encode this information.

Source routing stores the routing decisions for each hop on the path in the header flit, at the point when the packet is injected into the network. A header flit structure is illustrated in Fig. 3.6(a). At each hop, the corresponding address bits in the header are extracted, and directly used for routing decision with no further computation. Because of its speed, simplicity and scalibility, source routing is one of the most widely used routing methods. However, it is only viable for deterministic and oblivious routing.

Destination-based routing only stores the destination information in the header, as shown in Fig. 3.6(b). This information is repeatedly used for each hop on the path, to locally compute the routing decisions. Destination-based method is typically slower than source-routing due to the need

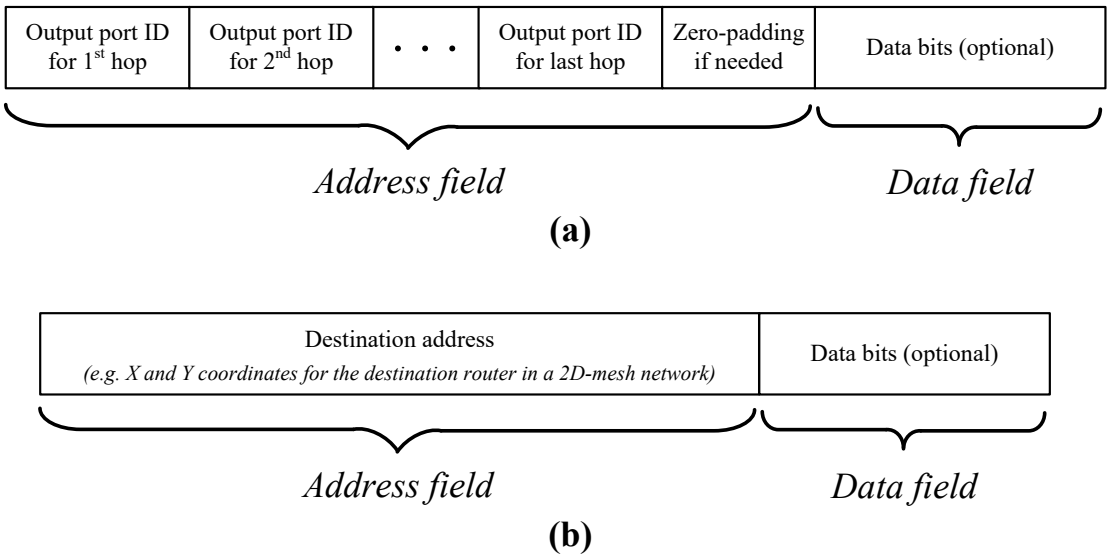


Figure 3.6: Header flit structure: (a) source routing; (b) destination-based

of per-router computation, but allows a smaller address field, and is a more general approach that can also be used for adaptive routing.

3.3 Flow Control Methods

3.3.1 Store-and-Forward

In *store-and-forward* flow control, a packet travels through the entire path as an entity. Each router along the routing path waits until the packet has been completely received, before it can forward the packet to the successor node. After the transmission begins, the packet is sent continuously without interruption.

In more detail, the packet must meet three prerequisites for starting a transmission: *(i)* the entire packet transmission completed from the predecessor to the current node; *(ii)* a packet-sized buffer allocated at the successor node; *(iii)* a free channel for transmitting the entire packet.

A timing diagram for store-and-forward flow control is shown in Fig. 3.7. Fig. 3.7(a) shows the structure of the network and the path of the packet, while Fig. 3.7(b) illustrates a 4-flit packet being forwarded over three hops without being stalled. The major drawback of store-and-forward is obvious – very high end-to-end system latency.

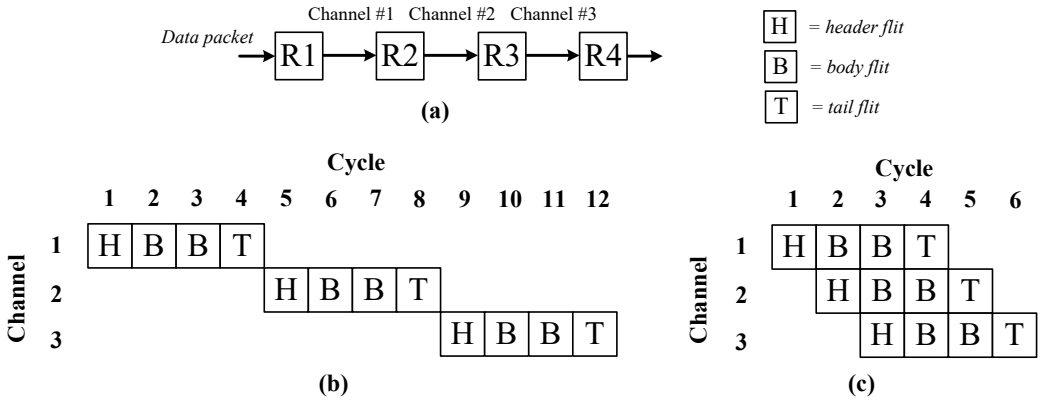


Figure 3.7: Timing diagrams for different flow control methods:
(a) path of the packet; (b) store-and-forward; (c) cut-through and wormhole routing

3.3.2 Cut-Through

Cut-through flow control largely overcomes the latency penalty for store-and-forward. The approach allows a node to forward a packet as soon as a section of the packet is received, without waiting for the arrival of the entire packet. The packet is still considered as an entity, and sent over the channel continuously, since the buffer space is guaranteed to hold the packet at the next router.

Fig. 3.7(c) shows a timing diagram for the same packet being forwarded through the same path using cut-through instead of store-and-forward. It is clear in the figure that a flit can be forwarded as soon as it arrives at the router, in ideal case.

3.3.3 Wormhole Routing

Wormhole routing operates like *cut-through*, but with channel and buffer allocated on a per-flit basis. Each packet is still treated as an entity. This means that, after the channel is allocated to a packet, no other packet can use the channel until the transmission is over. However, the flits in the packet are no longer guaranteed to be sent continuously, since a downstream router can stall the transmission in the middle when there is not enough space to hold more flits. Compared to *cut-through*, *wormhole flow control* largely decreases the buffer space requirement for achieving a similar network performance. Ideally, the timing diagram for *wormhole flow control* is the same as *cut-through*, as shown in Fig. 3.7(c). All networks and router designs in this thesis use wormhole routing.

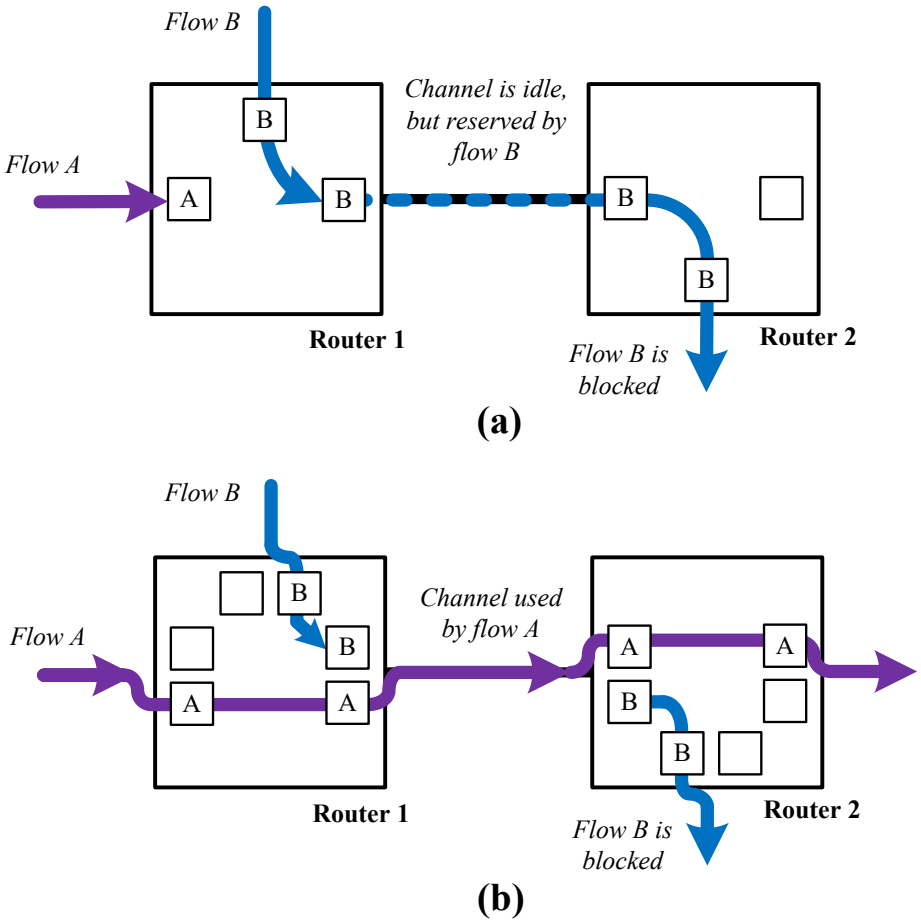


Figure 3.8: Apply virtual channel to wormhole routing:
(a) head-of-line blocking scenario; (b) solution with 2 VCs

3.3.4 Virtual Channels

Virtual channels are a commonly-used technique to resolve congestion and further increase the network throughput. It is almost always combined with wormhole routing to maximize the network performance [47]. A physical channel is allowed to be shared by multiple data flows. The technique enables flit-level interleaving on the channel; flits from different data flows are sent using round-robin. Each data flow, however, from its own perspective, still virtually has the full access of the physical channel, and conforms to all wormhole regulations and operations. Each packet is now assigned with a VC ID, which can be static throughout its transmission, or dynamically updated on a hop-by-hop basis.

Fig. 3.8 shows a well-known congestion scenario, also called *head-of-line blocking*, and how VC can resolve the congestion [46]. In Fig. 3.8(a), flow A is expected to send data from the west port of router 1 to the east port of router 2, through the channel connecting two routers. The channel is already reserved by flow B, as it is in the middle of a packet transmission. However, the south port of router B is currently congested, and flow B is stalled. According to wormhole routing, flow B will not release the channel until it transmits the entire packet. Therefore, the channel between two routers is idle and wasted. Fig. 3.8(b) illustrates a solution with 2 VCs. As the channel can be shared by two flows, flow A can successfully send its data without being blocked.

3.4 Synchronous Router Architecture and Operation

Routers are important components in any networks-on-chip. They direct the path of packets that are transmitted in the network. Although the thesis focuses on asynchronous networks-on-chip and asynchronous router designs, we start with an introduction for conventional synchronous routers. This will help the readers understand the technical contents in the remaining of the thesis, since the high-level structure and operation of asynchronous routers are largely similar to their synchronous counterparts.

3.4.1 Synchronous Router Structure without VC

The structure of a typical synchronous router is shown in Fig. 3.9. It is composed of *input units*, *output units* and a centralized *crossbar*, connecting those input and output units. Data first arrives at a certain input unit, then travels through the crossbar, and is finally sent out by a particular output unit.

Each *input unit* has an input channel, which is connected to an upstream neighbor router or a local terminal. By symmetry, each *output unit* has an output channel, which is connected to a downstream neighbor or the local terminal. The numbers of *input* and *output units* are not necessarily to be the same. Each *input* or *output unit* has a local control for flow control, allowing or stalling the arrival and outgoing data transmission of the unit. Data buffers can be added to further increase the throughput of the router, in exchange for extra area and power consumption.

The *crossbar* manages all traffic going from *input units* to *output units*. Its control, also called

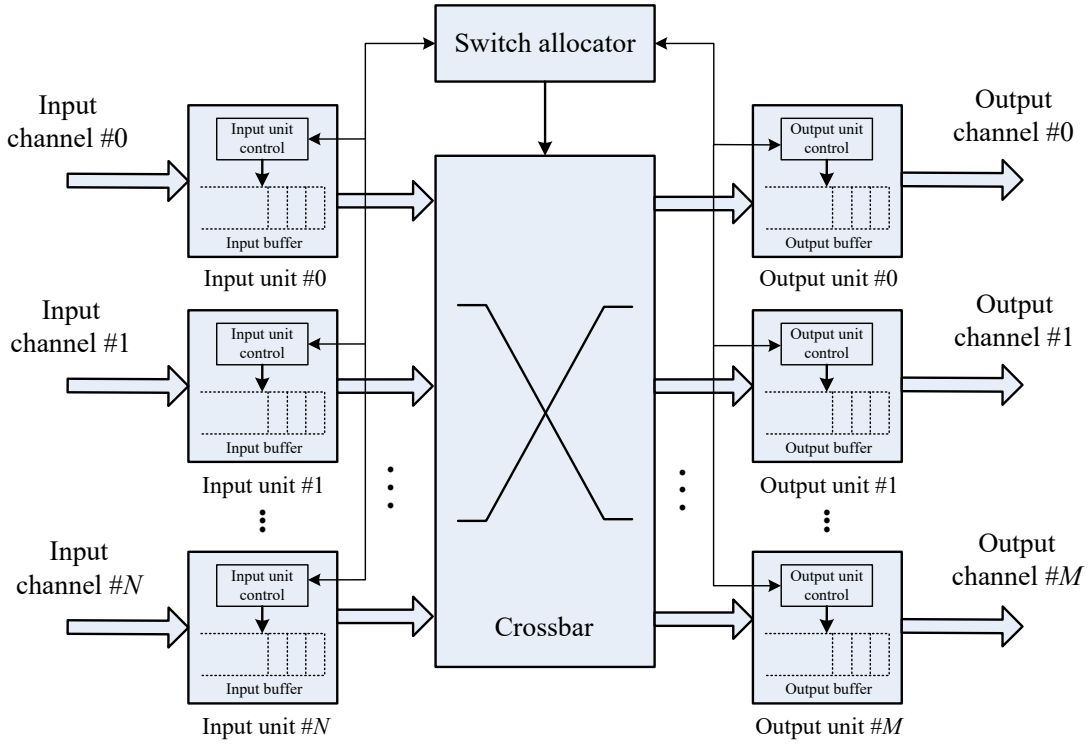


Figure 3.9: Synchronous router architecture (input/output buffers are optional)

the *switch allocator*, schedules these transmissions. We use fully-connected crossbars throughout the thesis. In a fully-connected crossbar, each pair of *input unit* and *output unit* is connected using dedicated datapath wires. Mux selection is used at the *output unit* side to select a particular *input unit* and allows data transmission from that *input unit* only. Incoming data from the remaining *input units* targeting the same *output unit* needs to wait until the current transmission is complete. Other crossbar implementations also exist [46], but are not introduced in this thesis.

3.4.2 Synchronous Router Structure with VCs: Two Structures

Virtual Channel (VC) capability can be added on top of the basic router implementation presented in Section 3.4.1. There are two widely-used structures for VC routers: *crossbar-sharing* and *crossbar-replication* [149].

The structure for a *crossbar-sharing* VC router is illustrated in Fig. 3.10, with an example of 2 VCs. The architecture is largely the same as the basic router without VCs. The basic functionality

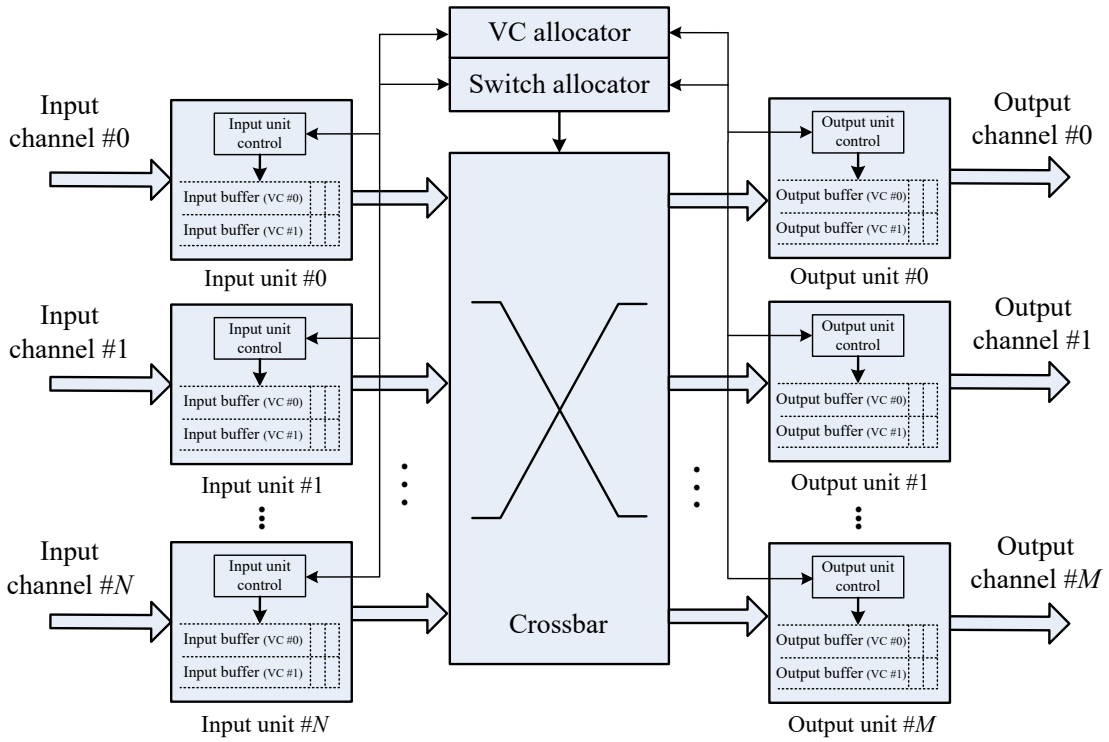


Figure 3.10: Synchronous VC router architecture: crossbar sharing

for each module remains the same, as presented in Section 3.4.1. In addition, the *input units* need to identify the data from different VCs. If static buffer management is used [103], as shown in the figure, data labeled with different VCs are stored in dedicated buffer queues, respectively.[1] The *output units* enable flit-level interleaving on the output channel, which send out a single flit per VC using round-robin. The *crossbar* has an additional control, a *VC allocator*, to assign a VC to each packet before the packet is transmitted from an *input unit* to an *output unit*.

The other widely-used VC router structure, *crossbar replication*, is shown in Fig. 3.11. The crossbar is replicated as many times as the number of VCs. Each crossbar is dedicated for processing data from a particular VC. VCs separate different traffic classes inside the router, which are mixed only on inter-router links. When a packet arrives, it is first assigned to a certain VC. Static buffer management is more suitable in this structure, Therefore, the packet is stored into the *input buffer*

[1] With adaptive buffer management, buffer spaces are shared with data from different VCs. The buffers are used more efficiently, while the control is more complex [103, 121, 161].

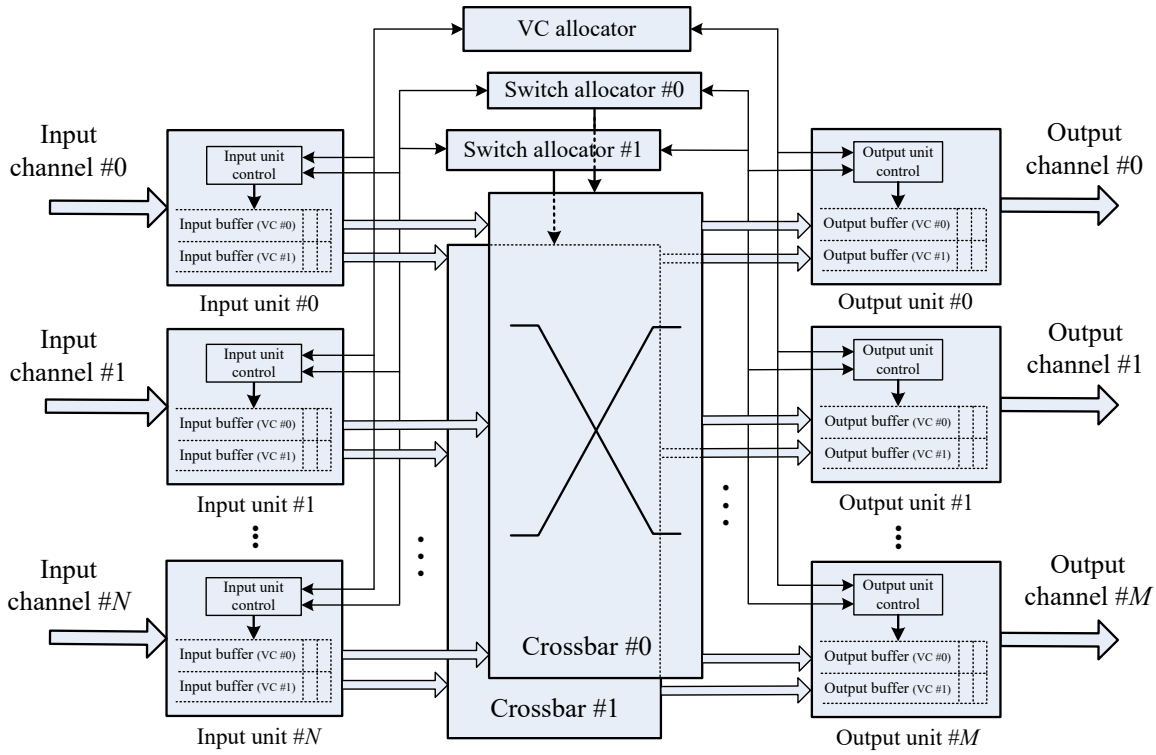


Figure 3.11: Synchronous VC router architecture: crossbar replication

for that particular VC. Then the packet traverses the corresponding crossbar. Finally, it is stored into the corresponding *output buffer* in the *output unit*, and merges with traffic of the other VCs on the output channel.

3.4.3 Router Pipelining

No matter which structure is used for the VC implementation, each flit requires similar steps to travel through the entire router. In a typical synchronous router, these steps are pipelined for achieving higher throughput [46].

The conventional pipelined operation for a 4-flit packet is shown in Fig. 3.12. The header flit incurs more steps than the body and tail flits, as it needs to set up the path and allocate resources for the entire packet. In an ideal case with no stalls, it requires five cycles (i.e. five steps) for a header flit to travel through each router. Header latency is an important parameter for network performance, since the header flit is always the bottleneck, while body and tail flits simply follow the header path.

Cycle	1	2	3	4	5	6	7	8
Header Flit	RC	VA	SA	ST	LT			
Body Flit #1				SA	ST	LT		
Body Flit #2					SA	ST	LT	
Tail Flit						SA	ST	LT

Figure 3.12: Pipelined operation for a 4-flit packet without stalls

The five steps for a header flit passing through a router are introduced in order: *(i) Routing Computation (RC)* calculates and determines the targeting output port for the packet. The operation is completed once the packet arrives at the *input unit*; *(ii) Virtual Channel Allocation (VA)* selects a VC and assigns the VC to the packet when there is available credit for that particular VC. An available credit usually corresponds to an empty buffer space at the *output unit*, or at the *input unit* of the downstream router when *output buffer* is not used; *(iii) Switch Allocation (SA)* searches for an available timing slot for the flit to travel through the switch; *(iv) Switch Traversal (ST)* actually sends the flit from the *input unit* to the *output unit*; *(v)* during *Link Traversal (LT)*, the flit is sent from the current router to the successor over the connected channel.

3.4.4 Pipeline Optimization: Speculation and Lookahead

Speculation and *Lookahead* are the two most common strategies used for router acceleration. For synchronous router implementations, these optimizations can reduce the number of pipeline stages, and thus improve network performance [46].

Speculation allows two pipeline stages to be performed in parallel, with the latter stage assuming that the previous stage will succeed. Both *SA* and *ST* stages can apply speculation technique.

For *speculative SA*, *SA* is now ideally performed in parallel with *VA*, as shown in Fig. 3.13(a). This can result in three different scenarios: *(i)* If both operations are successful, the speculation succeeds, since *VA* and *SA* are completed in one cycle. *(ii)* If only *VA* is completed, with an unsuccessful *SA*, the operation rolls back to a non-speculative case. The *SA* stage is delayed to the next clock cycle. *VA* does not need to be repeated again. *(iii)* Finally, if even *VA* cannot be completed,

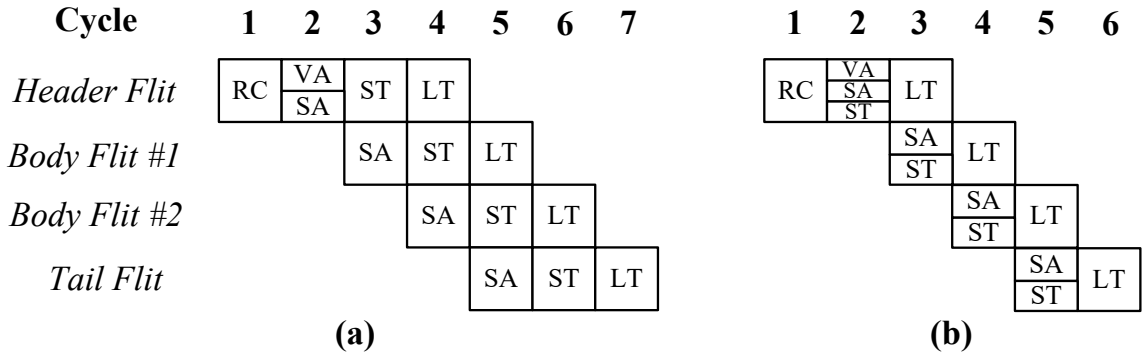


Figure 3.13: Pipeline speculation for a 4-flit packet without stalls:
(a) speculative *SA*; (b) speculative *SA*+*ST*

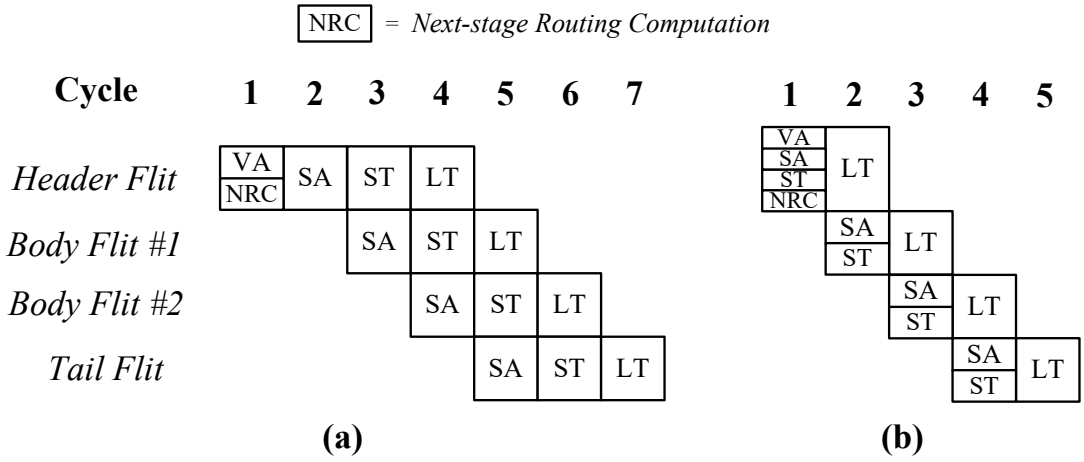


Figure 3.14: Lookahead routing computation for a 4-flit packet without stalls:
(a) lookahead *RC* only; (b) lookahead *RC* + speculative *SA*+*ST*

then both *VA* and *SA* are tried again in parallel for the next cycle. The result of *SA* in the current cycle is ignored, regardless of its success.

A similar speculation can be applied for *ST*. In this case, *ST* and *SA* are performed in parallel, ideally. Furthermore, combining both speculations above, three steps – *VA*, *SA* and *ST* – can thus be processed in a single clock cycle, as shown in Fig. 3.13(b).

Lookahead enables the *Routing Computation (RC)* of the current router to be completed in the predecessor node. When a header flit arrives at a router, the *RC* step is already completed and

therefore removed from the critical path. Unlike speculation, lookahead is always successful, if applied. The lookahead technique can be applied only when the input-output path for the next router is known in advance.

A timing diagram of *lookahead* strategy is shown in Fig. 3.14. The left figure shows the result when the technique is used alone, while the right figure shows when it is combined with speculation. Therefore, ideally, lookahead and speculation can shrink the pipeline depth to two cycles.

Chapter 4

A Low-Latency Asynchronous NoC for a Variant Mesh-of-Trees Topology

4.1 Introduction

This chapter introduces our first low-latency asynchronous network-on-chip [67]. The target network topology is the variant Mesh-of-Trees (MoT), combining a binary fan-out network (i.e. routing for each source node) and a binary fan-in network (i.e. arbitration) for each destination node, which was already introduced in Section 3.1.2. While this NoC topology is rarely used for embedded system-on-chip (SoC) platforms, it is receiving increasing attention as a foundation for shared-memory interface networks in high-performance parallel chip multiprocessors (CMP's), to provide needed bandwidth for globally uniform memory access. The network topology structure is again shown in Fig. 4.1.

Several recent shared-memory parallel processors are using MoT, or close variants, for core-to-memory (or cache) interfaces [10, 82, 89, 96, 97, 185]. Although MoT networks grow rapidly in size with the number of cores and memories, they are viable for medium-size parallel systems. In addition, extensions have been proposed to reduce area overhead through a hybrid MoT/butterfly topology, which maintains the throughput and latency benefits of MoT with the area advantages of butterfly [11].

In particular, among the above MoT networks, two of them use an asynchronous implementation. Our work is therefore carefully compared to those two networks. First, Horak *et. al* intro-

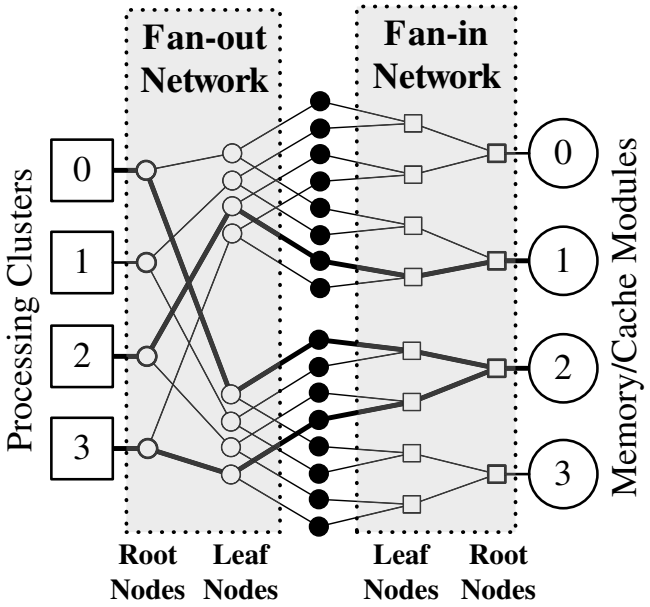


Figure 4.1: Mesh-of-trees: an efficient topology for connecting processors to memory

duces a new set of lightweight *routing* and *arbitration* primitives, and combines them to build a low-overhead asynchronous interconnection, which serves as our *baseline* approach [96, 97]. The NoC shows significant benefits under metrics of system latency, area and power. In particular, it provides much lower system latency (by 1.7×) than a comparable synchronous network operating at 800MHz, with identical throughput over the latter's entire operating range using a 90nm technology. Second, a more recent approach [82] adds limited *dynamic reconfiguration* capabilities to the network, using *prediction* based on local recent traffic history, resulting in further performance improvements for some benchmarks. This method will be called the *predictive* approach.

In practice, the communication network between cores and cache typically has light traffic, and the key bottleneck is transport time. However, both *baseline* and *predictive* approaches ([82, 96, 97]), are shown to hit a latency wall, due to the overhead incurred by two steps: arbitration resolution and input channel allocation. At medium to high input rates, and with adversarial traffic patterns, these overheads create a ceiling or limit to the overall end-to-end system latency.

Our proposed network uses a fundamentally different approach from previous high-performance asynchronous MoT networks, aimed at overcoming the latency bottleneck, while still maintaining

comparable throughput. In particular, a lightweight *monitoring network* is introduced, which provides rapid advance notification to all destination nodes, whenever a new flit enters the network. The network information allows each router node on its path to complete arbitration and pre-allocate the corresponding input channel, *before* the actual header flit arrival. As a result, in many traffic scenarios, the forward path through the network is directly optimized for end-to-end latency, and each arbitration node is configured to operate as a simple FIFO stage. A potential latency reduction of roughly 250 ps per switch: 150 ps for mutex set and 100 ps for channel opening. The approach is supported by the design of a new arbitration node for the MoT network. Significantly improved system latency results were obtained, across a diverse set of benchmarks. The proposed monitoring concept can not only facilitate early arbitration and channel pre-allocation for performance improvements, but may be suitable as well to observe run-time workload characteristics, for both synchronous and asynchronous NoC's. This direction is becoming important for dynamic voltage and frequency adjustment in the field of power management [29].

Since early arbitration and channel pre-allocation are applied to the fan-in network in the MoT variant topology, which is a network to merge traffic flows, a special and simple monitoring protocol is used. At a fan-in node, if monitoring tokens appear on either or both input channels, a monitoring token is immediately sent on the output channel, indicating something is coming within the fan-in cone, with no need to know which input monitoring actually wins the arbitration. Therefore, the monitoring network entirely bypasses the arbitration logic, and arbitration and monitoring forwarding operate in parallel. The monitoring output of a node is simply an OR of the two monitoring inputs. This optimization results in a considerably simpler and lightweight monitoring network design.

The remainder of this chapter is organized as follows. Section 4.2 presents existing latency acceleration techniques for general networks-on-chip. This discussion partially overlaps with Section 1.3.3 in Introduction chapter, but provides more details. Section 4.3 introduces the two asynchronous MoT networks, the *baseline* and *predictive* designs. They are prerequisites for understanding our new work. Before diving into the detailed network design, an overview of the new approach is presented in Section 4.4. Then, Section 4.5 provides the detailed design of the new network, and Section 4.6 extends the basic design to handle multi-flit packets. Simulation results for the proposed network are shown in Section 4.7. The chapter concludes in Section 4.8.

4.2 Related Work

Various acceleration techniques have been proposed for improving system latency in NoC's. However, most of them target synchronous NoC's.

An early single-cycle router in [156] speculatively allocates the switch as soon as a packet arrives at the router without any arbitration, assuming no other packets complete for the same output channel. However, it is only effective under congested scenarios where the input buffer contains adjacent flits, otherwise it relies on prediction. Router bypassing techniques have been also proposed in two recent approaches [122, 131], which provide additional dedicated high-speed VCs to enable single-cycle transmission. However, both networks assume baseline 5-ported routers with expensive 3-cycle operation, while our baseline uses low-radix routers with already heavily optimized performance (2.4 Gigaflits/sec throughput and 365 ps node latency in 90 nm technology [96, 97]) which is much more challenging to accelerate. In addition, significant restrictions are imposed in their approach: packets can only be accelerated within a small segment of 3-4 hops, then have to regain high-speed VCs for the next segment; acceleration is not allowed across turns; and both networks require packet priorities (either dynamically [122], or statically [131]).

In more recent work, WaveSync NoC [243] uses source-synchronous routing in a multi-synchronous domain, where the source clock is propagated on channels, which is a different focus from ours; it proposes acceleration techniques to avoid synchronization in non-congested router nodes. Another emerging technique, a mm-wave hybrid NoC [59], based on a small-worlds topology, uses wireless channels to connect a subset of long-range multi-hop links for acceleration. However, it relies the use of emerging technologies which are not yet fully available and which require high-cost transceivers.

A number of synchronous NoC's also use early information and aim for rapid pre-allocation. However, all of them use mesh-based topologies, and are closer to our work in Chapter 5. The introduction of these networks are thereby delayed to Section 5.2.

4.3 Background: Baseline and Predictive NoC Designs

In this section, two asynchronous networks, *baseline* and *predictive*, are introduced. They are a prerequisite to understand our new approach. Also, the proposed network is closely compared to

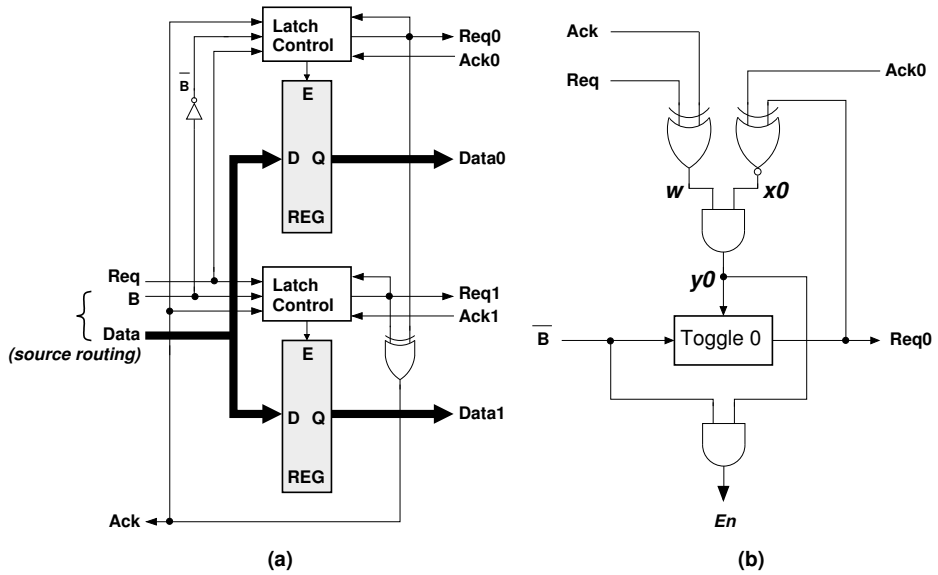


Figure 4.2: Baseline routing node: (a) top-level; (b) latch control

the these two previous networks in terms of area cost and performance. All the networks, including both previous designs and the new network, uses a transition-signaling (2-phase) protocol, along with single-rail bundled data encoding.

4.3.1 The Baseline Network

Two asynchronous components are used to create the original *baseline* network: *routing* and *arbitration* primitives [96, 97]. The variant MoT network can be directly constructed using these two fundamental building blocks.

Routing Primitive

The asynchronous routing primitive, shown in Fig. 4.2, receives one incoming stream and passes it to one of the two outgoing channels. When the node is empty, both top and bottom data registers are opaque. Following the transition-signaling bundled data protocol, the operation begins with new data arriving, including the binary routing signal B. After data inputs are stable, a request transition on *Req* occurs at the input channel. Then, the latch controller selected by the binary signal B, opens its latches, thereby allowing the data to flow to the designated output. To complete the asynchronous handshaking, three operations are performed in parallel: *(i)* data is sent out on the selected output channel, along with a request transition; *(ii)* the data register is made opaque again, to protect the

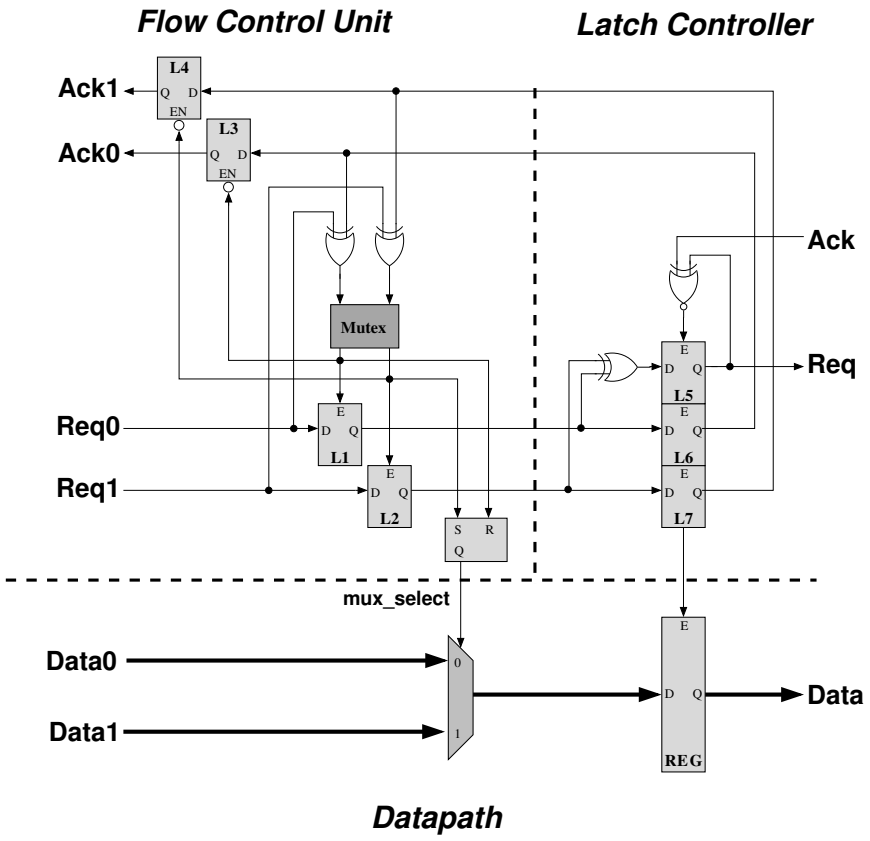


Figure 4.3: Baseline arbitration node

recently-received data; *(iii)* an acknowledgment transition is sent to the predecessor stage. This completes one full cycle of operation for the routing node.

Because the routing primitive has separate latch control and data registers for each output channel, the node is able to decouple processing between the two output routing channels. Even if one of the output channels is stalled awaiting acknowledgment, the other output channel can successively process *multiple* full transactions. This concurrency feature provides the capability of a limited virtual input channel, thereby providing significant system-level performance benefits [82, 96, 97].

Arbitration Primitive

The asynchronous arbitration primitive, shown in Fig. 4.3, mediates between two concurrent incoming streams of flits – enforcing mutual exclusion – and merges the result into a single serial outgoing stream.

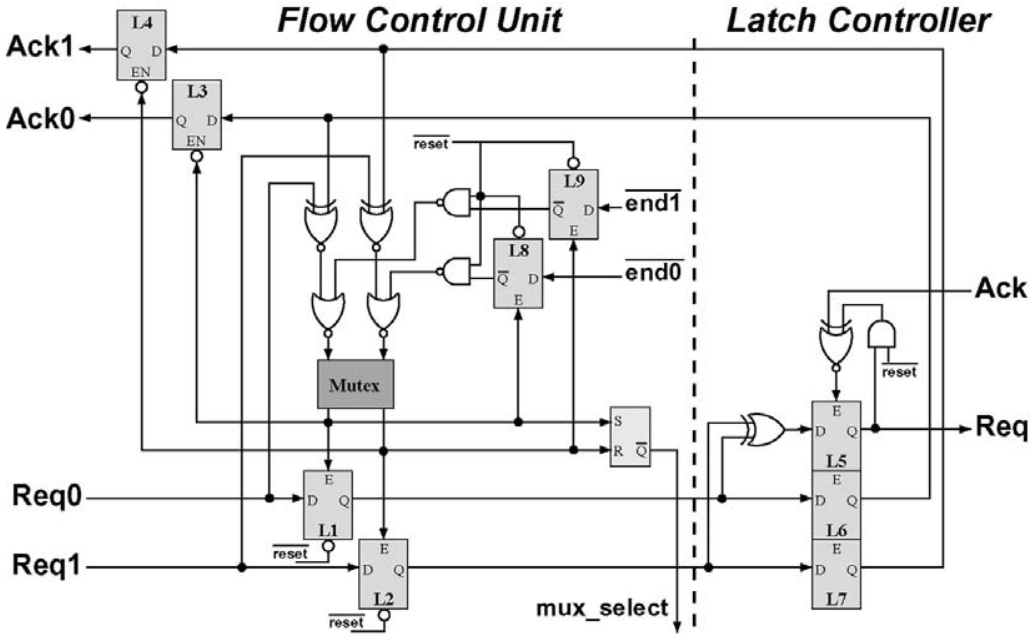


Figure 4.4: Baseline arbitration node: enhanced version to handle multi-flit packets

When the arbitration primitive is empty, the control latches L1 and L2 are opaque while all the other latches, including the data register, are transparent. Following the 2-phase bundled data protocol, operation begins when data appears at one of the input channels followed by a transition on corresponding req. To resolve contention between potentially concurrent incoming requests, the incoming request must first trigger a lock on the mutex. Next, two operations take place concurrently: 1) *mux_select* chooses the correct data input; and 2) either L1 or L2 is enabled, thereby forwarding the winning request. After the data is sent to the output channel, it is safely stored by making L5-L7 and the data register opaque. The mutex is then immediately reset and an acknowledge to the previous stage is generated. Finally, the right environment acknowledges the current stage and the entire operation is complete.

Enhanced Arbitration Primitive

An enhanced version of arbitration node is designed to support wormhole routing of multi-flit packets, shown in Fig. 4.4. As introduced in the Background chapter, Section 3.3.3, a flow-control unit, or flit, is the smallest granularity of message sent through the network. Wide packets are split into

multiple flits that travel contiguously through the network. In wormhole routing, once arbitration is won by a packet head flit in an arbitration node, each remaining flit in the packet must be guaranteed unarbitrated access through the node until the tail flit exits.

A 'kill your rival' protocol is implemented to bias the selection of the mutex, so that the next flit inside a multi-flit packet is guaranteed to advance without new arbitration. When the first flit wins the mutex, the opposing request input to the mutex is forced to zero, i.e. 'killed'. This either prevents future requests at the other mutex input from occuring, or in the case where a request was already pending, kills the opposing request until the entire multi-flit packet has advanced. The kill function is achieved using a NOR gate located at the input of the mutex [97].

4.3.2 The Predictive Network

The *predictive* network by Gill *et. al* [82] is the first attempt to optimize the *baseline* network. The network introduces a bi-modal arbitration node based on the earlier *baseline* approach. The node can operate in either normal mode or a 'single-channel-bias' mode. Since there are two input channels, it thus has three distinct modes: *default*, *bias-to-0* and *bias-to-1*.

The node enters bias-to-0 mode when recent traffic has been observed only on input channel 0, and input channel 1 is inactive (similarly, for entering bias-to-1 mode). In this case, it effectively operates as a fast-forward 'single input channel' node: its arbitration unit is entirely bypassed (operating vestigially in parallel, but is not observed), resulting in lower node latency. In addition, when single-input channel traffic is at moderate or high rate, higher node throughput is also obtained.

The node exits biased mode, reverting to default mode, when any flit arrives on the inactive input channel. Note that once the node is in biased mode, it always passes through default mode before changing to the other biased mode. Hence, all mode changes are between default and one of the biased modes.

In summary, a node's mode changes are determined entirely locally – at the node itself – based solely on its recent observed traffic history. Hence, adjacent nodes may be in entirely different states, at any time.

The *predictive* design uses the following *mode change* policy: *(i)* in default mode, if two successive flits from one input channel are processed by the node, change to biased; and *(ii)* in biased mode, if one flit arrives on the opposite input channel, revert to default.

4.4 Overview of the Approach

The section introduces the basic strategy of the proposed new approach. We also quickly review the operation and strategy of the *baseline* [96, 97] and *predictive* [82] networks, in order to have a head-to-head comparison.

The baseline design. In the baseline asynchronous MoT network [96, 97], the flit is forwarded through the fan-out network followed by the fan-in network, with a path determined by the flit's source-routed address bits. After a flit arrives at an arbitration node, the node first allocates and opens the channel. It then gets passed through and sent to the output. *Channel allocation only starts after the arrival of an actual header flit.* If there is contention, with incoming flits on both input channels, arbitration grants to exactly one of the requesting flits based on their relative arrival time. Since the design is asynchronous, arbitration is not performed at discrete clock boundaries, but in continuous time, using an analog arbiter. Once the winning flit is processed, the other flit can win arbitration.

The predictive approach. The predictive approach [82], as introduced before in Section 4.3.2, proposes enhanced arbitration nodes that can dynamically reconfigure themselves to accelerate the forward transmission of a stream of flits. In particular, if the node detects streaming activity on one input channel, and no incoming traffic on the other input channel, it *predicts* continued streaming activity (i.e. based on temporal locality) and puts the active input channel in an optimized state for improved performance.

Fig. 4.5(c) shows the network structure of the predictive approach, and Fig. 4.5(a) presents an arbitration node's operation. If all relevant nodes correctly predict and prepare for a flit in advance, an express path is created when the flit passes through the entire fan-in network. Each node operates normally as the baseline design, i.e. default mode, but enters a *single-channel-biased mode* when two consecutive flits arrive on a single channel. Once in biased mode, the arbiter is completely ignored and the biased channel is held open. As a result, *arbitration and channel allocation steps are entirely eliminated.* Subsequent flits arriving on the biased channel will be directly sent to the output. The node reverts to default mode when *any flit* arrives on the inactive input channel. The challenge with this approach is that misprediction or thrashing between modes may be induced, from adversarial traffic patterns.

A monitoring network is used to facilitate mode change operation for safety purposes only. It is

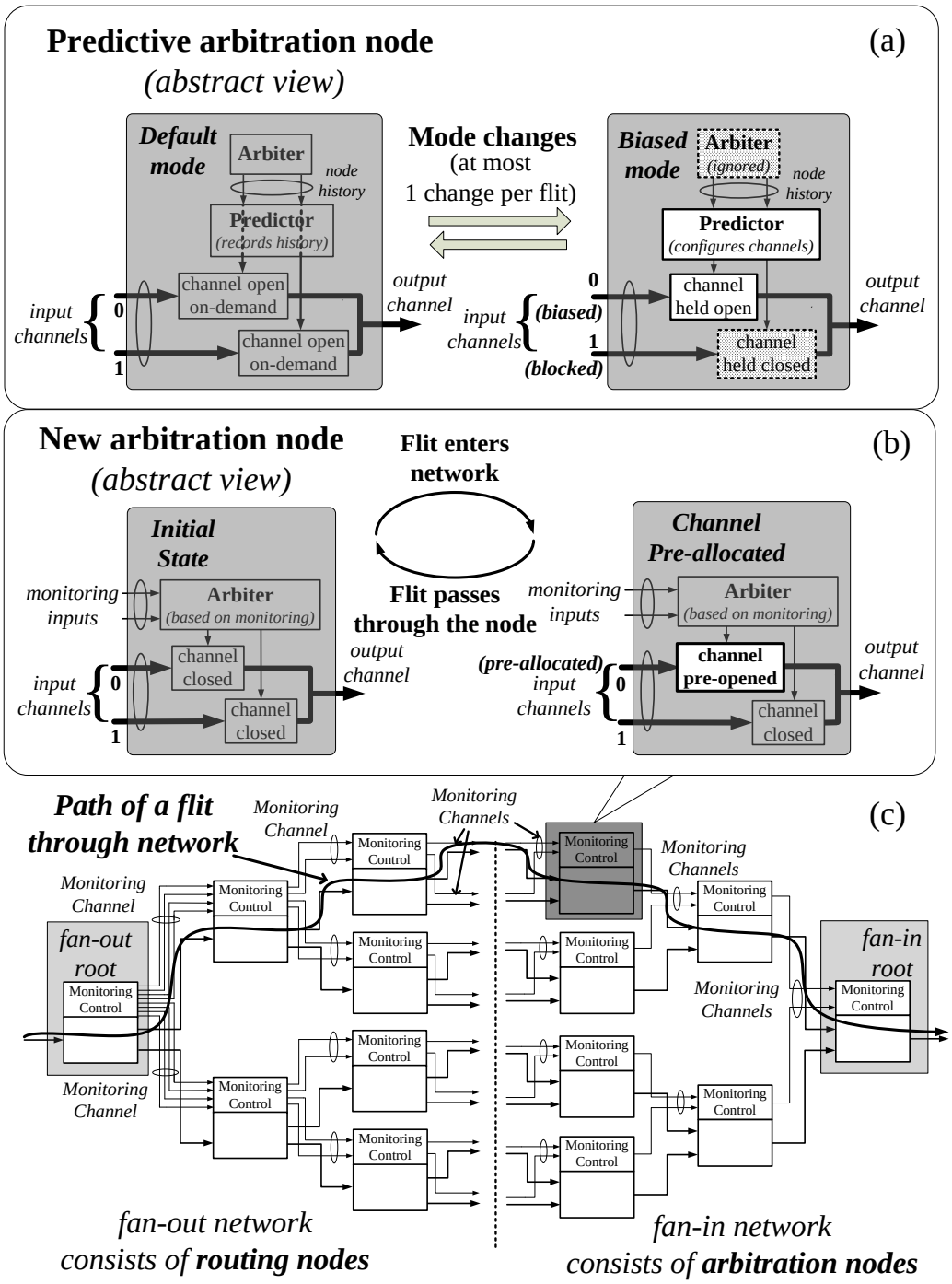


Figure 4.5: Block structure of MoT network: predictive and new approach

(a) Predictive node; (b) New node; (c) MoT network with monitoring

only used to enable a reversion from optimized (i.e. biased) to unoptimized (i.e. default) mode.

The new approach. This approach proposes an alternative protocol for accelerating the forward transmission of flits through the fan-in network. New arbitration nodes are designed, which *anticipate* the arrival of incoming traffic, rather than by prediction, by receiving an actual early notification through a monitoring network. Once an arbitration node has been notified of a pending header flit arrival, it rapidly performs early arbitration and pre-allocates the channel.

The overall network structure is the same as that of the predictive approach, but with fundamentally different operation. When a flit enters the network, injected by a source fan-out root node, advance notification is generated and fast forwarded to all the nodes along its path by the monitoring network. All arbitration nodes on the path *pre-allocate the corresponding input channel,* well before the actual flit arrives, as shown in Fig. 4.5(b). The input channel is then *kept open* until the flit passes through the node, as a combinational flow-through path with all latches transparent from a fan-in leaf to the fan-in root. When the flit arrives, it is directly sent through the pre-allocated channel to the output, *avoiding the overhead of a stop-and-go protocol*. The node then reverts back to the initial state and waits for the next operation. In the case of contention, arbitration is used to select one of the two input channels for pre-allocation; however, unlike the previous approaches, the arbitration is performed in advance, based on actual monitoring signals.

Several comparisons of the new vs. previous approaches are now highlighted. Similar to the baseline design, no history is recorded for the flits; this design decision allows a much simpler design. Similar to the predictive approach, in friendly cases, the arbitration and channel allocation are completely eliminated from the critical path, but using quite different strategies. However, the monitoring network has a major role in the new approach, as the node directly uses its advance notification to reconfigure in advance for higher performance. In contrast, in the predictive approach, monitoring serves a secondary role, to facilitate safe mode changes from optimized to default state.

4.5 Proposed Router Node Design

This section presents the design of the new network. Since there are two types of nodes in the MoT network, *routing* and *arbitration*, both of the new primitives will be introduced. We select to start with the new arbitration node because it contains most of the key changes, while the design of the

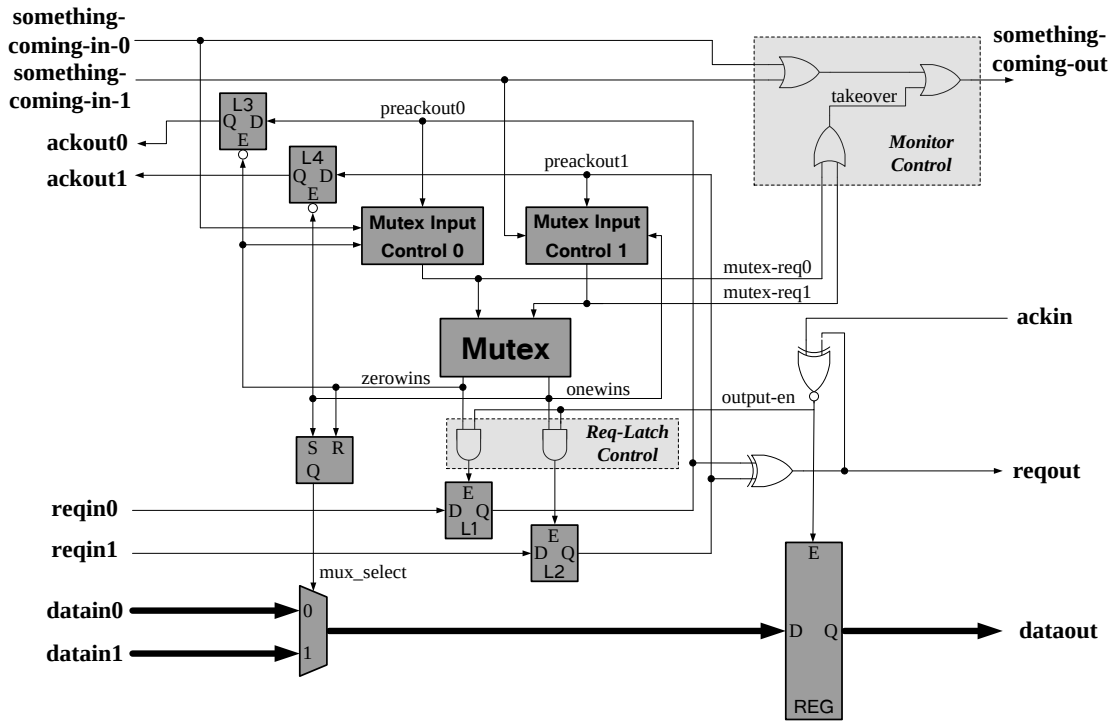


Figure 4.6: New arbitration primitive: single-flit design

new routing node is more straightforward.

4.5.1 Arbitration Node

The structure of the new node is illustrated in Fig. 4.6. Compared to the baseline design [96, 97], additional monitoring channels are added to support early arbitration capability. There exist three fundamental structural differences, compared to the baseline design: *(i)* two added *mutex input control* units, one for each input channel; *(ii)* a critical path from input to output with a single latch, while the previous designs use two latches; a new req-latch control is added to support this change; and *(iii)* a new *monitoring control* unit. These modifications facilitate early arbitration and allow the node to have extremely low latency, while still ensuring a simple design.

The *mutex-input controls* request and release the mutex during the operation. These controls initiate early arbitration, by asserting mutex request high whenever informed by early monitoring information. When a flit enters the network, *something-coming-in*, which is initially low, is asserted high, causing the control unit to request and win arbitration (i.e. mutex component). The control unit de-asserts its request, and releases the mutex, when the corresponding flit is sent to the output

channel.

The newly-added *req-latch control* unit enables a dramatic optimization in the node's forward path, reducing it from two latches [82, 96, 97] to a single one. This latch now serves two purposes: *(i)* channel pre-allocation based on the winning of mutex, and *(ii)* flow control. In contrast, the earlier designs use separate registers to handle the two purposes. This requires a significant protocol modification on the *req-latch control* unit, which now opens the latch in synchronization with both left and right channels: it awaits early arbitration to complete (assertion of a mutex output) and an acknowledge of the previous flit by the right environment, for flow control (*output-en* signal asserted high).

The *monitoring control* unit is part of the entire new monitoring network. It enables a rapid propagation of the monitoring information from the node's input to output. In particular, *something-coming-out* is asserted high as soon as either of the two *something-coming-in* signals is asserted high with only a two-gate delay latency. Since *something-coming-in* signals are not persistent (i.e. they can go low early), a *takeover* is needed to maintain *something-coming-out* high until the actual flit arrives and is sent to the output channel.

4.5.1.1 Operation

Before showing detailed component designs, the operation of the entire node is illustrated by three simple scenarios. Advanced scenarios can be simply built upon these three cases.

Single Flit Processing with No Contention

This scenario illustrates how early arbitration works in a friendly case. The incoming flit is assumed to arrive on channel 0, without loss of generality, and with no traffic on the other input channel. Initially, latches L1 and L2 are opaque, while all others (including data register) are transparent. First, when the flit enters the network, early arbitration is performed and channel 0 is pre-allocated, well before the actual flit arrival: *something-coming-in-0* is asserted high, causing the *mutex-input-control-0* unit to request the mutex. After the mutex is won (*zerowins* asserted high), latch L1 is pre-allocated (i.e. becomes transparent), and the data mux selects the top input channel. Eventually, the actual flit arrives. Since input channel 0 is now pre-allocated, the flit passes directly to the output channel. *A low forward latency is achieved: effectively one D-latch (L1) plus an XOR2.* Finally,

both L1 and the data register quickly close for data protection. In parallel, the mutex is released, allowing the left acknowledgment to be sent out through L3. In the end, the right environment acknowledges, and the entire operation is complete.

Two Flits Back-to-Back on the Same Channel

This case illustrates when a data stream arrives on only one input channel, e.g. channel 0. The first flit is processed similarly to the *single-flit* case above. In this scenario, though, the monitoring signal, *something-coming-in remains high,* indicating that there are more incoming flits. Note that the mutex is always briefly released between flits (*i.e. zerowins* de-asserted low), and *something-coming-in* is then re-sampled by the *mutex-input-control-0* unit for the second flit. The second flit on this input channel is then processed following the same procedure as the first one. Finally, *something-coming-in* is de-asserted low as no more flits are arriving. The mutex release between the two flits is forced to create a window for the other channel to have an opportunity to win the mutex. As a result, the arbitration is fair and starvation is avoided.

Contention between Two Channels

This scenario illustrates a simple case for the node to resolve contention. Two flits, one at each input channel, enter the network within a close time margin. Both *something-coming-in* inputs go high and the two *mutex-input controls* request the mutex almost simultaneously. Without loss of generality, assume channel 0 wins the mutex. The flit on channel 0 is processed normally by following the same procedure as in *single-flit* case, while the early arbitration on channel 1 cannot further proceed. As soon as channel 0 releases the mutex, channel 1 continues with its own early arbitration and the flit on channel 1 gets processed.

4.5.1.2 Details for Sub-Modules

Mutex Input Control

A more detailed view of this control unit and its operation are now presented. The new arbitration node contains two *Mutex Input Control* units, one for each input channel, as shown in Fig. 4.7(a). They are the core components that initiate early arbitration whenever notified by monitoring signals.

In processing each flit, the *Mutex Input Control* unit requests and releases the mutex exactly

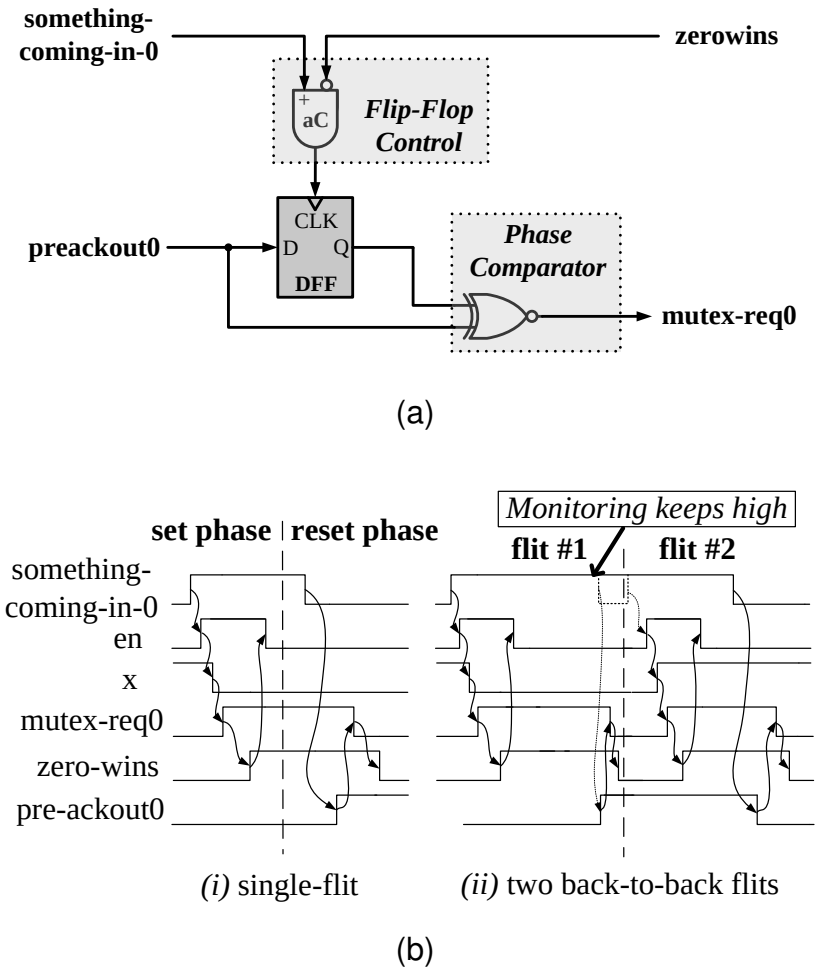


Figure 4.7: Mutex input control for single-flit design:
(a) implementation (b) timing diagram

once, generating a clean pulse at its only output *mutex-req0*, as shown in Fig. 4.7(b)*(i)*. An edge-triggered D-flipflop and a phase comparator are the core components for pulse generation. The phase comparator initially has different values at the two inputs and *mutex-req0* is zero. When a new flit enters the network and the local monitoring signal *something-coming-in-0* is asserted high, the *Flip-Flop Control* clocks the DFF. The two inputs of the phase comparator become equalized, and the control output (*mutex-req0*) is asserted high, resulting in a request to the mutex. (Once the mutex has been won, its *zerowins* output will then de-assert the *Flip-Flop Control* output.) Eventually, once the actual flit arrives at the node and is sent to the output channel, *pre-ackout* toggles. As a result, the bottom input to the phase comparator has a different value from the top

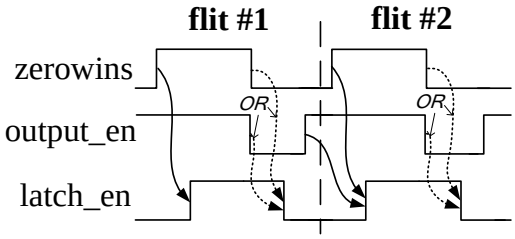


Figure 4.8: Timing diagram for req-latch control

input. Hence, the *mutex-req0* output is de-asserted low, thus releasing the mutex and also completing the output pulse.

A timing diagram for two back-to-back flits on the same channel is shown in Fig. 4.7(b)*(ii)*. If flits are very close, *something-coming-in-0* stays high; alternatively, with a wider gap, it will briefly go low.

Req-Latch Control

This unit is implemented by two simple AND2 gates, one for each input channel already shown in Fig. 4.6. A timing diagram of the processing of two successive flits, widely spaced, is illustrated in Fig. 4.8.

The output of each of the AND2 gates, which opens the corresponding input channel, is asserted high based on a synchronization of its two inputs. For each AND2 gate, its left input (*zerowins/onewins*) serves as a *request for input channel allocation,* controlling the corresponding latch (L1 or L2). It is asserted high when the corresponding monitoring signal has arrived and wins early arbitration, and is de-asserted low after the actual flit arrives and is transmitted to the output channel. In particular, the corresponding *preackout0/1* signal toggles, causing the mutex to reset, and thereby de-asserting the corresponding mutex output (i.e., *zerowins/onewins*). The right input (*output-en*) has the role of flow control; it is a busy flag indicating whether the successor is free to accept a new data item. It is initially de-asserted high, indicating a free output channel. Once the flit arrives and is sent out on this output channel, the *reqout* signal toggles and asserts the busy flag (*output-en*), as well as closing the data register. Once the right acknowledge *ackin* is toggled, the busy flag is de-asserted high.

For the case of downstream congestion, the right environment may take arbitrarily long to gen-

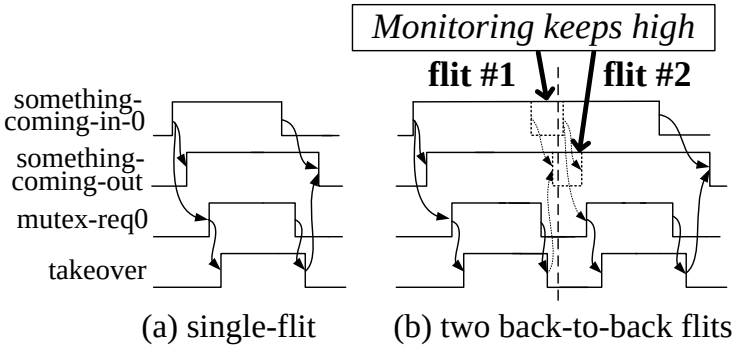


Figure 4.9: Timing diagram for fan-in monitoring control

erate its acknowledgment. Even so, if the advance monitoring indicates a second flit heading to the same input channel, the node has sufficient parallelism to begin the pre-allocation step: it can win arbitration (i.e. assert *zerowins/onewins* high), and request to enable the input channel. At this point, as soon as the downstream congestion is cleared, the corresponding latch, L1 or L2, will be finally be allowed to open again.

Monitoring Control

The unit is implemented using three simple AND2 gates, already shown in Fig. 4.6. The timing diagram for its operation is shown in Fig. 4.9.

Consider a simple simulation for an individual fan-in node and a single flit, as shown in Fig. 4.9(a). Assume the flit arrives on channel 0. When the flit enters the network, the node soon is notified: *something-coming-in-0* goes high and this information is quickly forwarded to *something-coming-out*. Concurrently, *takeover* is asserted, to maintain this output value. Eventually, the actual flit arrives, followed by *something-coming-in-0* going low. but *something-coming-out* stays high due to the *takeover* signal. Finally, after the flit is sent out on the output channel, the *takeover* is reset and allows *something-coming-out* to be de-asserted low.

In the case when there are two back-to-back flits approaching the node on channel 0, monitoring signals will stay high, until both incoming flits are processed. The timing diagram for this special case is illustrated by Fig. 4.9(b). Effectively, the monitoring signals are not normal handshaking signals, and no longer get reset when two consecutive flits are sufficiently close. It serves as a state bit, indicating the existence of incoming flit(s). Alternatively, runt pulses are possible for

the monitoring signal, i.e. a brief de-assertion, when the flits are somewhat further apart. This aggressive protocol is much more efficient than a classical deterministic handshaking protocol where each asserted signal is required in turn to be de-asserted.

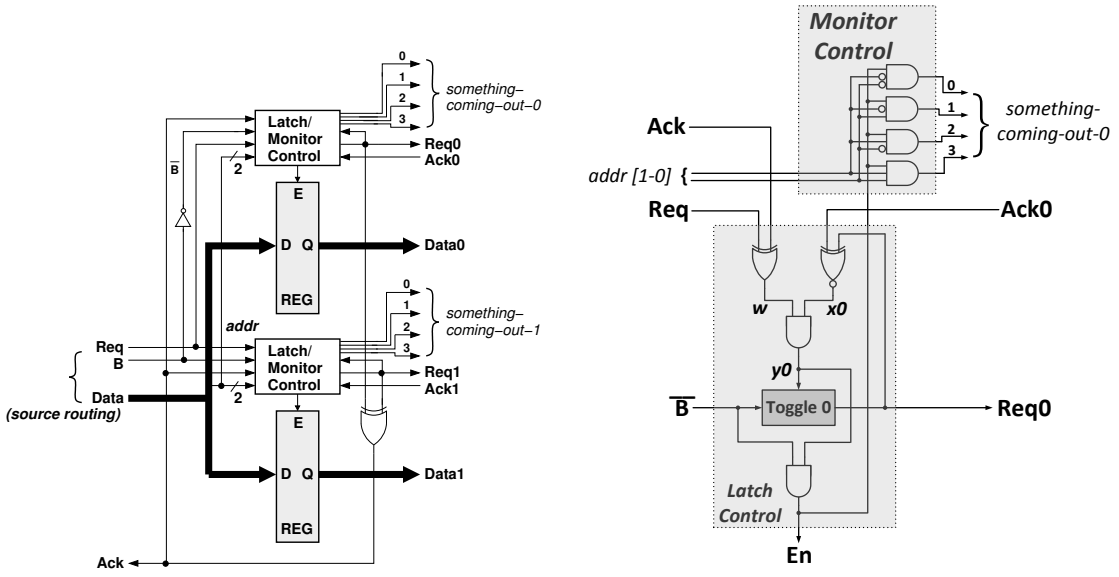
4.5.1.3 Timing Analysis

There are three key timing constraints residing in the arbitration node, which serve as design parameters for the physical layout level.

Monitoring operation: re-sampling time. Once asserted high, the monitoring input (*something-coming-in*) must be de-asserted early enough to prevent a stale value from being re-sampled by the *mutex-input control* unit, otherwise the input channel will be re-allocated with no incoming flit. In practice, given the gate-level implementation, this timing condition has adequate margins. Soon after a flit arrives, the corresponding monitoring input signal (*something-coming-in*) is de-asserted low and immediately forwarded to the *mutex-input control* unit, thus nullifying the sampling signal. Concurrently, after the flit is transmitted to the output channel, an entire mutex release cycle is initiated, followed by a re-acquisition, before the *mutex-input control* unit can re-sample the monitoring input.

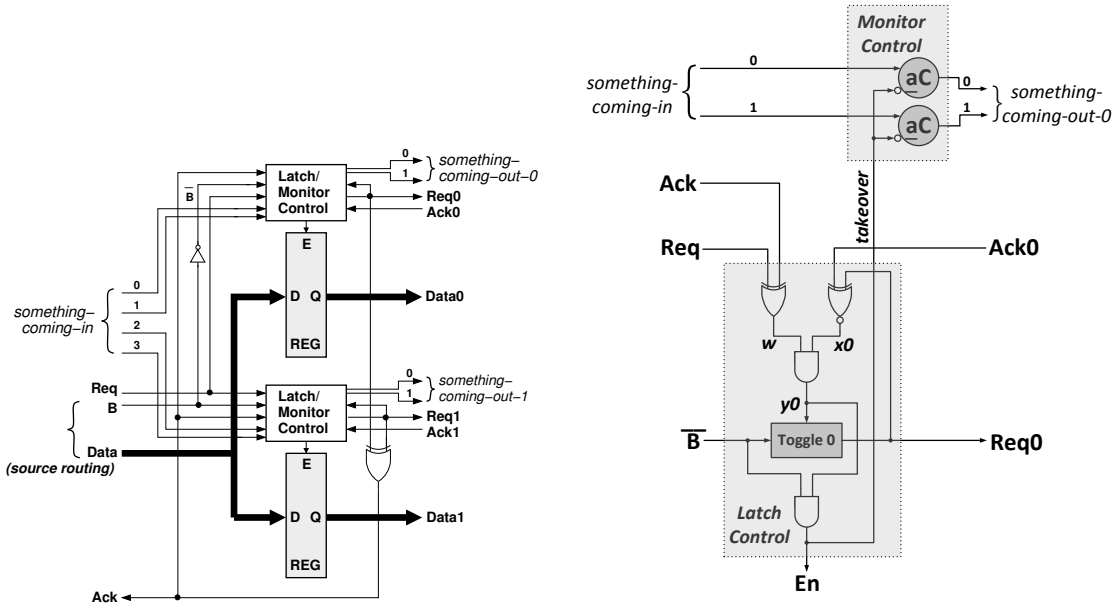
Hold time. Once data advances through the latch (L1/L2), it must be securely stored before new data arrives from the previous stage to prevent data overrun. In practice, L1/L2 closes nearly simultaneously with generating the left acknowledgment, because it involves small local paths. In contrast, the roundtrip time from generating the left acknowledgment to new data arrival involves one complete non-local channel traversal as well as paths through the left neighbor's control. The timing constraint, therefore, is easily satisfiable.

Pre-mature input channel opening. This timing constraint occurs whenever a flit is sent on the output channel. The right environment cannot acknowledge too quickly, before the mutex is released (*i.e. zerowins/onewins* goes low). Otherwise, L1/L2 will temporarily open for a short period; although no functional error can occur, possible glitches in control signals and other electrical issues may become a problem. In practice, as the right environment is normally another router of the same type, and the round-trip latency from generating req-out to receiving acknowledgment involves a complete non-local channel traversal, the timing constraint is also easily satisfiable.



(a) (b)

Figure 4.10: New root routing node: (a) top-level, (b) control logic



(a) (b)

Figure 4.11: New non-root routing node: (a) top-level, (b) control logic

4.5.2 Routing Node

In order for possible channel pre-allocation in arbitration nodes, lightweight monitoring control units are also added in each level of routing nodes. The new routing designs are shown in Fig. 4.10 and 4.11 for fan-out root and non-root nodes, respectively.

A monitoring transition is initiated at the fan-out root when the data enters the network and been quickly forwarded to all the routers along the path, including fan-out non-root nodes as well as fan-in nodes. However, unlike the arbitration primitives, the fan-out nodes only passes the monitoring information downstream without using it.

There are major differences in the protocol of monitoring signals between the new design and the predictive design of Gill *et. al* [82]. The time of monitoring assertion is the same but the time of de-assertion is significantly different. The monitoring channel of the predictive design uses a partial handshaking protocol and *something-coming-out* de-asserts only after we send out a flit and the right environment acknowledges. The new design, however, has no back pressure in monitoring protocol. *Something-coming-out* de-asserts soon after a flit is put onto the output channel.

4.5.3 Monitoring Network: A Quick Revisit

The monitoring network is the key innovation of the entire new design. This lightweight shadow network effectively anticipates the incoming traffic, when an actual new packet is injected into the network, and allow routers on the path to prepare in advance. Although the detailed design and operation for each of the fan-out fan-in monitoring control are already introduced in previous sections, it is still worthwhile re-examining it for its system-level operation.

Overall, the structure of the monitoring network is a shadow replica of the entire MoT network. For modularity, it is combined into the existing MoT network. Each node – routing or arbitration – has an attached small *monitoring control* unit, implemented by several gates. The monitoring control units for fan-out primitives, shown in Fig. 4.10 and 4.11, are similar to those in Gill *et. al* [82], with nearly identical functionality but using a different protocol in monitoring communication between neighbors. The *monitoring control* unit for each arbitration node was shown in Fig. 4.6.[1]

[1]The two top OR2 gates are combined using an OR3 gate in the actual implementation for improved performance.

Monitoring has distinct roles in the two halves of the network: in the fan-out part, monitoring signals only gets forwarded; in the fan-in part, monitoring signals get forwarded and are also used for channel pre-allocation at individual nodes. Whenever a flit enters the network through a root routing node, it initiates a monitoring signal transition. This signal is then rapidly forwarded to all the fan-out and fan-in nodes along the targeted path. A rapid wave of monitoring signals going high is detected as the relevant nodes get informed from the source to the destination only on this path. Then, monitoring signals are de-asserted low, node by node, as the actual flit proceeds through each router following the same path.

4.6 Multi-Flit Design

The previous Section 4.5 focused on a basic design to handle only single-flit packets. An enhanced version for the new arbitration node is now presented to support multi-flit capability, as in [96, 97].

The multi-flit network has identical routing primitives as the single-flit implementation, but with different arbitration nodes. The structure of a multi-flit fan-in node is shown in 4.12. It is largely identical to the single-flit one, except for a different *mutex-input-control* unit. The new control unit, shown in Fig. 4.13 replaces the one for *single-flit* design. In the new arbitration node, once the header flit wins in early arbitration, the mutex is held until the entire packet is processed. To implement this, the new *mutex-input-control* now takes a tail flag (*end0/1*) information from each flit. Normal assertion of *mutex-req* is performed as before, but the de-assertion (*i.e.* arbitration release) only occurs at tail flit.

Compared to the multi-flit design in [96, 97], the new approach no longer cycles the mutex on a per-flit basis, which was repeatedly won and released. Therefore, the new design has direct benefits in reduced switching activity, as well as improved throughput due to the complete elimination of this protocol overhead.

4.7 Experimental Results

Detailed evaluations are now presented, for pre-layout technology-mapped network primitives as well as 8×8 mesh-of-trees networks built upon these primitives. The results are compared to the

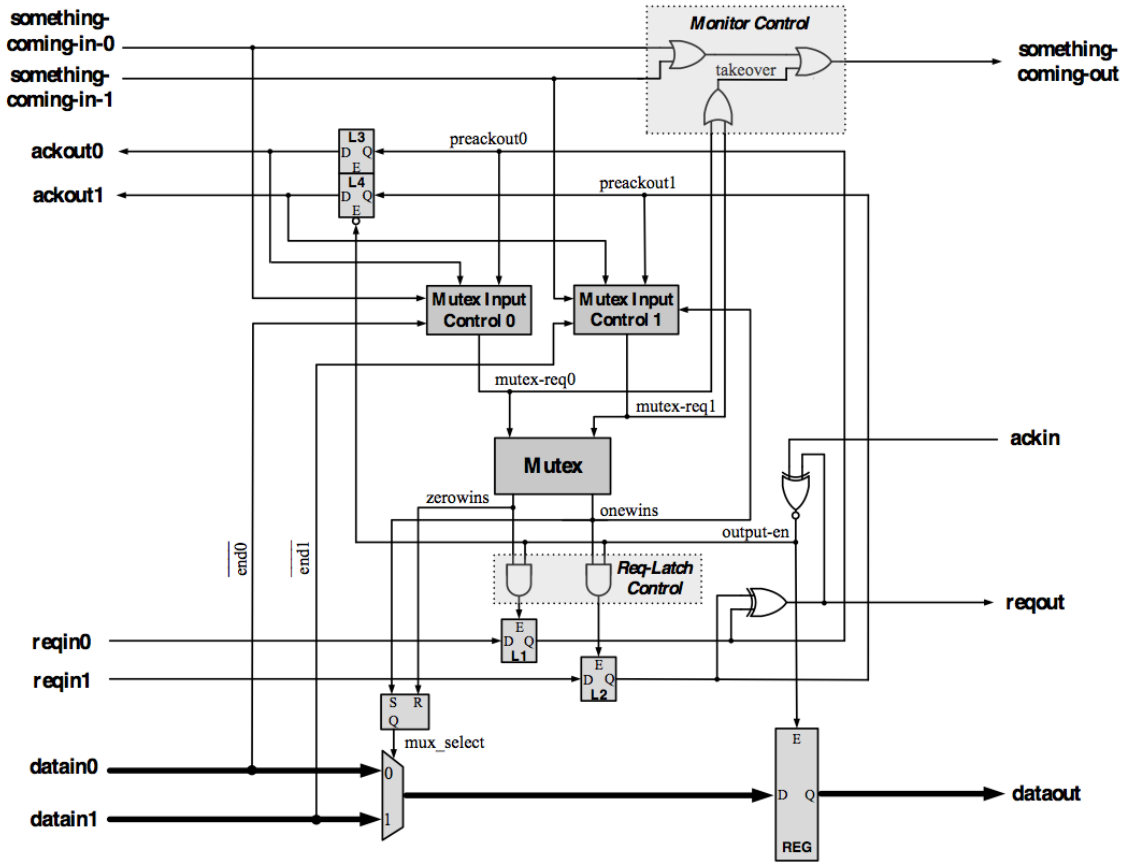


Figure 4.12: New arbitration primitive: multi-flit design

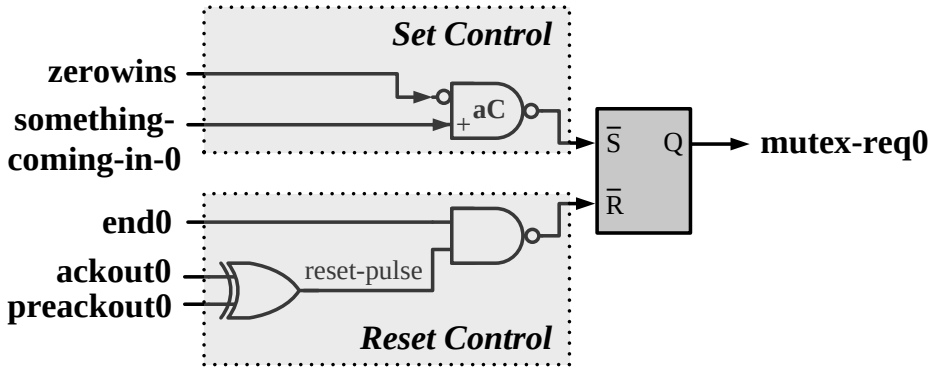


Figure 4.13: Mutex input control: multi-flit design

Table 4.1: Area comparison for pre-layout primitives (μm^2)

Node Type	Version		Included Components	New Implementations
Routing	*baseline*		***total***	**991.4**
			control only	201.1
	predictive	*root*	***total***	**1056.6**
			control only	266.4
		typical	***total***	**1033.7**
			control only	243.4
	new	*root*	***total***	**1042.5**
			control only	252.3
		typical	***total***	**1039.0**
			control only	248.7
Arbitration	*baseline*		***total***	**869.1**
			control only	190.5
	predictive		***total***	**1309.8**
			control only	632.4
	new		***total***	**947.3**
			control only	269.9

baseline design of Horak *et al.* [96, 97] as well as the *predictive* design of Gill *et al.* [82]. Both of the previous designs are re-implemented for fair comparison.

4.7.1 Asynchronous Primitives

An ARM 90nm SAGE-X standard cell library is used (the same as that used in [82]) for implementing the new and previous network primitives with a 32-bit wide datapath. Results are obtained using the Spectre simulator in Cadence Virtuoso environment at typical design corner with nominal temperature and supply voltage (1.0V, 27°C). As we use a different simulator from that of [82], reasonable results deviations are observed. All the results are obtained for the single-flit designs.

Area Comparison

The new primitives are shown to have only little area overhead over the *baseline* counterparts. In particular, the new arbitration primitive has significantly smaller area than the *predictive* one. Table 4.1 contains the results of total node area, as well as the control area alone, which excludes data

Table 4.2: Performance comparison for routing primitives

Version	**Latency*(ps)***	**Max. Throughput *(GFPS)***	
		Single	**Alternating**
Baseline	344	1.62	1.99
Predictive	344	1.58	1.99
New	344	1.62	1.99

path. The final layout node area is estimated by summing up the cell areas and then dividing by a packing factor of 0.8. For the routing primitive, the new design has a comparable area as *predictive* counterpart. When compared to the *baseline*, the control overhead of the new router primitive is within 5.2%, for both *typical* and *root* nodes, as the additional monitoring control is extremely simple. For the arbitration primitive, 28% less area is observed for the new node when compared to the *predictive* counterpart, due to a much simpler fan-in design. Even compared to the *baseline* arbitration primitive, the area overhead is less than 9%.

Latency and Throughput

Latency is calculated assuming an empty primitive, from an input request transition to the time that the primitive toggles its output request. A complete 3-level fan-in tree is implemented for evaluating maximum throughput under different steady-state traffic patterns in order to capture the realistic handshaking protocol overhead between neighboring nodes and avoid over-optimistic performance analysis. Then throughput is measured at the root primitive of the tree.

The new routing primitive has very close performance to both previous counterparts as expected, since effectively no design modification is done in the fan-out nodes except for additional monitoring controls off the critical path. The new arbitration primitive has significant latency reduction comparing to the *baseline*, and moderate to large latency improvement over the *predictive* arbitration primitive, depending on different operating modes that the *predictive* node operates in. There is a mix of throughput improvement and degradation for the new arbitration primitive, compared to the previous counterparts.

Routing primitive. Table 4.2 shows the latency and throughput results for the new routing prim-

Table 4.3: Performance comparison for arbitration primitives

Version	**Operating Mode**	**Latency*(ps)***	**Max. Throughput *(GFPS)***	
			Single	**Alternating**
baseline	N/A	416	1.35	2.28
predictive	Default	472	1.25	2.00
	Biased	264	1.77	N/A
new	N/A	215	1.42	2.43

itive. In order to show a complete range of performance, two different routing patterns are used for experiments: *single* (*i.e.* packets are only routed to one output port) and *alternating* (*i.e.* packets are routed to both output ports with a strict alternating pattern). Neither latency nor thoughput overhead was observed for the new routing nodes, in either *root* or *typical* versions. The throughput in the *single* case is even slightly better than *predictive* counterpart by an amount of 2.5%.

Arbitration primitive. Table 4.3 provides the performance results for the new arbitration primitive. Similarly, experiments are done under two different traffic patterns: *single* and *all*, respectively.

The new arbitration primitive has significant latency reduction comparing to the *baseline* (by 48%), and moderate to large latency improvement over the *predictive* arbitration primitive, depending on different operating modes that the *predictive* node operates in (54% improvement in *single* case and 19% in *all* case).

In terms of throughput, there is a mix of improvement and degradation for the new arbitration primitive, compared to the previous counterparts. In *single* case, data stream arrives on only one input channel. The new design has minor to moderate throughput improvements over the *baseline* and *predictive* (default mode), by 5.2% and 13.7% respectively, mostly due to a simple design with shorter forward path. However, about 20% worse throughput is observed, compared to the *predictive* (biased mode). This is because at high data input rate, there is not enough margin between two consecutive flits and the new design cannot fully complete channel pre-allocation before each flit arrival, while the *predictive* design operates in biased mode and the channel is always allocated. In *all* case, data streams arrive at both input channels. The new design also has better throughput, compared to both the *baseline* and *predictive* counterpart, in particular, 20% better throughput for

Table 4.4: Latency for monitoring control: node-level

Node Type	Node Level	Event	Transition Type	Latency *(ps)*
Routing	*root*	***Monitoring output*** *(something-coming-out)*	0→1	240
			1→0 (case #1)	396
			1→0 (case #2)	385
	typical		0→1	40
			1→0 (case #1)	257
			1→0 (case #2)	245
Arbitration	*N/A*	***Monitoring output*** *(something-coming-out)*	0→1	41
			1→0 (case #1)	263
			1→0 (case #2)	364
		Pre-allocation *(reqlatch-en)*	0→1	463

the latter comparison.

Monitoring signals. Table 4.4 evaluates the performance of the new monitoring signals, as they advance through a single node. For root routing primitive, the assertion (0 to 1) latency is measured from an input request transition to the time that *something-coming-out* is asserted high. For all the other routing and arbitration primitives, it is the time between *something-coming-in* and *something-coming-out*. The de-assertion delay (1 to 0) is divided into two cases. Case #1 is measured when a node is empty, from the time that a new request arrives until *something-coming-out* is de-asserted. Case #2 is for the case that a node is congested and is measured from the time that an acknowledge from the successor arrives until the time that *something-coming-out* deasserts. The assertion latency needs to be small, as the monitoring signal must be quickly forwarded, while the de-assertion latency is less important. Overall, for all non-root routing primitives as well as all arbitration primitives, the assertion delay is extremely fast: under 41ps. Pre-allocation time is also evaluated for the arbitration primitive, from *something-coming-in* assertion to the point when the channel is opened. The result, 463 ps, is fast enough for every fan-in node on the path to complete channel pre-allocation before actual flit arrival when the network is lightly loaded.

Design Complexity

The *new* arbitration node is also much simpler than the *predictive* one. It uses only a single analog mutex, while the *predictive* design requires five mutexes to solve complex mode change scenarios. The *predictive* node also contains complex *Policy* and *Safety* modules to store history information and maintain channel configuration. The *new* design requires none of these, and has simpler req- and ack-latch control units. However, the *new* design has two added small *mutex-input-controls* to allow channel pre-allocation.

4.7.2 Asynchronous Network

Experiments were also performed for both network latency and throughput, using 8×8 mesh-of-trees networks. We conduct two sets of experiments: for single-flit and multi-flit designs. Single-flit experiments compare three networks: the *baseline*, *predictive* and *new*. For multi-flit experiments, only two networks are compared, *baseline* and *new*, since the *predictive* design does not support multi-flit packets.

Experimental Setup

A similar network setup is used as in [82]. We use an asynchronous NoC simulator introduced in [82] developed by our group for testing, debugging and performance measurements, based on a PLI (Programming Language Interface) framework.

The initial generic framework was developed by Prof. Simha Sethumadhavan (Columbia University) for synchronous applications only. Networks are modeled in structural Verilog using the same 90nm standard library while the test environment is written in C. Packet source queues are installed at network inputs ports for recording of latency [46]. For single-flit experiments, the input environment generates flits at random intervals that conforms to an exponential distribution. For multi-flit experiments, flits within a packet are put into the source queue almost simultaneously and the mean time between packet headers follows an exponential distribution. We follow the standard procedure to ensure a long enough warm-up and measurement time for each benchmark [46], where one simulation with given warm-up and measurement phases is compared to another with both of these periods doubled, to check whether results are comparable.

Gate modeling. Our previous work of [82] employs a fixed-latency gate model by taking the

average of rising and falling transitions. Here a more accurate model is extracted by Cadence simulations, which assigns distinct latencies to rising and falling transitions for each pair of I/O paths of each gate, rather than using a single fixed latency model.

Benchmarks. Experiments are conducted for the same eight synthetic benchmarks as in [82], chosen to represent a wide range of network conditions. Three benchmarks provide friendly scenarios with no contention, where each fan-in node receives data on only one input channel: **1)** *Bit permutation* [46], **2)** *Digit permutation* [46] and **7)** *Random single source broadcast*. Three benchmarks provide moderate contention, where some fan-in nodes have light or no contention, and others have moderate contention: **4)** *Simple alternation with overlap*, **5)** *Random restricted broadcast with partial overlap* and **8)** *Partial streaming*. The two remaining benchmarks are most adversarial, with heavy contention at some fan-in nodes: **3)** *Uniform all-to-all random* [46] and **6)** *Hotspot8* [46]. The *predictive* network has particularly poor performance on these two, even worse than the *baseline* network.

Network-Level Simulation Results: Single-Flit Design

Overall, for the single-flit experiments, the *new* network has moderate to high latency reductions across all benchmarks, over both *baseline* and *predictive* networks, ranging from 13 to 39%. For throughput, the *new* network out-performs the *baseline* network, in six of eight benchmarks, with improvements up to 17%, and only minor degradation in the other two benchmarks. Compared to the *predictive* network, the new network has almost identical throughput, within 6%.

Latency. Fig. 4.14 contains detailed latency results for the three networks. Results are plotted as the average end-to-end network latency for each flit vs. the mean offered data input rate [82]. When lightly loaded, network latency in the *baseline* network is nearly 2400 ps for every benchmark. The *predictive* network has a latency around 1900 ps for friendly benchmarks but up to 3000 ps for the adversarial ones. Like the *baseline,* the *new* network has a quite stable latency over all benchmarks; however, the latter is overall much lower, around 1700 ps. Fig. 4.16 highlights network latencies at identical input rates. An offered throughput of 25% saturation rate is chosen; this rate is high enough to differentiate benchmarks while still retaining an uncongested network.

Compared to the *baseline* network, significant improvements are uniformly observed, ranging from 23% to 30%. Compared to the *predictive* network, the *new* network obtains improvements

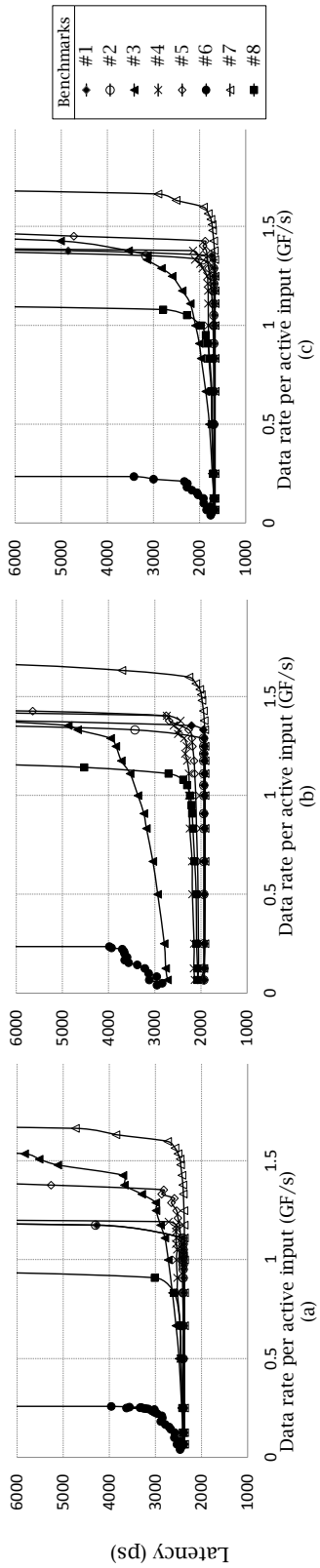


Figure 4.14: Network-level latency: (a) baseline network [96, 97]; (b) predictive network [82]; (c) new network

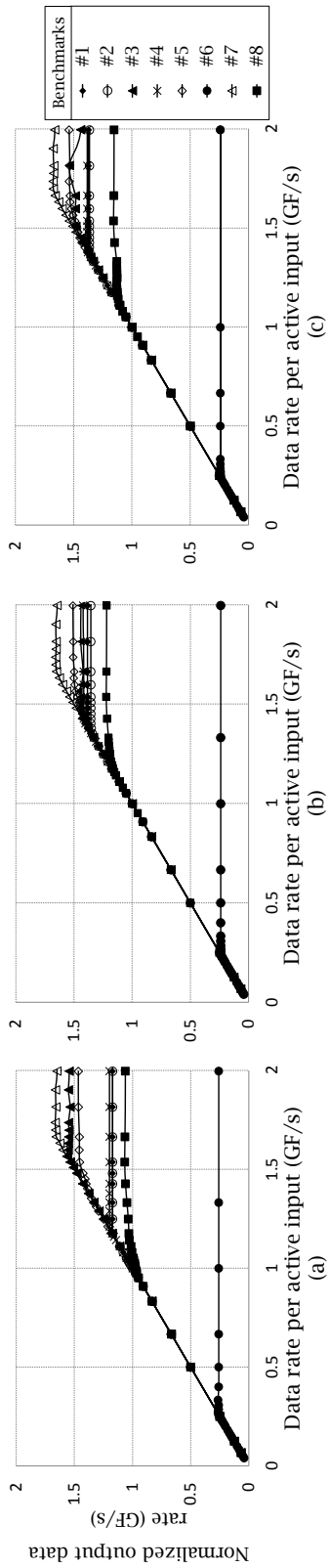


Figure 4.15: Network-level throughput: (a) baseline network [96, 97]; (b) predictive network [82]; (c) new network

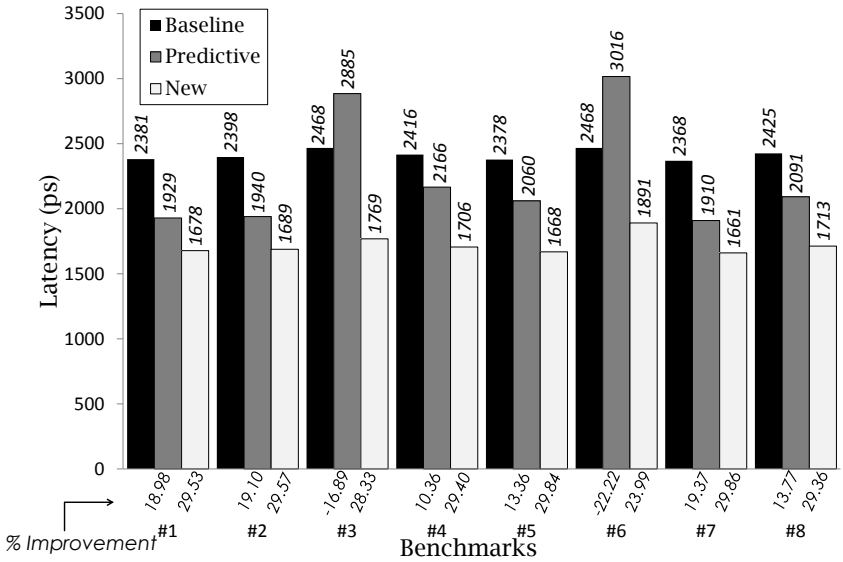


Figure 4.16: Latency comparison for 25% network load

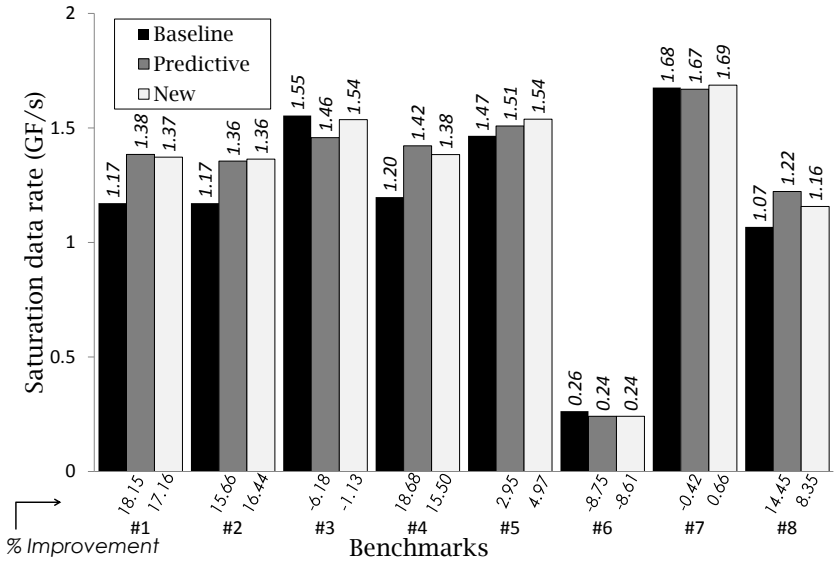


Figure 4.17: Saturation throughput comparison

from 13% to 21% for friendly and moderately adversarial benchmarks, with even higher improvements – near 38% – for the most adversarial benchmarks.

The two key contributions – merging latches on the forward path and channel pre-allocation – are now separately evaluated, to observe their impact. By merging latches, the network latency

is improved by roughly 250 ps (13%), when comparing the *predictive* and the *new* network in the three friendly benchmarks (1, 2 and 7). These benchmarks have persistent source-sink pair-wise streaming traffic with no contention. Hence, the involved fan-in routers in the *predictive* network stay in optimal configuration (i.e. held in the biased mode), and therefore this differential identifies the improvement due solely to the latch removal. For other benchmarks, the latency gains are always higher (up to roughly 38% in benchmarks 3 and 6), showing the additional contribution of the channel pre-allocation scheme.

The *stability* of the latency is also a key metric for memory access in shared-memory CMP's, capturing the degree of latency variation at fixed load under a variety of benchmarks. As highlighted in Fig. 4.16, while the *predictive* network has high latency variation, ranging from 1910 ps to 3016 ps, the *new* approach has *nearly uniform latency* over all benchmarks, which is another important benefit of the proposed approach.

Throughput. Fig. 4.15 explores network throughput, with results plotted as the output rate normalized to the number of active input sources vs. the mean offered data input rate [82]. Under fairly lightly-loaded traffic conditions, throughput tracks the input data rate. As the input rate increases, throughput results begin to level off to the *saturation throughput*. Fig. 4.17 provides a clean vision of the differences in saturation throughput values for the three networks. Compared to the *baseline,* the *new* network exhibits minor to moderate improvements – up to 17% – for six benchmarks, with up to 8% degradation on the two most adversarial ones. Compared to the *predictive* network, the *new* network exhibits nearly identical throughput – from 6% degradation to 6% improvement– across all benchmarks.

Monitoring Network Evaluation. Finally, network-level simulations show that the monitoring network can rapidly forward monitoring information and allows enough time for each fan-in node on the path to complete channel pre-allocation before actual flit arrives. In a zero-load network, the tightest margin between monitoring signal arrival and actual flit arrival is 590 ps, which occurs at leaf fan-in nodes.

Network-Level Simulation Results: Multi-Flit Design

Considering a multi-flit design, Figs. 4.18, 4.19 and 4.20 show the network-level latency and throughput results for the *baseline* and *new* networks using a fixed packet length of 8 flits. Only

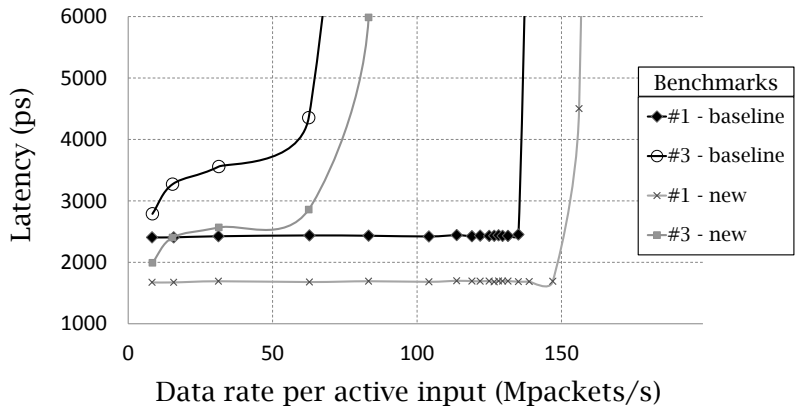


Figure 4.18: Latency for the networks with multi-flit capability

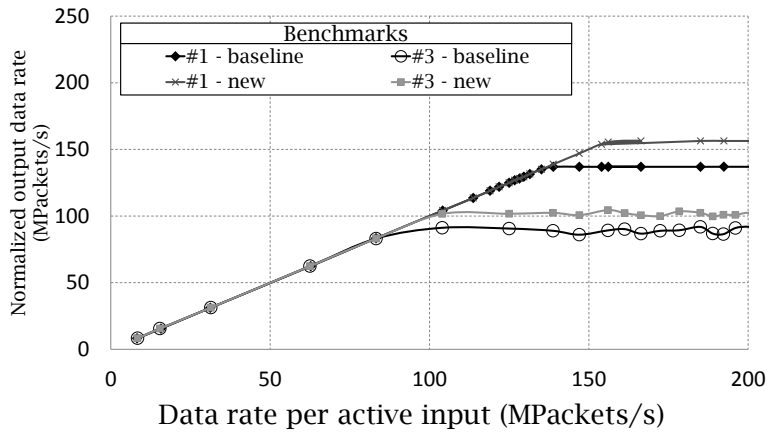


Figure 4.19: Throughput for the networks with multi-flit capability

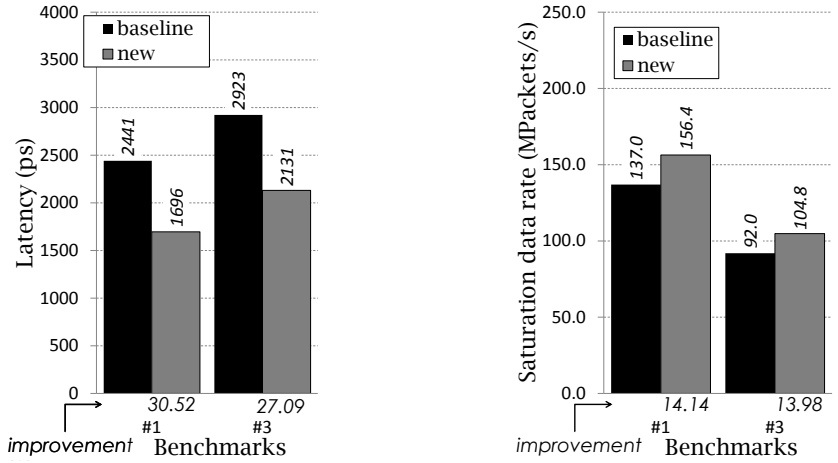


Figure 4.20: Performance comparison for multi-flit experiments

two representative benchmarks are picked for the experiment. Benchmark 1 (shuffle) is a friendly benchmark with no contension and benchmark 3 (all-to-all random), a widely-used benchmark that is moderate adversial. Significant improvements are observed in network latency (27% and 31%). Throughput is also considerably better (around 14%) for both tested benchmarks. These strong results prove the effectiveness of anticipation in the multi-flit case. Results are also obtained for a fixed packet length of 3 flits. Similar improvements in latency (28% and 31%) but smaller improvements in throughput (3% and 14%) are obtained.

4.8 Conclusions and Future Work

This chapter introduces a novel network protocol to address the system-latency bottleneck for high-performance asynchronous interconnection networks using a Mesh-of-Trees variant topology. The topology is efficient for connecting processing clusters to memory tiles in shared-memory chip multiprocessors. A lightweight shadow monitoring network is proposed to fast-forward information on arriving packets, in advance, allowing each node on its path to complete both arbitration and channel allocation before the data arrives.

Detailed experiments are performed to compare the proposed network with two of our earlier asynchronous designs [96, 97] at both router- and network-level. At router-level, the new asynchronous arbitration primitive only has small overhead when compared to the baseline, while is significantly simpler than the predictive counterpart (28% less in area). Also, the new arbitration node only has one mutex, while the predictive design requires five to solve complex mode change scenarios. At network-level, the new network shows stable improvements in end-to-end system latency over the baseline network, 24-30% improvements for all 8 benchmarks, while up to 38% latency improvements were obtained over the predictive network. In addition, the attached monitoring network is extremely simple and fast – only several gates are added for each node – providing enough slack for early arbitration and channel pre-allocation.

Future work will be focused on narrowing the window between channel reservation and the actual flit arrival, which can potentially increase the network utilization. Instead of triggering the pre-allocation right after the flit enters the network, it can be triggered at the middle of the network, but still allowing enough time for completing pre-allocation in time. Also, mixed-timing interfaces

will be created to connect the network with functional cores to form a GALS system for further experiments on real traffic benchmarks.

Chapter 5

A Low-Latency Asynchronous NoC for a 2D-Mesh Topology

5.1 Introduction

In the previous chapter, a novel monitoring technique is applied for a Mesh-of-Trees (MoT) variant topology. The lightweight monitoring network fast forwards early information of incoming traffic and allows routers on the path to prepare in advance. Although MoT topology and its variants are receiving increasing interest and attention, they are still used in only specialized applications. Mesh-based NoC structures are still the mainstream. Acceleration techniques for mesh-based topologies are more important and challenging: they have higher-radix switches with non-trivial routing, and with more complex resource sharing.

This chapter introduces and applies a monitoring network to a 2D-mesh topology, and presents our second asynchronous low-latency network-on-chip [105]. The new NoC is called AEoLiAN, ***A***synch-ronous ***E***arly-arbitration appr***o***ach for a ***Li***ghtweight ***A***ccelerated ***N***etwork. Similarly, a low-overhead monitoring network is introduced, for a 2D-mesh, which allows arbitration and channel allocation to be performed in advance of a packet through the network. However, compared to the optimized MoT network in Chapter 4, AEoLiAN has fundamentally new monitoring protocols and components.

The new monitoring protocol uses a fine-grain and synchronized router-by-router early arbitration, with monitoring and data advancing independently at different speeds. In contrast, the MoT

monitoring network informs all routers on a packet's path in rapid sequence. This coarse-grain approach does not work for 2D-mesh topology, because the 2D-mesh topology is much more complex and there is much more traffic interference in mesh-based structures: pre-allocation of routers too early for one data stream can block other data streams from using this node for a large amount of time, resulting in an unacceptable performance degradation and congestion. The more conservative synchronization protocol is selected to avoid such over provisioning, as well as to conquer deadlock issues.

The proposed 2D-mesh monitoring network is quite lightweight, compared to many recent early arbitration techniques used in synchronous NoC's. For example, SMART NoC [119], one of the most well-known optimized synchronous networks, requires substantial extra network resources, including additional VCs and link resources, to enable a complete multi-hop channel reservation for a packet in the preceding clock cycle. In contrast, our approach exploits fine-grain asynchronous operation, which allows rapid 'sub-cycle' pre-allocation decisions, on a hop-by-hop and node-by-node basis, resulting in lightweight dynamic resource allocation as a packet advances. Interestingly, given the decoupled and fine-grain operation of the new asynchronous NoC, only a narrow increase in channel width – 11 extra bits per channel – is needed to support the monitoring network, without the need to accommodate worst-case notification scenarios within each clock cycle as in synchronous approaches.

The AEoLiAN network was implemented and simulated on 6 diverse synthetic benchmarks in an 8×8 2D-mesh network topology. Uniform improvements in system latency are obtained over all the benchmarks and over a wide range of injection rates. Pre-allocation was shown to be nearly completed in advance, with only slight overlap, with the arrival of packet headers. Interestingly, the acceleration technique also provided significant throughput gains for several of the benchmarks, by reducing network congestion. A small number of additional experiments were also performed by perturbing packet size, network dimension and link length. The new NoC shows almost identical, or only slightly degraded, latency.

The remainder of the chapter is organized as follows. Section 5.2 introduces related work on various of early arbitration techniques used for synchronous NoC's. Section 5.3 presents the *baseline* network by Ghiribaldi *et al.* [80], which we build on and compare to. An overview of the approach is first provided in Section 5.4, before diving into the detailed network design. From Sec-

tion 5.5 to 5.7, implementation details of the network is presented. Section 5.8 presents an informal explanation for why the network has no deadlock. Section 5.9 provides major timing constraints of the network. Detailed experimental results for the proposed 2D-mesh NoC design is shown in Section 5.10. Finally, Section 5.11 concludes the chapter.

5.2 Related Work

A number of mesh-based synchronous NoC's use early information and aim for rapid pre-allocation. While several demonstrate effectiveness in latency improvements, most require major hardware resource allocation to support such acceleration on a per-clock cycle basis: multiple extra virtual channels (VCs), hybrid configuration networks requiring additional planes, and wide control or monitoring channels.

The SMART NoC [119] uses clockless repeated links to pre-allocate channels for multiple-hops simultaneously, allowing packets to potentially traverse multiple routers in one cycle. However, it requires wide monitoring channels with many extra VCs to obtain the full benefits of the approach. In particular, it includes an example (Fig. 7 [119]) with 24 distinct smart links emanating from each router, each with 2-4 address bits, and 12 VCs are proposed to obtain maximal benefits for its experimental results. Deja Vu [2] uses a hybrid network and requires an expensive dedicated monitoring plane, with the same channel width as the data plane, to support early arbitration; also, the environment must inject monitoring information into the network earlier than the data, unlike our proposed approach. The approach of [121] uses 'advanced bundles' to set switch allocation one cycle in advance; however, it builds on a slow 3-cycle baseline switch, and reported network-level performance had minimal latency improvement from 10-40% injected load and only modest improvements at other rates.

5.3 Background: Baseline NoC Design

The proposed work builds on a recent general-purpose asynchronous NoC router design with no latency acceleration, presented in [80]. This 5-ported switch uses a nearly entirely standard cell design, with low overhead and simple optimized control. It is a VC-less design, using destination addressing and X-Y routing, and uses a two-phase transition-signaling protocol, along with single-

rail bundled data encoding.

When compared to a highly optimized synchronous switch, *xpipesLite* [211], the baseline asynchronous switch demonstrated significant benefits: 71% reduction in switch area, 60-85% reduction in overall power, and a 44% average reduction in energy per flit, in the same technology. Unlike a number of prior asynchronous NoC's, which have been targeted to low or moderate throughput with high router node latency and area [27, 133], this switch also exhibited high throughput (903 ps average cycle time) and relatively low latency (1195 ps per router, header flit), in a low-power standard-Vth 40nm technology.

The router microarchitecture has two sets of components: Input Port Modules (IPMs) and Output Port Modules (OPMs). Five IPMs are connected through the crossbar to five OPMs, using a standard organization [46]. Each component is based on Mousetrap asynchronous pipelines [201, 204], which use a normally-transparent *capture-pass protocol* with single-level D-latch registers.

An IPM has a single input channel and four output channels, while an OPM has four input channels and one output channel. The IPM computes the current node's routing information, and propagates it to the designated OPM; input data is also broadcast 4-way to all OPMs. The role of the OPM is to *(i)* use the routing information to identify a valid request, *(ii)* resolve arbitration between competing requests, and *(iii)* allocate the designated output channel.

Details of an IPM and OPM module are introduced below, repectively. A more complete presentation can be found in [80].

5.3.1 Input Port Module

The structure of an IPM is shown in Fig. 5.1. Initially, the single-latch input register is normally transparent, as in Mousetrap pipelines, and all *Request Generators* are inactive.

The operation starts with a header flit arriving at the input channel. Following the transition-signaling bundled data protocol, $Data_{IN}$ becomes valid, and then a transition occurs on Req_{IN}. The header flit then directly passes through the input register, which is default-open, and immediately broadcast speculatively to all output channels. However, the header also activates *Packet Route Selector* to compute the real designated output port. The corresponding *Request Generator* is then activated, and asserts *PacketPathEnable* high on that port. The targeted OPM, which receives *PacketPathEnable* will identify the valid data, and sends acknowledgment back to the IPM. The remaining

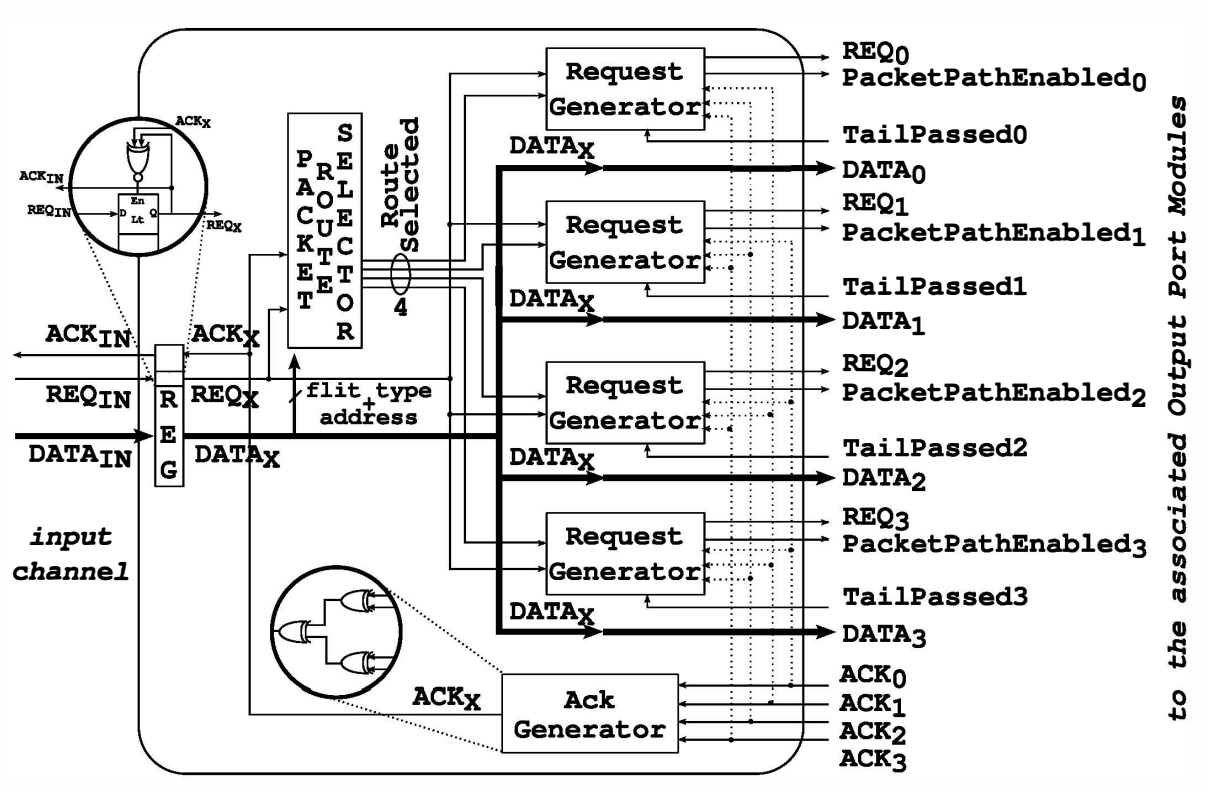


Figure 5.1: Baseline IPM architecture

OPMs simply ignore the data. After the header flit activates the corresponding *Request Generator*, *PacketPathEnable* signal remains high throughout the entire packet processing. Body and tail flits are directly transferred from IPM to the designated OPM at a very fast rate, as routing computation is no longer needed. Finally, after the tail flit is passed to the OPM and the acknowledge is received, the *Request Generator* is deactivated. The IPM is reset to the initial state and waits for the next packet processing.

5.3.2 Output Port Module

Fig. 5.2 shows the microarchitecture of the OPM. The module arbitrates between multiple incoming requests and merge them on the single output channel.

An asynchronous 4-way mutual exclusion element (mutex) performs arbitration, and a 4-way MUX selects the appropriate input data from the crossbar. The Tail Detector detects when a tail flit is sent out. The right data register uses a capture-pass protocol for flow control, and other components

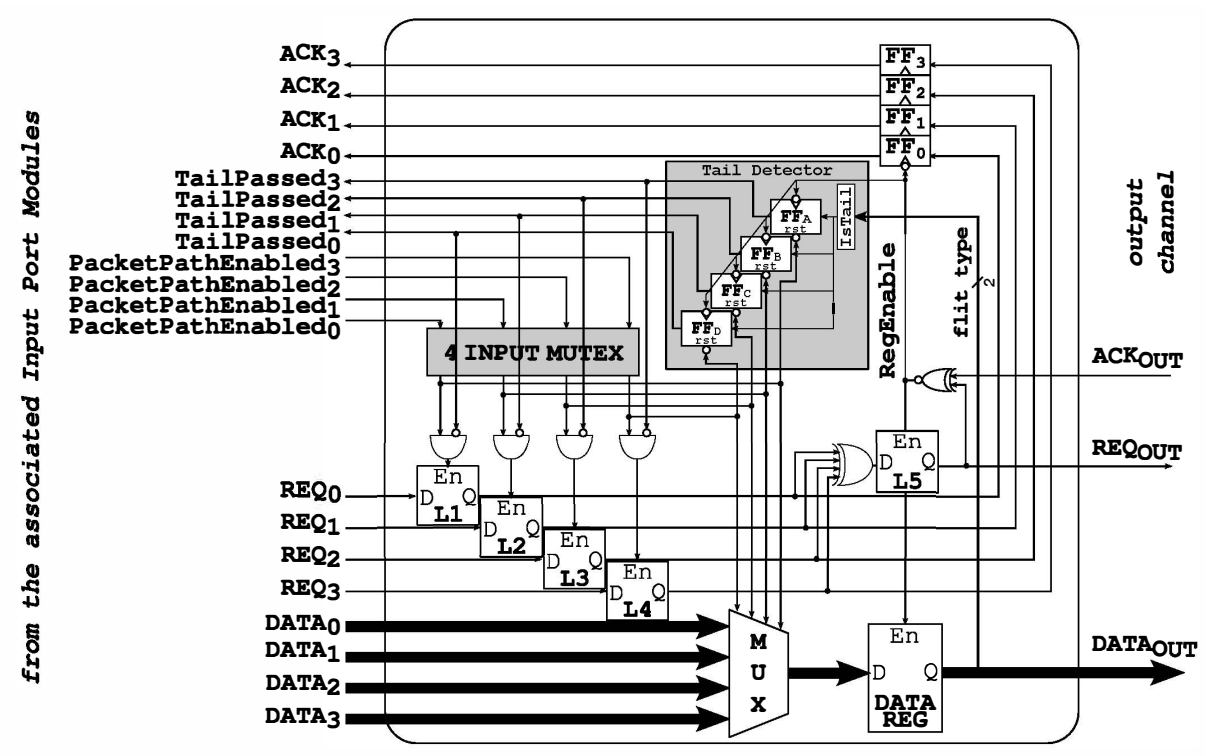


Figure 5.2: Baseline OPM architecture

manage handshaking with crossbar and output channel, as well as reset of the internal components.

Initially, the mutex is reset, with all inputs and outputs at zero. L1-L4 are normally opaque, blocking data requests until they win arbitration. L5 is a capture-pass latch, which is normally transparent. The operation begins when a header flit arrives from one of the IPMs, concurrently with the associated *PacketPathEnable* signal asserted high. The flit requests the mutex, wins the arbitration, and then the corresponding channel (L1-L4) will be open. The header flit is then directly put onto the output channel through the default-open output latch (L5). After the mutex is won, the channel will be held open throughout the entire packet processing. The remaining channels cannot win arbitration until the current packet is processed. Therefore, the body and tail flits of the current packet are directly sent out without being arbitrated again. Finally, after the tail flit sent out, a *TailPass* signal is sent back to the corresponding IPM, the IPM, in turn, deasserts *PacketPathEnable*, resets the mutex and closes the channel. The OPM is then reset back to the initial state.

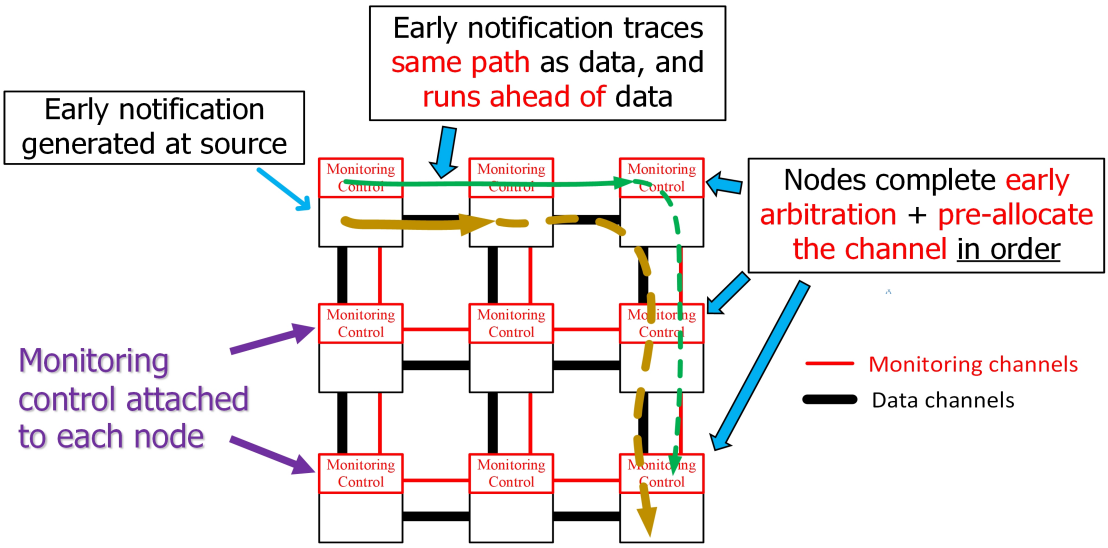


Figure 5.3: AEoLiAN overview: structure and operation

5.4 Overview of the Approach

The section introduces the basic strategy of AEoLiAN, and highlights the key features that are distinct from a previous baseline asynchronous NoC [80] with no early arbitration capability.

A structural and operational overview of AEoLiAN is shown in Fig. 5.3. The new NoC consists of two sub-networks: a lightweight and narrow monitoring network for fast forwarding the early notification of incoming data, and the standard datapath network for normal packet processing. Once a packet enters the network, its destination address in the header flit serves as the monitoring information, and it is rapidly advanced along the expected routing path through the monitoring network. Every router on the path performs early arbitration and channel pre-allocation based on the receipt of this information. Typically, the data only waits for a very short period in each router after it arrives, before the channel is completely allocated. In contrast, in the earlier NoC [80], arbitration is not allowed to start until the actual data arrives at each router, resulting in a long data wait time.

Lookahead routing is combined with the monitoring network, to further speed up its forward latency. A non-classical approach is proposed, which has higher parallelism to improve performance. There are two key distinctions. First, in a classical scheme [156], a packet must arrive at the current node before it initiates computation of lookahead routing for the successor node, while in the

proposed approach the computation is initiated earlier – *just after the advance monitoring arrives.* Second, the classical approach begins lookahead computation *after* switch allocation in the current node, while the proposed approach begins the computation *in parallel* with early arbitration. As a result, the above new protocol enhancements provide significantly accelerated performance.

In case of any contention, early arbitration is performed on a router-by-router basis. That is, if the monitoring loses the early arbitration at a certain router node, it is not allowed to further propagate, until it finally wins at the current router. The corresponding data packet also has to wait until the monitoring wins the arbitration and allocates the channel. In both cases, the monitoring proceeds as an arbitration 'wave,' by winning early arbitration at each router on the expected path. In other words, the monitoring network decides the processing order for input data streams in advance, and the actual data simply follows and replays the pre-determined order. This fine-grain 'router-by-router' early arbitration mechanism enables a simple design, which eliminates the need for over-provisioning the network as is required in some recent synchronous approaches, which use extra VCs to support multi-hop single-cycle transmissions, or expensive hybrid multi-plane networks. Instead, the fine granularity of asynchronous operation (a router node can forward monitoring in little more than 200 ps), allows 'sub-cycle' pre-allocation decisions on a hop-by-hop basis, using limited channel resources, based on ambient dynamic traffic.

Different handshaking protocols are used in the monitoring network vs. datapath. Data channels retain a 2-phase NRZ handshaking protocol with single-rail bundled data encoding, as in [80], to allow a high processing rate. On the other hand, the monitoring channels only operate on a slower per-packet basis, and use a 4-phase RZ protocol, with the packet destination encoded using single-rail bundled data. The use of the 4-phase protocol, with a clean 'return-to-zero,' simplifies the monitoring control, without sacrificing performance of the entire NoC in most of cases.

5.5 Proposed Router Node Design

The new asynchronous 5-port router builds on the baseline design with no early arbitration capability [80], including the integration of monitoring capability and an entirely new monitoring/data-path control, along with new protocols for synchronization of the monitoring and datapath. The basic structure and operation of the Input Port Module (*IPM*) and Output Port Module (*OPM*) are first

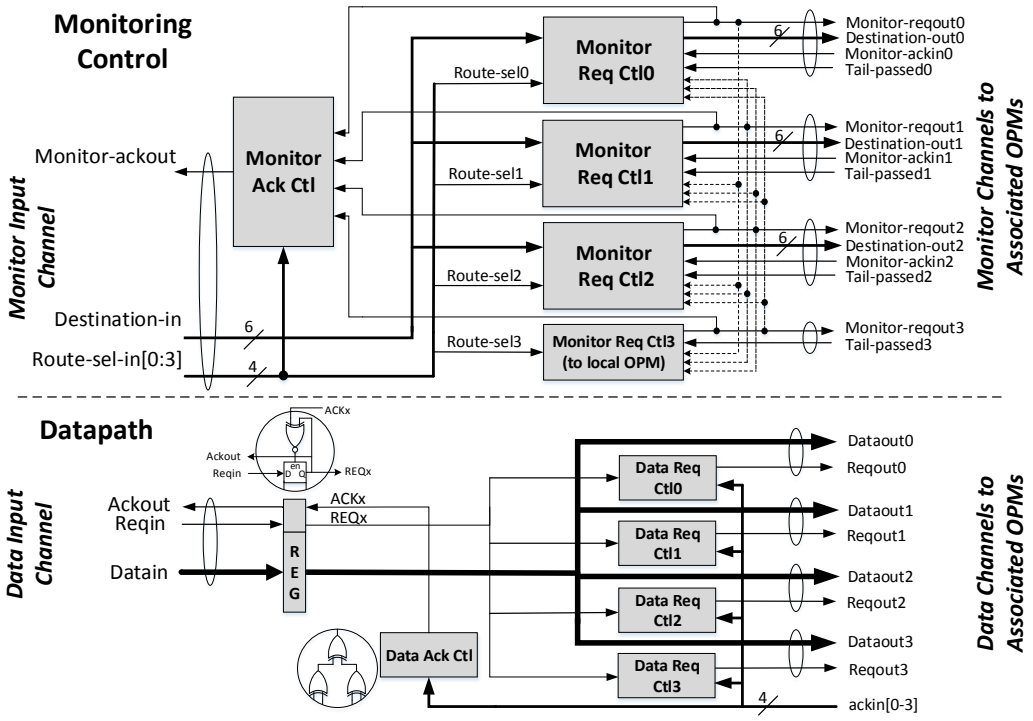


Figure 5.4: Proposed Input Port Module (IPM) architecture

introduced, in turn. Each contains a lightweight monitoring control and the datapath. Then the monitoring network is presented separately, including a more integrated view of its control components residing in the *IPM* and *OPM*. The section closes with timing analysis.

5.5.1 Input Port Module Architecture

An *Input Port Module*, shown in Fig. 5.4, routes the packet to the selected *Output Port Module*. It has a single input channel from the upstream neighbor, and four output channels that connect to 4 different *Output Port Modules* through a crossbar. Each channel is now augmented by a narrow monitoring sub-channel. Monitoring information typically arrives earlier than actual data, and initiates both early arbitration and channel pre-allocation.

5.5.1.1 Structure

An *Input Port Module* is divided into two independent blocks – the monitoring control on top, and the datapath at bottom.

The datapath consists of an input register, four Data Request Control units (one for each output port), and a Data Ack acknowledgment generator for the input channel. The datapath allows a packet to enter the *IPM* as soon as it arrives at the input channel, through the capture-pass input register. The packet is then quickly broadcast to all *OPMs*, with corresponding requests sent out by the four Data Request Controls. However, only the selected OPM will accept the data. All the remaining requests will be cancelled.

The monitoring control is part of the entire monitoring network. It consists of four *Monitoring Req Controls* on the right, each communicating with a different OPM, and a *Monitoring Ack Control* on the left, for handshaking with the predecessor router. Both control units use 4-phase handshaking protocols, and are initialized to the all-0 (i.e. reset) state. In more detail, the *Monitoring Req Control* initiates early arbitration as soon as the advance notification arrives, which includes the destination address and the lookahead information. It then asserts the appropriate *monitoring req* to one of the OPMs, and also forwards the packet destination along with the request. Only one of the control units is activated at a time. The unit serves two purposes: *(i)* a FIFO stage and flow control for the monitoring network, to maintain the monitoring information until the successor router safely stores it; and *(ii)* synchronization with datapath, to keep the monitoring request asserted throughout the processing of the entire packet. The *Monitoring Ack Control* simply merges the four *monitoring reqouts* and generates the acknowledgment to the predecessor, allowing a new monitoring token to arrive.

5.5.1.2 Operation

The basic operation of the *IPM* is illustrated by a friendly scenario with a single packet and no contention. Without loss of generality, this packet is assumed to be routed to *OPM0*. Initially, all handshaking signals are zero for both monitoring and data channels. In particular, a *1-hot* encoding is used for the lookahead routing address, *Route-sel-in*; this DI code serves as a bundling signal for the single-rail *Destination-in* address.

First, monitoring arrives on the input channel before the header flit, after a new packet is injected into the network, with an asserted *Route-sel-in[0]* signal. The *Monitor Req Ctl0* immediately asserts *Monitor-reqout0* high, and passes it, along with the destination address, to *OPM0*. Then an acknowledgment is generated to the left. In friendly case, the entire 4-phase monitoring handshak-

ing completes quickly on the left channel. In parallel, on the right channel, *OPM0* acknowledges on *Monitor-ackin0*, indicating early arbitration and channel pre-allocation are completed, and monitoring information has also been received by the downstream successor. Note that *Monitor-reqout0* still remains high.

Next, the header flit of the actual packet arrives. The header is broadcast to all four OPMs, similar to the approach in [80]. Since the designated channel is already pre-allocated in *OPM0*, the header is immediately sent to the next router. Each body flit is processed the same as the header. Eventually the tail flit arrives. After it is sent out, the *OPM0* asserts *Tail-Passed0* high, indicating the entire packet has been processed. At this point, *Monitor-reqout0* finally goes low, allowing the release of the arbiter and output channel. Finally, *Monitor-ackin0* and *Tail-Passed0* are de-asserted low almost simultaneously, thereby completing the entire operation.

In a contentious case, with two back-to-back incoming packets, the new monitoring token can arrive before *Monitor-reqout* is de-asserted for the first packet. However, the second request is not generated until the previous *Monitor-reqout* is de-asserted, even if it would request a different *OPM*, to avoid malfunction in the router.

5.5.2 Output Port Module Architecture

An *Output Port Module*, shown in Fig. 5.5, arbitrates between multiple incoming data streams that try to access the associated output channel. By symmetry, it has four input channels and one output channel, with each channel containing a monitoring sub-channel. Upon receiving early monitoring requests from the *IPM*, the *OPM* starts arbitration and channel allocation, well in advance.

5.5.2.1 Structure

The *OPM* also consists of a monitoring control and datapath, which synchronize with each other. The structure of the datapath is similar to [80], as described in the Background section.

The new monitoring control contains three important components:

(i) The *Packet Route Pre-Computation Unit* computes the lookahead routing information for the successor router node. The unit supports parallel computation for all the incoming monitoring channels, and select one of them based on the results of early arbitration. The routing information calculation is done in parallel with arbitration, and therefore its latency cost is entirely removed

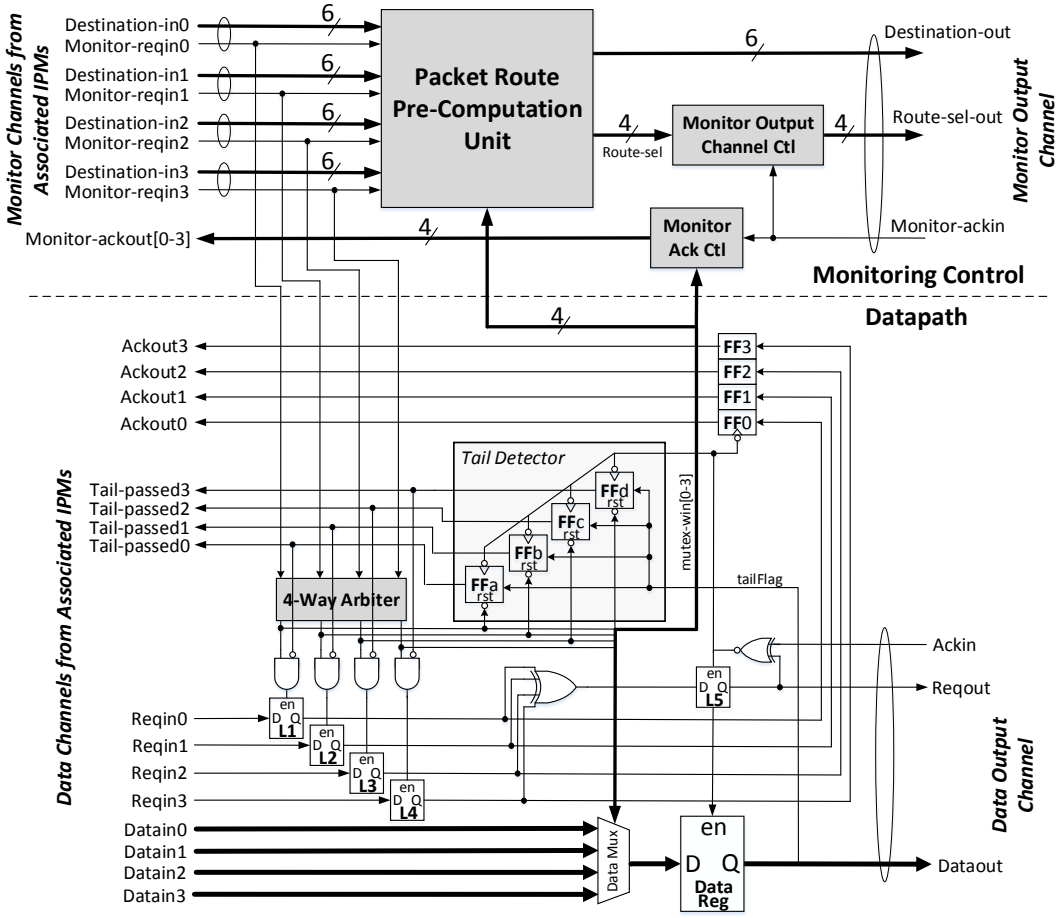


Figure 5.5: Proposed Output Port Module (OPM) architecture

from the critical path. In the proposed design, a simple X-Y routing is used. However, other types of routing algorithms can also be supported.

(ii) The *Monitor Output Control* quickly forwards the lookahead routing information from the Packet Route Pre-Computation Unit to the monitoring output channel. It is also a decoupling unit, enforcing monitoring flow control, which delays sending out a monitoring token until the monitoring output channel completes its previous 4-phase communication with the successor router node, Effectively, it allows the router to prepare the next monitoring input, in case the output channel incurs a long communication delay.

(iii) The *Monitor Ack Control* is simply a de-mux to route a handshaking acknowledgment from the downstream router back to the correct *IPM*.

5.5.2.2 Operation

A similar single-packet friendly case is illustrated for basic operation of the new OPM. The packet is assumed to arrive from *IPM0*. First, monitoring arrives from *IPM0*, with an asserted bundling request *Monitor-reqin0*, which is kept high throughout the processing of the packet. Two operations are initiated concurrently: *early arbitration* and *lookahead routing computation*. The mutex resolution and the completion of pre-computation are synchronized: both must be completed before the monitoring information is sent out through the *Monitor Output Channel Control*. At this point, latch *L1* is opened, i.e. the output channel is pre-allocated. In friendly case, the monitoring output channel on the right responds quickly, and completes the entire 4-phase handshaking without any other synchronization. When the actual packet arrives, the header flit is sent out as soon as *L1* opens, through the capture-pass output data register. The channel remains allocated; each body flit is processed exactly the same as the header flit. Finally, after the tail flit is placed onto the output channel. *Tail-Passed0* is asserted high. In turn, *IPM0* de-asserts *Monitor-Reqin0* low, and releases the mutex. The entire operation completes with de-assertion of *Tail-Passed0* and *Monitor-ackout0*.

In case of contention, two monitoring requests from different *IPMs* arrive almost simultaneously. Lookahead routing information will be computed in parallel for both. Based on the arbiter result, the winner processes the packet following the same procedure as in *single-packet* case. The losing channel, though it completes lookahead computation, does not send out this information. Its data packet must also wait, as its channel is not allocated. Eventually, the first packet is processed and releases the mutex. The losing channel then immediately continues its operation by winning the arbitration, allocating the channel, and forwarding lookahead information.

5.6 Monitoring Network: System- and Switch-Level Protocols and Design

In the previous Section 5.5, the structure and operation of monitoring controls were introduced individually for each of the *IPM* and *OPM*. Here, the monitoring network is re-examined in a more integrated way, from several different views: *(i)* at system level, the flow control of monitoring tokens – how these tokens are generated, stalled and forwarded through the entire path; *(ii)* at switch level, the interaction between the *IPM* and *OPM* monitoring controls; *(iii)* detailed implementation

of supporting monitoring components.

System-Level Operation

When a packet enters the network, a corresponding monitoring token is generated at the source node, and injected into the monitoring network. The token contains the destination address of the packet, and the lookahead routing information computed by each router on the path. Considering a single source-destination path, the monitoring network is effectively a linear FIFO, with extra synchronization with datapath. A monitoring token has to be kept until *(i)* the next FIFO stage (i.e. the successor router) safely stores it, and *(ii)* the corresponding packet is entirely processed by the current router. The monitoring FIFO can only store one token per router, only half capacity of the datapath. However, because each packet only requires a single monitoring token, the capacity of the monitoring network is sufficient in the case of multi-flit packets. By having fewer number of pipeline stages, the monitoring network is optimized for area and latency, without sacrificing performance.

Switch-Level Operation

The monitoring operation inside a single router is now considered. In particular, it involves the interaction between *IPMs* and *OPMs* across the crossbar. The monitoring channel connecting an *IPM* and an *OPM* effectively uses a modified 4-phase handshaking protocol. After the *IPM* sends out a request, the *OPM* has to acknowledge three separate events: *(i)* early arbitration is resolved and channel pre-allocation is done; *(ii)* the successor router has received and safely stored the monitoring token; and *(iii)* the corresponding packet has been processed and the monitoring token becomes stale. Event *(i)* and *(ii)* are acknowledged using a merged *monitor-ack* signal, while event *(iii)* is indicated by *Tail-passed*. After both acknowledgments are received, which can be in either order, the *monitor-reqout* is reset, followed by *monitor-ack* and *Tail-passed* de-asserted almost simultaneously. Then, a new monitoring token is allowed to be sent out, if one is pending.

Individual Control Units

Each sub-block within *IPMs* and *OPMs* is now presented to provide the final details for the monitoring implementations. The three key monitoring control units are: the *Monitor Req Control*, and

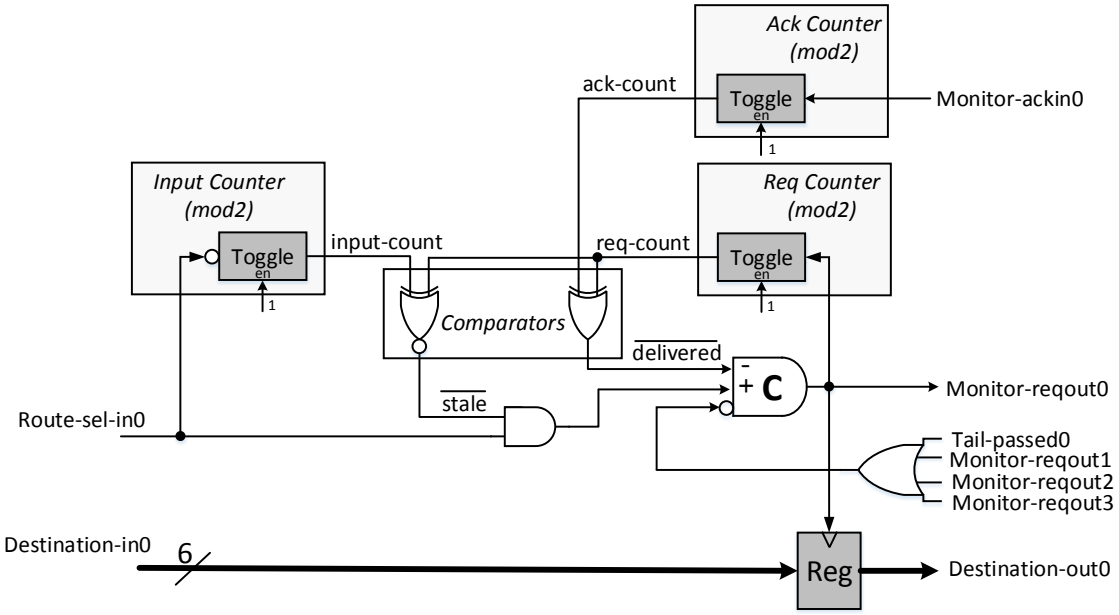


Figure 5.6: IPM details: Monitor Req Control

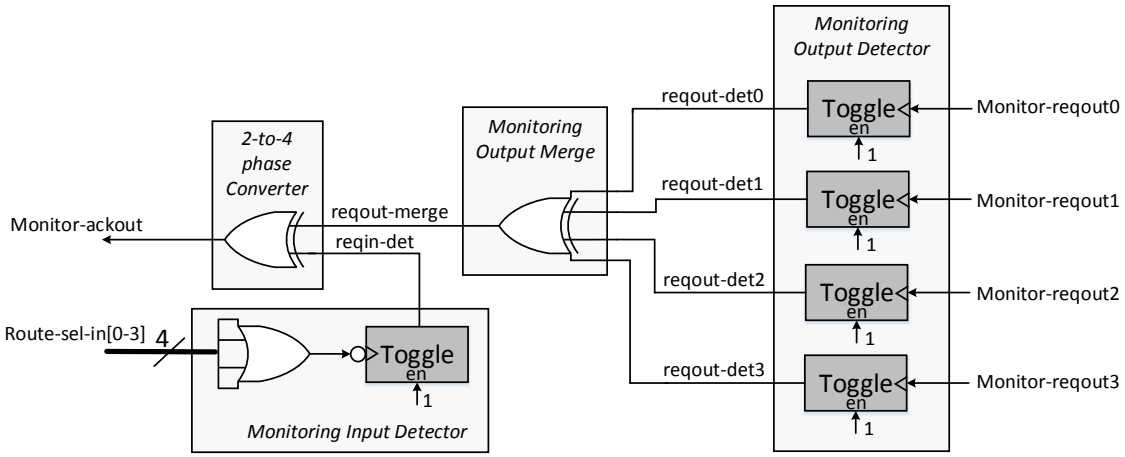


Figure 5.7: IPM details: Monitor Ack Control

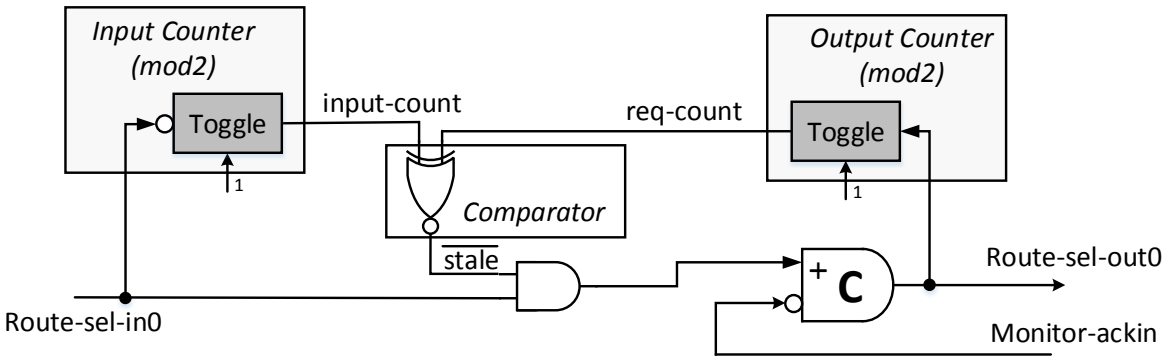


Figure 5.8: OPM details: Monitor Output Channel Control

the *Monitor Ack Control* in *IPM*; the *Monitor Output Channel Control* in *OPM*.

The *Monitor Req Control*, shown in Fig. 5.6,[1] implements the handshaking protocol introduced in the node-level operation above. The *one-hot Route-sel-in* is asserted high to indicate the arrival of a new monitoring token. If the unit is quiescent, *Monitor-reqout* is immediately sent to the *OPM*. After the *OPM* acks on both *Monitor-ackin* and *Tail-passed*, *Monitor-reqout* is reset. Then the *OPM* de-asserts both *Monitor-ackin* and *Tail-passed*. At this point, *Route-sel-in* is re-sampled to potentially start another operation. However, the unit can discriminate a stale *Route-sel* from a new token, by recording the spacing between two valid *Route-sel* tokens.

The operation of the *Monitor Ack Control* is trivial, as shown in Fig. 5.7. First, a monitoring token arrives by asserting *Route-sel-in*. When one of the monitoring reqouts goes high, indicating the monitoring token is safely stored, the control generates the monitoring ack to the predecessor. Then, the predecessor de-asserts *Route-sel-in*. Without any other synchronization, *Monitor-ackout* is reset to complete the entire 4-phase RZ handshaking protocol. The operation repeats for the next monitoring token.

The *Monitor Output Channel Control* in the *OPM* has four identical sub-blocks, one for each *Route-sel* wire. Each block, as shown in Fig. 5.8, is entirely a sub-design of *Monitor Req Control*. Similarly, when the unit is quiescent, *Route-sel* quickly propagates from input to output. However, the next *Route-Sel* cannot be generated until *(i)* a new token arrives, and *(ii)* the right monitoring channel completes the entire 4-phase handshaking protocol. Effectively the control can also be

[1] A generalized C-element asserts its output high when both the neutral input and the '+' input go high; it de-asserts its output low when both the neutral input and the '-' input go low.

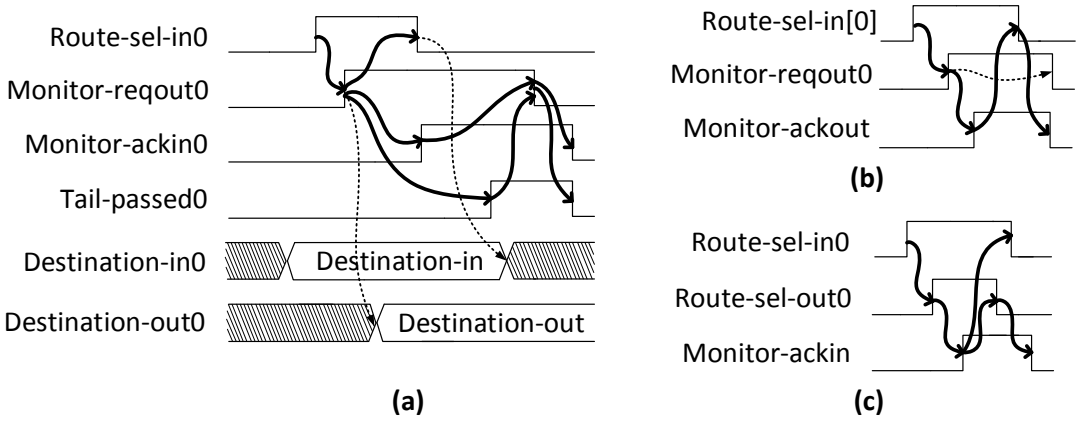


Figure 5.9: Timing digrams for single-packet processing:
(a) Monitor Req Ctl; (b) Monitor Ack Ctl; (c) Monitor Output Channel Ctl

treated as a decoupling unit, allowing the router to prepare a new monitoring token while the output channel incurs a long reset phase in the 4-phase communication.

The corresponding timing diagram for a friendly single-packet processing scenario is shown in Fig. 5.9, for each control unit presented above.

5.7 Local Input and Output Port Modules

While the previous subsections focused on a typical *IPM* and *OPM*, in a 5-port router, only four of the five *IPMs/OPMs* are typical, and the remaining one *IPM/OPM* needs to communicate with the network interface – namely the local *IPM/OPM*. The local *IPM* generates the monitoring information for each packet entering the network, and serves as the starting point of its entire monitoring path. The local *OPM* terminates the monitoring from further propagation, as there is no successor router. The designs of the local *IPM* and *OPM* are straightforward extensions of the typical ones, which provide the capability of initiating and terminating the monitoring signals.

5.8 Deadlock Analysis

The proposed network has no deadlock. In fact, it is deadlock non-increasing, compared to the baseline network. Because the baseline uses a dimensional-order routing, which conforms the turn

model and guarantees no deadlock, the proposed network, using the same routing algorithm, also has no deadlock. A short explanation is given below. A formal proof is beyond the scope of the thesis.

A monitoring token is always accompanied by an actual packet. The corresponding monitoring token of a packet is only generated after that packet is injected into the network. The monitoring serves as an *early header*, which operates just as the header flit in the original network. It requests and wins the arbitration and opens the channel, router by router in serial order. On the other hand, the proceeding of the monitoring is restricted. A monitoring token cannot be forwarded to the next router until the successor router has sent out the tail flit of the previous packet, and is fully ready to accept new data.

Therefore, if the monitoring token wins the arbitration and allocates the channel at a certain router, even though the monitoring signal is far ahead of the actual packet by any number of hops, it clears the entire path for the actual packet to this point. There is no impedance for the packet to proceed from the source to the current router. That is, within a finate amount of time after the monitoring, the actual packet is guaranteed to arrive. This property guarantees the deadlock freedom of the new network.

5.9 Timing Analysis

The new added monitoring network is carefully designed to contain only a few one-sided timing constraints, shown below, where both racing paths involve a global channel or intra-node crossbar communication. The remaining constraints involve a fast local path vs. a slower global channel or crossbar communication, and can be simply satisfied.

The rest of the timing constraints in the datapath must still be satisfied (e.g. the bundling timing constraints in the datapath, and several timing constraints related to the Mousetrap-style latch controls); several of these constraints are discussed in [80], for parts of the datapath that are invariant between the baseline and new designs. Note that single-rail bundled data asynchronous NoC's have been demonstrated as robust and correctly implemented, including at post-layout levels, in several recent designs [80, 97].

Input Bundling. A standard bundling timing constraint has to be satisfied for the monitoring input channel of each *IPM*. Typically, *Packet Destination* has to be stabilized before the arrival of

Route-sel, in order for the *Monitor Req Ctl* to safely store the destination information. However, the network can even tolerate a reversed skew, where *Route-sel* arrives slightly earlier than *Packet Destination*. This reversed bundling margin can be up to the delay of an AND2 and generalized C-element in series.

Output Bundling. A similar bundling timing constraint has to be satisfied on the monitoring output channel of each *OPM*. However, this constraint is easily satisfied, since the bundling request *Route-sel* has a longer path in the *OPM* through the *Monitor Channel Output Ctl*, while the destination information is directly sent out by the *Pre-Computation Unit*. Typically, *Packet Destination* is sent out earlier than *Route-sel* by a margin of an AND2 gate plus an aC delay.

Data Request Reset. Another timing constraint must be enforced in the crossbar between *IPM* and *OPM*. The scenario occurs after a complete packet has been processed, i.e. the tail flit is sent out by an *OPM*. The source *IPM* of the tail flit must quickly cancel the three incorrect data request toggles sent to the remaining *OPMs*, before one of these *OPMs* opens its channel due to a new channel pre-allocation. This constraint is simply satisfied, since a channel pre-allocation involves an arbiter operation, which is typically slow. The margin of the timing constraint is the 4-way arbiter latency plus an extra logic gate delay. As always with one-sided constraints, additional delays can be padded to increase this margin.

5.10 Experimental Results

Detailed evaluations are now presented for the new asynchronous early arbitration network, AEoLiAN, using technology-mapped pre-layout implementations. The new network is first simulated in an 8×8 2D-mesh on 6 synthetic benchmarks and compared to our previous *baseline* low-overhead asynchronous network without early arbitration capability [80], including area, latency and throughput. Then, a small number of additional experiments are performed to show the effectiveness of our approach under different network conditions, by changing packet size, network dimension and link length.

5.10.1 Experimental Setup

The *baseline* and *new* networks are implemented with a 64-bit wide datapath. A single router is first mapped using the FreePDK 45nm Namgate standard cells in Cadence Virtuoso environment. An

Table 5.1: Area comparison for pre-layout router nodes (μm^2)

Baseline	*New*	*Synchronous* (projected)
5081.4	8272.9	17369.5

accurate gate model is then extracted using Spectre simulation (at 1.0V, 27°C) to determine distinct latencies for rise and fall transitions, for every pair of I/O paths of each gate.

We developed an asynchronous NoC simulator for network evaluations, based on a Programming Language Interface (PLI) framework. Networks are modeled in structural Verilog with standard cells, using the gate models obtained above, while the test environment is written in C. Flits within a packet are placed into the source input queue simultaneously, and the mean time between packet headers follows an exponential distribution. We follow the standard procedure to ensure a long enough warm-up and measurement time [46].

Experiments are largely performed using an 8×8 2D-mesh topology with a fixed packet size of 5 flits. The link length and link delay between two neighbor routers are assumed to be 1mm and 100 ps, respectively, following the floorplan of a 45nm Freescale PowerPC e200z7 core as used in [119]. These parameters are changed later to run a small number of additional experiments.

Benchmarks

Experiments are conducted using six synthetic benchmarks, to cover a wide range of different network traffic conditions. Three permutation traffic patterns use different one-to-one mappings for source-destination pairs [46]: **1)** *Bit complement*, **2)** *Bit rotation*, and **3)** *Transpose*. Three additional benchmarks provide non-deterministic traffic to multiple destinations: **4)** *Uniform random,* in which each source is equally likely to send a packet to every destination; **5)** *Hotspot10,* in which each of the middle four routers receives approximately 10% of the packets from all the other nodes, while the remaining traffic is equally distributed to other destinations; and **6)** *Multiple-to-all,* in which the middle four routers can be the only sources and send packets to all the other routers with equal probability.

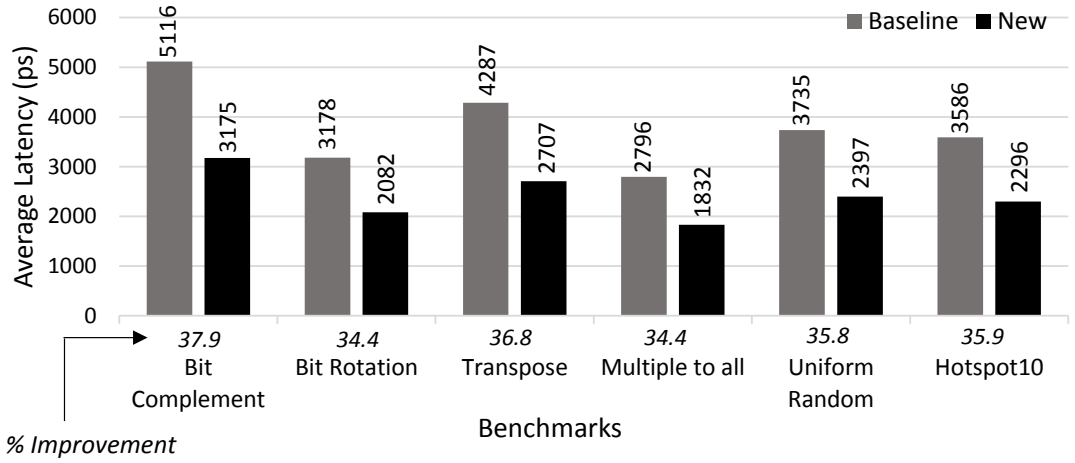


Figure 5.10: Latency comparison for 25% network load

5.10.2 Evaluation

Area Comparison

Table 5.1 compares router area for three designs – *new* and *baseline* asynchronous routers [80], as well as a state-of-the-art synchronous switch, xpipeLite [211]. For asynchronous routers, we divide the raw cell areas by a packing factor of 0.8 to obtain the final layout areas. The result of the synchronous switch is simply scaled, according to [80]. The *new* router has 63% more area than the *baseline*, but the absolute overhead is modest, since both routers only have basic functionality without including any input buffers or VCs; such overhead would be amortized with these extra features. On the other hand, the *new* design only costs less than half area than the synchronous counterpart, demonstrating its lightweight feature.

In terms of monitoring channel width, the overhead is only 11 extra bits, to support a 2D-mesh topology with 8×8 dimension. Interestingly, the width increases only logarithmically with network dimension. It also increases only linearly with router radix. Hence, the monitoring channel appears quite viable, in terms of overhead, for many applications.

Latency

Fig. 5.10 presents the network latency comparison at a particular input traffic rate, for the *new* network vs. the *baseline* [80]. We chose 25% because it is high enough to show the characteristics

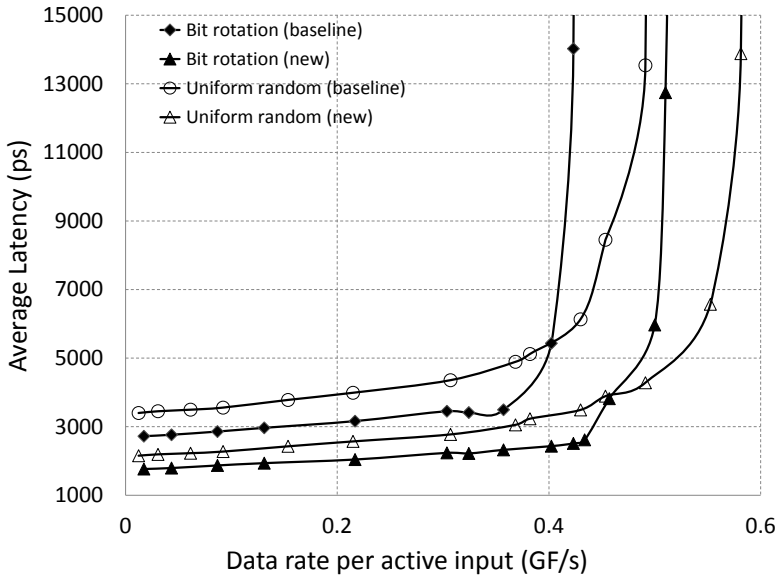


Figure 5.11: Latency for 'Bit rotation' and 'Uniform random'

of different benchmarks while retaining a network that is largely uncongested. Latency is calculated for the header flit. Overall, considerable latency improvements are obtained across all benchmarks, ranging from 34.4% to 37.9%. Even though the absolute latency values are quite different for the benchmarks, the improvements are quite stable, showing the effectiveness of the *new* network across a range of network conditions and traffic patterns.

Two benchmarks, *Bit rotation* and *Uniform random*, shown in Fig. 5.11, are picked for more detailed simulation, to highlight network latency under a complete spectrum of offered data input rates. The two benchmarks are selected as they have quite different characteristics. *Bit rotation* has a deterministic traffic flow between each distinct source-destination pair, while *Uniform random* is based on evenly-distributed communication throughout the network. For each benchmark, before saturation, the gap between two corresponding curves, *baseline* vs. the *new*, remains almost unchanged. The trend strongly indicates that the *new* network provides very good improvement for all data input rates, as long as the network is not saturated.

Throughput

Interestingly, while our main focus is on latency acceleration, noticeable throughput improvements are also obtained by the *new* network for most of the benchmarks. Fig. 5.12 provides a clean vision

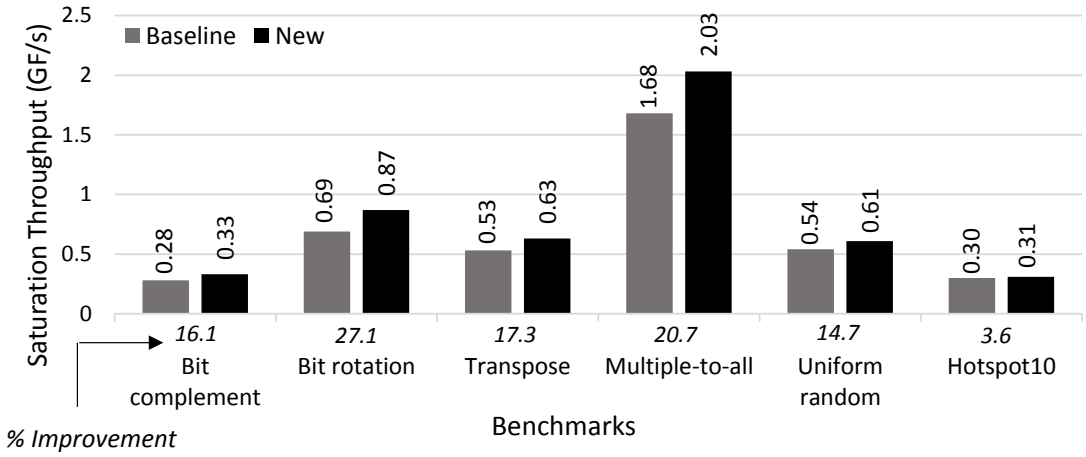


Figure 5.12: Saturation throughput comparison

of the differences in *saturation throughput* values for the two networks. This metric identifies the input rate beyond which no further improvement in network throughput can be obtained. Overall, in five of the six benchmarks, the *new* network exhibits consistent moderate to high improvements, ranging from 14.7 to 27.1%. For the remaining benchmark, *Hotspot10*, the *new* network shows only negligible gain.

The improvement in saturation throughput is closely correlated to the level of contention in network traffic. The more contention, the less improvement. These differences are due to the distinct operating scenarios of the network, as contention varies. In non-contentious traffic with high injection rates, the monitoring network in the new design shrinks the gap between two back-to-back packets, having the same *IPM-to-OPM* path. The lookahead information contained in the monitoring allows the second packet to be prepared in advance, hence it can go through faster. In contentious traffic, multiple packets from distinct input channels compete for the same output channel. In this case, even with the baseline network, due to high traffic, route computation for a waiting packet is entirely completed before the winning packet has released the channel, hence there is no significant acceleration due to early arbitration. In this case, baseline and new networks have similar operation. *Hotspot-10* is an extreme example.

Detailed throughput results for two benchmarks are shown in Fig. 5.13. The output rate is normalized to the number of active input sources, and the mean offered data rate is varied from zero-load to saturation.

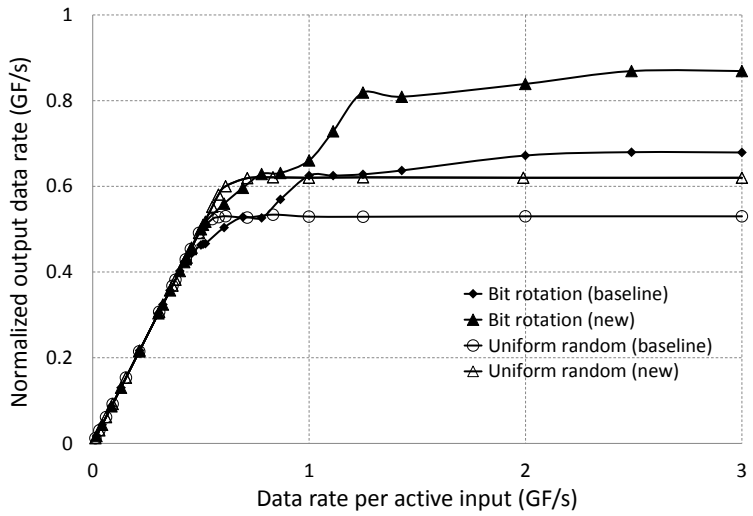


Figure 5.13: Throughput for 'Bit rotation' and 'Uniform random'

Comparison to Ideal Networks

Table 5.2 compares the *baseline* and *new* networks with two ideal networks, in terms of zero-load latency for the longest path in 8×8 2D-mesh network, through 15 routers and 14 hops. The goal is to estimate how close to ideal operation the actual network provides. The first ideal network assumes a fully-anticipatory monitoring network, in which channel pre-allocation can be entirely completed before actual data arrival. The second ideal network assumes a dedicated network where each source-sink pair is connected by an ideal point-to-point link, with no intermediate routers. The *new* network clearly shows a predictable large improvement over the *baseline*. However surprisingly, the *new* design also closely approaches the first ideal network within a small margin: only 488 ps. This indicates the channel is almost allocated before actual data arrives, with only a sub-ideal 35 ps late arrival time. For the comparison with the second ideal point, the dedicated network operates at 1400 ps while the *new* network is significantly slower. This outcome is understandable, as the second network assumes extremely unrealistic network resources with no routers on a 14-hop path. Overall, the absolute value of the *new* zero-load latency is extremely low: only 3476 ps ignoring link delays, and hence a 232 ps latency for each router.

Additional Experiments

Finally, a small number of additional experiments are performed, to gain insight into the new ap-

Table 5.2: Zero-load latencies for the longest path in 8×8 mesh (ps)

Baseline	*New*	*Ideal network #1 (fully pre-allocated)*	*Ideal network #2 (dedicated network)*
8087	4876	4388	1400

proach under varied parameters. Results are obtained using only the *Uniform random* benchmark. The experiments include: *(i) varying link length:* 2mm (200 ps) and 4mm (400 ps); *(ii) varying packet size:* single-flit and 8-flit packets; and *(iii) varying network dimension:* using a reduced 4×4 2D-mesh network. The parameters are changed one at a time, while keeping the remaining parameters as original.

Latency. In experiment *(i)*, when the link length uniformly increases from 1mm to 2mm and 4mm, latency improvements only decrease moderately (36%→28%→22%), as expected, as the optimizable portion of the path (i.e. the nodes) contributes a smaller proportion of the delay. In experiment *(ii)*, when varying packet size, the improvement is 36% and 34% for single-flit and 8-flit packets, respectively (compared to 36% for the earlier 5-flit packets), showing almost no impact. Finally, a latency improvement of 31% is observed in experiment *(iii)* of 4×4 2D-mesh, which is only slightly degraded over the 8×8 2D-mesh.

Saturation Throughput. Saturation throughput has mixed trends. In experiment *(i)*, when using longer link lengths (2mm and 4mm) for both the new and baseline NoC's, the improvement is only 4-5% over baseline in each case, while the original network (1mm length) exhibits a 15% improvement. In both examples, the system bottleneck becomes the round-trip communication between neighbor routers on the longer links, therefore the router optimization has a diminished role. In experiment *(ii)*, there was significant performance degradation for the single-flit packets, 33%, while the 8-flit packets retained the observed improvements. The former degradation indicates that the monitoring protocol becomes a bottleneck for single-flit traffic: single-flit packets arrive back-to-back, and the overhead of the four-phase monitoring protocol affects the processing rate. We believe that the protocol overhead, for this one special case, can be largely overcome by circuit-level monitoring link acceleration and protocol enhancement. In experiment *(iii)*, resulting improvements are almost identical between the 4×4 and 8×x8 2D-meshes.

Impact of Path Length on Proposed Approach. Concerns may be raised that our approach may not perform well for shorter path length or smaller networks, as monitoring might require several hops to get ahead of actual data. In practice, this is not the case. First of all, the above 4×4 2D-mesh experiment indicates a 31% latency improvement over baseline, vs. 36% for an 8×8 2D-mesh. On the other hand, the 3 permutation benchmarks picked for 8×8 network show highly-diverse average hop-count (4/6/8), but near-stable latency gains. Effectively, in our early arbitration protocol, after the first hop, a stable gap between monitoring and data is observed through the entire path, hence all remaining nodes are accelerated. As a result, stable latency improvements are obtained under widely varying network diameters.

Discussion

Unlike the monitoring protocol used in Chapter 4, a fine-grain and tightly-synchronized protocol is selected for the 2D-mesh topology. Early arbitration is performed on a router-by-router basis. That is, if the monitoring loses the early arbitration at a certain router node, it is not allowed to further propagate, until it finally wins at the current router. The corresponding data packet also has to wait until the monitoring wins the arbitration and allocates the channel.

Despite a conservative protocol, the network still shows dominating results over the baseline approach, which is attributed to two factors. First, a non-classical lookahead approach is used. Lookahead routing computation is initiated by the early monitoring information, instead of the actual data, and completed in parallel with early arbitration. Second, at the low-level implementation, the monitoring network bypasses both the input capture-pass register in the IPM and the output capture-pass register in the OPM, which considerably improves latency.

The net result of the combination of the above two factors is a well-structured design with a carefully-selected synchronized monitoring protocol, which considerably lifts the latency performance, while moderately improving the network throughput.

5.11 Conclusions and Future Work

This chapter introduces a new asynchronous NoC, AEoLiAN, to latency acceleration for a 2D-mesh topology. The method includes a lightweight monitoring network, with a modest increase in width: only 11 bits per channel. Uniform system latency results over 6 diverse benchmarks were obtained,

ranging from 34.4-37.9%, for moderate traffic, as well as substantial throughput benefits. As future work, we aim to migrate this approach to support VCs, and also to add mixed-timing interfaces to build a complete GALS system. Simulations will be performed on real benchmarks for more accurate results, while synchronization overhead is taken into account.

Chapter 6

An Industrial High-Performance and Low-Power Asynchronous NoC Router

6.1 Introduction

In the previous two chapters, a new asynchronous monitoring technique was proposed and applied to two different network topologies. The corresponding shadow monitoring networks notify the relevant routers early and let routers prepare for the incoming traffic in advance, and dynamically adjusting to network traffic conditions, at very low cost. The use of a monitoring approach results in considerable end-to-end latency improvements, as well as benefits to throughput.

However, the above two networks are only implemented at pre-layout (but post-mapped) levels, with buffers and virtual channel (VC) implementation largely ignored. In this chapter, a realistic industrial instantiation of a high-performance and low-power asynchronous router is presented, using a leading cutting-edge technology and a semi-automated design flow [104]. The work was in collaboration with, and completed during my internship at, AMD Research. This work also highlights the recent interest in industry in developing asynchronous NoC's, and in their potential as a viable solution for future large-scale digital systems.

The key contributions of the research are the following: *(i)* A basic 5-port asynchronous router is designed and implemented using an advanced industrial technology – a 14nm FinFET library. *(ii)* A novel end-to-end credit-based virtual channel (VC) flow control is proposed, which has fewer backward credit synchronizations to the upstream router, and is expected to outperform previous designs.

(iii) We provide the first published comparison for a high-performance asynchronous NoC router vs. an industrial synchronous NoC baseline router using such an advanced technology library. Unlike other baselines for research purposes, the synchronous design is used in recent high-end AMD processors and graphic products, to handle system-level configuration and power/performance monitoring and control. The comparison results are thus more persuasive and closer to reality. *(iv)* Finally, several industrial tools are used in the design process, for place-and-route (P&R) and design validation. These tools are modified from a standard synchronous design flow and thus open real future opportunities for industrial asynchronous NoC designs.

Besides the new VC technique, the asynchronous router also integrates low-latency input buffers using a circular FIFO design. Since buffers and VCs are commonly used in most of industrial NoC's, these router-level optimizations are important, and also orthogonal to network-level acceleration methods we proposed in Chapters 4 and 5.

The chapter is organized as the following. Section 6.2 presents the overall structure of the proposed asynchronous router, and the designs for high-performance buffers and VC controls. Section 6.3 introduces the complete design flow and the industrial tools been used. Section 6.4 shows the results for the proposed asynchronous router and compare them to the AMD commerical synchronous counterpart. Section 6.5 concludes the chapter. Due to confidentiality reasons, low-level design details are not shown, while all the results are presented in relative numbers only.

6.2 Proposed Asynchronous Router Design

6.2.1 Overall Router Structure

The router is designed for a 2D-mesh with a double-plane NoC, the same structure as the synchronous network to which we compare. Each router node in a double-plane structure contains two uncorrelated and identical sub-routers, as shown in Fig. 6.1. The *request plane* routes read and write request packets and the *response plane* delivers read and write responses. When a request arrives at the destination, a response packet is generated and sent back to the source node.

Each sub-router inherits a replicated-switch credit-based VC architecture [149]. As shown in Fig. 6.1, switches are replicated as many times as the number of VCs, e.g., Switches 0/1. VCs separate different traffic classes inside the router, which are mixed only on inter-router links. This

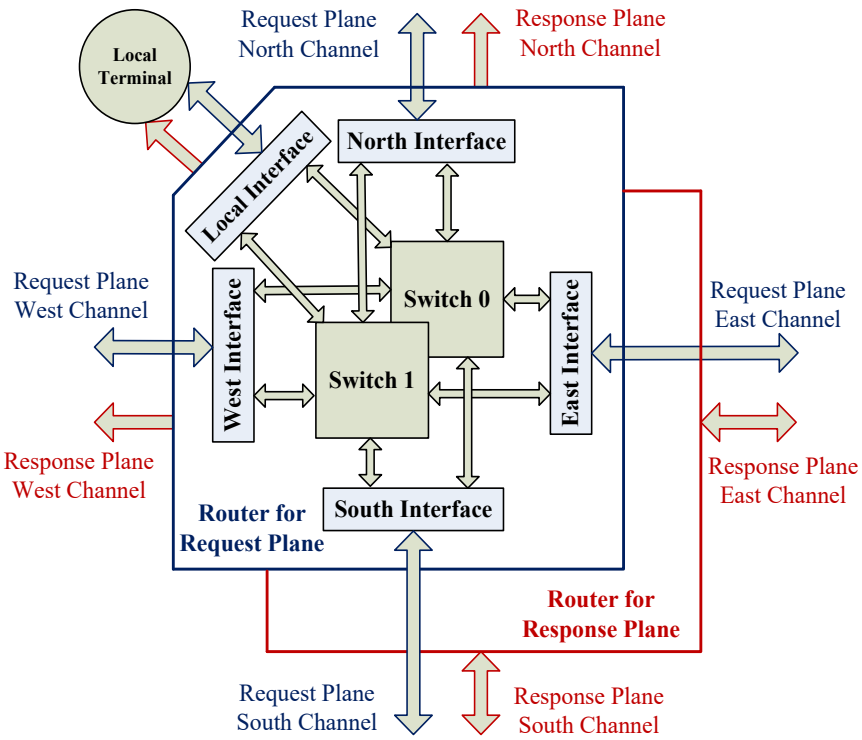


Figure 6.1: Node structure for proposed asynchronous double-plane router

structure outperforms a crossbar-sharing approach for asynchronous routers [149], although the latter is a typical approach for synchronous.

In each sub-router (ignoring the added VCs and buffers), the switch design is identical to our baseline in Chapter 5 [80]. When a packet arrives, it is first demuxed and routed to a certain switch, depending on its statically assigned VC, and then traverses the switch through arbitration, and finally merges with traffic of the other VC on the output channel.

6.2.2 Input Buffer

FIFOs are commonly used to provide additional storage capacity, in order to improve system-level performance. Input buffers are commonly used for NoC routers. Multiple registers can be placed one after the other to build a serial FIFO, but this introduces a severe latency penalty. Therefore, an efficient transition-signaling circular FIFO is used with constant low access time, which can provide

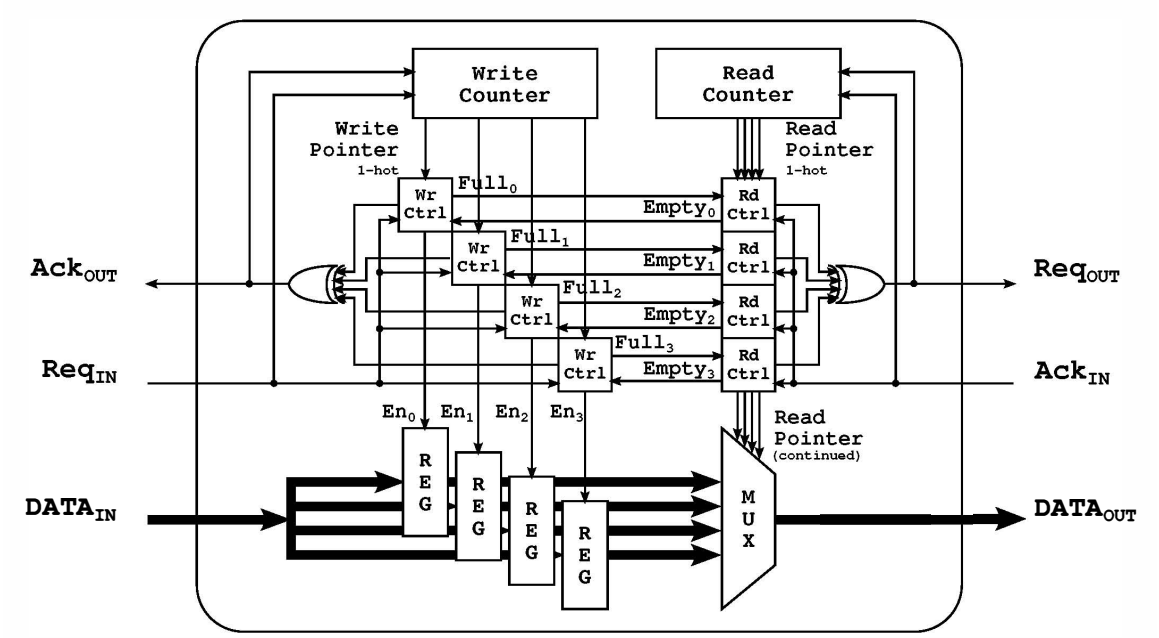


Figure 6.2: Input buffer circular FIFO: structure

much lower latency and cycle time. The FIFO structure is shown in Fig. 6.2. Its microarchitecture and operation have already been presented in [80].

Compared to a typical synchronous circular FIFO, the proposed asynchronous FIFO shows two significant and unusual advantages: *(i)* First, each asynchronous storage element is a single level-sensitive D-latch register, which has full storage capacity. In contrast, synchronous FIFO requires Flip-Flop (FF) or double-latch storage registers. A D-latch has approximately only half area and power cost as a FF, and usually has much lower latency. Since buffers usually occupy a large amount of area in routers, this is one of the key benefits for asynchronous switches. *(ii)* Second, written-in data can be immediately read out through a very short latency for the asynchronous circular FIFO. Write/read operations do not have to be aligned to clock cycle. On the contrary, it takes at least 2-3 cycles for any synchronous FIFOs from writing in to reading out a data item.

6.2.3 Proposed VC Flow Control

A new credit-based VC control is proposed, which outperforms the design of [149]. In [149], a protocol is used where the credit count is decreased whenever a flit is sent out and is increased

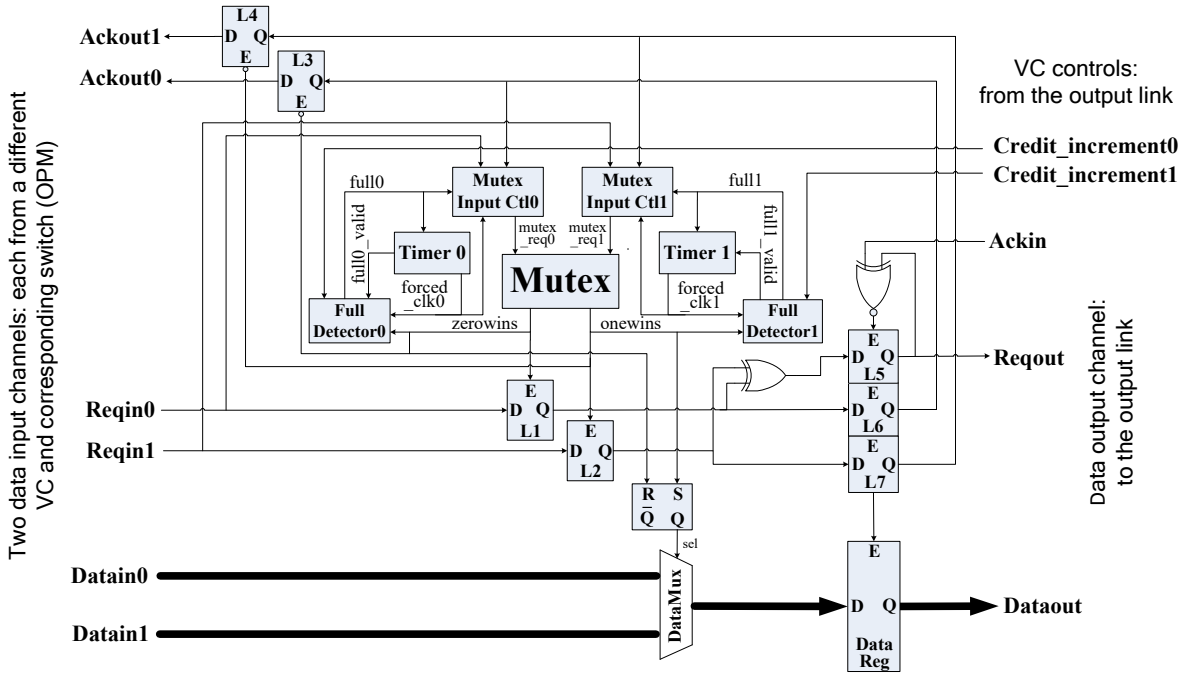


Figure 6.3: Proposed VC control for an output channel interface

whenever the successor releases an input buffer slot. These two operations are mutually exclusive, and treated symmetrically at the same priority. As a result, a non-critical credit-increment operation can thus potentially block a critical credit-decrement, and delay sending out a flit. In contrast, the new VC approach only updates the credit when a flit is sent out. Credit-increment requests are queued and are only updated along with the next credit-decrement request. This new 'lazy-update' scheme prevents unnecessary credit-increment updates and potentially increases the throughput.

Fig. 6.3 shows the microarchitecture of the new VC flow control. The module is part of the router interface. It takes two input data streams from two VCs, performs flit-level arbitration and merges them into a single output stream. The *Full Detector* and *Timer* units are the core of the new VC control. The *Full Detector* updates the credit every time a flit is sent out. The update considers all queued credit-increment requests as well as the current credit-decrement. The unit also prevents further arbitration requests for the corresponding VC when no credit is available. In that case, *Timer* is also activated, which constantly checks at a configurable given rate if any credits are released. When there is available credit again, the blocking is released.

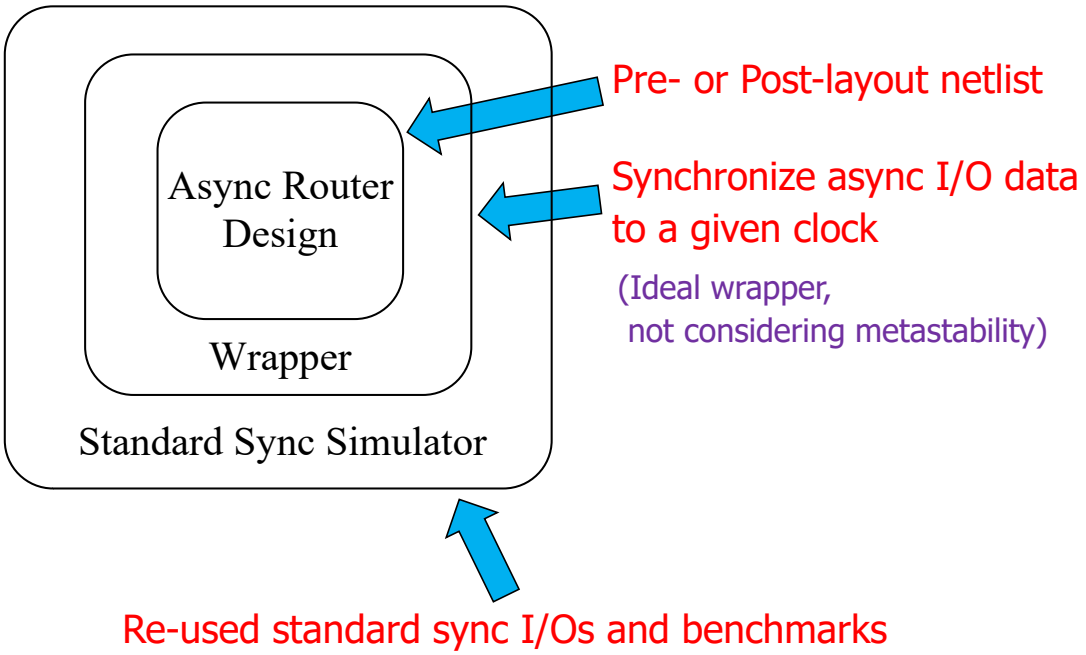


Figure 6.4: Design validation tool illustration

6.3 Design Flow and Tools

6.3.1 Design Validation Tool

Functional validation at gate-level are performed for both pre- and post-layout designs for a single node. The validation tool is migrated from that used in the AMD industrial setting for the synchronous design. As shown in Fig. 6.4, a wrapper is added for the asynchronous router to synchronize the input and output data to a given clock. Therefore, the asynchronous router with wrapper can be simply plugged into the existing tool, and uses any standard benchmarks as for synchronous testing.

6.3.2 Design Flow and P&R Tool

An illustration of the entire design flow is shown in Fig. 6.5. The design was first manually synthesized by mapping each gate to a real commercial cell library. Asynchronous one-sided timing constraints are satisfied by manually adding proper inverter-chain delays. Manual mapping prevents logic synthesis automation, which can potentially create control glitches. More systematic research

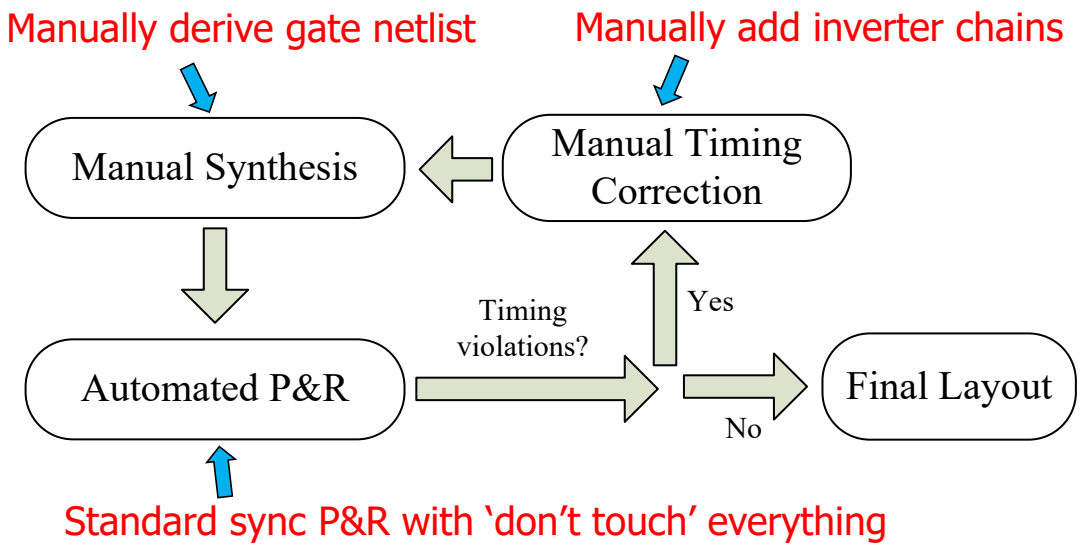


Figure 6.5: Design flow illustration

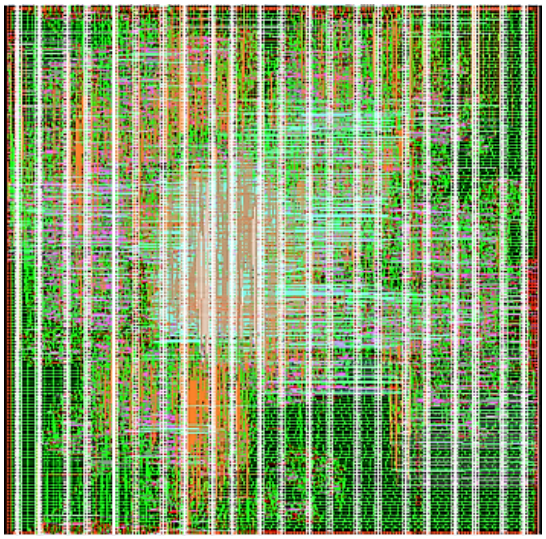


Figure 6.6: Actual layout for the proposed asynchronous router

solutions for asynchronous logic synthesis automation have been proposed in [80], which are not included due to the extensive effort required to re-instrument the stable industrial flow. However, it is expected that no serious obstacles appear to their inclusion in the future.

The P&R flow uses a standard automated synchronous approach, without logic optimization. The final asynchronous layout is shown in Fig. 6.6, which was used for post-layout design valida-

tion.

In sum, compared to the fully-automated synchronous design flow, the asynchronous router is implemented using a different semi-automated process. The synchronous design is, in fact, somewhat more optimized. Several optimization techniques are used during design synthesis, including logic optimization with gate transformations, gate selection from different categories (i.e. high- or low-VT gates), as well as low-level transistor sizing. There optimizations were turned off for manual synthesis of the asynchronous router, to ensure hazard freedom on control paths. Minimum-size gates with typical VT are selected for most parts of the design. Higher-drive gates are only used where the design has large fan-out. However, even with these disadvantages in the design flow, the asynchronous results still dominate the synchronous counterpart.

Although post-layout results could not be reported at this time, due to the use of advanced commercial technology, this flow demonstrates the viability of incorporating asynchronous physical design into a leading industrial environment. Furthermore, the strong initial asynchronous results are highly encouraging, and are expected to contribute to industrial motivation to invest in asynchronous CAD tool development.

6.4 Experimental Results

Experiments are performed for a single pre-layout router node in terms of area, latency and power. The synchronous baseline is a typical 3-cycle router, with fine-grain clock-gating. It also has some additional functionality for error detection and router configuration; these contribute only 1-4% area and power increase, with negligible performance impact. All results are presented in relative numbers only, due to confidentiality reasons. Based on industrial experience, it is expected that pre-layout comparisons and post-layout comparisons will be similar for such as small router design.

The results for the basic comparison is shown in Fig. 6.7. Both asynchronous and synchronous routers are configured to have 2 VCs, each with buffer depth of 7. Each router is synthesized using a low-power industrial 14nm library (0.65V, TT corner, 273K). The synchronous router is synthesized targeting a 1 GHz clock rate, based on the performance requirements of several high-end AMD products, using a standard automated flow, while the asynchronous router is synthesized manually, as indicated. Evenly distributed random traffic is sent from all input ports to all output

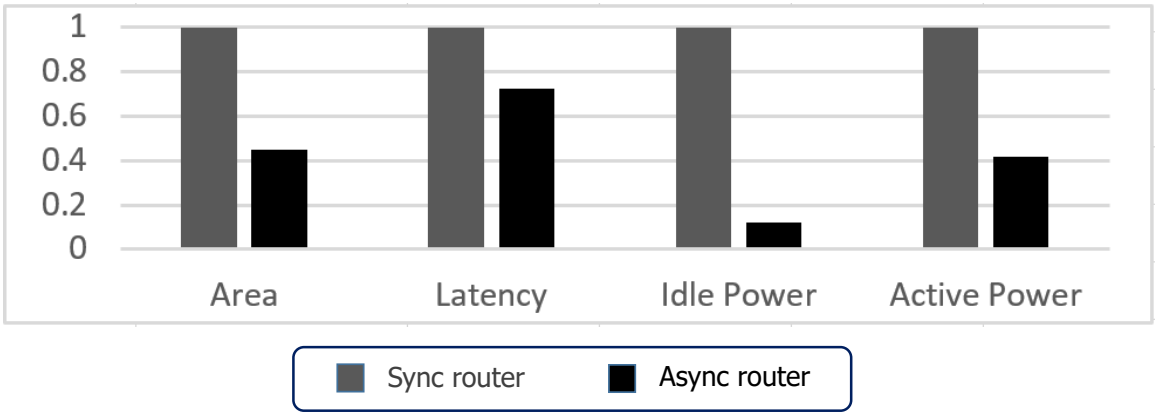


Figure 6.7: Asynchronous vs. synchronous router: basic comparison

ports, with a random packet size between 2 and 6. The asynchronous router dominates the results: 55% lower area adn 28% latency improvement, with 88% and 58% savings in idle and active power, respectively.

There are three reasons why the asynchronous design can achieve fundamentally better results than the synchronous counterpart.

The first reason is related to the clock distribution and clock power consumption. The synchronous router requires a fairly complicated clock tree to distribute the clock, while the asynchronous design entirely eliminates the clock distribution. This is a considerable save in both power and area cost. Also, asynchronous components consume dynamic power only when they are activated. Effectively, the asynchronous router provides clock gating at an arbitrary granularity without any instrumentation. The synchronous design, however, requires extra hardware to implement fine-grain clock-gating.

Second, the asynchronous router uses single-stage latch registers for data storage, compared to FF's used in the synchronous router. Interestingly, even using single-latch architectural registers, the asynchronous design requires only simple one-sided (i.e. worst-case) timing constraints to be satisfied (unlike several synchronous single-latch approaches which have min/max constraints or pulse-mode operation). The input and/or output buffers always consume a large amount of, sometimes even dominate, the area and power consumption of a router. Since an asynchronous-style single-stage D-latch register only consumes roughly half area and power as a typical FF, this be-

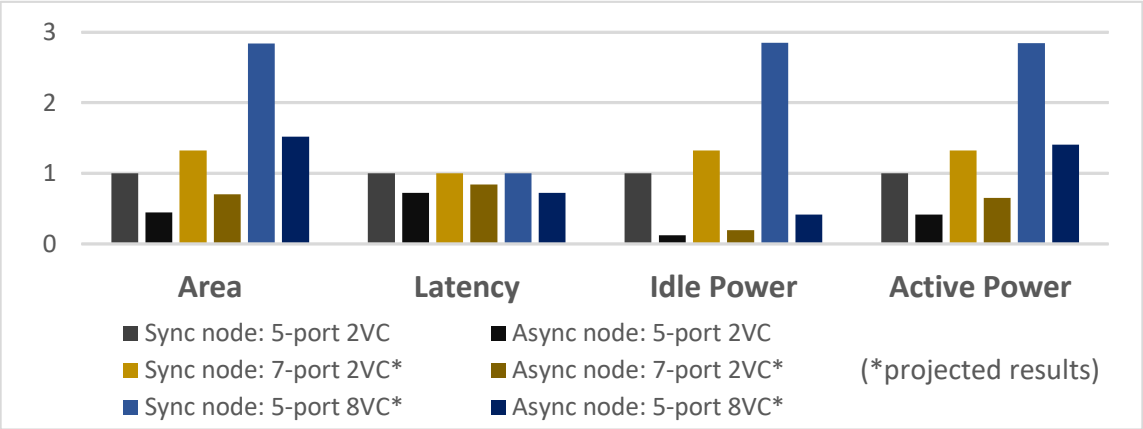


Figure 6.8: Asynchronous vs. synchronous router: projected results

comes another major source of area and power savings. Also, single-level latches are only activated when data exists, while the FF's are activated every clock cycle. What is more, some of these asynchronous latch-based data registers use a capture-pass protocol and are normally transparent, so data can go through these registers directly whenever it arrives. However, FF-based registers require the clock to be broadcast to each of the FF in the register bank, which considerably degrades the latency performance.

Finally, unlike the synchronous router, the overall performance of the asynchronous router is an average among all flits (i.e. header, body and tail flits), which is not limited to the worst-case. In particular, in the asynchronous router, the header flit sets up the switch path and has the worst latency. The body and tail flits only need to follow the pre-setup path, including pre-open channel latches, and default-open flow control registers. These non-blocking latches ensure a much faster processing speed for body and tail flits. In contrast, for the synchronous baseline router, the clock rate needs to be set according to the worst case – the header flit. There is no performance speed-up for body and tail flits.

Also, unrelated to our results, asynchronous NoC's can handle varied link lengths with little overhead. The synchronous NoC's must discretize each link communication to a fixed number of clock cycles. For example, for short links, where data can be transmitted much less than a clock cycle, a full cycle is still allocated for the link traversal. For asynchronous NoC's, however, the link transmission latency is entirely flexible and not aligned to clock cycles.

As additional experiments, Fig. 6.8 presents estimated results for two useful alternative designs: *(i)* a 7-port router with 2 VCs, and *(ii)* a 5-port router with 8 VCs. The former is important for 3D stacking, and the latter represents a more realistic VC configuration. For both synchronous and asynchronous routers, area and power costs noticeably increase in *(i)* and *(ii)*, due to higher radix or more VCs, while latency is largely unchanged. However, relative asynchronous area and power benefits are largely maintained, though latency improvements are reduced for the 7-port configuration.

6.5 Conclusions

This chapter focuses on a real-world instantiation of a high-performance asynchronous router in an industrial setting. A high-performance and low-power asynchronous router is implemented, integrating low-latency input buffers, and a novel end-to-end credit-based virtual channel (VC) flow control approach. The new VC approach has fewer backward credit synchronizations to the upstream router – only updating the credit count when a flit is sent out – and thus is expected to outperform previous designs. The router is implemented using a cutting-edge 14nm FinFET library, and compared directly to an AMD commercial synchronous counterpart. To the best of our knowledge, this is the first comparison for an asynchronous router vs. an industrial synchronous baseline using such an advanced technology library. Also, several industrial tools are used in the design process, for place-and-route (P&R) and design validation. The development of this project at AMD Research, and the harnessing of some synchronous CAD tools in the design flow, highlight industrial interest in developing asynchronous NoC's, and their use as a potentially viable solution for future large-scale commercial digital systems.

Chapter 7

A High-Throughput Asynchronous Multi-Resource Arbiter

7.1 Introduction

While previous chapters present complete NoC or router designs, including new techniques for performance optimization, this chapter focuses on a single important component that is almost always used in any NoC: an arbiter. Arbiters are critical for NoC performance, and sometimes the bottleneck for the router as well as the entire network. In synchronous designs, the arbitration problem is fundamentally simplified: resources are allocated based on distinct discrete clock cycles. In contrast, asynchronous arbiters are required in asynchronous NoC's. The asynchronous arbitration problem deals with competing requests and resource assignments in continuous time, unaligned to clock cycles, which may arrive arbitrarily close together (e.g. a few picoseconds or less), therefore the problem is fundamentally more challenging [80, 104, 105, 148, 166].

Generally speaking, there are two types of arbiters: *single-resource arbiters* and *multi-resource arbiters*. Single-resource arbiters, also called single-resource multi-way arbiters, are used to decide the serial order in which the requests are served when several clients want to access a single shared resource concurrently. Multi-resource arbiters generalize the case: they assign multiple interchangeable resources among a number of requesting clients.

In asynchronous NoC's, single-resource arbiters are usually used for channel arbitration when

multiple incoming data streams compete for the same single output channel. There is already a large amount of existing work on asynchronous single-resource arbiters, with different structures and interesting trade-offs [62, 63, 80, 108, 148, 160, 206, 242]. Several simple arbiter examples were introduced in Section 2.3.3 as asynchronous background. In particular, a new scalable arbiter design approach was recently presented in [148], and applied to a family of arbiters with different numbers of input requests (practically up to 16). The designs have simple structures, targeting an overall high performance, as well as an increased robustness and impartiality across all inputs.

In this chapter, our focus is instead on multi-resource arbiters, which are more complex and for which there are only limited existing efficient designs and only little recent research. In asynchronous NoC's, multi-resource arbiters are typically used for VC allocation for packets to select any available VC from a VC pool of the same service level [62, 63]. Also, multi-resource arbiters can be used for a novel type of NoC – asynchronous *spatial-division multiplexing* (SDM) NoC's [126, 128, 208]. SDM is a promising direction for future NoC's, where inter-router links are subdivided into sub-channels for concurrent data transmission. The SDM technique has been proved to be an alternative strategy to the classic VC approach for mitigating head-of-line blocking, in order to target high network throughput. Multi-resource arbiters are used to select any one of potentially several free sub-channels as an output channel for a data transmission.

The chapter proposes a new high-throughput asynchronous multi-resource arbiter based on fine-grain pipelining [106]. The allocation of a resource to a client is divided into several steps, where multiple successive client-resource pairs can be selected rapidly in sequence, and the completion of the assignments can overlap in parallel. The proposed arbiter is well-structured and highly scalable, and can be simply extended to a design with arbitrary numbers of clients and resources. In addition, a new static four-phase asynchronous pipeline structure is introduced and applied to the arbiter design, incorporating a highly-concurrent handshaking protocol. In particular, experiments are performed at two interesting and different design points: 4×3 and 8×4. Results show that the proposed arbiter achieves considerable lower cycle time than a previous baseline design at both design points. Interestingly, in spite of the pipelining, the new arbiter exhibits slightly better latency in light traffic.

The remaining of the chapter is organized as follows. Section 7.2 introduces existing approaches for multi-resource arbiter. Most previous designs are from Yakovlev's group, and we use one of their

designs as baseline for comparison. Section 7.3 then presents the structure and the operation for the baseline design. Through Sections 7.4 to 7.7, the proposed asynchronous multi-resource arbiter is introduced. Before diving into details of the design, we start with an overview of the approach in Section 7.4, and Section 7.5 presents a new high-performance asynchronous pipeline style. The pipeline is actually a novel static implementation of the dynamic HC pipeline [202], which is a new general structure that can be used for a variety of asynchronous applications. The pipeline is applied extensively in our proposed arbiter design in modified ways; then, Sections 7.6 and 7.7 present the details of the proposed arbiter design and the timing analysis, respectively. Experimental results for the new arbiter and its comparison with the baseline are given in Section 7.8. Finally, the chapter concludes in Section 7.9.

7.2 Related Work

There is only limited prior work on asynchronous multi-resource arbiter design. Yakovlev's group is the only one having several recent publications, including two different approaches: *serial assignment* and *parallel assignment*.

A serial design [85, 86] allows only one resource allocation at a time. The implementation is simple, with an elegant structure, which exhibits good scalability and has relatively low latency. The protocol, however, is quite conservative and slow when there are multiple waiting clients and resources, resulting in poor throughput.

Two different parallel designs have also been proposed. A multi-token ring arbiter [85, 86] consists of a ring of client cells and resource tokens continuously propagating in the ring. A client cell obtains a resource when needed, and puts the token back after usage. The arbiter is expected to have high performance. However, the paper highlights several significant shortcomings. First, each cell always requires two arbitrations in series: to resolve if a token in the ring will be acquired by the cell, and also if it must compete with a token that the cell wants to put back into the ring. Also, the design is not suitable for nearly-equal client and resource numbers, since the result is a congested ring which has slow movement of tokens. Finally, the design is typically for passive resources only.

Another alternative parallel design uses a monolithic arbiter with synchronization based on discrete time frame division [196]. In each time frame, a static set of requesting clients and resources

are maximally matched, as in a synchronous design. Any late-arriving requests must be deferred until a later time frame. The results shows some performance improvement over the serial design. The design, however, requires a mutex for each client and resource channel, as well as a large number of high fan-in gates and C-elements, and hence is not scalable for larger arbiter size. In addition, many details of the implementation are not included, and results are largely missing.

The *committee problem* [19] is a more complex multi-resource arbitration problem. While it is largely orthogonal to the basic multi-resource arbitration problem on which we focus, the idea of client competition and parallel resource assignments also exists. In a committee problem, a set of professors is assigned to committee groups. The problem is to schedule committee meetings concurrently if all the professors in the group are available, while no professor can attend multiple meetings simultaneously. Several solutions were proposed [19]. In the best design, each committee in parallel examines its members, and locks available members during the examination. The committee starts the meeting when all the members are available. Otherwise, it releases the obtained professors and restarts a meeting request. While exploiting the parallelism in protocol, the design is based on a communicating sequential process (CSP) two-phase model specification, which results in quite slow and complex hardware.

7.3 Background: Baseline Multi-Resource Arbiter

This section reviews the non-pipelined serial asynchronous multi-resource arbiter, invented by Golubcovs and Yakovlev *et al.* [85, 86]. The design uses a simple and clean protocol, and appears to be the best existing asynchronous solution. It serves as the baseline for the comparison. We start with a black-box view of the arbiter with introduction on its channel protocols. Then the detailed structure of the design is introduced, followed by a simple simulation.

7.3.1 External Channel Protocols

A black-box structural overview of a NxM baseline asynchronous multi-resource arbiter is shown in Fig. 7.1. There are N client channels on the left, and M resource channels on the right.

Clients can initiate concurrent and independent requests, and are granted when resources are available. The resource ID contains the result of allocation, and is delivered along with the grant

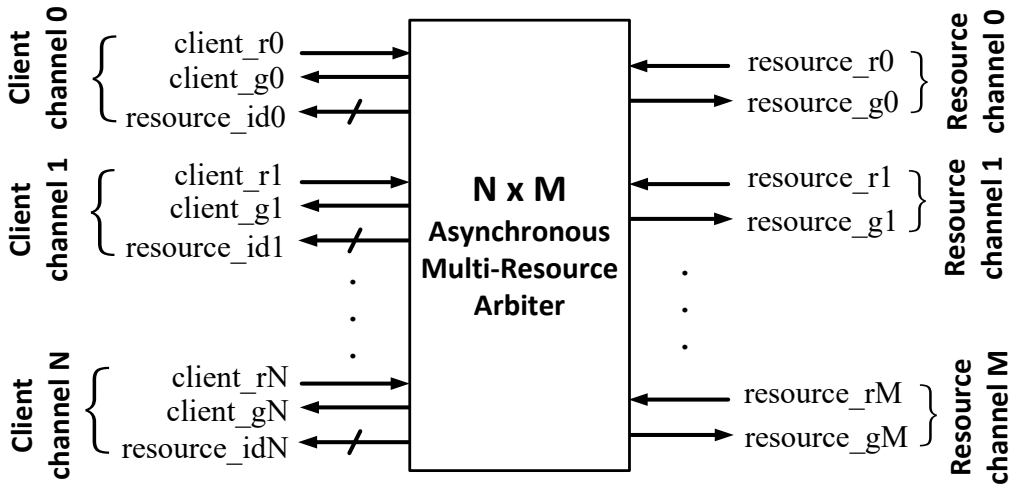


Figure 7.1: Baseline asynchronous multi-resource arbiter: a black-box overview

to a winning client. Likewise, resources also have to proactively report their availability through resource requests, and therefore are called *active resources* [85, 86]. The resource channel then operates symmetrically to the client channel, but without ID information. As an alternative, *passive resources* automatically become available to clients immediately after each utilization, without any explicit asserted request. Active resources can be useful to properly handle when a resource fails or it turns off to save power. In addition, an active protocol can be simply transformed back to a passive protocol, if a resource immediately reports its availability after it is released. The baseline design assumes active resources, which our new design also inherits.

All the client and resource channels use a four-phase communication protocol. The entire lifetime of a client making use of a resource actually involves three operation phases, as shown in Fig. 7.2. The resource first needs to be matched with the client (*assignment phase*). During this phase, the client and resource requests are asserted high, and both channels are granted. The client then holds and uses the resource for an arbitrarily long time (*utilization phase*). There is no channel activation in this phase. Finally, the assignment is released (*release phase*), during which client and resource requests are de-asserted low, followed by both grants de-asserted low.

The challenge of multi-resource arbitration largely concentrates on getting an efficient design for the assignment phase. Hence, this is the main focus of our research.

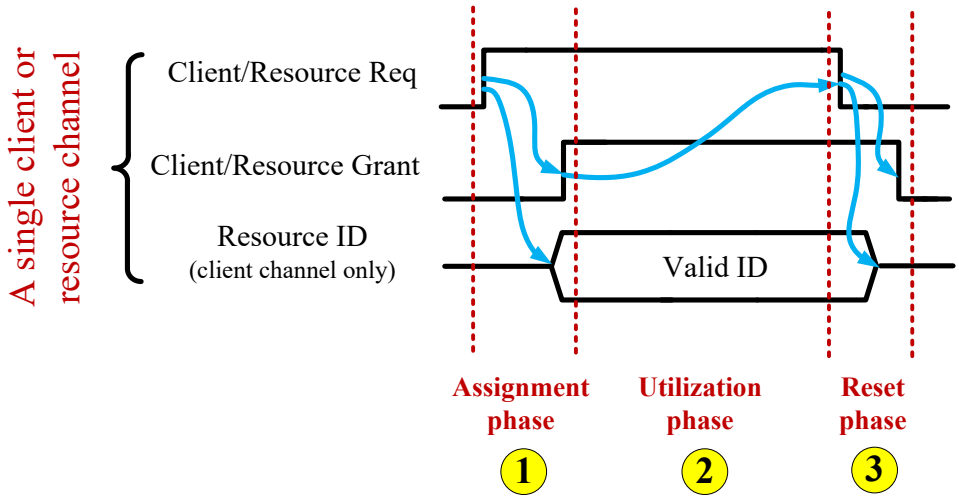


Figure 7.2: External channel protocol

7.3.2 Structure

Fig. 7.3 shows the architecture of a 4×3 baseline arbiter example, with four client channels and three resource channels. As indicated before, each channel contains a four-phase request and a grant. The resource IDs for client channels are 1-hot encoded, and indicate the assignment result.

The entire design is divided into three major blocks. The *Client Generator* arbitrates between concurrent client requests and produces a single winner. The *Resource Generator* has a similar structure for resources. The *Assignment Unit* pairs the two winners, then records the assignment and generates the grants.

There are several important components of the *Assignment Unit*. The core of the unit is a 4×3 cell bit array, which stores a snapshot of the currently active assignments. In particular, each cell presents the state of a distinct client-resource pair: a 1 value indicates 'assigned', and a 0 value means 'not assigned'. At any time, there can be multiple assigned client-resource pairs.

In the array, restricted operation is enforced, where only a single cell can be written to 1 at any time, which is the essence of a serial approach. The Client Winner Generator sends information on a single client winner to the array, and the Resource Winner Generator sends similar information on a single resource winner, and the array then records the winning pair in the corresponding cell. After assignment, the two *Grant Generators* inform the client-resource pair of the start of the utilization

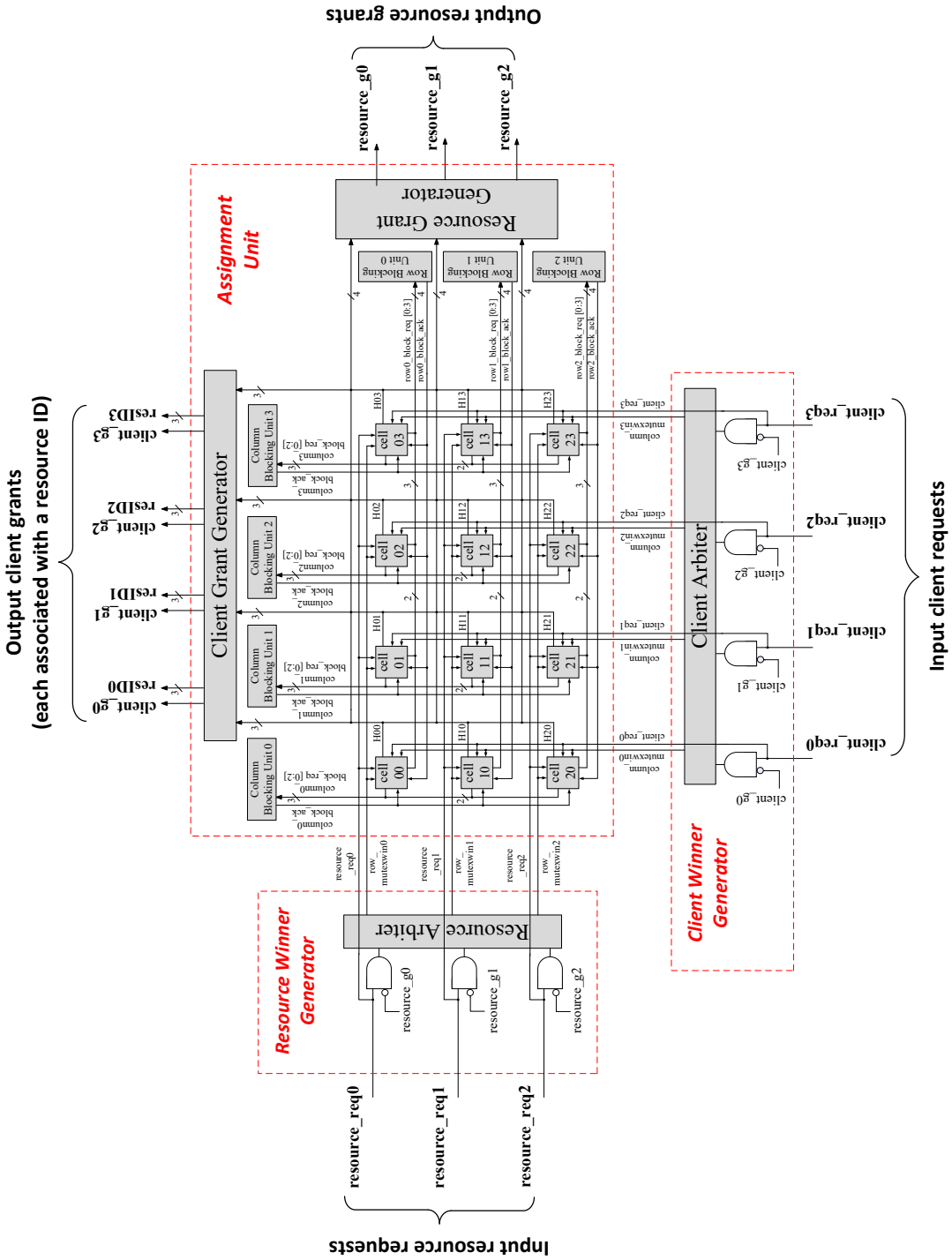


Figure 7.3: Baseline asynchronous multi-resource arbiter

phase by asserting the corresponding grants. When the client and resource request to release the assignment (i.e. reset phase of handshaking channels), the corresponding cell is cleared (i.e. reset to 0), thereby ending the utilization phase. Finally, *blocking units* are provided in each row and column; when the *Assignment Unit* writes a cell, these units prevent any spurious writing to other cells in the same row or column.

7.3.3 Operation

As illustration, a simple scenario is simulated, where a single client and resource request are received and paired together, both arriving on channel #0. The client request is assumed to arrive first, and quickly arbitrated to become the winner. The resource winner is generated similarly. Next, *cell00* synchronizes on the winner pair, and initiates row and column blocking in parallel, before the assignment can be recorded into the cell. The operation disables all other cells in the same row and column. A single round-trip communication between the cell and the blocking unit is required to complete each blocking operation. is recorded, and grants are generated. Finally, both arbiters are reset right after grant generation, and row/column blocking is released, to allow the next serial assignment. At any time, an assigned cell can be cleared, without synchronization with ongoing assignments.

While the above protocol is well-structured, it has significant performance overhead. In particular, a sequence of steps – arbitration, cell selection, row and column blocking, and cell write – must be followed in series, for a single assignment, before any new assignment can be started, resulting in a critical throughput bottleneck. The goal of the new approach is to break through the serial limitation, and allow both pipelining and limited overlapped concurrent operation (e.g. in cell writes).

7.4 Overview of the Approach

The basic strategy of the new approach is now introduced, before delving into details in the new design.

As shown in Fig. 7.4(a), the baseline protocol is simple but conservative. Assignments are performed successively without any overlapping. An assignment begins with two concurrent arbitration

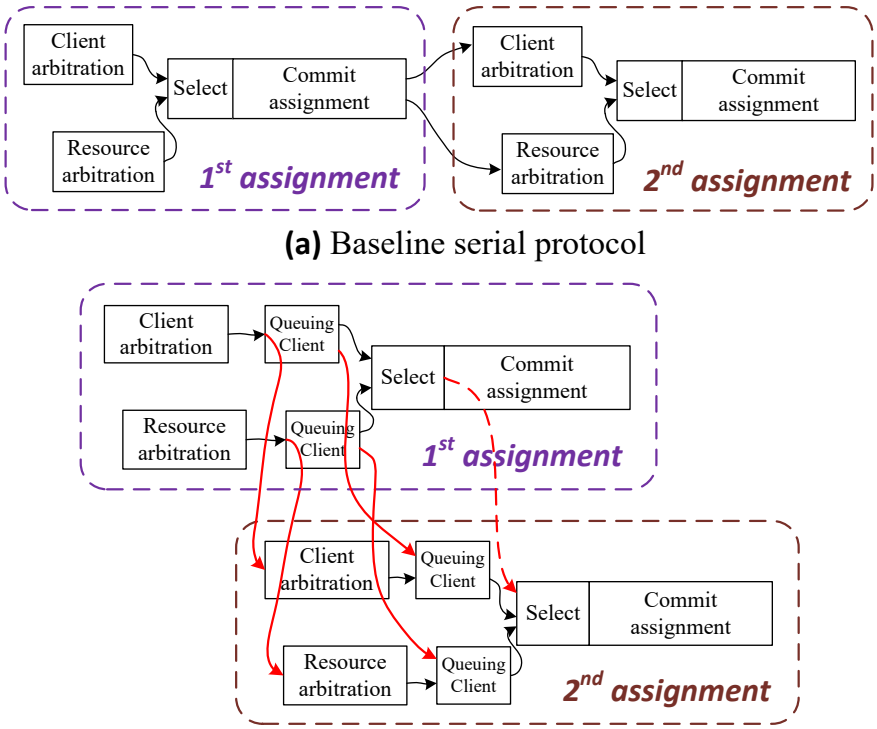


Figure 7.4: Protocol comparison: baseline vs. new

steps, 'client arbitration' and 'resource arbitration', where a client winner and a resource winner are determined. As soon as both arbitrations are complete, the pair is selected (i.e. the corresponding cell synchronizes row/column requests, then activates blocking) and the assignment is committed (i.e. the cell is written, then client and resource grants are generated, followed by release of the two arbiters). A 'commit' saves the pair and the informs both client and resource. The 'commit' is always followed by the 'select' without any delay.

The proposed protocol is shown in Fig. 7.4(b); it includes fine-grain pipelining to deliver higher throughput. In particular, as shown, the assignment operation is now broken into four sub-steps, including a new inserted queueing step. The result is that multiple assignments can be active simultaneously, operating on different sub-steps. The queue is a special multi-token circular FIFO, which provides additional parallelism, by allowing the arbiter to complete multiple successive operations even when the downstream pipeline is congested. It also offers fast data access time. Finally, the 'commit assignment' sub-step is distinct: this last operation is not pipelined, but instead full parallel

overlap of multiple 'commit assignment' sub-steps are supported by a concurrent hardware unit.

7.5 Proposed Static HC Pipeline

Before we present the proposed arbiter design, a new high-performance asynchronous pipeline is first introduced. This pipeline is then intensively used in the new arbiter with some modifications. In addition, it can be treated as a general stand-alone pipeline approach to be used in many non-arbiter applications, which advances the state-of-art of high-performance asynchronous pipelines.

Our proposed design effectively implements the high-capacity dynamic pipeline [202] using static logic. The pipeline protocol is highly concurrent with only one backwards synchronization arc, while static logic provides a better level of robustness.

The section is organized as the following. The pipeline protocol is first presented, then followed by a design overview. Finally, we introduce the related work, and compare the new pipeline with an existing well-known design through careful analysis.

7.5.1 Pipeline Protocol

The overall protocol of the new pipeline is shown in Fig. 7.5(a), and the actual design is illustrated in Fig. 7.5(b). Forward and backward synchronization are commonly-used terminologies in asynchronous pipelines [166, 202, 165, 239]. A forward synchronization is always enforced in any asynchronous pipeline: data needs to be processed and forwarded stage by stage. A backwards synchronization, on the other hand, is for the successor stage notifying and coordinating with the current stage to continue the operation.

The new static HC pipeline inherits a four-phase communication protocol from the conventional HC pipeline. The four-phase communication protocol is selected in order to comply with external requests and grants, which are also level signals. Expensive protocol conversion is thus avoided. The new protocol also only has a single backwards synchronization arc as the dynamic HC pipeline, for high throughput.

Each pipeline stage cycles through three phases, as shown in Fig. 7.5(a), which are largely mapped from the original dynamic HC pipeline. The phases have different names, since static logic is used. Initially, stage N is in the 'eval' phase. As soon as a new data enters the stage, it captures the

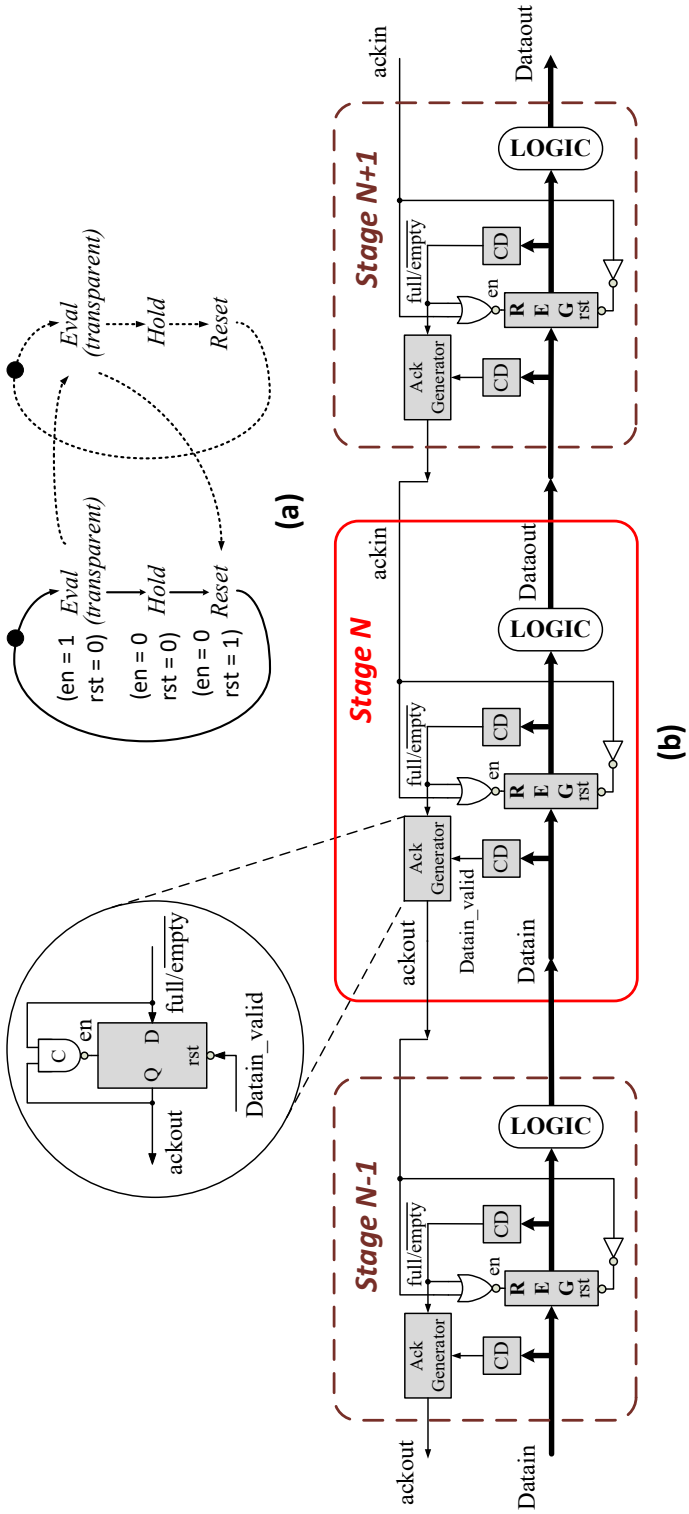


Figure 7.5: Proposed pipeline: (a) operation protocol; (b) structure

data and moves to the 'hold' phase. In parallel, the data is first evaluated and then sent to the next stage. In addition, the stage also acknowledges the previous stage N-1, with its single backwards synchronization arc (which will in turn initiate the 'reset' of the predecessor). Once the successor stage N+1 has completed its evaluation phase, it enables stage N to complete an entire cycle, without any further synchronization – reset, evaluate a new data item, and hold it.

The new protocol largely follows the HC pipeline, with a small modification required due to the use of static logic. In particular, an *early done* optimization is still used, as in the dynamic HC pipeline [202], where the current stage acknowledges the previous stage at the start of its evaluation. However, a new *early isolate* optimization is now needed: a stage must be isolated as soon as a new data arrives, i.e. when its evaluation begins. In contrast, the original HC pipeline used a *late isolate*, only after evaluation is complete. This optimization is needed with the static design: while in the original HC, dynamic gates and registers are inherently immune to the reset of their inputs (after start of evaluation), the static HC pipeline must isolate the D-latch registers to prevent reset inputs from propagating forward.

7.5.2 Pipeline Design and Structure

An example of a proposed 3-stage static HC pipeline is shown in Fig. 7.5(b). A 4-phase handshaking protocol is used for control. The data is DI encoded. Data registers are composed of single-level D-latches. Each register is controled by an *enable* and a *reset* signal. In each phase, a distinct configuration is applied for the two controls. There are two completion detection units (CDs) per stage. They are used for detecting a valid data, and placed on the left and right of the register, respectively. This is similar to the pre-charged half-buffer (PCHB) pipeline, invented by A. Lines [134]. The NOR2 gate is used to control the opening and closing of the register, while the *Ack generator* sends the acknowledgment to the left.

7.5.3 Related Work and Comparison

There are only a few asynchronous pipelines using a four-phase protocol and static logic datapath [72, 129, 244]. Most have two backwards synchronization arcs per cycle between two adjacent stages [129, 244]. Only the *fully-decoupled* pipeline in [72] has a protocol with only one back synchronization arc, and uses a similar static HC protocol as ours. However, the Furber/Day design has

significant performance overhead and complex controls.

In particular, a careful analysis indicates that compared to Furber/Day's approach, the proposed pipeline *halves* the forward latency per stage, and has slightly better cycle time. The forward latency of the proposed pipeline is only a 'D to Q' delay of a single D-latch, while Furber/Day's design involves two asymmetric C-elements (aCs), and one of them is a 4-input aC with relatively high stack. For the cycle time, the new pipeline has a result that is estimated as equal to 13 simple logic gates, while Furber/Day's design involves 14 simple logic gates. Also, the stage control of the new pipeline is simple, containing two low-cost CDs for detecting a valid small delay-insensitive (DI) code (e.g. 1-hot data in our case), two standard cells and a regular 2-input C-element. In contrast, the Furber/Day's pipeline contains four non-standard aCs per stage. Some of them are 3- and 4-input aCs with relatively high stacks.

7.6 Proposed Asynchronous Multi-Resource Arbiter

In this section, the new multi-resource arbiter is presented in detail. We start with its overall structure and operation, then followed by the design of all sub-module blocks and their operation. While the focus is on a particular instance, a 4×3 arbiter, the design can easily be generalized to higher dimensions.

7.6.1 Structure

Fig. 7.6 shows the architecture of the entire design. The arbiter has four clients and three resources. A four-phase communication protocol is used, both for external client and resource channels and all internal handshaking channels.

The structure is divided into three blocks. The new *Client Winner Serializer* generates client winners serially, and enqueues the winners. Similarly, the *Resource Winner Serializer* creates a queue of available resources. Each of the serializer units is also a fine-grain pipeline, allowing multiple data items to be processed simultaneously. In contrast, in the baseline approach, the client and resource units process and produce only a single winner at a time. The new *Assignment Unit* then selects client-resource pairs from the two queues, and commits the assignments. Interestingly, this unit effectively allows full parallelism: the selections are taken concurrently, and multiple parallel

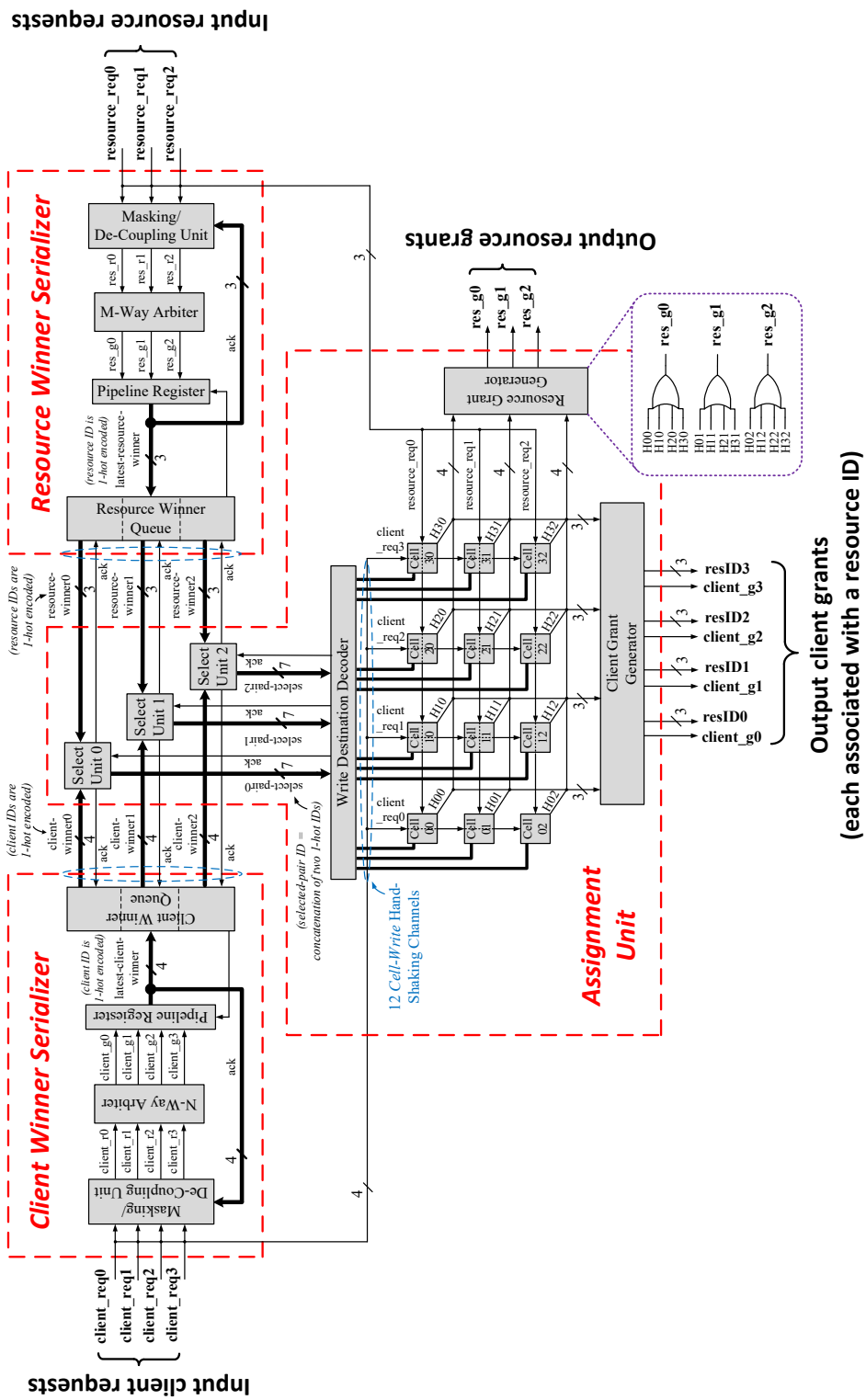


Figure 7.6: Proposed asynchronous multi-resource arbiter

assignments are allowed to be recorded into the cell. In contrast, the baseline *Assignment Unit* allows only a single selection and a commit at any time. Both the pipelining and parallelism features introduced above are highlighted in the new protocol, as shown in Fig. 7.4. Finally, at a macro level, the three units form a non-linear pipeline: the *Assignment Unit* is a *join* stage, which merges the data from the two serializer stages [169]. The three blocks are now introduced in turn.

The *Client Winner Serializer* has four client requests as inputs, and has three output channels. Each output channel contains a single client winner, 1-hot encoded, and an acknowledgment. The unit is further pipelined into three stages. The left stage generates a client winner using a normal single-resource arbiter. The middle is a fast single FIFO stage, which decouples the arbiter from the right stage. The right stage is a multi-token *Client-Winner Queue*, which stores successive winners, and provides parallel access to the *Assignment Unit*. The queue depth is three, which equals the maximum number of concurrent assignments allowed in the design.

The *Resource Winner Serializer* has almost the same structure and operation as the *Client Winner Serializer*.

The *Assignment Unit* obtains two sets of winners from the two serializers, clients and resources respectively, and also receives direct client and resource requests from the external channels. In turn, the unit produces grants for these external client and resource channels. The unit provides three key functions: *(i)* select (i.e. make final decisions) on client-resource pairs, performed by the *Select Units*; *(ii)* commit the assignments, where assignments are recorded into the *cell array*, and external grants are generated; and *(iii)* clear the assignments, in which the assignment record is cleared in the *array*, and the external grants are also de-asserted.

The cell array in the new design has a somewhat similar structure and role as the baseline approach. However, the new cell array is much simpler, since select operations are extracted out to a linear number of *Select Units*, while the baseline design folds select and recording of commits into each cell in the quadratic array. In addition, unlike the original design, the array now allows *concurrent writes*: each cell has a dedicated channel for its own write operation, hence the array can accommodate multiple parallel threads. In contrast, in the baseline design, the array has a single write access shared among all cells, hence writes are fully serialized.

The *Grant Generators* have the same role and design as those in the baseline approach. They receive the assignment record from the *array* and generate grants to external client and resource

channels. Whenever an assignment is recorded in the array, a pair of grants for the corresponding client and resource channels is asserted simultaneously. On the other hand, the pair of grants is de-asserted when the assignment is cleared.

7.6.2 Mapping Proposed Pipeline to the Design

The new static HC pipeline is used in multiple locations in the overall design, with variations on the basic stucture. While two of the three major units in the structure, *Client Winner Serializer* and *Resource Winner Serializer*, extensively apply the new pipeline technique, the *Assignment Unit* is largely a parallel unit with no pipelining. The discussion now focuses on the three-stage pipeline within the *Client Winner Serializer*, as an example. The other serializer has similar structure.

In the pipeline, the middle and right stages largely make direct use of the new static HC pipeline, while the left stage is a special case, interfacing to the clients, where a simpler protocol can be used.

The left stage is effectively a parallel to series protocol converter: its pipeline register, *Masking/De-Coupling Unit*, interfaces between the slower external handshaking operation on the multi-channel parallel client interface (i.e. which maintain persistent access to a resource during the *entire* utilization phase) and the fast handshaking operation required with the serial N-way arbiter of the stage. Fundamentally, each client channel interface has separate control. A conservative communication protocol is selected for each control slice of this stage, with two backwards synchronization arcs, since high-throughput is not required for the interface with each client channel. This protocol is known as Williams' *PS0* protocol [166, 239].

The middle stage is a simple *Pipeline Register*. It directly uses a copy of the static HC pipeline stage, with a small modification. In particular, since the left stage uses a *PS0* protocol, the left acknowledgment from the middle stage provides two backwards synchronization arcs, with simplified control. As a result, the left CD of the middle stage is eliminated.

The right stage is not a simple operator, but rather a multi-token circular FIFO. It is fundamentally a static HC stage, but with multiple output channels, which allow parallel access to the *Assignment Unit*. Therefore, the static HC control is basically copied for each FIFO slot, with partially merged control.

7.6.3 Operation

The operation of the proposed multi-resource arbiter is now illustrated by two scenarios: *(i)* a friendly case to show the basic operation, where there is only a single client request and a single available resource; *(ii)* a bursty case where all client requests arrive within a small time window while all resources are available. The latter scenario shows how the client-resource pairs can be selected rapidly in sequence and assignments are completed in parallel with overlap.

Friendly Scenario

Without loss of generality, the client and resource requests are assumed to arrive both on channel #0. Initially, all handshaking signals are de-asserted low, the two single-resource arbiters are reset, and the two queues are empty. The cell array stores 0s in all cells.

Assume the resource request arrives first, with asserting *resource_req0* high. It is immediately forwarded by the default-transparent *Masking/De-Coupling Unit*, and then requests the M-way arbiter (i.e. *client_r0* asserted high) and wins the arbitration (i.e. *client_g0* asserted high). The *Pipeline Register* then captures the winner, and asserts its acknowledge high back to the *Masking/De-Coupling Unit*, which in turn releases the M-way arbiter (i.e. enters reset phase). In parallel, the winner is also forwarded to, and stored in, the *Resource Queue*. The queue immediately outputs the winner to the *Assignment Unit* (by asserting a valid output data high) and the *Select Unit0* is partially enabled. The queue also asserts its acknowledgment to the *Pipeline Register*. Eventually, a client request arrives, which follows a similar procedure, and the client queue likewise outputs the winner on its top channel.

Next, the select and commit operations are performed. In particular, *Select Unit0* synchronizes the client and resource winners, and generates the winner pair's ID (i.e. *select_pair0* ID is asserted and becomes valid), which is sent to the *Write Destination Decoder*. The *decoder* translates the pair into the unique output channel to the array, which is attached to the corresponding assignment cell. A 'write request' is immediately asserted high to *Cell00*. The assignment is then recorded in the array by writing a 1 into the cell. At this point, all activated channels in the *Assignment Unit* already have asserted their requests high without receiving an acknowledgment (i.e. *client-/resource-winner0* channels, *select-pair0* channel, and the input channel to *cell00*).

After completing the cell write, two threads occur concurrently: *(i) cell00* broadcasts the new

stored value to both *Grant Generators*, resulting in external grants asserted high for both active client and resource channels; and *(ii)* an acknowledgment is sent backwards from the assignment cell to the two queues (via the decoder and the select units). At this point, all related channels indicated above complete the active phase of their 4-phase handshaking. The occupied slots in both queues are first cleared (i.e. freed up); the *select unit* and the *decoder* are then reset in serial order.

Finally, after an arbitrarily long time, the client and resource each finish their utilization phases. The assignment is then released. The external requests, *client_req0* and *resource_req0*, are both de-asserted low, which can occur in any order. These two de-asserted requests are directly forwarded to *Cell00*, which is cleared (i.e. 0 is written). The two external grants are then de-asserted accordingly.

Bursty Scenario

In this more complex scenario, as there are four client requests and three available resources, three assignments are completed. Each assignment is similar to the friendly case, but with significant overlap of operation.

First, the three resource requests arrive in any order, informing their availability, with no client requests presented. These three resources are stored into the resource queue, loosely depending on their arrival time. They are placed onto different output channels of the FIFO, in order from top to bottom, and in turn partially enable the three *Select Units*. Next, four client requests arrive almost simultaneously. Similar to the resources, the first three client winners are stored into the client queue, quickly one after the other, and sent out on different output channels, while the fourth must wait at the input of the queue. As soon as each client winner appears on the output of the queue, the corresponding *Select Unit* is fully enabled and the selection proceeds as in the friendly case. The three selections occurs in sequence, since the *Select Units* receive requests in order; however, the hardware units are actually decoupled, hence in practice the operations may in fact overlap.

Finally, commit operations occur for these multiple assignments. Unlike the baseline, the design allows fully-parallel access to the array, and the commits can all occur concurrently, including cell writes and external grant generation. Each separate thread observes the same protocol as for the friendly case.

7.6.4 Details for Sub-Modules

This sub-section provides further implementation details of each sub-module shown in the design structure.

7.6.4.1 Masking/De-Coupling Unit

Fig. 7.7 introduces the *Masking/De-Coupling Unit* for the client side. The unit is a decoupled and customized register. It decouples the long-term handshaking protocol with the client interface, from the rapid handshaking and release of the arbiter. The design takes multiple client requests as inputs from left, and has a single output channel to the right. The arbiter can receive multiple competing input requests, as desired.

A detailed individual element is shown in Fig. 7.7(b) for one of the client input channels. The element is initially transparent. After the client request arrives and passes through, the latch becomes opaque. When the output request wins the mutex and the winner being stored by the *Pipeline Register*, the element is acknowledged, and forces a reset on its own output, thereby releasing the arbiter and allowing another client to win. There are two pre-conditions for the element to become transparent again (i.e., evaluate phase), which occur in order: *(i)* the right acknowledgment is de-asserted when the corresponding winner is safely stored into the FIFO; and *(ii)* the external client request goes low, after the resource is assigned, used, and finally released.

7.6.4.2 Pipeline Register

The *Pipeline Register*, as shown in Fig. 7.8, is a normal static HC register with simplification. As discussed before, the left part of the control and left CD are deleted, and a more conservative PS0-style acknowledgment is sent to the left.

7.6.4.3 Client and Resource Winner Queues

The two queues are implemented as circular FIFOs to store the client and resource winners, respectively. The FIFO is serial-in-parallel-out, allowing data immediately to appear on the output channel after it is stored into the queue. It is fundamentally a static HC stage, interfacing with multiple output channels in a round-robin fashion. It is also the key component to create parallel threads for

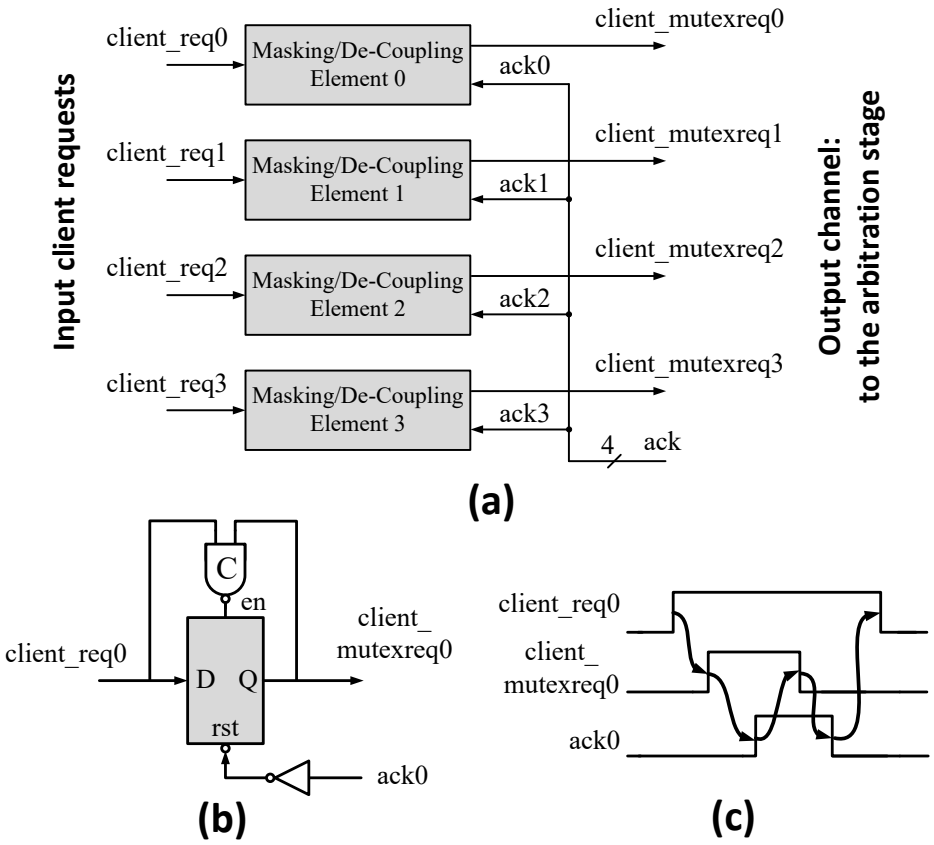


Figure 7.7: Masking/de-coupling unit:

(a) structure; (b) implementation; (c) timing diagram

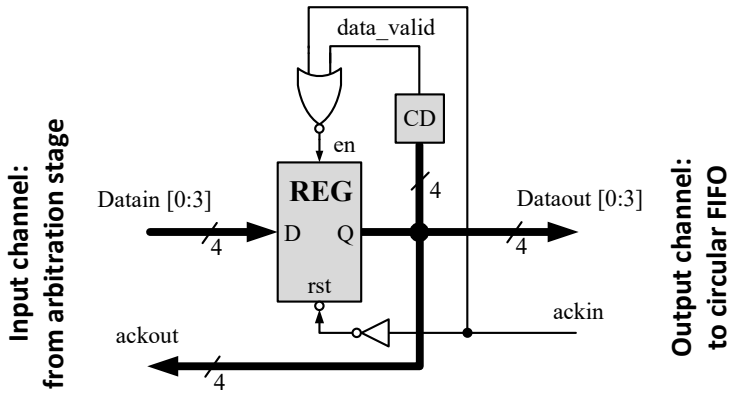


Figure 7.8: Pipeline register

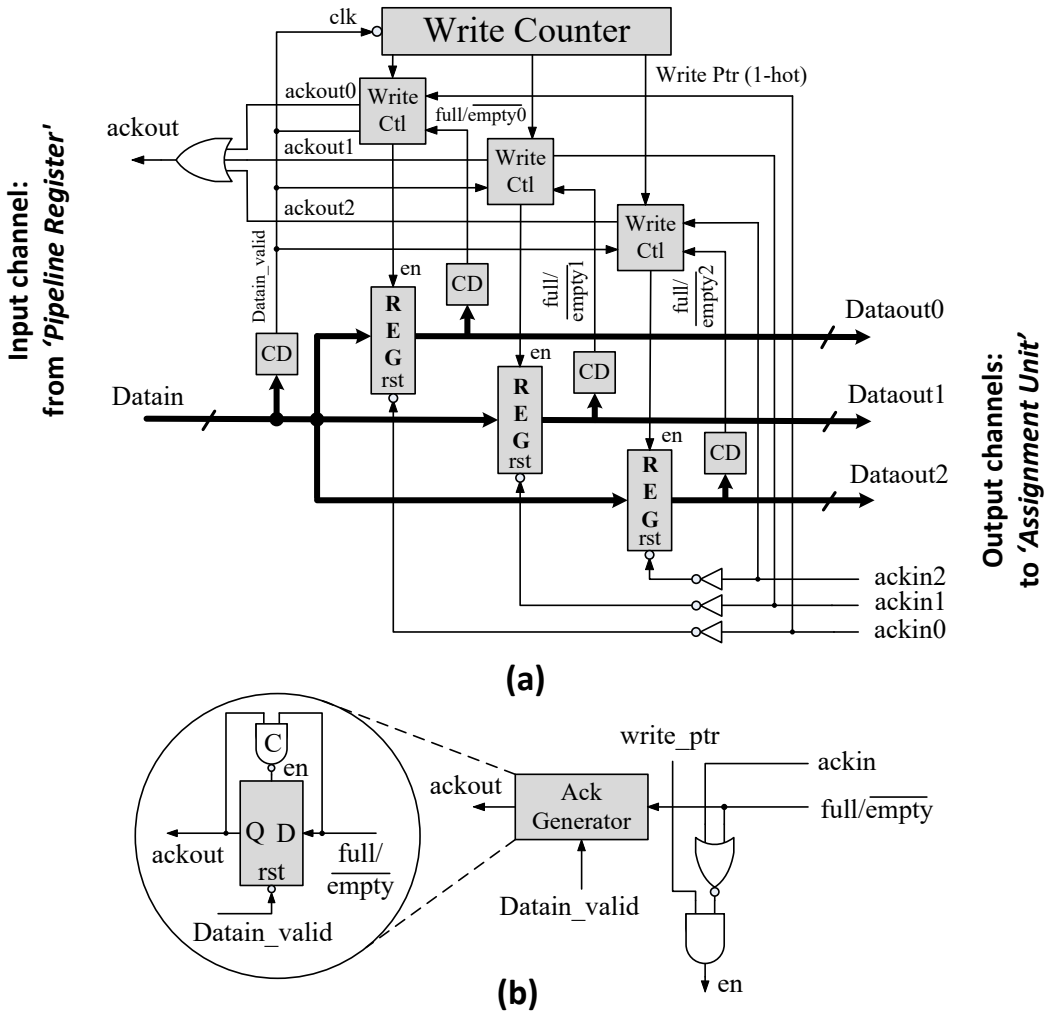


Figure 7.9: Client winner queue: (a) structure; (b) implementation for 'write control'

client-resource selections.

The structure of the queue is shown in Fig. 7.9. The design is modified from a non-parallel-out transition-signaling circular FIFO proposed in [80]. The data registers, implemented by D-latches, are storage elements. The *Write Counter* outputs a round-robin 1-hot pointer to select an active register to write.

A write operation starts when a new data arrives at the input channel. The data register is default-open, and the data immediately passes through. The state of the slot is changed to 'full', and the register is closed with an acknowledgment asserted to the left. The input channel then completes

the reset phase of the four-phase protocol without other synchronization, during which the write pointer is moved to the next slot. The register will open again when the following two conditions hold: *(i)* the pointer selects the slot again; *(ii)* the data is read out.

A read operation simply starts as soon as a new data is stored into the FIFO. A read request is put onto on the output channel. When the entire four-phase handshaking is completed on the output channel, the data is read out. The state of slot is also changed back to 'empty'.

7.6.4.4 Select Unit

Each *Select Unit* synchronizes a pair of client and resource winners and makes the final selection of the pair. The unit actually has no logic inside: the two input winner IDs (each using a 1-hot DI code) are concatenated to form a selected-pair ID, which is a new DI code, and sent on the output channel. The acknowledgment input is simply forked and broadcast to the two serializers.

7.6.4.5 Write Destination Decoder

The detailed implementation of the *Write Destination Decoder* is shown in Fig. 7.10. The unit translates each selection result to an output channel activation. Whenever a selection result arrives at any of the three input channels, a write request is immediately placed onto the associated output channel for the pair. The unit allows up to three parallel decoding operations. Decoded cell destinations never overlap, since the pending clients are never paired to the same resource, or vice versa. In practice, however, decoding is very fast, and thus each request is completed without overlapping.

The unit contains twelve *Req Generators* (one for each output channel) and three *Ack Generators* (one for each input channel), and effectively serves as a crossbar. A *Req Generator* detects a distinct client-resource pair, which can arrive from any of the input channels. The *Req Generator* also routes the acknowledgment from the output channel back to the source input channel. Each *Ack Generator*, in turn, merges those directed acknowledgments from all *Req Generators*.

7.6.4.6 Individual Cell of the Array

An individual cell, shown in Fig. 7.11, is a single-bit memory, and stores the 'assignment state' for a particular client-resource pair. The unit receives a four-phase input channel from the *Decoder*, and two direct inputs of the entire arbiter design: the external client and resource requests, which

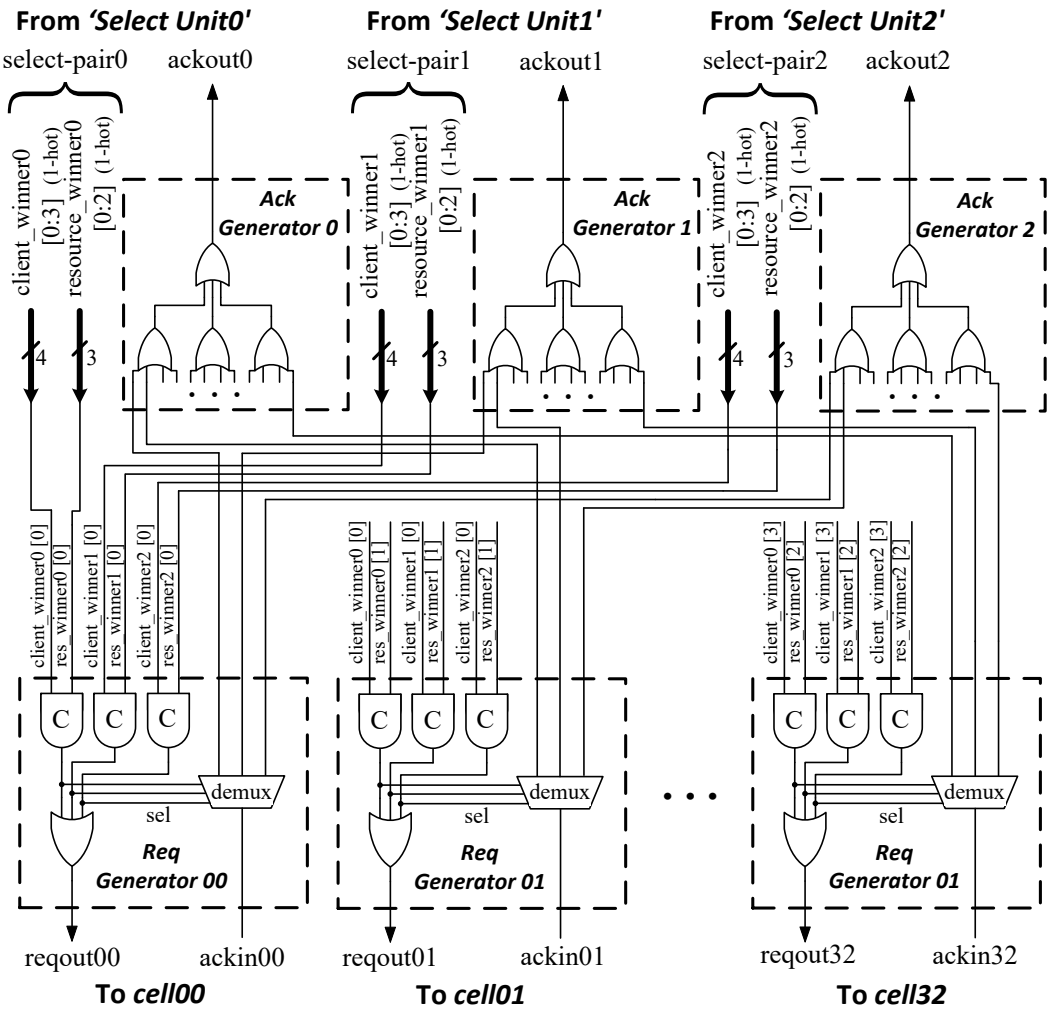


Figure 7.10: Write destination decoder

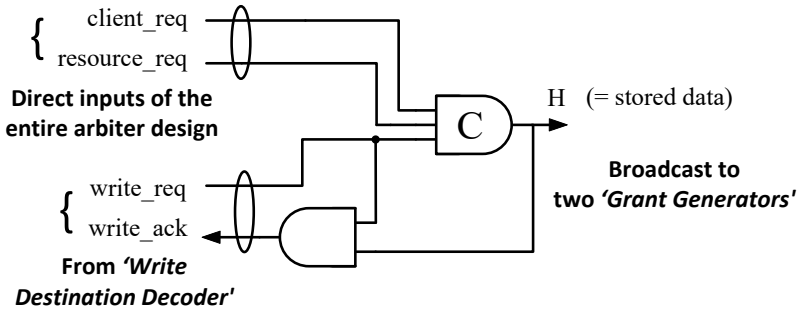


Figure 7.11: Individual cell implementation

are associated with the assignment pair. The output is the single-bit data stored by the cell. The cell is considerably simplified when compared to the baseline design, since the selection operation is extracted out, and blocking capability is no longer needed and therefore deleted.

There are only two cell operations, 'write' and 'clear', which must strictly alternate. In a 'write', the assignment is recorded by writing a 1 into the cell. The client and resource requests are first asserted high in any order, followed by a write request asserted high. The cell is then written, and acknowledges the decoder unit. The cell value is also broadcast to both grant generators to grant the active external client and resource channels. Finally, the input channel completes the reset phase of the handshaking protocol with the decoder. On contrary, a 'clear' operation writes a 0 into the cell. The event occurs when the external client and resource channels each release the assignment. The two external release requests, i.e. client/resource requests de-asserted, communicate directly with the cell, which is then cleared.

7.7 Timing Analysis

The proposed asynchronous multi-resource arbiter involves a few simple pipeline-related timing constraints. In contrast, the *Assignment Unit* has entirely QDI[1] operation, with largely a sequential protocol on parallel threads. Since the left and right pipelined units basically use a static HC protocol, most timing constraints are directly transformed from those in HC dynamic pipeline [202]. Also, timing constraints on the N-way arbiters are already listed in [148]. The baseline approach, presented in [85], also has several one-sided timing constraints; however, these are not explicitly identified in this chapter.

The new design has three pipeline stages. Two of them, the linear *Pipeline Register* and a *circular FIFO*, largely operate as HC style, and have similar timing constraints. The *Masking/De-Coupling Unit* is a non-standard pipeline element, and operates differently. The timing constraints of the *Pipeline Register* and the *circular FIFO* will be discussed, while the constraints of *Masking/De-Coupling Unit*, which are similar and much easier to satisfy in practice, will be omitted.

[1] *Quasi-delay insensitive (QDI)* is a type of robust asynchronous design style. QDI circuits operate correctly assuming arbitrary gate delays, but all wires at each fanout point must have roughly equal delays, i.e. an *isochronic fork* assumption [141, 166].

Timing constraints for *Pipeline Register*

(i) Hold time. After data passes through the pipeline register, there is a race between two events: (a) the closing of the latch to safely store the data, needs to occur earlier than (b) data reset on the input channel. Since (a) is a local operation for the register control only involving two logic gates, while (b) contains a round-trip communication with the left arbitration stage to release the arbiter, the constraint can typically be simply satisfied.

(ii) Register re-open time. After data goes through the pipeline register, there is another race between two events: (a) the data reset on the input channel, needs to occur earlier than (b) data is stored downstream and the current register is re-opened. Otherwise, a stale data item will be stored. In practice, event (b) requires a complete four-phase communication (i.e. 2 round-trips) with the FIFO stage, while (a) only has a fast arbiter reset operation, as in *(i)*. The timing constraint can also be simply satisfied.

Timing constraints for *circular FIFO*

This stage also includes the above two timing constraints, as well as an additional constraint.

(iii) Write pointer update. A race starts right after the input channel resets the data, between two events: (a) the write pointer is moved to the next slot, which needs to occur earlier than (b) new data arrival on the input channel, otherwise, the new data will be stored into the same FIFO slot. The constraint can be easily satisfied, because the pointer update involves only a local simple operation through the *Write Counter*, while a round-trip communication with the left stage is required for a new data to arrive.

7.8 Experimental Results

Detailed evaluations are now presented for the new asynchronous multi-resource arbiters. Results are obtained for two different sizes, 4×3 and 8×4. Each of the proposed arbiters is compared to the *baseline* non-pipelined design with the same size [85], in terms of area, performance and robustness. Both technology-mapped implementations are at pre-layout level, using the same standard cell library, for fair comparison.

These two design points are carefully selected. The 8×4 arbiter is substantially larger than the

4×3 one, with more complicated bookkeeping for concurrent assignments. Also, they are fundamentally different arbiter classes. The 4×3 arbiter has similar number of clients and resources, while the 8×4 arbiter has many clients with few resources. Finally, with two design points, a flavor of scaling trends can be observed.

7.8.1 Experimental Setup

All arbiter designs, the two *baseline* and two *new* designs, are mapped using the FreePDK 45nm Nangate library. Analog mutual exclusion elements (mutexes) are implemented with a combination of two NAND2 gates and four transistors [183]. The remaining parts of the designs use standard cells. The asynchronous sequential C-elements and aC-elements are implemented using combinational gates with feedback. Results are obtained using the Spectre simulator in the Cadence Virtuoso environment at a typical design corner with nominal temperature and supply voltage (1.0V, 27°C).

7.8.2 Simulation Results

Area Comparison

Pre-layout areas are compared for the *baseline* vs. *new* arbiters, at both design points. The final layout area is estimated by summing up the raw cell areas and dividing by a packing factor of 0.8. For 4×3 arbiters, the *baseline* design has an area of 398.3μm^2, while the *new* arbiter occupies 667.7μm^2. A 67.6% increase of area is observed for the 4×3 arbiter size. For 8×4 arbiters, the *baseline* and *new* designs have an area of 868.2μm^2 and 1271.5μm^2, respectively. The area overhead of the *new* approach becomes less: only 46.5%. However, for both design points, the absolute area overhead is quite small. As an application example – an asynchronous SDM NoC [208] –, the multi-resource arbiter is estimated to occupy one order of magnitude smaller area than the router in which it is used.[1]

More insight is gained by focusing on the cell array, which is the core component of both *baseline* and *new* designs. The NxM array is the only quadratic area contributor (i.e. growing with NxM). Interestingly, the new array has significantly smaller area than the baseline: 174.8 vs. 213.9

[1] According to the implementation in [105], a basic asynchronous router has an area of 5081.4 μm^2, while the *new* arbiter area is only 667.7 μm^2 using a similar technology library.

Table 7.1: Average latency comparison (ps)

		Latency					
		1st assignment	2nd assignment	3rd assignment	4th assignment	Average	Average improvement
4x3 Arbiter	*Baseline*	552	1110	1673	N/A	1112	12.8%
	New	544	968	1397	N/A	970	
8x4 Arbiter	*Baseline*	710	1558	2401	3229	1975	24.5%
	New	687	1217	1758	2305	1492	

μm^2, respectively, for the 4×3 arbiters; and 456.0 vs. 570.3 μm^2, respectively, for the 8×4 arbiters. In order not to overstate the benefits, the array area of the *new* design also includes its periphery (i.e. the decoder). This area improvement is stable: 18.2% (4×3) and 20.0% (8×4). Since the remaining components in *baseline* and *new* arbiters grow only linearly in area, a smaller array provides better scalability for larger arbiter sizes, as the size of the array will eventually dominate the entire design area. Therefore, it is expected that the overall area overhead of the *new* approach will become even smaller beyond the 8×4 design point.

Zero-Load Latency

In order to obtain zero-load latency results, each arbiter is set up with no client requests while all resources are available. Latency is then calculated from the arrival of a client request to the assertion of the client grant. Comparable results are observed for *baseline* and *new* approaches at both design points. For 4×3 arbiters, the *baseline* has a latency of 555 ps, while the *new* arbiter has a result of 537 ps. For 8×4 arbiters, the latencies are 712 ps and 689 ps, for *baseline* and *new* designs, respectively. Interestingly, in spite of the overhead of fine-grain pipelining, the proposed approach obtains slightly better latency. This is largely due to the deletion of 'blocking' from the critical path of assignment commitment.

Average Latency: Bursty Client Requests

Table 7.1 shows the average latency when a burst of client requests arrives. The arbiter starts with no client requests and all available resources. All client requests then arrive within a small time frame, and three assignments are successively completed. The latency of each assignment is recorded and

Table 7.2: Cycle time comparisons (ps)

	4x3 Arbiter			8x4 Arbiter		
	Baseline	***New***	***Improvement***	***Baseline***	***New***	***Improvement***
Case #1: back-to-back assignments	591	452	23.5%	887	566	36.2%
Case #2: congestion scenario	789	742	6.0%	1059	1024	3.3%

the average is taken. The scenario examines an adversarial case where the arbiter needs to complete many resource allocations at its maximum rate.

The *new* 4×3 arbiter shows a moderate improvement of 12.8%, when compared to the 4×3 *baseline* design. When the arbiter size grows, a larger relative improvement of 24.5% is observed for the *new* 8×4 arbiter, when compared to its corresponding *baseline* approach. As shown in the table, for each of the design points, the *baseline* and *new* designs have similar latencies for the first assignment, because the first one is largely the same as the zero-load case, and the *baseline* and *new* arbiters have comparable zero-load performances. However, latency differences increase rapidly as more assignments being tested. The reason is that the *new* arbiter allows back-to-back assignments to overlap and be performed in parallel, while the *baseline* design only allow serial operation.

Cycle Time #1: Back-to-Back Assignments

Cycle times for back-to-back assignments are shown in Table 7.2. The benchmark shows the maximum rate for assignment completion, which can be directly translated to throughput performance. The same benchmark is performed as in the 'average latency' case. The metric is the time interval between two consecutive assignments being granted. The *new* arbiters show large improvement for both 4×3 and 8×4 design points: 23.5% and 36.2%, respectively, over the *baseline* designs. The result again validates the benefits of fine-grain pipelining.

Discussion: separating the benefits of pipelining vs. parallelism. Each of the two techniques used in the proposed design, *pipelining and parallelism*, has its own benefits. The *pipelining* technique decouples the winning of arbitration from the actual assignment operation. The client and resource winner pairs are therefore provided to the middle *assignment unit* at a very fast rate. Compared to the baseline design, when there are multiple active requests, the overhead of winner arbi-

tration is removed from the critical path. The *parallelism* technique further enables the *assignment unit* a faster rate of absorbing winner pairs. This is because each assignment operation is still much slower than the rate of winner pair generation. Each selected pair is now absorbed as soon as it arrives. By combining the two techniques, *pipelining and parallelism*, the throughput is now entirely determined by the rate of pair selection.

Cycle Time #2: Congestion Scenario

Finally, cycle time is obtained for a typical congestion scenario, also as shown in Table 7.2. During setup, all the clients and resources are busy: a maximum number of assignments (e.g. three in 4×3 arbiters and four in 8×4 arbiters) are already completed, and an extra client is waiting. The time is computed from when an assignment is released to when the same resource is allocated to the waiting client. The *new* arbiters show slightly better cycle time, by 6.0% and 3.3%, respectively, for two different design points, 4×3 and 8×4. It is understandable, because the scenario only involves one assignment, and effectively is a variant of a zero-load latency scenario, while the proposed pipeline technique requires multiple overlapping operations to exhibit its benefits.

The 'congestion scenario' actually has a dual case: a single active client channel with back-to-back sequential requests, and all resources are available. In this case, we also expect similar and no worse cycle time results for the *new* approach over the *baseline*.

Robustness: Grant-Overlapping Problem

A robustness challenge for arbiters has been identified as the grant-overlapping (GO) problem [148], where a new grant may be illegally asserted before the previous grant has been de-asserted. Such behavior is a specification violation, though in some environments it can be safely tolerated. The problem, previously defined for single-resource arbiters, can be simply extended to multi-resource designs, for each individual resource: where two distinct grants can briefly overlap with identical resource ID. However, GO exists for neither the *baseline* nor the *new* design. In both arbiters, a resource needs to proactively inform the arbiter its availability every time after it is used and released. This protocol thus provides a gap between two usages for the same resource and prevents the GO problem.

7.8.3 Summary and Discussion of Scaling Trends

Overall, the *new* arbiter has significant performance benefits over the *baseline*. As a highlight, when the arbiter size grows from 4×3 to 8×4, the cycle time improvement for processing back-to-back assignments increases from 23.5% to 36.2%. The performance benefit of the *new* approach comes from two factors. First, the two pipelined serializers (client and resource) allow high-rate back-to-back arbitration of the input streams, decoupled from the operation of the assignment unit, unlike the *baseline*. Second, the assignment unit is instrumented for parallel (i.e. overlapping) operation, hence it can process multiple back-to-back assignments simultaneously, also unlike the *baseline*. Interestingly, the performance improvement increases with larger number of clients and resources, since the larger arbiters and assignment unit in the *baseline* become a greater serial bottleneck, while the *new* design maintains decoupled pipelined arbitration and concurrent assignment.

In terms of area, the *new* arbiter has moderate overhead compared to *baseline*: 67.6% (4×3) and 46.5% (8×4). However, as indicated earlier, the absolute area of the arbiter is quite small: only 403.3 μm^2 for the larger 8×4 size. In addition, it typically occupies only a small fraction of area of a NoC switch. In terms of scaling trends, the quadratic array is always smaller in the *new* design, and its size starts to dominate in larger arbiters (see earlier), leading to continued area overhead reduction as the size increases. These encouraging trends strongly indicate that beyond the 8×4 design point, the *new* arbiter will have a even better performance gain with a smaller overhead. Also, the *new* approach is overall suitable for a wide range of arbiter classes.

7.9 Conclusions and Future Work

The chapter introduces a new asynchronous multi-resource arbiter. The proposed design targets high-throughput by using a fine-grain pipeline protocol. The client-resource pairs are now selected rapidly in sequence, and multiple back-to-back assignments can largely overlap. The proposed arbiter is simple and highly scalable, which can be simply extended to a design with arbitrary numbers of clients and resources. It can be directly used in existing asynchronous NoC's for performance optimization. In addition, the well-structured stand-alone design promises its future use in SDM NoC's, which opens up new NoC directions. In addition, a new static four-phase asynchronous pipeline is introduced, with only a single backwards synchronization arc per cycle, and has been

successfully incorporated into the microarchitecture. The results are obtained for two new arbiters with different sizes, 4×3 and 8×4. Large improvements in throughput are observed at both design points, when the new design is compared to the corresponding baseline serial approach.

In terms of future work, the proposed asynchronous multi-resource arbiter will be deployed in real asynchronous NoC's, either for VC allocation or SDM area. Also, it will be integrated and tested in a variety of other applications, e.g. asynchronous Multiple-Input-Multiple-Output (MIMO) queues and asynchronous RAID storage systems, to prove its high-quality as a stand-alone design.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis presented a set of design solutions for performance optimization and evaluation of asynchronous on-chip interconnection networks, in order to significantly advance the state-of-art and prove the viability of asynchronous and GALS networks-on-chip, for the use of high-performance and low-energy systems. The contributions are instantiated in three ways. First, practical acceleration techniques are proposed to focus on optimizing the system latency, in order to break through the latency bottleneck in the memory interfaces of many on-chip parallel processors. While these novel acceleration technologies lift the network performance considerably, the designs are very cost efficient in terms of both area and power. Second, an advanced high-performance and low-power asynchronous router is instantiated and validated using a leading industrial technology library, in collaboration with AMD Research. And for the first time, an asynchronous router design is compared to a commercial energy-efficient synchronous NoC baseline in an advanced technology, i.e. 14nm FinFET. Finally, a high-performance multi-resource asynchronous arbiter design is developed. This small but important component can be directly used in existing asynchronous NoC's for performance optimization. In addition, the well-structured stand-alone design promises its future use in opening up new NoC directions.

Latency acceleration for asynchronous NoC's. A new concept, named a *monitoring network*,

is introduced. Monitoring networks are lightweight shadow networks used for fast-forwarding anticipated traffic information. The routers are notified with incoming traffic in the near future, and therefore allowed to initiate and perform arbitration and channel allocation in advance. Corresponding NoC designs are proposed to implement the protocol. The technique is successfully applied to two network topologies which belong to two different categories – a *mesh-of-trees (MoT) variant* structure and a *2D-mesh* structure. The solution has only small overhead, without creating an extra network plane; nor extra VCs are needed. Considerable and stable latency improvements are observed in all kinds of traffic patterns, with moderate throughput gains.

An industrial instantiation for a high-performance and low-power asynchronous router. The work was completed and in collaboration with AMD Research, which provides an industrial implementation of an asynchronous NoC router in a leading technology. The proposed router integrates several advanced techniques, including a low-latency circular FIFO for buffer design, and a novel end-to-end 'lazy' credit-based virtual channel (VC) flow control. This asynchronous router is then compared to an AMD commercial synchronous router using a realistic 14nm FinFET library. This is the first such comparison, to the best of our knowledge, for an asynchronous router vs. an industrial synchronous baseline using an advanced technology library. The synchronous design is used in recent high-end AMD processors and graphic products. In addition, a semi-automated design flow is created for the asynchronous router design. In particular, industrial tools are used for place-and-route (P&R) and design validation. These tools are modified from a standard synchronous design flow and therefore open real future opportunities for industrial asynchronous NoC designs. Significant benefits in terms of area, latency and power were obtained.

A pipelined high-throughput asynchronous multi-resource arbiter. Finally, a new fine-grain pipelined asynchronous multi-resource arbiter is proposed. This is an understudied but important research area. Multi-resource arbitration is used in existing asynchronous NoC's for VC allocation. Also, it serves as a key component for a potential future NoC direction – spatial-division multiplexing (SDM) NoC's. Besides the NoC applications, it can be treated as a great stand-alone design for future large-scale parallel systems. The new arbiter design aims for high throughput while retains a low latency. The allocation of a resource to a client is divided into several steps. Multiple successive client-resource pairs can be selected rapidly in sequence, and the completion of the assignments can overlap in parallel.

8.2 Future Work

There are a few potential areas for future work.

Post-layout evaluation and fabrication. Most of the designs in this thesis, including the early arbitration NoC's in Chapters 4 and 5, as well as the arbiter design in Chapter 7 are implemented and evaluated only at pre-layout level. In order for more accurate results and comparisons for the performance and power of the designs, full layout and possibly fabrication are required.

Build complete GALS systems. This thesis only focuses on the NoC itself. The designs are either for the interconnection networks only (Sections 4, and 5), or for individual routers (Section 6) and the router's sub-components (Section 7). The next step is to use these networks and components as the communication fabric to build complete GALS systems. These GALS systems can be high-performance chip multiprocessors (CMP's) or lower-power embedded systems. Mixed-timing interfaces typically need to be implemented and added to connect the synchronous cores with the network-on-chip. In the performance comparison, the synchronization overhead will be thereby taken into account, and results are closer to the reality.

Simulations on real parallel benchmark suites. In the near future, after a complete CMP system is built, real benchmarks suites for parallel computing, such as PARSEC and SPLASH-2, can be used for testing instead of synthetic benchmarks. These benchmarks are much closer to real-world large-scale multi-threaded commercial applications, and therefore more accurate performance and power numbers can be obtained.

Synthesis flow integration. In the next step, an automated design flow for NoC synthesis and verification can be integrated. The lack of CAD tools is one of the key reasons to prevent asynchronous style designs from industrial consideration. Our next step is to optimize the design flow in Section 6, and integrate it with emerging tools [147], in order to bring our designs a step closer to the industry.

Practical applications for the proposed multi-resource arbiter. The proposed asynchronous multi-resource arbiter in Section 7 will be deployed in real asynchronous NoC's, either for VC allocation or SDM area. Also, it will be integrated and tested in a variety of other applications, e.g. asynchronous Multiple-Input-Multiple-Output (MIMO) queues and asynchronous RAID storage systems, to prove its high-quality as a stand-alone design.

Bibliography

[1] A.B. Abdallah. *Multicore systems on-chip: Practical software/hardware design*. Atlantis Press, 2013.

[2] A.K. Abousamra, R.G. Melhem, and A.K. Jones. Deja Vu switching for multiplane NoCs. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 11–18, 2012.

[3] N. Agarwal, T. Krishna, L.-S. Peh, and N.K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 33–42, 2009.

[4] V. Akella, N.H. Vaidya, and G.R. Redinbo. Asynchronous comparison-based decoders for delay-insensitive codes. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 47(7):802–811, 1998.

[5] F. Akopyan, J. Sawada, A. Cassidy, R.A.-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J.B. Kuang, R. Manohar, W.P. Risk, B. Jackson, and D.S. Modha. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, 2015.

[6] T. Amert, N. Otterness, M. Yang, J. Anderson, and F.D. Smith. GPU scheduling on the NVIDIA TX2: Hidden details revealed. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 1–12, 2017.

[7] R.M. Badia and J. Cortadella. High-level synthesis of asynchronous systems: Scheduling and process synchronization. In *Proceedings of European Conference on Design Automation (EDAC)*, pages 70–74, 1993.

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

[9] W.J. Bainbridge, W.B. Toms, D.A. Edwards, and S.B. Furber. Delay-insensitive point-to-point interconnect using M-of-N codes. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 132–140, 2003.

[10] A.O. Balkan, M.N. Horak, and U. Vishkin. Layout-accurate design and implementation of a high-throughput interconnection network for single-chip parallel processing. In *IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 21–28, 2007.

[11] A.O. Balkan, G. Qu, and U. Vishkin. An area-efficient high-throughput hybrid interconnection network for single-chip parallel processing. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 435–440, 2008.

[12] A. Bardsley and D.A. Edwards. The Balsa asynchronous circuit synthesis system. In *Forum on Design Languages*, 2000.

[13] P. Beerel, G.D. Dimou, and A.M Lines. Proteus: An ASIC flow for GHz asynchronous designs. *IEEE Design and Test*, 28(5):38–51, 2011.

[14] P. Beerel and T.H.-Y. Meng. Automatic gate-level synthesis of speed independent circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 581–587, 1992.

[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, S. Miermont, and P. Vivet. Dynamic voltage and frequency scaling architecture for units integration within a GALS NoC. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 129–138, 2008.

[17] L. Benini, E. Flamand, D. Fuin, and D. Melpignano. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 983–987, 2012.

[18] L. Benini and G. De Micheli. Networks on chip: A new SoC paradigm. *IEEE Transactions on Computers*, 35(1):70–78, 2002.

[19] I. Benko and J. Ebergen. Delay-insensitive solutions to the committee problem. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 228–237, 1994.

[20] K. Bergman, L.P. Carloni, A. Biberman, J. Chan, and G. Hendry. *Photonic network-on-chip design*. Springer, 2014.

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

[22] D. Bertozzi, L. Benini, and G. De Micheli. Error control schemes for on-chip communication links: The energy-reliability tradeoff. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):818–831, 2005.

[23] E. Bezati, S.C. Brunet, M. Mattavelli, and J.W. Janneck. Coarse grain clock gating of streaming applications in programmable logic implementations. In *Proceedings of the IEEE Electronic System Level Synthesis Conference (ESLsyn)*, pages 1–6, 2014.

[24] K. Bhardwaj, W. Jiang, and S.M. Nowick. Achieving lightweight multicast in asynchronous NoCs using a continuous-time multi-way read buffer. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 1–8, 2017.

[25] G. Birtwistle and A. Davis. *Asynchronous Digital Circuit Design*. Workshops in Computing, Springer-Verlag, 1995.

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

[27] T. Bjerregaard and J. Sparsoe. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1226–1231, 2005.

[28] P. Bogdan, R. Marculescu, S. Jain, and R.T. Gavila. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platform under highly variable workloads. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 35–42, 2012.

[29] P. Bogdan, R. Marculescu, S. Jain, and R.T. Gavila. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 21–28, 2012.

[30] P. Bogdan, P.P. Pande, M.H. Amrouch, and J.H. Shafique. Power and thermal management in massive multicore chips: Theoretical foundation meets architectural innovation and resource allocation. In *Proceedings of the IEEE International Conference on Compliers, Architectures and Synthesis of Embedded Systems (CASES)*, pages 1–2, 2016.

[31] E. Brunvand. *Translating concurrent communicating programs into asynchronous circuits*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[32] M. Cannizzaro, W. Jiang, and S.M. Nowick. Practical completion detection for 2-of-N delay-insensitive codes. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 151–158, 2010.

[33] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 162–163, 2015.

[34] D.M. Chapiro. *Globally-asynchronous locally-synchronous systems*. PhD thesis, Department of Computer Science, Stanford University, 1984.

[35] G. Chen, M.A. Anders, H. Kaul, S.K. Satpathy, S.K. Mathew, S.K. Hsu, A. Agarwal, R.K. Krishnamurthy, V. De, and S. Borkar. A 340 mV-to-0.9 V 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16x16 network-on-chip in 22 nm Tri-Gate CMOS. *IEEE Journal of Solid-State Circuits*, 50(1):1444–1454, 2015.

[36] Y. Chen. *Digital signal processing with signal-derived timing: analysis and implementation*. PhD thesis, Department of Electrical Engineering, Columbia University, 2017.

[37] Y. Chen, X. Zhang, Y. Lian, R. Manohar, and Y. Tsividis. A continuous-time digital IIR filter with signal-derived timing, agile power dissipation and synchronous output. In *Proceedings of the IEEE Symposium on VLSI Circuits (VLSIC)*, pages C272–C273, 2017.

[38] Y. Chen, X. Zhang, Y. Lian, R. Manohar, and Y. Tsividis. A continuous-time digital IIR filter with signal-derived timing and fully agile power consumption. *IEEE Journal of Solid-State Circuits*, 53(2):418–430, 2018.

[39] W. Choi, K. Duraisamy, R.G. Kim, J.R. Doppa, P.P. Pande, R. Marculescu, and D. Marculescu. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *Proceedings of the IEEE International Conference on Compliers, Architectures and Synthesis of Embedded Systems (CASES)*, pages 1–10, 2016.

[40] J.F. Christmann, E. Beigne, C. Condemine, N. Leblond, P. Vivet, G. Waltisperger, and J. Willemin. Bringing robustness and power efficiency to autonomous energy harvesting microsystems. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 62–71, 2010.

[41] W.A. Clark and C.E. Molnar. Macromodular computer systems. *Computers in Biomedical Research*, 4:45–85, 1974.

[42] W.S. Coates, J.K. Lexau, I.W. Jones, S.M. Fairbanks, and I.E. Sutherland. FLEETzero: An asynchronous switching experiment. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 173–182, 2001.

[43] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Complete state encoding based on the theory of regions. In *Proceedings of the IEEE International*

Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), pages 36–47, 1996.

[44] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: A tool for manipulating concurrent specification and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, 1997.

[45] J. Cortadella, A. Kondratyev, L. Lavagno, and C.P. Sotiriou. Desynchronization: Synthesis of asynchronous circuits from synchronous specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):1904–1921, 2006.

[46] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, Inc., 2003.

[47] W.J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):192–205, 1992.

[48] W.J. Dally and C.L. Seitz. The torus routing chip. *Distributed Computing*, 1(4):187–196, 1986.

[49] W.J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 684–689, 2001.

[50] S. Das, J.R. Doppa, P.P. Pande, and K. Chakrabarty. Energy-efficient and reliable 3D network-on-chip (NoC): Architectures and optimization algorithms. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, 2016.

[51] S. Das, A. Fan, K.-N. Chen, C.S. Tan, N. Checka, and R. Reif. Technology, performance, and computer-aided design of three-dimensional integrated circuits. In *Proceedings of the ACM International Symposium on Physical Design (ISPD)*, pages 108–115, 2004.

[52] I. David, R. Ginosar, and M. Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Transactions on on Computers*, 41(1):2–11, 1992.

[53] I. David, R. Ginosar, and M. Yoeli. Self-timed is self-checking. *Journal of Electronic Testing*, 6(2):219–228, 1995.

[54] R. David, P. Bogdan, R. Marculescu, and U. Ogras. Dynamic power management of voltage-frequency island partitioned networks-on-chip using Intel's single-chip cloud computer. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 1–2, 2011.

[55] M. Davies, A. Lines, J. Dama, A. Gravel, R. Southworth, G. Dimou, and P. Beerel. A 72-port 10G ethernet switch/router using quasi-delay-insensitive asynchronous design. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 103–104, 2014.

[56] A.L. Davis. The architecture and system method of DDM1: A recursively structured data driven machine. In *Proceedings of the ACM Annual Symposium on Computer Architecture (ISCA)*, pages 210–215, 1978.

[57] A.L. Davis and S.M. Nowick. *Asynchronous Circuit Design: Motivation, Background and Methods*. Chapter in *Asynchronous Digital Circuit Design*, G. Birtwistle and A. Davis editors, Springer-Verlag (Workshops in Computing Series), pages 1–49, 1995.

[58] M.E. Dean, T.E. Williams, and D.L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (LEDR). In *Proceedings of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI*, pages 55–70, 1991.

[59] S. Deb, K. Chang, X. Yu, S.P. Sah, M. Cosic, A. Ganguly, P.P Pande, B. Belzer, and D. Heo. Design of an energy-efficient CMOS-compatible NoC architecture with millimeter-wave wireless interconnects. *IEEE Transactions on Computers*, 62(12):2382–2396, 2012.

[60] S. Deb, A. Ganguly, P.P. Pande, B. Belzer, and D. Heo. Wireless NoC as interconnection backbone for multicore chips: Promises and challenges. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):228–239, 2012.

[61] Y. Deng and W. Maly. 2.5D system integration: A design driven system implementation schema. In *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 450–455, 2004.

[62] R. Dobkin, R. Ginosar, and I. Cidon. QNoC asynchronous router with dynamic virtual channel allocation. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, page 218, 2007.

[63] R. Dobkin, R. Ginosar, and A. Kolodny. QNoC asynchronous router. *Integration, the VLSI Journal*, pages 103–115, 2009.

[64] J. Ebergen, J. Gainsley, J. Lexau, and I. Sutherland. GasP control for domino circuits. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 12–22, 2005.

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

[66] V. Ekanayake, C. Kelly, and R. Manohar. An ultra low-power processor for sensor networks. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 27–36, 2004.

[67] G. Faldamis, W. Jiang, G. Gill, and S.M. Nowick. A low-latency asynchronous interconnection network with early arbitration resolution. In *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 329–336, 2014.

[68] K.M. Fant. *Logically Determined Design: Clockless System Design with NULL Convention Logic*. Wiley-Interscience, 2005.

[69] M. Ferretti and P.A. Beerel. High performance asynchronous design using single-track full-buffer standard cells. *IEEE Journal of Solid-State Circuits*, 41(6):1444–1454, 2006.

[70] R.M. Fuhrer, B. Lin, and S.M. Nowick. Symbolic hazard-free minimization and encoding of asynchronous finite state machines. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 604–611, 1995.

[71] R.M. Fuhrer, S.M. Nowick, and M. Theobald. *MINIMALIST: An Environment for the Synthesis, Verification and Testability of Burst-Mode Asynchronous Machines*. Technical report, Department of Computer Science, Columbia University, 1999.

[72] S.B. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 4(2):247–253, 1996.

[73] S.B. Furber, A. Efthymiou, and M. Singh. A power-efficient duplex communication systems. In *Alex Yakovlev and Reinder Nouta, editors, Asynchronous Interfaces: Tools Techniques, and Implementations*, pages 145–150, 2000.

[74] S.B. Furber, F. Galluppi, S. Temple, and L.A. Plana. The SpiNNaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[75] S.B. Furber, J.D. Garside, S. Temple, J. Liu, P. Day, and N.C. Paver. AMULET2e: An asynchronous embedded controller. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 290–299, 1997.

[76] S.B. Furber, D.R. Lester, L.A. Plana, J.D. Garside, E. Painkras, S. Temple, and A.D. Brown. Overview of the SpiNNaker sytem architecture. *IEEE Transactions on Computers*, 62(12):2454–2467, 2013.

[77] S.B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 11–16, 1996.

[78] J.D. Garside, W.J. Bainbridge, A. Bardsley, D.M. Clark, D.A. Edwards, S.B. Furber, J. Liu, D.W. Lloyd, S. Mohammadi, J.S. Pepper, O. Petlin, S. Temple, and J.V. Wood. AMULET3i – an asynchronous system-on-chip. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 162–175, 2000.

[79] D. Gebhardt, J. You, and K.S. Stevens. Comparing energy and latency of asynchronous and synchronous NoCs for embedded SoCs. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 115–122, 2010.

[80] A. Ghiribaldi, D. Bertozzi, and S.M. Nowick. A transition-signaling bundled data NoC switch architecture for cost-effective GALS multicore systems. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 332–337, 2013.

[81] G. Gill, A. Agiwal, M. Singh, F. Shi, and Y. Makris. Low-overhead testing of delay faults in high-speed asynchronous pipelines. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 46–56, 2006.

[82] G. Gill, S.S. Attarde, G. Lacourba, and S.M. Nowick. A low-latency adaptive asynchronous interconnection network using bi-modal router nodes. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 193–200, 2011.

[83] G. Gill and M. Singh. Automated microarchitectural exploration for achieving throughput targets in pipelined asynchronous systems. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 117–127, 2010.

[84] C.J. Glass and L.M. Ni. The turn model for adaptive routing. *Journal of the ACM*, 41(5):874–902, 1994.

[85] S. Golubcovs, D. Shang, F. Xia, A. Mokhov, and A. Yakovlev. Concurrent multiresource arbiter: Design and applications. *IEEE Transactions on Computers*, 62(1):31–44, 2013.

[86] S. Golubcovs, D. Shang F. Xia, A. Mokhov, and A. Yakovlev. Modular approach to multi-resource arbiter design. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 107–116, 2009.

[87] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal network on chip: Concepts, architectures and implements. *IEEE Design and Test of Computers*, 22(5):414–421, 2005.

[88] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S.W. Keckler, and D. Burger. On-chip interconnection networks of the TRIPS chip. *IEEE Micro*, 27(5):41–50, 2007.

[89] B. Grot, D. Hardy, P. Lotfi-Kamran, B. Falsafi, C. Nicopoulos, and Y. Sazeides. Optimizing data-center TCO with scale-out processors. *IEEE Micro*, 32(5):52–63, 2012.

[90] P. Guerrier and A. Greiner. A generic architecture for on-chip packet-switched interconnections. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 250–256, 2000.

[91] J. Hansen and M. Singh. A fast branch-and-bound approach to high-level synthesis of asynchronous systems. In *Proceedings of the IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*, pages 107–116, 2010.

[92] J. Hansen and M. Singh. Multi-token resource sharing for pipelined asynchronous systems. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1191–1196, 2012.

[93] M. Hayenga, N.E. Jerger, and M. Lipasti. SCARAB: A single cycle adaptive routing and bufferless network. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 244–254, 2009.

[94] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Oberg, M. Millberg, and D. Lindqvist. Network on chip: An architecture for billion transistor era. In *Proceedings of the IEEE Nordic Microelectronics Conference (NORCHIP)*, pages 1–8, 2000.

[95] R. Ho, K.W. Mai, and M.A. Horowitz. The future of wires. In *Proceedings of the IEEE*, pages 490–504, 2000.

[96] M.N. Horak, S.M. Nowick, M. Carlberg, and U. Vishkin. A low-overhead asynchronous interconnection network for GALS chip multiprocessors. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, pages 43–50, 2010.

[97] M.N. Horak, S.M. Nowick, M. Carlberg, and U. Vishkin. A low-overhead asynchronous interconnection network for GALS chip multiprocessors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4):494–507, 2011.

[98] J. Hu and R. Marculescu. DyAD: Smart routing for networks-on-chip. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 260–263, 2004.

[99] M. Imai and T. Yoneda. Duplicated execution method for NoC-based multiple processor systems with restricted private memories. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 463–471, 2011.

[100] M. Imai and T. Yoneda. Improving dependability and performance for fully asynchronous on-chip networks. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 65–76, 2011.

[101] S. Ishihara, M. Hariyama, and M. Kameyama. A low-power FPGA based on autonomous fine-grain power gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(8):1394–1406, 2011.

[102] H. Jacobson, E. Brunvand, G. Gopalakrishnan, and P. Kudva. High-level asynchronous system design using the ACK framework. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 1–11, 2000.

[103] N. Jiang, J. Balfour, D.U. Becker, B. Towles, W.J. Dally, G. Michelogiannakis, and J. Kim. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 86–96, 2013.

[104] W. Jiang, D. Bertozzi, , G. Miorandi, S.M. Nowick, W. Burleson, and G. An asynchronous NoC router in a 14nm FinFET library: Comparison to an industrial synchronous counterpart. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 732–733, 2017.

[105] W. Jiang, K. Bhardwaj, G. Lacourba, and S.M. Nowick. A lightweight early arbitration method for low-latency asynchronous 2D-mesh NoC's. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[106] W. Jiang and S.M. Nowick. A high-throughput asynchronous multi-resource arbiter using a pipelined assignment approach. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 1–9, 2017.

[107] W. Jiang and G. Sadowski. Self-timed router with virtual channel control. US Patent Application No. 15/085,783 (2016).

[108] M.B. Josephs and J.T. Yantchev. CMOS design of the tree arbiter element. *IEEE Transactions on VLSI Systems*, 4(4):472–476, 1996.

[109] J.W. Joyner, P. Zarkesh-Ha, and J.D. Meindl. A stochastic global net-length distribution for a three-dimensional system-on-a-chip (3D-SoC). In *Proceedings of the IEEE International ASIC/SOC Conference*, pages 147–151, 2001.

[110] A.B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A power-area simulator for interconnection networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(1):191–196, 2012.

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

[112] E. Kasapaki and J. Sparso. Argo: A time-elastic time-division-multiplexed NoC using asynchronous routers. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 45–52, 2014.

[113] A. Khoche and E. Brunvand. Testing micropipelines. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 239–246, 1994.

[114] G. Kim, K. Lee, Y. Kim, S. Park, I. Hong, K. Bong, and H.-J. Yoo. A 1.22 TOPS and 1.52 mW/MHz augmented reality multicore processor wih neural network NoC for HMD applications. *IEEE Journal of Solid-State Circuits*, 50(1):113–124, 2015.

[115] J. Kim, J. Balfour, and W. Dally. Flatten butterfly topology for on-chip networks. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 37–40, 2007.

[116] Y. Kim, D. Shin, J. Lee, Y. Lee, and H.-J. Yoo. A 0.55 V 1.1 mW artificial intelligence processor with on-chip PVT compensation for autonomous mobile robots. *IEEE Transactions on Circuits and Systems*, PP(99):1–14, 2017.

[117] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, 2006.

[118] A. Kondratyev, L. Sorensen, and A. Streich. Testing of asynchronous circuits by 'inappropriate' means. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 171–180, 2002.

[119] T. Krishna, C.-H. Chen, W.-H. Kwon, and L.-S. Peh. SMART: Single-cycle multihop traversals over a shared network on chip. *IEEE Micro*, 34(3):43–56, 2014.

[120] M. Krstic, E. Grass, F.K. Gurkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design and Test*, 24(5):430–441, 2007.

[121] A. Kumar, P. Kundu, L.-S Peh, A.P. Singh, and N.K. Jha. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 63–70, 2007.

[122] A. Kumar, L.-S Peh, P. Kundu, and N.K. Jha. Express virtual channels: Towards the ideal interconnection fabric. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 150–161, 2007.

[123] D.S. Kung. Hazard-non-increasing gate-level optimization algorithms. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 631–634, 1992.

[124] J. Lbanez-Guzman, C. Laugier, J.D. Yoder, and S. Thrun. *Autonomous driving: Context and state-of-the-art*. Springer, 2012.

[125] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes*. Morgan Kaufmann Publishers, 1992.

[126] 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 IEEE/ACM/IFIP International Conference on the Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 81–86, 2005.

[127] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor. Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip. *IEEE Transactions on Computers*, 57(9):1182–1195, 2008.

[128] A. Leroy, D. Milojevic, D. Verkest, F. Robert, and F. Catthoor. Concepts and implementation of spatial division multiplexing for guaranteed throughput in networks-on-chip. *IEEE Transactions on on Computers*, 57(9):1182–1195, 2008.

[129] M. Lewis, J. Garside, and L. Brackenbury. Reconfigurable latch controllers for low power asynchronous circuits. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 27–35, 1999.

[130] M. Li, Q.-A. Zeng, and W-.B. Jone. DyXY: A proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 849–852, 2006.

[131] Z. Li, J. Wu, L. Shang, R.P. Dick, and Y. Sun. Latency criticality aware on-chip communication. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1052–1057, 2009.

[132] M. Ligthart, K. Fant, R. Smith, A. Taubin, and A. Kondratyev. Asynchronous design using commercial HDL synthesis tools. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 1–12, 2000.

[133] A. Lines. Asynchronous interconnect for synchronous SoC design. *IEEE Micro*, 24(1):32–41, 2004.

[134] A.M. Lines. *Pipelined Asynchronous Circuits*. Technical report, Department of Computer Science, California Institute of Technology, 1998.

[135] T.-T. Liu, L.P. Alarcon, M.D. Pierson, and J.M. Rabaey. Asynchronous computing in sense amplifier-based pass transistor logic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(7):883–892, 2009.

[136] D.W. Lloyd and J.D. Garside. A practical comparison of asynchronous design styles. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 36–45, 2001.

[137] J. Luo, A. Elantably, V.D. Pham, C. Killian, D. Chillet, S. Le Beux, O. Sentieys, and I. O'Connor. Performance and energy aware wavelength allocation on ring-based WDM 3D optical NoC. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1372–1375, 2017.

[138] R. Marculescu, U.Y. Ogras, L.-S. Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):3–21, 2009.

[139] A.J. Martin. *Programming in VLSI: from Communicating Processes to Delay-Insensitive Circuits*. Technical report, Department of Computer Science, California Institute of Technology, 1989.

[140] A.J. Martin. *Asynchronous datapaths and the design of an asynchronous adder*. Technical report, California Institute of Technology, 1991.

[141] A.J. Martin, S. Burns, T.K. Lee, D. Borkovic, and P.J. Hazewindus. The design of an asynchronous microprocessor. In *Advanced Research in VLSI*, pages 351–373, 1989.

[142] A.J. Martin, M. Nystrom, and C.G. Wong. Three generations of asynchronous microprocessors. *IEEE Design and Test*, 20(6):9–17, 2003.

[143] P.B. McGee, M.Y. Agyekum, M.A. Mohamed, and S.M. Nowick. A level-encoded transitions signaling protocol for high-throughput asynchronous global communication. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 116–127, 2008.

[144] P.A. Merolla, J.V. Arthur, R.A.-Icaza, A.S. Cassidy, J. Sawada, F. Akopyan, B.L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S.K. Esser, R. Appuswamy, B. Taba, A. Amir, M.D. Flickner, W.P. Risk, R. Manohar, and D.S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[145] G. De Micheli and L. Benini. Networks on chip: 15 years later. *IEEE Transactions on Computers*, 50(5):10–11, 2017.

[146] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1–6, 2004.

[147] G. Miorandi, M. Balboni, S.M. Nowick, and D. Bertozzi. Accurate assessment of bundled-data asynchronous NoCs enabled by a predictable and efficient hierarchical synthesis flow. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 10–17, 2017.

[148] G. Miorandi, D. Bertozzi, and S.M. Nowick. Increasing impartiality and robustness in high-performance N-way asynchronous arbiters. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 108–115, 2015.

[149] G. Miorandi, A. Ghiribaldi, S.M. Nowick, and D. Bertozzi. Crossbar replication vs. sharing for virtual channel flow control in asynchronous NoCs: A comparative study. In *Proceedings of the IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6, 2014.

[150] M. Modarressi, H. Sarbazi-Azad, and M. Arjomand. A hybrid packet-circuit switched on-chip network based on SDM. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 566–569, 2009.

[151] A. Mokhov, V. Khomenko, and A. Yakovlev. Flat arbiters. *Fundamenta Informaticae, No. 1-2*, pages 63–90, 2011.

[152] M. Moreira, A. Neutzling, M. Martins, A. Reis, R. Ribas, and N. Calazans. Semi-custom NCL design wth commercial EDA frameworks: Is it possible? In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 53–60, 2014.

[153] M. Moreira, B. Oliveira, F. Moraes, and N. Calazans. Impact of C-elements in asynchronous circuits. In *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 437–443, 2012.

[154] P.R. Morrow, C.-M. Park, S. Ramanathan, M.J. Kobrinsky, and M. Harmes. Three-dimensional wafer stacking via Cu-Cu bonding integrated with 65-nm strained-Si/low-k CMOS technology. *IEEE Electron Device Letters*, 27(5):335–337, 2006.

[155] D.E. Muller and W.S. Bartky. A theory of asynchronous circuits. In *International Symposium on the Switching Theory in Harvard University*, pages 204–243, 1959.

[156] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 188–197, 2004.

[157] R. Mullins, A. West, and S. Moore. The design and implementation of a low-latency on-chip network. In *Proceedings of the ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 164–169, 2006.

[158] J. Murray, P. Wettin, P.P. Pande, and B. Shirazi. *Sustainable wireless network-on-chip architectures*. Morgan Kaufmann, 2016.

[159] C. Myers. *Asynchronous Circuit Design*. John Wiley and Sons, 2001.

[160] S.R. Naqvi and A. Steininger. A tree arbiter cell for high speed resource sharing in asynchronous environments. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1–6, 2014.

[161] C.A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M.S. Yousif, and C.R. Das. ViChaR: A dynamic virtual channel regulator for network-on-chip routers. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 333–346, 2006.

[162] S.M. Nowick and D.L. Dill. Synthesis of asynchronous state machines using a local clock. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 192–197, 1991.

[163] S.M. Nowick and D.L. Dill. Exact two-level minimization of hazard-free logic with multiple-input changes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(8):986–997, 1995.

[164] S.M. Nowick and C.W. O'Donnell. On the existence of hazard-free multi-level logic. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 109–120, 2003.

[165] S.M. Nowick and M. Singh. High-performance asynchronous pipelines: An overview. *IEEE Design and Test of Computers*, 28(5):8–22, 2011.

[166] S.M. Nowick and M. Singh. Asynchronous design – part 1: Overview and recent advances. *IEEE Design and Test*, 32(3):5–18, 2015.

[167] S.M. Nowick and M. Singh. Asynchronous design – part 2: Systems and methodologies. *IEEE Design and Test*, 32(3):19–28, 2015.

[168] S.M. Nowick, K.Y. Yun, P.A. Beerel, and A.E. Dooply. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 210–223, 1997.

[169] R.O. Ozdag and P.A. Beerel. High-speed QDI asynchronous pipelines. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 13–22, 2002.

[170] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance evaluation and design trade-offs for network-on-chip interconnect architectures. *IEEE Transactions on Computers*, 54(8):1025–1040, 2005.

[171] R. Parikh, R. Das, and V. Bertacco. Power-aware NoCs through routing and topology reconfiguration. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[172] L.-S. Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA-01)*, pages 164–169, 2001.

[173] O.A. Petlin and S.B. Furber. Built-in self-testing of micropipelines. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 22–29, 1997.

[174] G. Philip, B. Christopher, and P. Ramm. *Handbook of 3D Integration: Technology and Application of 3D Integrated Circuits*. Wiley-VCH, 2008.

[175] S.J. Piestrak. Membership test logic for delay-insensitive codes. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 194–204, 1998.

[176] S.J. Piestrak and T. Nanya. Towards totally self-checking delay-insensitive systems. In *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing (FTCS)*, pages 228–237, 1995.

[177] A. Pinto, L.P. Carloni, and A.L. Sangiovanni-Vincentelli. Efficient synthesis of networks on chip. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 1–5, 2003.

[178] M. Pirretti, G.M. Link, R.R. Brooks, N. Vijaykrishnan, M. Kandemir, and M.J Irwin. Fault tolerant algorithms for network-on-chip interconnect. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI Emerging Trends in VLSI Systems Design (ISVLSI)*, pages 1–6, 2004.

[179] L.A. Plana. *Contributions to the Design of Asynchronous Macromodular Systems*. PhD thesis, Department of Computer Science, Columbia University, 1999.

[180] L.A. Plana and S.H. Unger. Pulse-mode macromodular systems. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 348–353, 1998.

[181] V. Puente, J.A. Gregorio, F. Vallejo, and R. Beivide. Immunet: A cheap and robust fault-tolerant packet routing mechanism. In *Proceedings of the IEEE Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2004.

[182] A. Pullini, F. Angiolini, S. Murali, D. Atienza, G. De Micheli, and L. Benini. Bringing NoCs to 65 nm. *IEEE Micro*, 27(5):75–85, 2007.

[183] J. Rabaey, A. Chandrakasan, and B. Nokolic. *Digital Integrated Circuits: A Design Perspective (2nd Edition)*. Prentice-Hall, 2003.

[184] M. Radetzki, C. Feng, X. Zhao, and A. Jantsch. Methods for fault tolerance in networks-on-chip. *ACM Computing Surveys*, 46(1):8:1–38, 2013.

[185] A. Rahimi, I. Loi, M.R. Kakoee, and L. Benini. A fully-synthesizable single-cycle interconnection network for share-L1 processor clusters. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1–6, 2011.

[186] A.M. Rahmani, M.-H. Haghbayan, A. Miele, P. Liljeberg, A.Jantsch, and H. Tenhunen. Reliability-aware runtime power management for many-core systems in the dark silicon era. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2):427–440, 2017.

[187] R.B. Reese, S.C. Smith, and M.A. Thornton. Uncle – An RTL approach to asynchronous design. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 65–72, 2012.

[188] W.F. Richardson and E. Brunvand. Precise exception handling for a self-timed processor. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 32–37, 1995.

[189] D. Rostislav, V. Vishnyakov, E. Friedman, and R. Ginosar. An asynchronous router for multiple service levels networks on chip. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 1–10, 2005.

[190] P.K. Sahu and S. Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):1987–2000, 2013.

[191] E. Salminen, A. Kulmala, and T.D. Hamalainen. Survey of network-on-chip proposals. In *OCP-IP White Paper*, pages 1–13, 2008.

[192] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli. SunFloor 3D: A tool for networks on chip topology synthesis for 3-D systems on chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):1987–2000, 2010.

[193] C.L. Seitz. *System timing*. Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[194] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottethodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA)*, pages 432–443, 2005.

[195] A. Shacham, K. Bergman, and L.P. Carloni. Photonic network-on-chip for future generations of chip multiprocessors. *IEEE Transactions on Computers*, 57(9):1246–1260, 2008.

[196] D. Shang, F. Xia, and A. Yakovlev. Highly parallel multi-resource arbiters. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 4117–4120, 2010.

[197] A. Sheibanyrad, A. Greiner, and I. M.-Panades. Multisynchronous and fully asynchronous NoCs for GALS architecture. *IEEE Design and Test of Computers*, 25(6):572–580, 2008.

[198] P. Shepherd, S.C. Smith, J. Holmes, A.M. Francis, N. Chiolino, and H.A. Mantooth. A robust, wide-temperature data transmission system for space environments. In *IEEE Aerospace Conference (AERO)*, pages 1–13, 2013.

[199] F. Shi, Y. Makris, S.M. Nowick, and M. Singh. Test generation for ultra-high-speed asynchronous pipelines. In *Proceedings of the IEEE International Test Conference (TEST)*, pages 1–10, 2005.

[200] A.K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 1–10, 2013.

[201] M. Singh and S.M. Nowick. MOUSETRAP: High-speed transition-signaling asynchronous pipelines. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 9–17, 2001.

[202] M. Singh and S.M. Nowick. The design of high-performance dynamic asynchronous pipelines: High-capacity style. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(11):1270–1283, 2007.

[203] M. Singh and S.M. Nowick. The design of high-performance dynamic asynchronous pipelines: Lookahead style. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(11):1256–1269, 2007.

[204] M. Singh and S.M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(6):684–698, 2007.

[205] M. Singh, J.A. Tierno, A. Rylyakov, S. Rylov, and S.M. Nowick. An adaptively pipelined mixed synchronous-asynchronous digital FIR filter chip operating at 1.3 Gigahertz. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(7):1043–1056, 2010.

[206] T. Singh and A. Taubin. A highly scalable GALS crossbar using token ring arbitration. *IEEE Design and Test of Computers*, 24(5):464–472, 2007.

[207] B. Sinharoy, J.A. Van Norstrand, R.J. Eickemeyer, H.Q. Le, J. Leenstra, D.Q. Nguyen, B. Konigsburg, K. Ward, M.D. Brown, J.E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J.W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach, T. Karkhanis, and K.M. Fernsler. IBM POWER8 processor core microarchitecture. *IBM Journal of Research and Development*, 59(1):2:1–21, 2015.

[208] W. Song and D. Edwards. Asynchronous spatial division multiplexing router. *Microprocessors and Microsystems*, 35(2):85–97, 2011.

[209] R.F. Sproull, I.E. Sutherland, and C.E. Molnar. The counterflow pipeline processor architecture. *IEEE Design and Test*, 11(3):48–59, 1994.

[210] M. Stan and W. Burleson. Low-power CMOS clock drivers. In *Proceedings of the International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, pages 149–156, 1995.

[211] S. Stergiou, F. Angiolini, S. Carta, L. Raffo, D. Bertozzi, and G.D. Micheli. xpipesLite: A synthesis oriented design library for networks on chip. In *Proceedings of the IEEE Design, Automation and Test in Europe (DATE)*, pages 1188–1193, 2005.

[212] K.S. Stevens, S. Rotem, R. Ginosar, P. Beerel, C.J. Myers, K.Y. Yun, R. Koi, C. Dike, and M. Roncken. An asynchronous instruction length decoder. *IEEE Journal of Solid-State Circuits*, 36(2):217–228, 2001.

[213] I.E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, 1989.

[214] D. Sylvester and K. Keutzer. A global wiring paradigm for deep submicron design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2):242–252, 2000.

[215] A. Takamura, M. Kuwako, M. Imai, T. Fujii, M. Ozawa, I. Fukasaku, Y. Ueno, and T. Nanya. TITAC-2: An asynchronous 32-Bit microprocessor based on scalable-delay-insensitive model. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 288–294, 1997.

[216] P. Teehan, M. Greenstreet, and G. Lemieux. A survey and taxonomy of GALS design styles. *IEEE Design and Test*, 24(5):418–428, 2007.

[217] J. Teifel and R. Manohar. Highly pipelined asynchronous FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 133–142, 2004.

[218] N. Teimouri, M. Modarressi, and H. Sarbazi-Azad. Power and performance efficient partial circuits in packet-switched networks-on-chip. In *Proceedings of Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 509–513, 2013.

[219] H. Terada, S. Miyata, and M. Iwata. DDMPs: Self-timed super-pipelined data-driven multimedia processors. *Proceeding of the IEEE Design and Test*, 87(2):282–295, 1999.

[220] M. Theobald and S.M. Nowick. Fast heuristic and exact algorithms for two-level hazard-free logic optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1130–1147, 1998.

[221] Y. Thonnart, E. Beigne, and P. Vivet. A pseudo-synchronous implementation flow for WCHB QDI asynchronous circuits. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 73–80, 2012.

[222] A.W. Topol, D.C. La Tulipe, L. Shi, D.J. Frank, K. Bernstein, S.E. Steen, A. Kumar, G.U. Singco, A.M. Young, K.W. Guarini, and M. Leong. Three-dimensional integrated circuits. *IBM Journal of Research and Development*, 50(4/5):491–506, 2006.

[223] S.H. Unger. *Asynchronous Sequential Switching Circuits*. New York, NY: Wiley, 1969.

[224] L.G. Valiant. A scheme for fast parallel communication. *SIAM Journal of Computing*, 11(2):350–361, 1982.

[225] K. van Berkeal, J. Kessels, M. Roncken, R. Saeijs, and F. Schalij. The VLSI-programming language Tangram and its translation into handshake circuits. In *Proceedings of the European Conference on Disign Automation (EDAC)*, pages 384–389, 1991.

[226] C.H. van Berkel and C.E. Molnar. Beware the three-way arbiter. *IEEE Journal of Solid-State Circuits*, 34(6):840–848, 1999.

[227] K. van Berkel. *Handshake Circuits: An Asynchronous Architecture for VLSI Programming*. Cambridge University Press, 1993.

[228] K. van Berkel and A. Bink. Single-track handshaking signaling with application to micropipelines and handshake circuits. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 122–133, 1996.

[229] K. van Berkel, M.B. Josephs, and S.M. Nowick. Scanning the technology: Applications of asynchronous circuits. In *Proceedings of the IEEE*, pages 223–233, 1999.

[230] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 96–107, 1998.

[231] H. van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proceedings of the IEEE International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, pages 1–12, 1998.

[232] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.

[233] T. Verhoeff. Delay-insensitive codes–an overview. *Distributed Computing*, 3(1):1–8, 1988.

[234] C. Vezyrtzis, W. Jiang, S.M. Nowick, and Y. Tsividis. A flexible clockless digital filter. In *Proceedings of the IEEE European Solid-State Circuit Conference (ESSCIRC)*, pages 65–68, 2013.

[235] C. Vezyrtzis, W. Jiang, S.M. Nowick, and Y. Tsividis. A flexible event-driven digital filter with frequency response independent of input sample rate. *IEEE Journal of Solid-State Circuits*, 49(10):2292–2304, 2014.

[236] P. Vivet, Y. Thonnart, R. Lemaire, C. Santos, E. Beigne, C. Bernard, F. Darve, D. Lattard, I. M.-Panades, D. Dutoit, F. Clermidy, S. Cheramy, A. Sheibanyrad, F. Petrot, E. Flamand, J. Michailos, A. Arriordaz, L. Wang, and J. Schloeffel. A 4x4x2 homogeneous scalable 3D network-on-chip circuit with 326 MFlit/s 0.66 pJ/b robust and fault tolerant asynchronous 3D links. *IEEE Journal of Solid-State Circuits*, 52(1):33–49, 2017.

[237] P. Wehner, J. Rettkowski, T. Kleinschmidt, and D. Gohringer. MPSoCSim: An extended OVP simulator for modeling and evaluation of network-on-chip based heterogeneous MPSoCs. In *Proceedings of the IEEE International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 390–395, 2015.

[238] D. Wiklund and D. Liu. Switched interconnect for system-on-a-chip designs. In *Proceedings of the IP 2000 Europe Conference (IP2000)*, pages 1–6, 2000.

[239] T.E. Williams. *Self-timed rings and their application to division*. PhD thesis, Department of Electrical Engineering and Computer Science, Stanford University, 1991. *(http://vlsiweb.stanford.edu/people/alum/pdf/9105_Williams_SelfTimedRings.pdf)*.

[240] T.E. Williams and M.A. Horowitz. A zero-overhead self-timed 160-ns 54-b cmos divider. *IEEE Journal of Solid-State Circuits*, 26(11):1651–1661, 1991.

[241] D.H. Woo and H.-H.S. Lee. Extending Amdahl's law for energy-efficient computing in the many-core era. *IEEE Transactions on Computers*, 41(12):24–31, 2008.

[242] A. Yakovlev, A. Petrov, and L. Lavagno. A low latency asynchronous arbitration circuit. *IEEE Transactions on VLSI Systems*, 2(3):372–377, 1994.

[243] Y.S. Yang, R. Kumar, G. Choi, and P. Gratz. WaveSync: A low-latency source synchronous bypass network-on-chip architecture. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 241–248, 2012.

[244] K.Y. Yun, P.A. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In *Proceedings of the IEEE International Symposium of Asynchronous Circuits and Systems (ASYNC)*, pages 17–28, 1996.

[245] Z. Zhang, A. Greiner, and S. Taktak. A reconfigurable routing algorithm for a fault-tolerant 2D-Mesh network-on-chip. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 441–446, 2008.