Design and Implementation of Low-latency, Low-power Reconfigurable On-Chip Networks

by

Chia-Hsin Chen

B.S., National Taiwan University (2007)
S.M., Massachusetts Institute of Technology (2012)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

February 2017

Author .... Signature redacted
Department of Electrical Engineering and Computer Science
October 14, 2016

Certified by..... Signature redacted ....
Li-Shiuan Peh
Professor
Thesis Supervisor

Accepted by ...... Signature redacted ....
Leslie A. Kolodziejski
Professor
Chair of the Department Committee on Graduate Students

Design and Implementation of Low-latency, Low-power Reconfigurable On-Chip Networks

by

Chia-Hsin Chen

Submitted to the Department of Electrical Engineering and Computer Science
on October 14, 2016, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

In this dissertation, I tackle large, low-latency, low-power on-chip networks. I focus on two key challenges in the realization of such NoCs in practice: (1) the development of NoC design toolchains that can ease and automate the design of large-scale NoCs, paving the way for advanced ultra-low-power NoC techniques to be embedded within many-core chips, and (2) the design and implementation of chip prototypes that demonstrate ultra-low-latency, low-power NoCs, enabling rigorous understanding of the design tradeoff of such NoCs.

I start off by presenting DSENT (joint work), a timing, area and power evaluation toolchain that supports flexibility in modeling while ensuring accuracy, through a technology-portable library of standard cells [108]. DSENT enables rigorous design space exploration for advanced technologies, and have been shown to provide fast and accurate evaluation of emerging opto-electronics. Next, low-swing signaling has been shown to substantially reduce NoC power, but requires custom circuit design in the past. I propose a toolchain that automates the embedding of low-swing cells into the NoC datapath, paving the way for low-swing signaling to be part of future many-core chips [17]. Third, clockless repeated links have been shown to be embeddable within a NoC datapath, allowing packets to go from source to destination cores without being latched at intermediate routers. I propose SMART$_{app}$, a design that leverages theses clockless repeaters for configuration of a NoC into customized topologies tailored for each applications, and present a synthesis toolchain that takes each SoC application as input, and synthesize a NoC configured for that application, generating RTL to layout [18].

The thesis next presents two chip prototypes that I designed to obtain on-depth understanding of the practical implementation costs and tradeoffs of high-level architectural ideas. The SMART NoC chip is a 3×3 mm^2 chip in 32 nm SOI realizing traversal of 7 hops within a cycle at 548 MHz, dissipating 1.57 to 2.53 W. It enables a rigorous understanding of the tradeoffs between router clock frequency, network latency and throughput, and is a demonstration of the proposed synthesis toolchain. The SCORPIO 36-core chip (joint work) is an 11×13 mm^2 chip in 45 nm SOI demonstrating snoopy coherence on a scalable ordered mesh NoC, with the NoC taking just 19 % of tile power and 10 % of tile area [19, 28].

Thesis Supervisor: Li-Shiuan Peh
Title: Professor

Acknowledgments

First of all, I would like to thank my research advisor, Prof. Li-Shiuan Peh. It was really nice working with her and learning from her not only technical knowledge but also attitude in life and in research. Thanks to her, I had the chance to attend Princeton and MIT, and to participate in tons of interesting projects in addition to my own research projects.

I would also like to thank my committee members, Prof. Joel Emer, Prof. Srini Devadas and Prof. Vivienne Sze, for helping me shape the thesis as well as providing insightful feedback and comments on my thesis.

I thank all my group mates, Bin Lin, Niket Agarwal, Kostas Aisopos, Manos Koukoumidis, Tushar Krishna, Sunghyun Park, Bhavya Daya, Jason Gao, Woo Cheol Kwon, Pablo Ortiz, and Suvinay Subramanian. It was a great experience that I collaborated with most of them on plenty of projects throughout my long 8 years of Ph.D. Specifically, I would like to Tushar and Suvinay for all the endless, sometimes last minute, technical discussions; without them, my thesis would not have any progress.

Even though I was an EE student in college, but there are just so many things in circuits and measurements that I had not learned. Thanks to Chen Sun, Arun Paidimarri, and Phillip Nadeau, I learned a lot on digital circuits, implementations, and measurements. I even get my first job as a digital circuit designer/engineer.

Studying aboard in the US and being away from home are tough and lonely. I thank Hung-Wen Chen, Yin-Wen Chang, Max Hsieh, Yu-Chung Hsiao, Dawsen Hwang and Hsin-Jung Yang from MIT, Joecy Lin, Alex Huang and Jeremiah Tu from my chorus group, as well as Karen Chang, my best roommate, for their accompany and all the fun moments together. Hsin-Jung Yang is my best friend at MIT; we had a great time together: having meals, watching soap operas, chitchatting, discussing all sorts of matters including research ideas, and supporting each other during deadlines. She is the first person I would turn to whenever I am in a bad mood or encounter any obstacles. I would definitely miss the daily fun snack time we had together. Even though Karen Chang and I were roommates for only a few months, she accompanied me and filled me with pure positive energy when I was putting on my final sprint toward my thesis and defense, and dragged me out of my room to try out many interesting and fun things that I probably would never attempt. Jeremiah Tu lured me into playing LoL, which helps me make good virtual friends, and served as my best way to relieve stress and release tension.

Even though I have been away from home for 8 years with only few short visits, my deepest gratitude goes to my family, my parents and my brother, for their support throughout my life and always being by my side. Without them, I will not be here and become Dr. Chen.

Lastly, this year is not only the year that I become Dr. Chen but also the turning point of my life. I thank all the people who help, support and encourage to be myself and smoothly transition from Owen to Amy. I appreciate all the efforts they make to quickly accept my new identity, let me become the person I want to be, and show no discrimination. I am extremely grateful to have them around me. ♡

Contents

Abstract **iii**

Acknowledgments **v**

Contents **vii**

List of Figures xiii

List of Tables xvii

1 Introduction **1**

1.1 Network-on-Chip 2

1.2 Dissertation Overview 3

1.2.1 DSENT – Design Space Exploration of Networks Tool (Chapter 3) 3

1.2.2 Low-Power Crossbar Generator Tool (Chapter 4) 4

1.2.3 SMART$_{app}$– Low-Latency Network Generator Tool for SoC Applications (Chapter 5) 4

1.2.4 SMART Network Chip (Chapter 6) 5

1.2.5 SCORPIO – A 36-core Shared Memory Processor Demonstrating Snoopy Coherence on a Mesh Interconnect (Chapter 7) 5

1.3 Dissertation Contribution 6

1.3.1 NoC Toolchains 6

1.3.2 NoC Chip Prototypes 7

2 Background 9

2.1 Network-on-Chip (NoC) 9

2.1.1 Topology 10

2.1.2 Routing Algorithm 10

2.1.3 Flow Control Mechanism 11

2.1.4 Microarchitecture 11

2.2 Low-Power Link – Low-Swing Signaling 13

2.3 Low-Latency Link – Opto-Electrical Signaling 13

2.3.1 Photonic Link 14

2.3.2 Prior Photonic NoC Architectures 15

2.4 Low-Latency and Low-Power Routers 15

2.5 Reconfigurable NoC Topologies 16

2.6 In-network Coherence and Filtering 16

3 DSENT – Design Space Exploration of Networks Tool 19

3.1 Motivation 19

3.2 Existing NoC Modeling Tools 20

3.3 DSENT Framework 21

3.3.1 Framework Overview 22

3.3.2 Power, Energy, and Area Breakdowns 23

3.4 DSENT Models and Tools for Electronics 24

3.4.1 Transistor Models 24

3.4.2 Standard Cells 25

3.4.3 Delay Calculation and Timing Optimization 26

3.4.4 Expected Transitions 28

3.4.5 Summary 28

3.5 DSENT Models and Tools for Photonics 28

3.5.1 Photonic Device Models 29

3.5.2 Interface Circuitry 29

3.5.3 Ring Tuning Models 30

3.5.4 Optical Link Optimization 31

3.5.5 Summary 32

3.6 Model Validation 32

3.7 Example Photonic Network Evaluation 34

3.7.1 Scaling Electrical Technology and Utilization Tradeoff 35

3.7.2 Photonics Parameter Scaling 38

3.7.3 Thermal Tuning and Data Rate 38

3.8 Summary 40

4 Low-Power Crossbar Generator Tool **43**

4.1 Motivation 43

4.2 Background 46

4.2.1 Limitations to current synthesis flow 48

4.3 Datapath Generator 49

4.3.1 Building Block Pre-characterization 51

4.3.2 Layout Generation 52

4.3.3 Verification and Extraction 55

4.3.4 Post-characterization and Selection 55

4.4 Evaluation 56

4.4.1 Generated vs. Synthesized Datapath 56

4.4.2 Case Study 58

4.5 Summary 60

5 SMART - Low-Latency Network Generator Tool for SoC Applications **61**

5.1 Motivation 61

5.2 Background - Clockless Repeated Links 63

5.3 SMART Network Architecture 65

5.3.1 Router Microarchitecture 65

5.3.2 Routing 67

5.3.3 Flow Control 68

5.4 Tool Flow 69

5.4.1 Physical Implementation 70

5.4.2 Application Mapping 71

5.5 Case Study 72

5.5.1 Configurations 72

5.5.2 Performance Evaluation 73

5.5.3 Power Analysis: 74

5.6 Summary 75

6 SMART Network Chip **77**

6.1 Motivation 77

6.2 Design Analyses of SMART on Process Limitation 79

6.2.1 Repeated Link 79

6.2.2 Data Path 80

6.2.3 Control Path 81

6.2.4 Summary 83

6.3 Chip Architecture 83

6.3.1 NIC and Tester Microarchitecture 85

6.3.2 Router Microarchitecture 86

6.4 Implementation Consideration 90

6.5 Evaluation 91

6.5.1 Setup 91

6.5.2 Area 92

6.5.3 Timing – Static Timing Analysis (STA) 94

6.5.4 Timing – Measurement 96

6.5.5 Power – Simulation 98

6.5.6 Power – Measurement 100

6.5.7 Sources of Discrepancies 100

6.5.8 Insights 101

6.6 Summary 102

7 SCORPIO - A 36-core Shared Memory Processor Demonstrating Snoopy Coherence on a Mesh Interconnect 105

7.1 Motivation 105

7.2 Globally Ordered Mesh Network 107

7.2.1 Walkthrough Example 109

7.2.2 Main Network Microarchitecture 111

7.2.3 Notification Network Microarchitecture 115

7.2.4 Network Interface Controller Microarchitecture 116

7.3 36-Core Processor with SCORPIO NoC 118

7.3.1 Processor Core and Cache Hierarchy Interface 119

7.3.2 Coherence Protocol 120

7.3.3 Functional Verification 122

7.4 Architecture Analysis 124

7.4.1 Performance 126

7.4.2 NoC Design Exploration for 36-Core Chip 127

7.4.3 Scaling Uncore Throughput for High Core Counts 129

7.5 Architectural Characterization of SCORPIO Chip 132

7.5.1 L2 Service Latency 132

7.5.2 Overheads 134

7.6 Chip Measurements and Lessons Learned 135

7.7 Related Work 136

7.8 Summary 139

8 Conclusion 141

8.1 Dissertation Summary 141

8.1.1 Development of NoC Design Toolchains 141

8.1.2 Design and Implementation of Chip Prototypes 143

8.2 Future Research Directions 143

A SMART Network Architecture Targeting Many-core System Applications 145

A.1 Motivation 145

A.2 SMART Router and Terminology 146
A.3 SMART in a *k*-ary 1-Mesh 148
A.3.1 SMART-hop Setup Request (SSR) 149
A.3.2 Switch Allocation Global: Priority 151
A.3.3 Ordering 152
A.3.4 Guaranteeing Free VC/buffers at Stop Routers 153
A.3.5 Additional Optimizations 154
A.3.6 Summary 155
A.4 SMART in a *k*-ary 2-Mesh 155
A.4.1 Bypassing routers along dimension 155
A.4.2 Bypassing routers at turns 156
A.5 Summary 157

Bibliography **159**

List of Figures

1-1 Core count trend over the years 1

2-1 Router Microarchitecture 12

2-2 A typical opto-electronic NoC including electrical routers and links, and a wavelength devision multiplexed intra-chip photonic link 14

3-1 DSENT Framework with Examples of Network-related User-defined Models 22

3-2 Standard cell model generation and characterization. In this example, a NAND2 standard cell is generated. 25

3-3 Mapping Standard Cells to RC Delays 26

3-4 Incremental Timing Optimization 27

3-5 Comparison of Network Energy per bit vs. Network Throughput . . . 36

3-6 Energy per bit Breakdown at Various Throughputs 36

3-7 Sensitivity to Waveguide Loss 37

3-8 Sensitivity to Heating Efficiency 38

3-9 Comparison of Thermal-Tuning Strategies at 16.5 Tb/s Throughput . . . 39

4-1 2-bit 4×4 crossbar schematic 46

4-2 Logical 4:1 Multiplexer (a) and Two Realizations (b)(c) 46

4-3 Simplified datapath 47

4-4 Standard synthesis flow 48

4-5 Proposed Datapath Generator's Tool Flow 50
4-6 Schematic of Transmitter and Receiver 51
4-7 Transmitter Abstract Layout 52
4-8 Example Single-bit Crossbar Layout with 6 Inputs and 6 Outputs 52
4-9 4-bit Crossbar Abstract Layout with 1 Port Connecting to the Link . . . 53
4-10 Selected Wire Shielding Topology 54
4-11 Example 6×6 64-bit Datapath Layout with One Link Shown 55
4-12 Energy per bit Sent of 64-bit Datapaths 57
4-13 Crossbar Area with Various Architectural Parameters 58
4-14 Five-port Router in a Mesh Network 58
4-15 Synthesized Router with Generated Low-swing Datapath 59

5-1 Mesh Reconfiguration for Three Applications. All links in bold take one-cycle. 62
5-2 VLR Schematic 63
5-3 SMART Router Microarchitecture and Pipeline 66
5-4 SMART NoC in Action with Four Flows (The number next to each arrow indicates the traversal time of that flow.) 67
5-5 Tool Flow 69
5-6 One-bit SMART Crossbar 70
5-7 32-bit Tx Block Layout 70
5-8 Generated 4x4 NoC Layout 70
5-9 Performance 73
5-10 Power Breakdown 74

6-1 Achievable HPC_{max} for Repeated Links at 1 GHz. 80
6-2 Energy and Area versus HPC_{max} for Crossbar 81
6-3 Implementation of SA-G at W_{in} and E_{out} for 1D version of $SMART_{cycle}$. 81
6-4 Energy and Area versus HPC_{max} for 1D version of SA-G 82
6-5 Energy and Area versus HPC_{max} for 2D version of SA-G 82
6-6 Chip Layout 84

6-7 Node Microarchitecture 84
6-8 Router Microarchitecture 86
6-9 Router Pipeline 87
6-10 Router Pipeline 88
6-11 Folded network with router pitch of 1 mm 90
6-12 Area Breakdown 92
6-13 Router Critical Paths 93
6-14 Chip Critical Path 95
6-15 Flit/Credit Only Path Delay 97
6-16 Flit/Credit + SSR Path Delay 97
6-17 Leakage Power Breakdown 98
6-18 Dynamic Power Breakdown 99
6-19 Measured Power 99
6-20 Average Latency versus Injection Rate 102

7-1 Proposed SCORPIO Network 108
7-2 Time Window for Notification Network 108
7-3 Walkthrough Example (from T1 to T3) 110
7-4 Walkthrough Example Cont. (from T4 to T5) 111
7-5 Walkthrough Example Cont. (from T6 to T7) 112
7-6 Router Microarchitecture 113
7-7 Notification Router Microarchitecture 115
7-8 Network Interface Controller Microarchitecture 117
7-9 36-core Chip Layout with SCORPIO NoC 119
7-10 36-core Chip Schematic 120
7-11 *sync* Test for 2 Cores 123
7-12 Normalized Runtime and Latency Breakdown 126
7-13 Normalized Runtime with Varying Network Parameters 128
7-14 Pipelining effect on performance and scalability 130
7-15 L2 Service Time Breakdown (*barnes*) 132

7-16 L2 Service Time Histogram (*barnes*) 134
7-17 Tile Overheads 135

A-1 SMART Router Microarchitecture 146
A-2 Example of Single-cycle Multi-hop Traversal 146
A-3 k-ary 1-Mesh with dedicated SSR links. 148
A-4 SMART Pipeline 148
A-5 SMART Example: No SSR Conflict 150
A-6 SMART Example: SSR Conflict with Prio=Local 150
A-7 SMART Example: SSR Conflict with Prio=Bypass 151
A-8 k-ary 2-Mesh with SSR Wires From Shaded Start Router 156
A-9 Conflict Between Two SSRs for N_{out} Port 157
A-10 SMART_2D SA-G priorities 157

List of Tables

3-1 DSENT Parameters 24

3-2 DSENT Validation Points 33

3-3 Network Configuration 34

3-4 Default Technology Parameters 35

3-5 Sweep Parameters Organized by Section 35

4-1 Inputs to Proposed Datapath Generator 49

4-2 Pre-characterization Results 51

4-3 Performance of 1 mm Link of Two Organizations 55

4-4 Example Generated Datapaths 56

4-5 Router Specifications 58

5-1 Simulation Results of Max Number of Hops per Cycle 65

5-2 4x4 NoC Configuration 72

6-1 Chip specification 85

6-2 Flit Link Length and Delay 95

6-3 Clock Skew (ns) 96

7-1 SCORPIO chip features 121

7-2 Regression Tests 123

7-3 Request Categories 132
7-4 Comparison of multicore processors 137

1-1 Terminology 147

1
Introduction

Advances in CMOS technology have enabled increasing transistor density on a chip. Due to the power wall, general-purpose computer architects have stopped using the extra transistors to increase the complexity of a single processing core. Instead, they have embraced a more power/area efficient approach, using the additional transistors to increase the number of processing cores and run these cores in parallel to obtain higher performance. Meanwhile, in the embedded domain, system-on-chip (SoC) designers have also started adding more and more general-purpose/application-specific intellectual

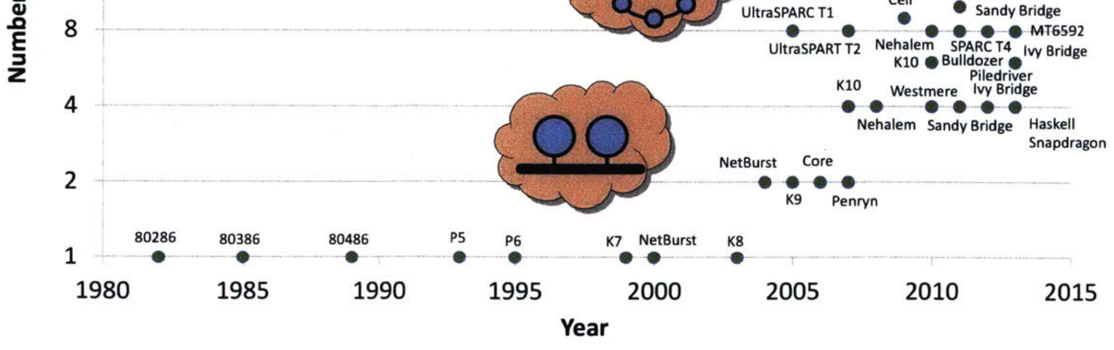


Figure 1-1: Core count trend over the years

property (IP) with the emergence of diverse computation-intensive applications over the past few years, and this has intensified with the proliferation of smart phones. Figure 1-1 shows the number of cores on chip of some well-known architectures from Intel, AMD, Oracle (Sun Microsystems), IBM, Tilera and others. Starting from 2004, the number of cores has continued to increase. While desktop processors have employed 8 to 16 cores, high-throughput targeted processors have reached more than 48 cores. This trend is expected to continue with future architectures incorporating tens or hundreds of cores.

1.1 Network-on-Chip

One or more on-chip networks (NoCs) are used to support efficient communication among the cores. A decade ago, when the number of the cores was few, buses were adopted from the off-chip network to serve as the communication medium. However, as the number of cores increases, buses cannot sustain the ever-increasing bandwidth demand and incur high packet delivery latency, which worsens the system performance significantly. To overcome the shortcomings of the buses, two extremes (in terms of crossbar radix size) of the NoC topologies are used: flat crossbar and ring. A flat crossbar enables direct all-to-all communication between cores, providing both high throughput and low delivery latency. However, the crossbar structure requires a large amount of silicon and wiring resources, which grows quadratically with the number of cores. On the other hand, while a ring does not suffer from the resource issue, its throughput is limited and the delivery latency grows proportional to the core count in the system.

Systems with higher core count incorporate more complex network topologies, such as meshes, to alleviate the resource and performance issues of rings and flat crossbars. These topologies usually consist of several smaller crossbars and employ more direct connections than rings between cores. The use of several small crossbars lowers the complexity of the resource required for a flat crossbar from quadratic to linear in the number of cores. Meanwhile, more connections between cores allow a lower network diameter, allowing delivery latency to scale sub-linearly with network size. Throughout this dissertation, I will refer to a small crossbar along with its flow control logic as a *router*, and the point-to-point wires that connect the routers/cores as *links*.

While the use of routers could enhance the link utilization, which effectively reduces the need of excessive amount of links and improves the throughput, routers also come with some disadvantages. The more routers on the path from a source core to a destination core, the more latency and power cost[1]. These costs are significant as compared to the ideal scenario, where a path with the same length only consists of a link and no routers. In the past ten years, many works have been proposed to improve NoC performance while keeping its power consumption at a reasonable level. These works can be roughly classified into four categories: topology, routing algorithm, flow control technique, and physical implementation.

1.2 Dissertation Overview

This dissertation aims to make ultra-low-latency, low-power NoCs for future many-core systems. Chapter 2 provides necessary background on NoC and an overview on signaling techniques along with past research proposals for low-latency and low-power NoCs. The rest of this section provides an overview of each project involves in this dissertation and its associated chapter.

1.2.1 DSENT - Design Space Exploration of Networks Tool (Chapter 3)

Opto-electronic links have been shown to have potential to replace copper wires as an ultra-low-latency, low-energy interconnect for NoCs. However, architecting and exploring of the design space for opto-electronic NoCs are difficult with the lack of fast, accurate models that capture photonics and electronics. This chapter presents DSENT, a NoC cost evaluation tool that provides timing, power and area information for both electrical

[1]A unidirectional ring represents a worst case scenario, where a core intends to send a packet to the core connected to the upstream router. The packet needs to traverse all other routers before it reaches the destination, resulting a minimum packet delivery latency bound of $N - 2$, where N is the number of cores in the network.

and emerging photonic NoCs with a given set of NoC parameters. The tool is designed to provide fast, yet accurate estimation (within seconds) to help researchers to quickly evaluate various network proposals and their impact on the overall system. [108]

This is joint work with Chen Sun. I focused on electrical components' modelings and validation, while Chen focused on photonic components' modeling. Specifically, I developed models for electrical primitive cells and basic components that are essential for any NoC designs and validated the models with the place-and-routed designs for various architectural parameters using a commercial 45 nm SOI technology node.

1.2.2 Low-Power Crossbar Generator Tool (Chapter 4)

In addition to the opto-electronic signaling, low-swing signaling is another signaling technology that can substantially reduce NoC power, but has required custom design in the past. I identify that the datapath of a router, crossbar and link, is one of the major power consumption source, and incorporate low-swing signaling techniques into the datapath to lower its power consumption. As the existing VLSI tool chain does not support low-swing circuit integration, I develop a tool chain along with a layout generation tool that takes architectural parameters and generates a layout of a low-power datapath integrated with provided low-swing circuits. [17]

1.2.3 SMART$_{app}$- Low-Latency Network Generator Tool for SoC Applications (Chapter 5)

Clockless repeated links can be embedded within a NoC datapath, allowing packets to traverse from a source to a destination multiple hops away within a single cycle, without needing to be latched at intermediate routers. These clockless repeaters enable configuration of a NoC into customized topologies tailored for each application, what we term as SMART$_{app}$. In this chapter, I present the SMART$_{app}$ architecture, and I propose a tool flow that takes SoC applications as input, synthesizes a NoC that reconfigures its topology for each application, along with its register-transfer level (RTL) netlist and layout. [18]

1.2.4 SMART Network Chip (Chapter 6)

In addition to the $\text{SMART}_{\text{app}}$, $\text{SMART}_{\text{cycle}}$, a joint work with Tushar Krishna (briefly described in Appendix A), is a variant of SMART network that targets manycore system applications. While both $\text{SMART}_{\text{app}}$ and $\text{SMART}_{\text{cycle}}$ dramatically reduce the packet delivery latency, this latency benefit relies on CMOS process characteristics and careful physical implementation. To demonstrate its feasibility, I designed and implemented a chip prototype of a 64-node SMART NoC, with switchable modes between $\text{SMART}_{\text{app}}$ and $\text{SMART}_{\text{cycle}}$. In this chapter, I discussed the various decisions I made in the design of the test chip, driven by detailed characterization of the design on the targeted process. Furthermore, I present the silicon measurements that enabled an in-depth understanding of the tradeoffs between router clock frequency, network latency and throughput.

1.2.5 SCORPIO - A 36-core Shared Memory Processor Demonstrating Snoopy Coherence on a Mesh Interconnect (Chapter 7)

Servers are moving toward shared memory many-core architectures, and NoCs have been proposed as the communication fabric that can scale to handle such shared memory many-core processors. However, power has been a limiting constraint, and low-power scalable mesh NoCs have not been demonstrated to be able to handle the high bandwidth, low latency requirements of snoopy cache coherent many-core processors. In this chapter, I present the 36-core SCORPIO test chip that tackles this challenge - it incorporates global ordering support within the mesh NoC, while maintaining low latency and low power, bringing mesh NoCs into mainstream snoopy coherence many-core chips. This is a joint project where I led the chip design and implementation. I will discuss the design decisions made, present the RTL simulations that evaluate the scalability of the chip to 100 cores, the detailed timing, area and power analysis. The analysis showed that the 36-core test chip can attain 1 GHz (833 MHz post-layout) at 28.8 W on 45 nm SOI, with the NoC just taking up 10 % of tile area and 19 % of tile power, demonstrating that low-latency, low-power mesh NoCs can be realized for mainstream snoopy coherence. [19, 28]

This is joint work with Bhavya Daya, Woo-Cheol Kwon, Suvinay Subramanian, Sunghyun Park and Tushar Krishna. I co-led the SCORPIO project with Bhavya Daya, with her as the architecture lead while I was the chip RTL and design lead. Specifically, I participated in the architecture design and oversaw the chip RTL implementation. I was in charge of implementing the interface between the proposed network and commercial memory controller, as well as some functionalities in the L2 cache controller. I performed the physical implementation, taking the chip RTL to layout.

1.3 Dissertation Contribution

In this dissertation, I focus on two key challenges in the realization of such NoCs in practice:

- The development of NoC design toolchains that can ease and automate the design of large-scale NoCs, paving the way for advanced ultra-low-power NoC techniques to be embedded within many-core chips.
- The design and implementation of chip prototypes that demonstrate ultra-low-latency, low-power NoCs, enabling rigorous understanding of the design tradeoffs of such NoCs.

In the following, I expand on my contributions in addressing these challenges in these projects.

1.3.1 NoC Toolchains

- I proposed and developed a fast, yet accurate electrical NoC timing, area and power modeling tool. It is validated and shown to be within 20 % of circuit-level Spice simulations. DSENT has since been incorporated within gem5 [**gem5**] and McPAT [70] and widely used in the architectural community.
- I developed a tool chain along with a layout generation tool that takes architectural parameters and generates a layout of a low-power datapath integrated with provided low-swing driver/receiver circuits. I proposed and developed a low-power crossbar

layout generation tool that enables, for the first time, the automated design of large-scale NoCs with custom low-swing cells. It was the first demonstration of a generated low-swing crossbar and link within a fully-synthesized NoC router.

- I proposed and developed a toolchain that synthesizes and configures a single-cycle multi-hop NoC into customized topologies tailored for each application. It enables the automated generation of a single-cycle multi-hop NoC given an application task graph, automatically carried through layout, enabling significant reduction in packet delivery latency for the targeted SoC applications.

1.3.2 NoC Chip Prototypes

- I designed and fabricated the SMART NoC chip prototype, demonstrating for the first time through chip measurements that SMART enables up to 7 hops to be traversed within a cycle, and can be realized at low area/power overhead. However, as the critical path is stretched, reducing max frequency from 817 to 548 MHz, the overall maximum delay savings reduce from $7\times$ to $(7 \times 1.2\text{ns}/1.8\text{ns} = 3.2\times)$.
- I co-designed and implemented the SCORPIO chip prototype which showed that ordering can be supported within a scalable, mesh NoC, realizing a 36-core snoopy cache coherence with high performance (833 MHz post layout), at reasonable area (10 %) and power (19 %) overhead.

2 Background

In this chapter, we provide an overview of the necessary background on on-chip interconnection networks. In addition, we also present background on techniques that can be applied to the network designs to achieve low-power and/or low-latency.

2.1 Network-on-Chip (NoC)

The network-on-chip (NoC) is a network that enables communications between various nodes on the same chip, such as general processing cores, specialized cores, caches, as well as memory controllers, etc. We define a stream of communication between two nodes as a communication *flow*. If the flows or the flow patterns between any two nodes are deterministic, then it is possible to design a tailored network, which is common in the system-on-chip (SoC) domain. However, in other domains such as general-purpose multicore processors, potentially any node would communicate with any other nodes and a network that supports all-to-all communication is required.

The primary features that characterize a NoC are its topology, routing algorithm, flow control mechanism and microarchitecture. We describe each of these briefly. A more thorough discussion can be found in [27, 94].

2.1.1 Topology

A NoC comprises a set of routers and links that connect the nodes on the same chip. The topology is the physical connection of these routers and links. Some common topologies are *bus*, *crossbar*, *ring*, *mesh*, *clos* and *flattened butterfly*.

The topology determines the minimum distance, or number of *hops*, between communicating nodes, where a hop is referred as the unit distance between adjacent routers. A high hop count typically indicates a high network delay to deliver a message and this is the issue that we tackle in this dissertation. The topology also determines the path diversity, which is the number of alternate shortest paths between a source and a destination. The path diversity improves the robustness of the network as well as the fault tolerance.

Crossbars and rings are popular topologies in current off-the-shelf multicore processors and graphics processing units (GPU). However, as the number of nodes in a network increases, a network with only a crossbar may be too complex and not feasible, while the ring topology may not be able to fulfill bandwidth and latency requirements. As a result, among all other topologies, the mesh topology is a popular topology used in many research proposals [36, 42, 43, 110, 119], because of its regular structure and scalability. We use this topology extensively in this dissertation.

2.1.2 Routing Algorithm

The routing algorithm determines how a message is forwarded in the network from its source to destination. In general, routing algorithms can be classified into three categories, i.e. deterministic, oblivious, and adaptive. While using a deterministic routing algorithm, messages always traverse the same path for the same source-destination pair. These deterministic routing algorithms are easy to implement at low area and power cost. On the contrary, both oblivious and adaptive routing algorithms allow messages to traverse different paths for the same source-destination pair. The difference between these two algorithms is that the oblivious routing algorithm chooses a route without considering any current network's state; while the adaptive routing algorithm uses network's state to determine the route.

Dimensional-ordered routing (DOR) algorithm, or XY (YX) routing algorithm, is a commonly used algorithm for the mesh network, which is simple and guarantees deadlock freedom. While routing with this algorithm, messages are first routed along the X (or Y) dimension and then along the Y (or X) dimension.

2.1.3 Flow Control Mechanism

The flow control mechanism defines how a message is forwarded in a network; more specifically how network resources (buffers and links) are allocated. A good flow control mechanism allocates these resources efficiently to achieve high throughput and low latency.

Flow control mechanisms can be classified based on the granularity at which resource allocation occurs. Circuit switching allocates all the links along the route from the source to destination at once for each message. Even though the circuit switching mechanism achieves low latency and does not require buffers in the routers, it often leads to poor bandwidth utilization. Mechanisms such as store-and-forward and virtual-cut-through dissemble messages into packets that can be fitted into the router's buffer, interleaving them on links by allocating resources at packet level to improve utilization.

The packet can be dissembled into an even smaller unit, called a *flit*. Virtual channel (VC) flow control is an example of flow control that allocates buffers and links at the flit level. Unlike packet-level flow control mechanisms that require buffer allocation for the whole packet at the next router, virtual channel flow control allows flits to move forward to the next router as long as there are buffers for the flits. A virtual channel is essentially a buffer queue in the router, and flits in different VCs can be multiplexed onto the links to further improve the resource utilization. VCs can also be used to guarantee deadlock freedom in the network or in the system. In cache-coherent systems, VCs are often used to break coherence protocol level deadlocks.

2.1.4 Microarchitecture

Figure 2-1 shows an example router microarchitecture for a two-dimensional mesh network that uses VC flow control. The router has five input and output ports, corresponding

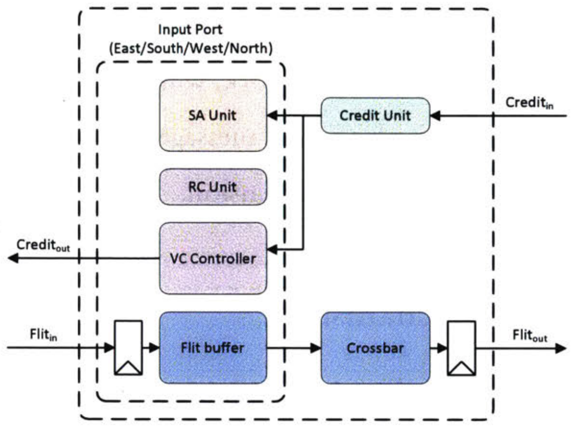


Figure 2-1: Router Microarchitecture

to its four neighboring directions: north (N), south (S), east (E), west (W), and a local or core port (L/C). Essentially, the router consists of input buffers, route computation logic, virtual channel selectors, switch allocators, and a crossbar. A typical router performs the following actions:

- **Buffer Write (BW):** Buffer the incoming flit.
- **Route Compute (RC):** If the incoming flit is the head of a packet, compute the route to determine the output port to depart from.
- **Switch Allocation (SA):** Arbitrate among buffered flits for the crossbar access as well as link access.
- **VC Selection (VS):** Select and reserve a VC at the next router from a pool of free VCs [63] for the head flit that won the SA.
- **Switch Traversal (ST):** Forward the flits that won the SA from their input ports to output ports.
- **Link Traversal (LT):** Forward the flits from the output ports to the next routers.

Depending on the clock frequency, the routers are typically pipelined into two or more stages to move from one router to another. Therefore, at the minimum, it takes two cycles to traverse one hop. In case of contention, flits may be buffered and hence take more cycles to move to the next router.

2.2 Low-Power Link - Low-Swing Signaling

Current on-chip network architectures require both long interconnects for the connection of processor cores, and small wire spacing for higher bandwidth. This trend has significantly increased wire capacitance and resistance. Unfortunately, physical properties of the on-chip interconnects are not scaling well with transistor sizes. In general, the low-swing technique can lower energy consumption and propagation delay but at the cost of a reduced noise margin [97]. Most existing low-swing on-chip interconnects (lower supply voltage drivers [97, 127], cut-off drivers [34, 126, 127] and charge sharing techniques [40, 68, 125]), however, are optimized for low-power signaling to maximize energy efficiency at the link level, leading to increase in propagation delay caused by reduced driving current. While pre-emphasis techniques such as equalization [41, 55, 77] can generate energy-efficient low-swing signaling along with the inherent channel loss of global links without sacrificing propagation speed, their application to an NoC with only relatively short router-to-router links, such as a mesh, is limited due to huge area overheads of the equalized drivers, poor bandwidth density of differential wiring and lack of point-to-point global wiring space.

Noise is one of the main concerns while using low-swing signaling techniques. Some of the noise concerns in low-swing designs can be mitigated by sending data differentially, which helps eliminate common-mode interference. However, this takes up two wires which doubles the capacitance and area. Adding shielding wires also helps reduce crosstalk and could potentially lower voltage-swing, but it also adds coupling capacitance and area. Increasing the sensitivity of the receiver helps lower voltage-swing on the wires, but it often needs a larger sized transistor or more sophisticated receiver design that has larger footprint and capacitance.

2.3 Low-Latency Link - Opto-Electrical Signaling

Recognizing the potential scaling limits of electrical interconnects, architects have proposed emerging nanophotonic technology as another option for both on-chip and off-chip networks [10, 66, 89, 113]. As optical links avoid capacitive, resistive and signal integrity constraints imposed upon electronics, photonics allows for ultra-low latency and efficient realization of physical connectivity that is costly to accomplish electrically.

2.3.1 Photonic Link

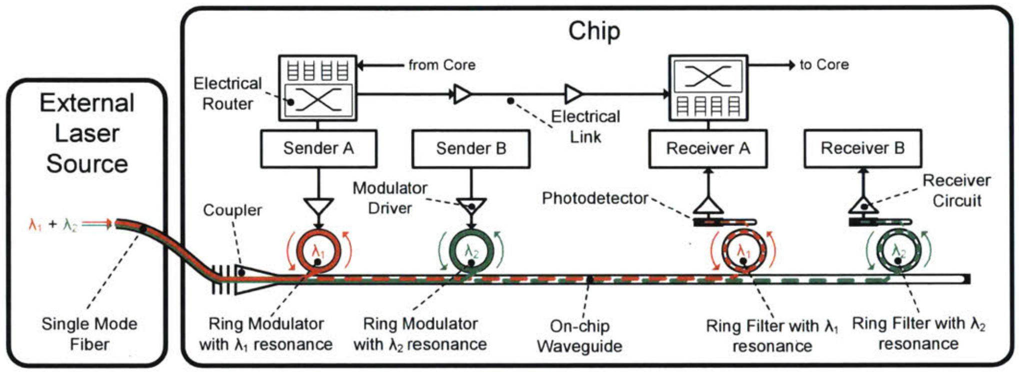


Figure 2-2: A typical opto-electronic NoC including electrical routers and links, and a wavelength devision multiplexed intra-chip photonic link

Waveguides, Couplers, and Lasers: Waveguides are the primary means of routing light within the confines of a chip. Vertical grating couplers [109] allow light to be directed both into and out-of the plane of the chip and provide the means to bring light from a fiber onto the chip or couple light from the chip into a fiber. In this dissertation (Chapter 3), we assume commercially available off-chip continuous wave lasers, though we note that integrated on-chip laser sources are also possible [45, 71].

Ring Resonators: The optical ring resonator is the primary component that enables on-chip wavelength division multiplexing (WDM). When coupled to a waveguide, rings perform as notch filters; wavelengths at resonance are trapped in the ring and can be potentially *dropped* onto another waveguide while wavelengths not at resonance pass by unaffected. The resonant wavelength of each ring can be controlled by adjusting the device geometry or the index of refraction. As resonances are highly sensitive to process mismatches and temperature, ring resonators require active thermal tuning [33].

Ring Modulators and Detectors: Ring modulators modulate its resonant wavelength by electrically influencing the index of refraction [96]. By moving a ring's resonance in and out of the laser wavelength, the light is modulated (on-off keyed). A photodetector, made of pure germanium or SiGe, converts optical power into electrical current, which can then be sensed by a receiver [32] and resolved to electrical ones and zeros. Photodetectors standalone are generally wideband and require ring filters for wavelength selection in WDM operation.

The dynamics of a wavelength-division-multiplexed (WDM) photonic architecture are shown in Figure 2-2. Wavelengths are provided by an external laser source and coupled into an on-chip waveguide. Each wavelength is modulated by a resonant ring modulator dropped at the receiver by a matching ring filter. Using WDM, a single waveguide can support dozens of independent data-streams on different wavelengths.

2.3.2 Prior Photonic NoC Architectures

Many photonics-augmented architectures have been proposed to address the interconnect scalability issue posed by rapidly rising core-counts, The Corona [113] architecture uses a global 64 $\times$ 64 optical crossbar with shared optical buses employing multiple matching ring modulators on the same waveguide. Firefly [89] and ATAC [66] also feature global crossbars, but with multiple matching receive rings on the same waveguide in a multi-drop bus configurations. The photonic clos network [50] replaces long electrical links characteristic of clos topologies with optical point-to-point links (one set of matching modulator and receiver ring per waveguide) and performs all switching electrically. Phastlane [23] and Columbia [39] networks use optical switches in tile-able mesh-like topologies.

2.4 Low-Latency and Low-Power Routers

A plethora of research in NoCs over the past decade coupled with technology scaling has allowed the actions within a router to move from serial execution to parallel execution, via lookahead routing [27], simplified VC selection [63], speculative switch arbitration [76, 82], non-speculative switch arbitration via lookaheads [58, 61, 62, 64, 91] to bypass buffering and so on. This has allowed the router delay to drop from 3 to 5 cycles in industry prototypes [42, 43] to 1-cycle in academic NoC-only prototypes [62, 91], resulting in 2-cycle-per-hop traversal.

2.5 Reconfigurable NoC Topologies

Prior works on reconfigurable NoCs motivated the need for application-specific topology reconfiguration and proposed various NoC architectures that support reconfiguration. Application-Aware Reconfigurable NoC [79] adds extra switches next to each router (a second crossbar in principle), and presets static routes based on application traffic. VIP [80] supports reconfiguration virtually, by prioritizing a virtual channel (VC) in the network to always get access to the crossbars, enabling single-cycle-per-hop for flits on this VC. ReNoC [105, 107] adds an extra topology switch (a set of muxes) at the output ports for each router and presets them to enable static routes in the network before the application is run. Skip-links [48] dynamically reconfigures the topology based on the traffic at each router when application is run, and sets up the crossbars to allow flits to bypass buffering and arbitration stages at intermediate routers.

2.6 In-network Coherence and Filtering

Various proposals, such as Token Coherence (TokenB), Uncorq, Time-stamp snooping (TS), and INSO extend snoopy coherence to unordered interconnects. TokenB [74] performs the ordering at the protocol level, with tokens that can be requested by a core wanting access to a cacheline. TokenB assigns T tokens to each block of shared memory during system initialization (where T is at least equal to the number of processors). Each cacheline requires an additional $2 + \log T$ bits. Although each token is small, the total area overhead scales linearly with the number of cachelines.

Uncorq [106] broadcasts a snoop request to all cores followed by a response message on a logical ring network to collect the responses from all cores. This enforces a serialization of requests to the same cacheline, but does not enforce sequential consistency or global ordering of all requests. Although read requests do not wait for the response messages to return, the write requests have to wait, with the waiting delay scaling linearly with core count, like physical rings.

TS [73] assigns logical time-stamps to requests and performs the reordering at the destination. Each request is tagged with an ordering time (OT), and each node maintains a guaranteed time (GT). When a node has received all packets with a particular OT, it increments the GT. TS requires a large number of buffers at the destinations to store all packets with a particular OT, prior to processing time. The required buffer count linearly scales with the number of cores and maximum outstanding requests per core. For a 36-core system with 2 outstanding requests per core, there will be 72 buffers at each node, which is not practical and will grow significantly with core count and more aggressive cores.

INSO [5] tags all requests with distinct numbers (snoop orders) that are unique to the originating node which assigns them. All nodes process requests in ascending order of the snoop orders and expect to process a request from each node, If a node does not inject a request, it is expected to periodically expire the snoop orders unique to itself. While a small expiration window is necessary for good performance, the increased number of expiry messages consume network power and bandwidth. Experiments with INSO show that the ratio of expiry messages to regular messages is about 25 for a time window of 20 cycles. At the destination, unused snoop orders still need to be processed leading to wasteful consumption of cycles and worsening of ordering latency.

3

DSENT - Design Space Exploration of Networks Tool

3.1 Motivation

With the rise of many-core chips that require substantial bandwidth from the NoC, integrated photonic links have been investigated as a promising alternative to traditional electrical interconnects [10, 50, 66, 89, 113], because photonic links avoid the capacitive, resistive and signal integrity constraints imposed upon electronics. Photonic technology, however, is still immature and there remains a great deal of uncertainty in its capabilities. Whereas there has been significant prior work on electronic NoC modeling (see Section 3.2), evaluations of photonic NoC architectures have thus-far not yet evolved past the use of fixed energy costs for photonic devices and interface circuitry, whose values also vary from study to study. To gauge the true potential of this emerging technology, inherent interactions between electronic/photonic components and their impact on the NoC need to be quantified.

In this chapter, we propose a unified framework for photonics and electronics, DSENT (Design Space Exploration of Networks Tool) [108], that enables rapid cross-hierarchical area and power evaluation of opto-electronic on-chip interconnects[1]. We design DSENT for two primary usage modes. When used standalone, DSENT functions as a fast

[1]We focus on the modeling of opto-electrical NoCs in this chapter, though naturally, DSENT's electrical models can also be applied to pure electrical NoCs as well

design space exploration tool capable of rapid power/area evaluation of hundreds of different network configurations, allowing for impractical or inefficient networks to be quickly identified and pruned before more detailed evaluation. When integrated with an architectural simulator [3, 78], DSENT can be used to generate traffic-dependent power-traces and area estimations for the network [67].

DSENT makes the following contributions:

- Presents the first tool that is able to capture the interactions at electronic/photonic interface and their implications on a photonic NoC.
- Proposes the first network-level modeling framework for electrical NoC components featuring integrated timing, area, and power models that are accurate (within 20 %) in the deep sub-100 nm regime.
- Identifies the most profitable opportunities for photonic network optimization in the context of an entire opto-electronic network system. In particular, we focus on the impact of network utilization, technology scaling and thermal tuning.

The rest of the chapter is organized as follows. Section 3.2 provides an overview on prior NoC modeling. We describe the DSENT framework in Section 3.3 and present its models for electrical and optical components in Section 3.4 and 3.5, respectively. Validation of DSENT is shown in Section 3.6. Section 3.7 presents an energy-efficiency-driven network case-study and Section 3.8 summarizes the chapter.

3.2 Existing NoC Modeling Tools

Several modeling tools have been proposed to estimate the timing, power and area of NoCs. Chien proposed a timing and area model for router components [22] that is curve-fitted to only one specific process. Peh and Dally proposed a timing model for router components [93] based on logical effort that is technology independent; however, only one size of each logic gate and no wire model is considered in its analysis. These tools also only estimate timing and area, but not power.

Among all the tools that provide power models for NoCs [8, 9, 51, 115], Orion [51, 115], which provides parametrized power and area models for routers and links, is the most widely used in the community. However, Orion lacks a delay model for router components, allowing router clock frequency to be set arbitrarily without impacting energy/cycle or area. Furthermore, Orion uses a fixed set of technology parameters and standard cell sizing, scaling the technology through a gate length scaling factor that does not reflect the effects of other technology parameters. For link components, Orion supports only limited delay-optimal repeated links. Orion does not model any optical components.

PhoenixSim [16] is the result of recent work in photonics modeling, improving the architectural visibility concerning the trade-offs of photonic networks. PhoenixSim provides parameterized models for photonic devices. However, PhoenixSim lacks electrical models, relying instead on Orion for all electrical routers and links. As a result, PhoenixSim uses fixed numbers for energy estimations for electrical interface circuitry, such as modulator drivers, receivers, and thermal tuning, losing many of the interesting dynamics when transistor technology, data rate, and tuning scenarios vary. PhoenixSim in particular does not capture trade-offs among photonic device and driver/receiver specifications that result in an area or power optimal configuration.

To address shortcomings of these existing tools, we propose DSENT to provide a unified electrical and optical framework that can be used to model system-scale aggressive electrical and opto-electronic NoCs in future technology nodes.

3.3 DSENT Framework

In our development of the generalized DSENT modeling framework, we observe the constant trade-offs between the amount of required user input and overall modeling accuracy. All-encompassing technology parameter sets can enable precise models, at the cost of becoming too cumbersome for predictive technologies where only basic technology parameters are available. Overly simplistic input requirements, on the other hand, leaves significant room for inaccuracies. In light of this, we design a framework that allows for a

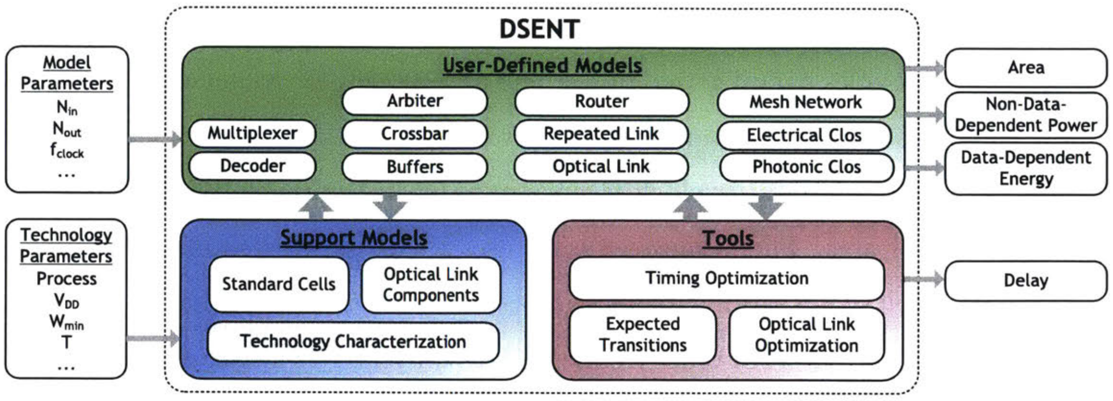


Figure 3-1: DSENT Framework with Examples of Network-related User-defined Models

high degree of modeling flexibility, using circuit- and logic-level techniques to simplify the set of input specifications without sacrificing modeling accuracy. In this section, we introduce the generalized DSENT framework and key features of our approach.

3.3.1 Framework Overview

DSENT is written in C++ and utilizes the object-oriented approach and inheritance for hierarchical modeling. The DSENT framework, shown in Figure 3-1, can be separated into three distinct parts: user-defined models, support models, and tools. To ease development of user-defined models, much of the inherent modeling complexity is off-loaded onto support models and tools. As such, most user-defined models involve just simple instantiation of support models, relying on tools to perform analysis and optimization. Like an actual electrical chip design, DSENT models can leverage instancing and multiplicity to reduce the amount of repetitive work and speed up model evaluation, though we leave open the option to allow, for example, all one thousand tiles of a thousand core system to be evaluated and optimized individually. Overall, we strive to keep the run-time of a DSENT evaluation to *a few* seconds, though this will vary based upon model size and complexity.

3.3.2 Power, Energy, and Area Breakdowns

The typical power breakdown of an opto-electronic NoC can be formulated as the following:

$$\begin{aligned} P_{\text{total}} &= P_{\text{electrical}} + P_{\text{optical}} \\ P_{\text{electrical}} &= P_{\text{router}} + P_{\text{link}} + P_{\text{interface}} + P_{\text{tuning}} \\ P_{\text{optical}} &= P_{\text{laser}} \end{aligned}$$

The optical power is the wall-plug laser power (lost through non-ideal laser efficiency and optical device losses). The electrical power consists of the power consumed by electrical routers and links as well as electric-optical interface circuits (drivers and receivers) and ring tuning. Power consumption can be split into *data-dependent* (DD) and *non-data-dependent* (NDD) parts. Non-data-dependent power is defined as power consumed regardless of utilization or idle times, such as leakage and un-gated clock power. Data-dependent power is utilization-dependent and can be calculated given an energy per each event and frequency of the event. Crossbar traversal, buffer read and buffer write are examples of high-level events for a router. Power consumption of a component can thus be written as $P = P_{\text{NDD}} + P_{\text{DD}} = P_{\text{NDD}} + \sum E_i f_i$, where P_{NDD} is the total non-data-dependent power of the module and E_i, f_i are the energy cost of an event and the frequency of an event, respectively.

Area estimates can be similarly broken down into their respective electrical (logic, wires, etc.) and optical (rings, waveguides, couplers, etc.) components. The total area is the sum of these components, with a further distinction made between active silicon area, per-layer wiring area, and photonic device area (if a separate photonic plane is used).

We note that while the area and non-data-dependent power can be estimated statically, the calculation for data-dependent power requires knowledge of the behavior and activities of the system. An architectural simulator can be used to supply the event counts at the network- or router-level, such as router or link traversals. Switching events at the gate- and transistor-level, however, are too low-level to be kept track of by these means, motivating a method to estimate transition probabilities (Section 3.4.4).

Table 3-1: DSENT Parameters

(a) Process (NMOS)

Parameter	**45 nm SOI**	**11 nm TG**	**Unit**
Nominal Supply Voltage (VDD)	1.0	0.6	V
Minimum Gate Width	150	40	nm
Contacted Gate Pitch	200	44	nm
Gate Capacitance / Width	1.0	2.42	fF/μm
Drain Capacitance / Width	0.6	1.15	fF/μm
Effective On Current / Width [84]	650	738	μA/μm
Single-transistor Off Current	200	100	nA/μm
Subthreshold Swing	100	80	mV/dec
DIBL	150	125	mV/V

(b) Interconnect (Global Wire Layer)

Parameter	**45 nm SOI**	**11 nm TG**	**Unit**
Minimum Wire Width	150	120	nm
Minimum Wire Spacing	150	120	nm
Wire Resistance (Min Pitch)	0.700	0.837	Ω/μm
Wire Capacitance (Min Pitch)	0.150	0.167	fF/μm
Resistivity	24.1	25.1	nΩ m
Wire Thickness	255	250	nm
Dielectric Thickness	250	220	nm
Dielectric Constant	2.76	2.76	

3.4 DSENT Models and Tools for Electronics

As the usage of standard cells is practically universal in modern digital design flows, detailed timing, leakage, and energy/op characterization at the standard-cell level can enable a high degree of modeling accuracy. Thus, given a set of technology parameters, DSENT constructs a standard cell library and uses this library to build models for the electrical network components, such as routers and repeated links.

3.4.1 Transistor Models

We strive to rely on only a minimal set of technology parameters (a sample of which is shown in Table 3-1) that captures the major characteristics of deep sub-100 nm technolo-

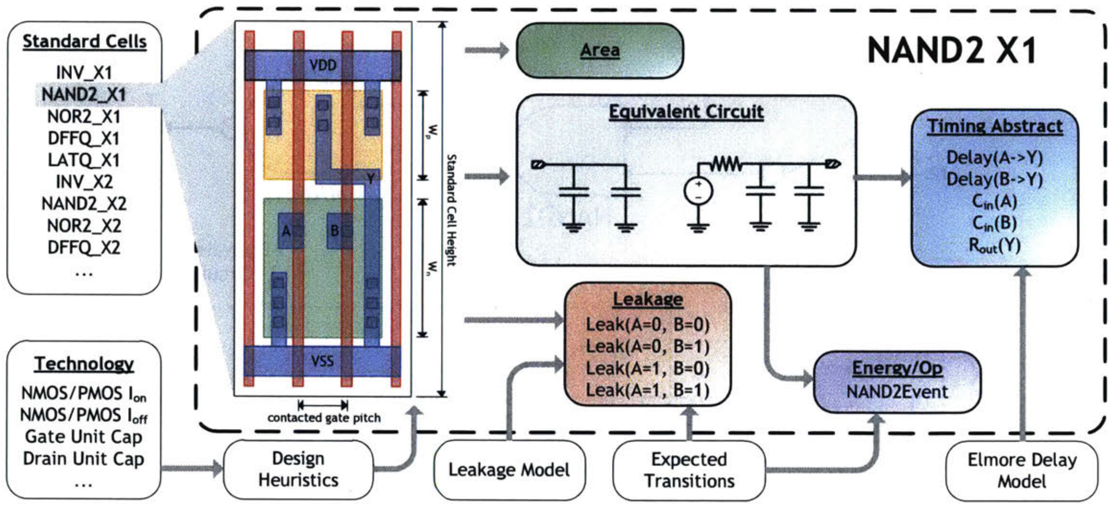


Figure 3-2: Standard cell model generation and characterization. In this example, a NAND2 standard cell is generated.

gies without diving into transistor modeling. Both interconnect and transistor properties are paramount at these nodes, as interconnect parasitics play an ever larger role due to poor scaling trends [95]. These parameters can be obtained and/or calibrated using ITRS roadmap projection tables [47] for predictive technologies, or characterized from SPICE models and process design kits when available.

Currently, DSENT supports the 45, 32, 22, 14 and 11 nm technology nodes. Technology parameters for the 45 nm node are extracted using SPICE models. Models for the 32 nm node and below are projected [53] using the virtual-source transport of [54] and the parasitic capacitance model of [118]. A switch from planar (bulk/SOI) to tri-gate transistors is made for the 14 and 11 nm nodes.

3.4.2 Standard Cells

The standard-cell models (Figure 3-2) are portable across technologies, and the library is constructed at run-time based on design heuristics extrapolated from open-source libraries [85] and calibrated with commercial standard cells.

We begin by picking a global standard cell height, $H = H_{\text{ex}} + \alpha(1 + \beta)W_{\text{min}}$, where β represents the P-to-N ratio, W_{min} is the minimum transistor width, and H_{ex} is the extra height needed to fit in supply rails and diffusion separation. α is heuristically picked such

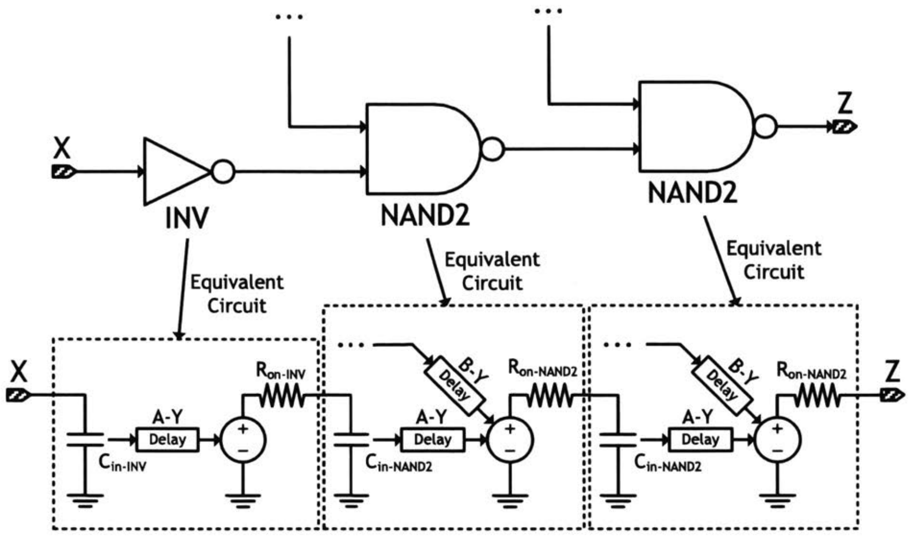


Figure 3-3: Mapping Standard Cells to RC Delays

that large (high driving strength) standard cells do not require an excessive number of transistor folds and small (low driving strength) cells do not waste too much active silicon area. For each standard cell, given a drive strength and function, we size transistors to match pull-up and pull-down strengths, folding if necessary. As lithography limitations at deep sub-100 nm force a fixed gate orientation and periodicity, the width of the cell is determined by the max of the number of NMOS or PMOS transistors multiplied by the contacted gate pitch, with an extra gate pitch added for separation between cells.

Currently, DSENT provides an essential set of standard cells that are commonly used in VLSI design, e.g., INV, BUF, NAND2, NOR2, LATQ, DFFQ, DFFRPQ, DFFSRPQ, MUX2, XOR2, and ADDF; DSENT also provides cells with various number of foldings ranged from 1 to 16.

3.4.3 Delay Calculation and Timing Optimization

To allow models to scale with transistor performance and clock frequency targets, we apply a first-order delay estimation and timing optimization method. Using timing information in the standard cell models, chains of logic are mapped to stages of resistance-capacitance (RC) trees, shown in Figure 3-3. An Elmore delay estimate [37, 97] between two points i and k can be formed by summing the product of each resistance and the total

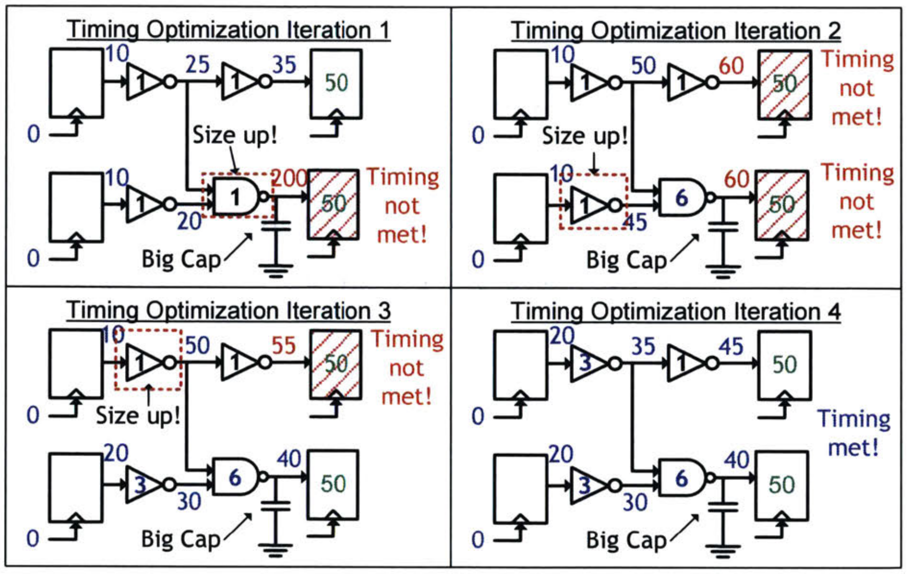


Figure 3-4: Incremental Timing Optimization

downstream capacitance it sees:

$$T_{\mathrm{d},i\rightarrow k} = ln(2)\sum_{n=i}^{k}\sum_{m=n}^{k} R_n C_m \tag{3.1}$$

Note that any resistances or capacitances due to wiring parasitics is automatically factored along the way. If a register-to-register delay constraint, such as one imposed by the clock period, is not satisfied, timing optimization is required to meet the delay target. To this end, we employ a greedy incremental timing optimization algorithm, as shown in Figure 3-4. We start with the identification of a critical path. Next, we find a node to optimize to improve the delay on the path, namely, a small gate driving a large output load. Finally, we size up that node and repeat these three steps until the delay constraint is met or if we realize that it is not possible and give up. Our method optimizes for minimum energy given a delay requirement, as opposed to logical-effort based approaches employed by existing models [15, 70, 93], which optimize for minimum delay, oblivious to energy. Though lacking the rigorousness of timing optimization algorithms used by commercial hardware synthesis tools, our approach runs fast and performs well given its simplicity.

3.4.4 Expected Transitions

The primary source of data-dependent energy consumption in CMOS devices comes from the charging and discharging of transistor gate and wiring capacitances. For every transition of a node with capacitance C to voltage V, we dissipate an energy of $E = \frac{1}{2}CV^2$. To calculate data-dependent power usage, we sum the energy dissipation of all such transitions multiplied by the clock frequency and activity factors, $P_{\text{DD}} = \sum \alpha_i C_i V_i^2 f_i$. Node capacitance C_i can be calculated for each model and, for digital logic, V_i is the supply voltage, f_i is the clock frequency and α_i is the activity factor. The frequency of occurrence, $\alpha_i f_i$, however, is much more difficult to estimate accurately as it depends on the pattern of bits flowing through the logic. As event counts and signal information at the logic gate level are generally not available except through structural netlist simulation, DSENT uses a simplified expected transition probability model [72] to estimate the average frequency of switching events. Probabilities derived using this model are also used with state-dependent leakage in the standard cells to form more accurate leakage calculations.

3.4.5 Summary

DSENT models a technology-portable set of standard cells from which larger electrical components such as routers and networks are constructed. Given a delay or frequency constraint, DSENT applies (1) timing optimization to size gates for energy-optimality and (2) expected transition propagation to accurately gauge the power consumption. These features allow DSENT to outpace Orion in estimating electrical components and in projecting trends for future technology nodes.

3.5 DSENT Models and Tools for Photonics

Chen Sun led the modeling of photonics devices briefly described in this section as background.

A complete on-chip photonic network consists of not only the photonic devices but also the electrical interface circuits and the tuning components, which are a significant fraction of the link energy cost. In this section we present how we model these components in DSENT.

3.5.1 Photonic Device Models

Similar to how it builds the electrical network model using standard cells, DSENT models a library of photonic devices necessary to build integrated photonic links. The library includes models for lasers, couplers, waveguides, ring resonators, modulators and detectors. The total laser power required at the laser source is the sum of the power needed by each photodetector after applying optical path losses:

$$P_{\text{laser}} = \sum P_{\text{sense},i} 10^{\text{loss}_i/10} \tag{3.2}$$

where $P_{\text{sense},i}$ is the laser power required at photodetector i and loss_i is the loss to that photodetector, given in dB. Note that additional link signal integrity penalties (such as near-channel crosstalk) are lumped into loss_i as well.

3.5.2 Interface Circuitry

The main interface circuits responsible for electrical-to-optical and optical-to-electrical conversion are the modulator drivers and receivers. The properties of these circuits affect not only their power consumption, but also the performance of the optical devices they control and hence the laser power [33].

Modulator Driver: We adopt the device models of [33] for a *carrier-depletion* modulator. We first find the amount of charge ΔQ that must be depleted to reach a target extinction ratio, insertion loss, and data rate. Using equations for a reverse-biased junction, we map this charge to a required reverse-biased drive voltage (V_{RB}) and calculate the effective capacitance using charge and drive voltage $C_{\text{eff}} = \Delta Q / V_{\text{RB}}$. Based on the data rate, we size a chain of buffers to drive C_{eff}. The overall energy cost for a modulator driver can be expressed as:

$$E_{\text{driver}} = \frac{1}{\gamma} \Delta Q \max(\text{VDD}, V_{\text{RB}}) + E_{\text{buf}}(C_{\text{eff}}, f) \tag{3.3}$$

where γ is the efficiency of generating a supply voltage of V_{RB} and $E_{\mathrm{buf}}(C_{\mathrm{eff}}, f)$ is the energy consumed by the chain of buffers that are sized to drive C_{eff} at a data rate f.

Receiver: We support both the TIA and integrating receiver topologies of [33]. For brevity, we focus the following discussion on the integrating receiver, which consists of a photodetector connected across the input terminals of a current sense-amplifier. Electrical power and area footprints of the sense-amplifier is calculated based on sense-amplifier sizing heuristics and scaled with technology, allowing calculation of switching power. To arrive at an expression for receiver sensitivity (P_{sense}), we begin with an abbreviated expression for the required voltage buildup necessary at the receiver sense amp's input terminal:

$$V_{\mathrm{d}} = V_{\mathrm{s}} + V_{\mathrm{os}} + V_{\mathrm{m}} + \Phi(\mathrm{BER})\sqrt{\sum \sigma_n^2} \tag{3.4}$$

which is the sum of the sense-amp minimum latching input swing (V_{s}), the sense-amp offset mismatch (V_{os}), a voltage margin (V_{m}), and all Gaussian noise sources multiplied by the number of standard deviations corresponding to the receiver bit error rate (BER). The required input can then be mapped to a required laser power requirement, P_{sense} at the photodetector:

$$P_{\mathrm{sense}} = \frac{1}{R_{\mathrm{pd}}} \frac{\mathrm{ER}}{\mathrm{ER} - 1} V_{\mathrm{d}} C_{\mathrm{in}} \frac{2f}{1 - 2fT_{\mathrm{j}}} \tag{3.5}$$

where R_{pd} is the photodetector responsivity (in terms of A/W), ER is the extinction ratio provided by the modulator, C_{in} is the total parasitic capacitance present at the receiver input node, f is the data rate of the receiver, and T_{j} is the clock uncertainty. The factor of 2 stems from the assumption that the photodetector current is given only half the clock period to integrate; the sense-amp spends the other half in the precharge state.

Serializer and Deserializer: DSENT provides models for a standard-cell-based serializer and deserializer (SerDes) blocks, following a mux/de-mux-tree topology [38]. These blocks provide the flexibility to run links and cores at different data rates, allowing for exploration of optimal data rates for both electrical and optical links.

3.5.3 Ring Tuning Models

An integrated WDM link relies upon ring resonators to perform channel selection. Sensitivity of ring resonances to ring dimensions and the index of refraction leaves them particularly vulnerable to process- and temperature-induced resonance mismatches [14, 86, 88], requiring active closed-loop tuning methods that add to system-wide power consumption [50]. In DSENT, we provide four models for four alternative ring tuning approaches [33]: *full-thermal tuning*, *bit-reshuffled tuning*, *electrically-assisted tuning*, and *athermal tuning*.

Full-thermal tuning is the conventional method of heating using resistive heaters to align their resonances to the desired wavelengths. Ring heating power is considered non-data-dependent, as thermal tune-in and tune-out times are too slow to be performed on a per-flit or per-packet basis and thus must remain always-on. Bit-reshufflers provide freedom in the bit-positions that each ring is responsible for, allowing rings to tune to its closest wavelength instead of a fixed absolute wavelength. This reduces ring heating power at the cost of additional multiplexing logic. Electrically-assisted tuning uses the resonance detuning principle of carrier-depletion modulators to shift ring resonances. Electrically-tuned rings do not consume non-data-dependent ring heating power, but is limited in tuning range and requires bit-reshufflers to make an impact. Note that tuning distances too large to be tuned electrically can still be bridged using heaters at the cost of non-data-dependent heating power. Athermal tuning represents an ideal scenario in which rings are not sensitive to temperature and all process mismatches have been compensated for during post-processing.

3.5.4 Optical Link Optimization

Equation 3.3 and 3.5 suggest that both the modulator driver's energy cost and the laser power required at the photodetector depend on the specification of extinction ratio (ER) and insertion loss (IL) of the modulator on the link. This specification can be used to tradeoff power consumption of the modulator driver circuit with that of the laser. This is an optimization degree of freedom that DSENT takes advantage of, looping through different combinations to find one that results in the lowest overall power consumption.

3.5.5 Summary

DSENT provides models not only for optical devices but also for the electrical backend circuitry including modulator driver, receiver and ring tuning circuits. These models enable link optimization and reveal tradeoffs between optical and electrical components that previous tools and analysis could not accomplish using fixed numbers.

3.6 Model Validation

We validate DSENT results against SPICE simulations for a few electrical and optical models. For the receiver and modulator models, we compare against a few early prototypes available in literature (fabricated at different technology nodes) to show that our results are numerically within the right range. We also compare our router models with a post-place-and-route SPICE simulation of a textbook virtual channel router and with the estimates produced by Orion2.0 [51] at the 45 nm SOI technology node. To be fair, we also report the results obtained from a modified Orion2.0 where we replaced Orion2.0's original scaling factors with characterized parameters for the 45 nm SOI node and calibrated its standard cells with those used to calibrate DSENT. Overall, the DSENT results for electrical models are accurate (within 20 %) compared to the SPICE simulation results. We note that the main source of inaccurate Orion2.0 results is from the inaccurate technology parameters, scaling factors, and standard cell sizing. The re-calibrated Orion2.0 reports estimations at the same order of the SPICE results. The remaining discrepancy is partly due to insufficient modeling detail in its circuit models. For example, pipeline registers on the datapath and the multiplexers necessary for register-based buffers are not completely modeled by Orion2.0.

Table 3-2: DSENT Validation Points

(a) Photonic Devices

Model	Ref. Point	DSENT	Unit	Config
Ring Modulator Driver	[29]-50	60.87 (21.74 %)	fJ/bit	11 Gb/s, ER = 10 dB, IL = 6 dB
Receiver	[32]-52	43.02 (− 14.0 %)	fJ/bit	3.5 Gb/s, 45 nm SOI

(b) Router

Model	Ref. Point	Orion2.0	Orion2.0 (re-calibrated)	DSENT	Unit	Config
Buffer	SPICE-6.93	34.4 (396 %)	3.57 (− 48.5 %)	7.55 (8.94 %)	mW	• 6 input/output ports
Crossbar	SPICE-2.14	14.5 (578 %)	1.26 (− 41.1 %)	2.06 (− 3.74 %)	mW	• 64 bit flit width
Control	SPICE-0.75	1.39 (85.3 %)	0.31 (− 58.7 %)	0.83 (10.7 %)	mW	• 8 VCs per port
Clock Dist.	SPICE-0.74	28.8 (3791 %)	0.36 (− 51.4 %)	0.63 (− 17.5 %)	mW	• 16 buffers per port
Total	SPICE-10.6	91.3 (761 %)	5.56 (− 47.5 %)	11.2 (5.66 %)	mW	• 1 GHz clock frequency
Total Area	Encounter-0.070	0.129 (84.3 %)	0.067 (− 4.29 %)	0.062 (− 11.2 %)	mm^2	• 0.16 flit injection rate

Table 3-3: Network Configuration

(a) Network

Parameter	**Value**
Number of tiles	256
Chip area (divided equally amongst tiles)	400 mm^2
Packet length	80 Bytes
Flit width	128 bits
Core frequency	2 GHz
Clos configuration (m, n, r)	16, 16, 16
Link latency	2 cycles
Link throughput	128 bits/core/cycle

(b) Router

Parameter	**Value**
Number pipelines stages	3
Number virtual channels (VC)	4
Number buffers per VC	4

3.7 Example Photonic Network Evaluation

Though photonic interconnects offer potential for improved network energy-efficiency, they are not without their drawbacks. In this section, we use DSENT to perform an energy-driven photonic network evaluation. We choose a 256-tile version of the 3-stage photonic clos network proposed by [50] as the network for these studies. Like [50], the core-to-ingress and egress-to-core links are electrical, whereas the ingress-to-middle and middle-to-egress links are photonic. The network configuration parameters are shown in Table 3-3. While DSENT includes a broader selection of network models, we choose this topology because there is an electrical network that is logically equivalent (an electrical clos) and carries a reasonable balance of photonic and electrical components. To obtain network-level event counts with which to animate DSENT's physical models, we implement the clos network in Garnet [3] as part of the gem5 [12] architecture simulator. Though the gem5 simulator is primarily used to benchmark real applications, we assume a uniform random traffic pattern to capture network energy at specific loads. Given network event counts, DSENT takes a few seconds to generate an estimation.

Table 3-4: Default Technology Parameters

Technology Parameters	Default Values
Process technology	11 nm TG
Optical link data rate	2 Gb/s
Laser efficiency	0.25
Coupler loss	2 dB
Waveguide loss	1 dB/cm
Ring drop loss	1 dB
Ring through loss	0.01 dB
Modulator loss (optimized)	0.01 to 10.0 dB
Modulator extinction (optimized)	0.01 to 10.0 dB
Photodetector Capacitance	5 fF
Link bit error rate	1×10^{-15}
Ring tuning model	Bit-Reshuffled [13, 33]
Ring heating efficiency	100 K/mW

Table 3-5: Sweep Parameters Organized by Section

Section	Sweep Parameter	Sweep Range
3.7.1	Electrical Process	45 nm SOI, 11 nm TG
3.7.2	Waveguide Loss	0.0 to 2.5 dB
	Ring Heating Efficiency	10 to 400 K/mW
3.7.3	Tuning Model	Full-Thermal, Bit-Reshuffled [13, 33], Electrically-Assisted [33]
	Link Data Rate	2 to 32 Gb/s per λ

In the following studies, we investigate the impact of different circuit and technology assumptions using energy cost per bit delivered by the network as our evaluation metric. Unless otherwise stated, the default parameters set in Table 3-4 are used. The parameters we sweep are organized by section in Table 3-5.

3.7.1 Scaling Electrical Technology and Utilization Tradeoff

We first compare the photonic clos network with an electrical equivalent, where all photonic links are replaced with electrical links of equal latency and throughput (128 wires, each at 2 GHz). We perform this comparison at the 45 nm SOI and 11 nm Tri-

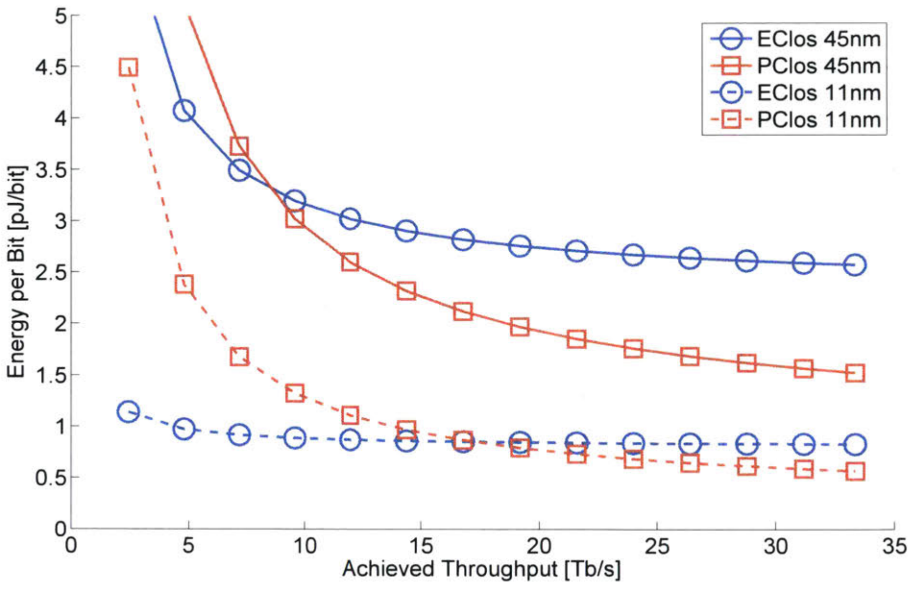
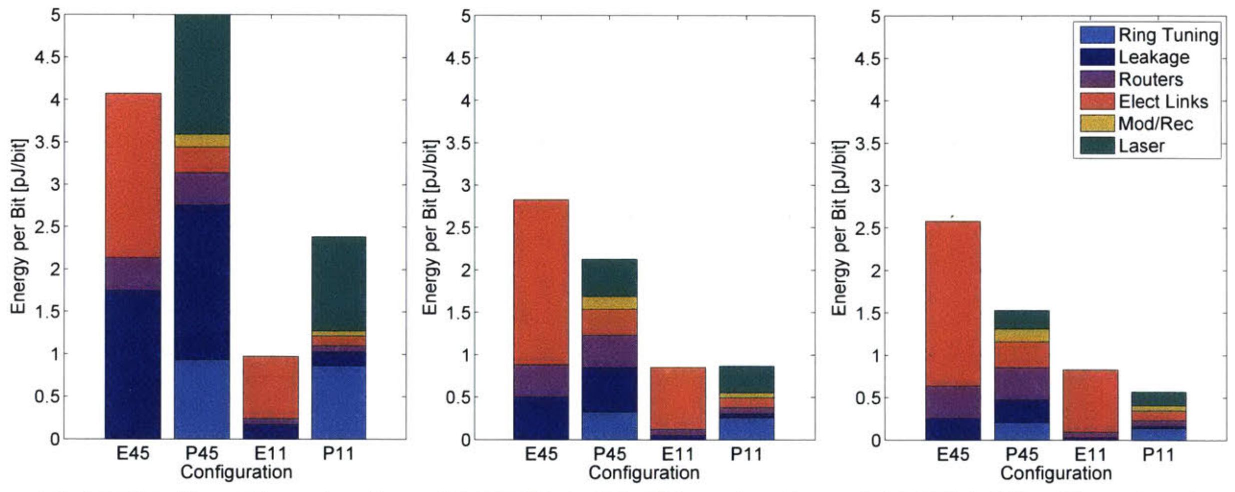


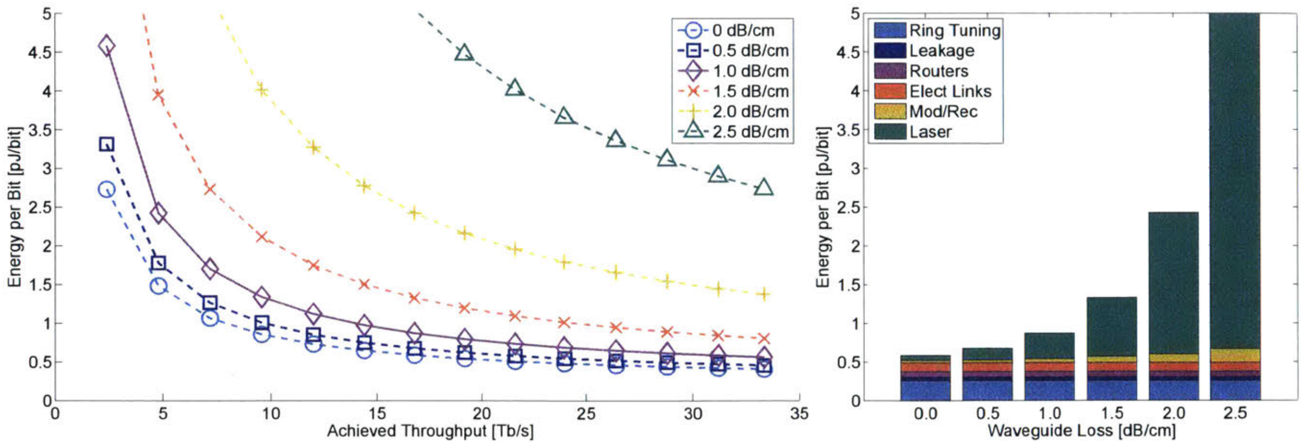
Figure 3-5: Comparison of Network Energy per bit vs. Network Throughput



(a) 4.5 Tb/s (Low Throughput) (b) 16.5 Tb/s (Med Throughput) (c) 33 Tb/s (Max Throughput)

Figure 3-6: Energy per bit Breakdown at Various Throughputs

Gate technology nodes, representing present and future electrical technology scenarios, respectively. Energy per bit is plotted as a function of achieved network throughput (utilization) in Figure 3-5 and a breakdown of the energy consumption at three specific throughputs is shown in Figure 3-6. The utilization is plotted up to the point where the network saturates, which is defined as when the latency reaches 3× the zero-load latency.



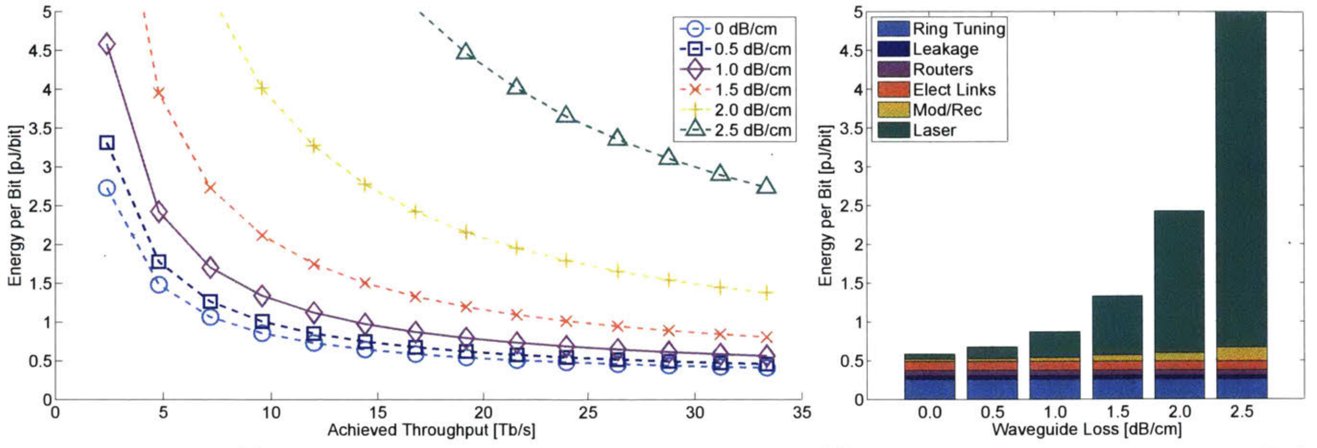
(a) Energy per bit vs. throughput

(b) Energy per bit Breakdown at 16 Tb/s Throughput

Figure 3-7: Sensitivity to Waveguide Loss

Note that in all configurations, the energy per bit rises sharply at low network utilizations, as non-data-dependent (NDD) power consumption (leakage, un-gated clocks, etc.) is amortized across fewer sent bits. This trend is more prominent in the photonic clos as opposed to the electrical clos due to a significantly higher NDD power stemming from the need to perform ring thermal tuning and to power the laser. As a result, the electrical clos becomes energy-optimal at low utilizations (Figure 3-6a). The photonic clos presents smaller data-dependent (DD) switching costs, however, and thus performs more efficiently at high utilization (Figure 3-6c).

Comparing 45 and 11 nm, it is apparent that both photonic and electrical clos networks benefit significantly from electrical scaling, as routers and logic become cheaper. Though wiring capacitance scales slowly with technology, link energies still scale due to a smaller supply voltage at 11 nm (0.6 V). Laser and thermal tuning cost, however, scale marginally, if at all, allowing the electrical clos implementation to benefit more. In the 11 nm scenario, the electrical clos is more efficient up to roughly half network of the saturation throughput. As networks are provisioned to not operate at high throughputs where contention delays are significant, energy efficiency at lower utilizations is critical.



(a) Energy per bit vs. throughput

(b) Energy per bit Breakdown at 16 Tb/s Throughput

Figure 3-8: Sensitivity to Heating Efficiency

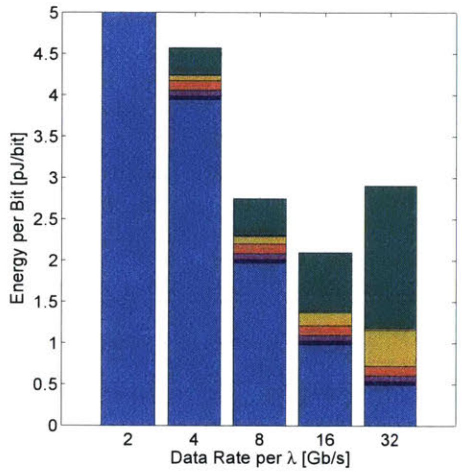
3.7.2 Photonics Parameter Scaling

For photonics to remain competitive with electrical alternatives at the 11 nm node and beyond, photonic links must similarly scale. The non-data-dependent laser and tuning power as particularly problematic, as they are consumed even when links are used sporadically.

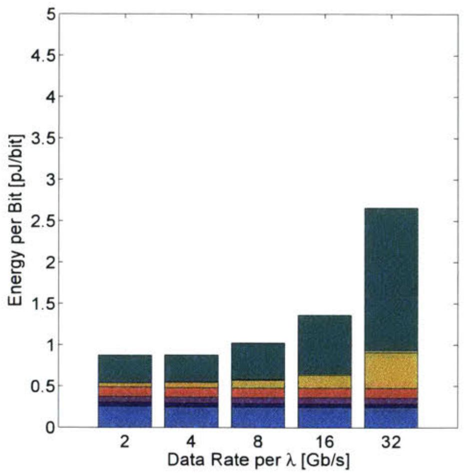
In Figure 3-7 and 3-8, we evaluate the sensitivity of the photonic clos to waveguide loss and ring heating efficiencies, which affect laser and tuning costs, using the 11 nm electrical technology model. We see that our initial loss assumption of 1 dB/cm brings the photonic clos quite close to the ideal (0 dB/cm) and the network could tolerate up to around 1.5 dB/cm before laser power grows out of proportion. Ring tuning power will also fall with better heating efficiency. However, it is not clear whether a 400 K/mW efficiency is physically realizable and it is necessary to consider potential alternatives.

3.7.3 Thermal Tuning and Data Rate

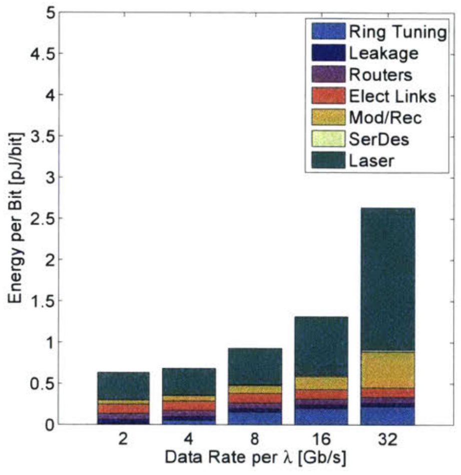
Per wavelength data rate of an optical link is a particularly interesting degree of freedom that network designers have control over. Given a fixed bandwidth that the link is responsible for, an increase in data rate per wavelength means a decrease in the number of WDM wavelengths required to support the throughput. In other words, since the



(a) Full-Thermal Tuning (conservative)



(b) Bit Reshuffled Tuning (default)



(c) Electrically-Assisted Tuning (optimistic)

Figure 3-9: Comparison of Thermal-Tuning Strategies at 16.5 Tb/s Throughput

throughput of each link is 128 bits/core/cycle at a 2 GHz core clock, a data rate of 2, 4, 8, 16 and 32 Gb/s per wavelength (λ) implies 128, 64, 32, 16 and 8 wavelengths per link. This affects the number of ring resonators and, as such, can impact the tuning power.

Under the more conservative full-thermal (no bit-reshuffling) tuning scenario (Figure 3-9a), the energy spent on ring heating is dominant and will scale proportionally with the number of WDM channels (and thus inversely with per wavelength data rate). Modulator and receiver energies, however, grow with data rate as a result of more aggressive circuits. Laser energy cost per bit grows with data rates due to a relaxation of modulator insertion loss/extinction ratios as well as clock uncertainty becoming a larger fraction of the receiver evaluation time. Routers and electrical links remain the same, though a small fraction of energy is consumed for serialization/deserialization (SerDes) at the optical link interface. These trends result in an optimal data rate between 8 to 16 Gb/s, where ring tuning power is balanced with other sources of energy consumption, given the full-thermal tuning scenario.

This trend is no longer true once bit-reshuffling (the default scenario we assumed for Section 3.7.1 and 3.7.2) is considered, shown in Figure 3-9b. Following the discussion in Section 3.5.3, a bit-reshuffler gives rings freedom in the channels they are allowed to tune to. At higher data rates, there are fewer WDM channels and hence rings that require tuning. However, the channel-to-channel separation (in wavelength) is also greater. Given the presence of random process variations, sparser channels means each ring requires, on average, more heating in order to align its resonance to a channel. These two effects cancel each other out. Since the bit-reshuffler logic itself consumes very little power at the 11 nm node, ring tuning costs are small and remain relatively flat with data rate.

If electrical-assistance is used (Figure 3-9c), tuning power favors high WDM channel counts (low data rates). This is a consequence of the limited resonance shift range that carrier-depletion-based electrical tuners can achieve. At high WDM channel counts where channel spacing is dense, rings can align themselves to a channel by electrically biasing the depletion-based tuner without a need to power up expensive heaters. By contrast, when channels are sparse, ring resonances will often have to be moved a distance too far for the depletion tuner to cover and costly heaters must be used to bridge the distance. As such, the lowest data rate, 2 Gb/s per wavelength, is optimal under this scenario. A well-designed electrically-assisted tuning system could completely eliminate non-data-dependent tuning power. Hence, it is a promising alternative to aggressive optimization of ring heating efficiencies.

3.8 Summary

Integrated photonic interconnects is an attractive interconnect technology for future many-core architectures. Though it promises significant advantages over electrical technology, evaluation of photonics in existing proposals have relied upon significant simplifications. To bring additional insight into the dynamic behavior of these active components, we developed a new tool, DSENT, to capture the interactions between photonics and electronics. By introducing standard-cell-based electrical models and interface circuit models, we complete the connection between photonic devices and the rest of the opto-electrical network. In addition to providing fast and accurate evaluations, DSENT keeps an essential set of technology parameters that can be easily obtained and updated for predictive technologies. Using our tool, we show that the energy-efficiency of a photonic NoC is poor at lower utilizations due to non-data-dependent laser and tuning power. We released DSENT open-source [30]. Till today, DSENT has been incorporated into gem5 [12] and, used significantly, e.g., [21, 57, 59, 65, 67, 116, 117].

4

Low-Power Crossbar Generator Tool

4.1 Motivation

Crossbar is the fundamental building block that connects input ports to output ports. A 1-bit $N\times M$ crossbar consists of $N\times M$ interconnected wires that are controlled by switches and enable any port to connect to any other port. The outputs of a crossbar connect to links that then connect to an IP block or a router. The crossbar and links thus together form the datapath of a NoC. Apart from the clocking power consumed by the buffers, the datapath dominates the NoC power consumption. Fabricated chips from academia, such as MIT RAW [111] and UT TRIPS [98], use RTL synthesis to generate the datapath, and the ratio of datapath power consumption and the total on-chip network power consumption are reported to be 69 and 64 %, respectively. Intel Teraflops [42] uses a custom-designed double-pumped crossbar with a location based channel driver to reduce the channel area and peak channel driver current [112], and is thus able to reduce datapath power to 32 % of the total on-chip network power. Other circuit techniques that have been proposed to reduce this power consumption involve dividing the crossbar wires into multiple segments and partially activating selected segments [69, 114] based on the input and output ports. These circuit techniques present only the capacitance between the input and output port, and disable/reduce other capacitances. They are thus successful in reducing wasteful power consumption. However, they still require complete charging/discharging of the long wires from the input port to the output port and the core-core links, which are significant power consumers.

Low-swing signaling techniques can help mitigate the wire power consumption. The energy benefits of low-swing signaling have been demonstrated on-chip from 10 mm equalized global wires [55], through 1 to 2 mm core-to-core links [99], to less than 1 mm within crossbars [62, 102, 120]. However, such low-swing signaling circuits, which can be viewed as analog circuits, require full custom design, resulting in substantial design time overhead. Circuit designers have to manually design schematic/netlists, optimize logic gates for each timing path, and size individual transistors. Moreover, layout engineers have to manually place all the transistors and route their nets with careful consideration of circuit symmetry and noise coupling. This custom design process leads to high development cost, long and uncertain verification timescales, and poor interface to other parts of a many-core chip, which are mostly RTL-based.

In the past, designers faced similar challenges while integrating low-power memory circuits with the VLSI CAD flow, with their sense amplifiers, self-timed circuits and dynamic circuits. Memory compilers, which are now commonplace, have solved the problem and enabled these sophisticated analog circuits to be automatically generated, subject to variable constraints specified by the users. This chapter proposes to similarly automate and generate low-swing signaling circuits as part of the datapath (crossbar and links) of a NoC, thereby integrating such circuits within the CAD flow of many-core chips, enabling their broad adoption.

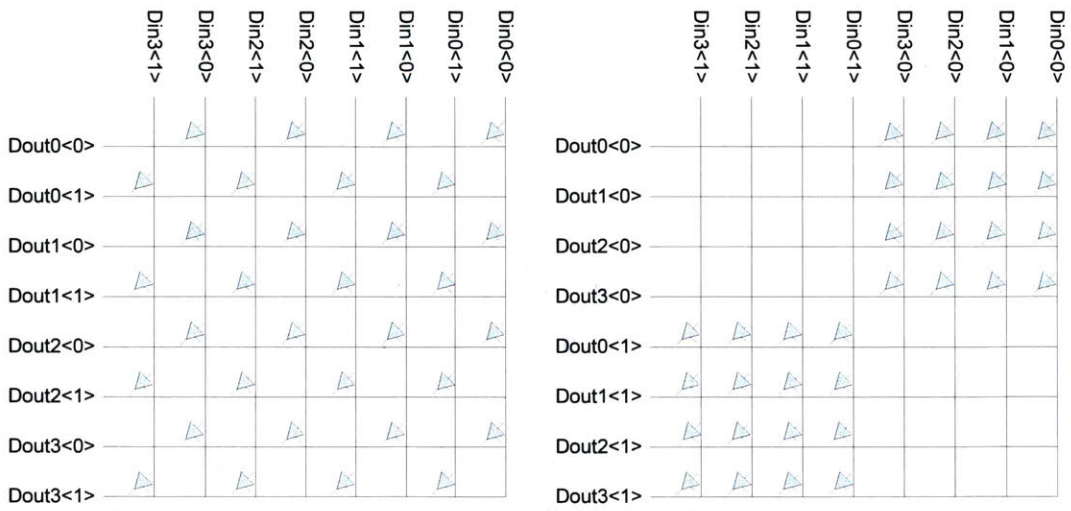
Since crossbars and links are such an essential component of on-chip networks, there have been efforts in the past to automate their generation. Sredojevic and Stojanovic [103] presented a framework for design-space exploration of equalized links, and a tool that generates an optimized transistor schematic. However, they rely on custom-design for the actual layout. ARM AMBA [7], STMicroelectronics STBus [104], Sonics MicroNetworks [122], and IBM CoreConnect [44] are examples of on-chip bus generators; DX-Gt [101] is a crossbar generator; and $\times$ pipes [26] is a network interface, switch and link generator. These tools are aimed at application specific network-on-chip (NoC) component generation, but they all stop at the synthesizable HDL level, i.e. they generate

RTL, and then rely on synthesis and place-and-route tools to generate the final design. This is not the most efficient way to design crossbars, as we show below in Section 4.4 highlighting that a synthesized crossbar design consumes significantly more power than a custom low-swing crossbar.

In this chapter we present a NoC datapath generator [17], which is the first to integrate low-swing links in an automated manner. It is also the first to generate a noise-robust layout at the same time, embedded within the synthesis flow of a 5-port NoC router in 45 nm SOI. Our tool takes a low-swing driver as input and ensures (1) a crosstalk noise-robust routing, (2) supply noise-robust differential signaling, and (3) crosstalk-controlled full-shielded links, in the generated datapath. To the best of our knowledge, our tool provides the following contributions to the low-power NoC community in the following important ways:

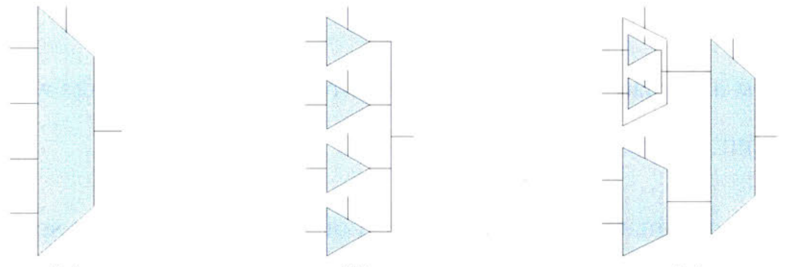
- It is the first automated generation of noise-robust low-swing links within the crossbar, and between routers.
- It is the first automated layout generation of a crossbar for a user specified number of ports, channel-width, and target frequency.
- It is the first demonstration of a generated low-swing crossbar and link within a fully-synthesized NoC router.
- Our automatically generated low-swing crossbar achieves an energy savings of 50 %, at the same targeted frequency of the synthesized crossbar, at 3 to 4 times the area overhead. Relative to the entire router, the larger footprint of the crossbar is manageable, at just 30 % of the overall router area.

The rest of the chapter is organized as follows. Section 4.2 presents some background on crossbars and low-swing link design. Section 4.3 explains our low-swing crossbar and link generator. Section 4.4 provides a case study of the datapaths generated using our tool, and Section 4.5 summarizes the chpater.



(a) Port-sliced Organization (b) Bit-sliced Organization

Figure 4-1: 2-bit 4 × 4 crossbar schematic



(a) (b) (c)

Figure 4-2: Logical 4:1 Multiplexer (a) and Two Realizations (b)(c)

4.2 Background

A $N \times M$ crossbar connects N inputs to M outputs with no intermediate stages, where any inputs can send data to any non-busy outputs. Figure 4-1 shows the schematic of a 2-bit 4 × 4 crossbar. In effect, a 1-bit $N \times M$ crossbar consists of M $N:1$ multiplexers, one for each output. The $N:1$ multiplexer can be realized as one logic gate or cascaded smaller $N':1$ multiplexers, where $N' < N$, as shown in Figure 4-2. A custom-circuit designer often favors the former implementation due to the layout regularity, as it enables various optimization techniques. However, this implementation suffers from the fact that the intrinsic delay of the multiplexer grows with N. Synthesis tools usually use the latter approach that cascades smaller multiplexers to implement a $N:1$ multiplexer with

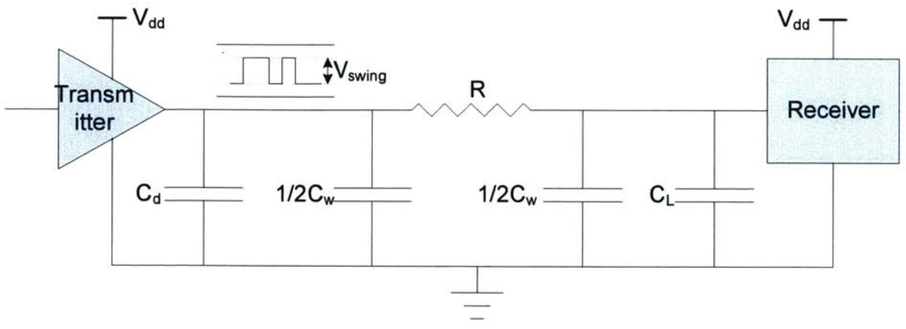


Figure 4-3: Simplified datapath

arbitrary N. By using this approach, the intrinsic delay grows with $\log N$ instead of N. However, it may lead to higher power consumption since more multiplexers are used.

Two gate organizations are possible for many-bit crossbars, as shown in Figure 4-1. One organization, port-slicing, groups all the bits of a port close to each other. The other organization, bit-slicing, groups all the gates of a bit together. The former approach eases routing (since all bits for an input/output port are grouped together), and minimizes the span of the control wires that operate the multiplexers for each input port. However, using the former approach leaves lot of blank spots that increases area, and folding the crossbar over itself to reduce area is non-trivial. The latter approach, on the other hand, minimizes the distance between the gates that contribute to the same output bit. This design is easier to optimize for area by placing all the bit-cells together and eliminating blank spaces, but requires more complicated routing to first spread out and then group all bits from a port.

In addition to a crossbar, links and receivers form a datapath. Different design decisions for these components would result in trade-offs in area, power and delay. From the perspective of sending a signal, a datapath can be simplified to three components connected together: a transmitter, a wire, and a receiver, as shown in Figure 4-3. The corresponding delay and energy consumption can be formulated as follows:

$$\text{Energy} = (C_\text{d} + C_\text{w} + C_\text{L})\text{VDD}V_\text{swing}$$

$$\text{Delay} = ((C_\text{d} + C_\text{w} + C_\text{L})V_\text{swing}/I_\text{av})$$

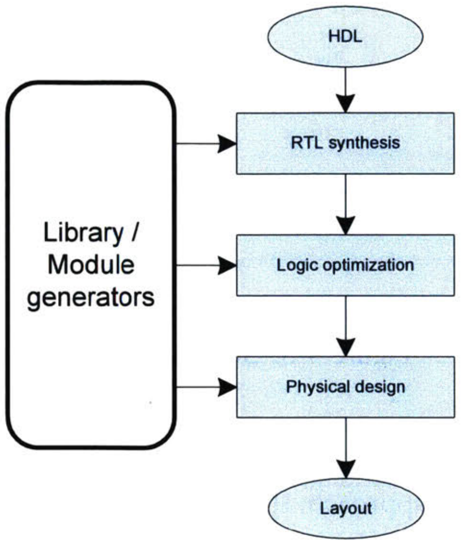


Figure 4-4: Standard synthesis flow

where C_d is the output capacitance of the transmitter, C_w is the wire capacitance, C_L is the input capacitance of the receiver. VDD is the power supply of the circuit, and V_{swing} is the voltage swing on these capacitors. I_{av} is the average (dis)charge current. In general, lowering the capacitance, reducing the voltage swing, and increasing the (dis)charging current can help in reducing energy consumption and delay.

A transmitter with larger sized transistors would have larger (dis)charging current which would decrease the delay. But it has larger footprint and C_d. Greater wire spacing lowers the coupling capacitance between wires but it takes larger metal area. Increasing wire width could reduce the wire resistance but it also increases capacitance and metal area.

4.2.1 Limitations to current synthesis flow

Given a hardware description of a crossbar, the existing synthesis flow, like the one shown in Figure 4-4, with a standard cell library could synthesize and realize a crossbar circuit. Unfortunately, the existing synthesis flow and standard cell libraries are designed for full voltage-swing digital circuits. New features in certain CAD tools enable low power designs by supporting multiple power domains and power shutdown techniques. However, none of them support analysis and layout for low voltage swing operations.

Table 4-1: Inputs to Proposed Datapath Generator

Type	Inputs
Architectural parameters	Number of input ports (N)
	Number of output ports (M)
	Data width in bits (W)
	Link length (L)
User preferences	Input port location
	Output port location
	Link wire width and spacing
Technology related information	Standard cell library
	Metal layer information
	Transmitter and receiver design
	Second power supply level (if needed)
System design constraints	Target frequency, power, area

Moreover, place-and-route tools are often too general and cannot take full advantage of the regularity of a crossbar and fail to generate an area-efficient layout. Therefore, a system designer needs to custom-design a low-swing crossbar, which is time-consuming and error-prone.

4.3 Datapath Generator

In this section we present our crossbar and link generator for low-swing datapaths. The low-swing property is enabled by replacing the cross-points of a crossbar with low-swing transmitters, and adding receivers at the end of the links to convert low-swing signals back to full-swing signals. The data links that connect transmitters and receivers are equipped with shielding wires to improve signal integrity. As shown in Table 4-1, our proposed datapath generator takes architectural parameters (e.g. the number of inputs and outputs, data width per port, link length), user layout preferences (e.g. port locations, link width and spacing), and technology files (e.g. standard cell library, targeted metal layers, TX and RX cells), and generates a crossbar and link layout that meets specified user preferences and system design constraints: area, power, and delay. The output files of our proposed datapath generator are fed directly into a conventional synthesis tool flow, which is similar

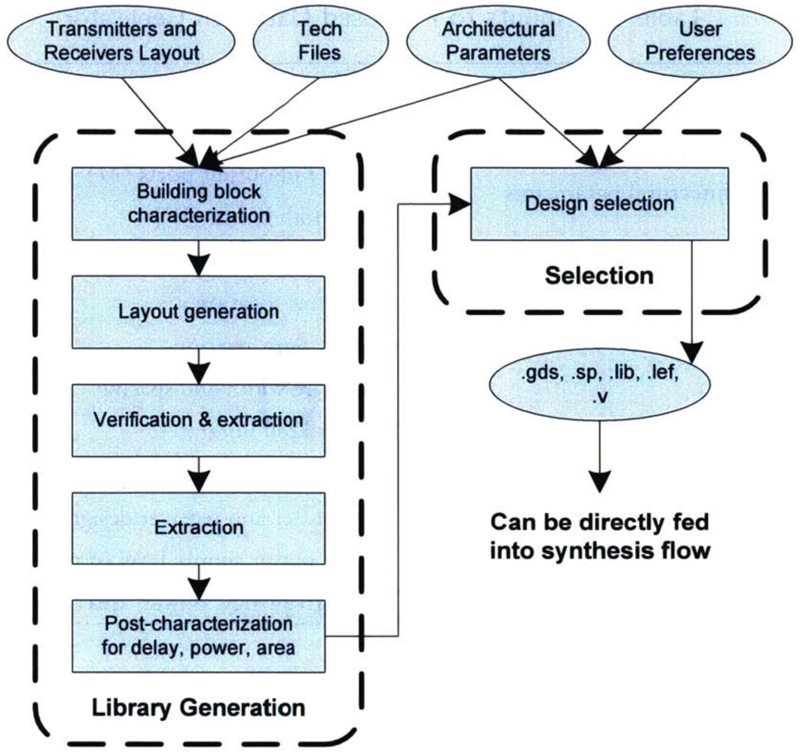
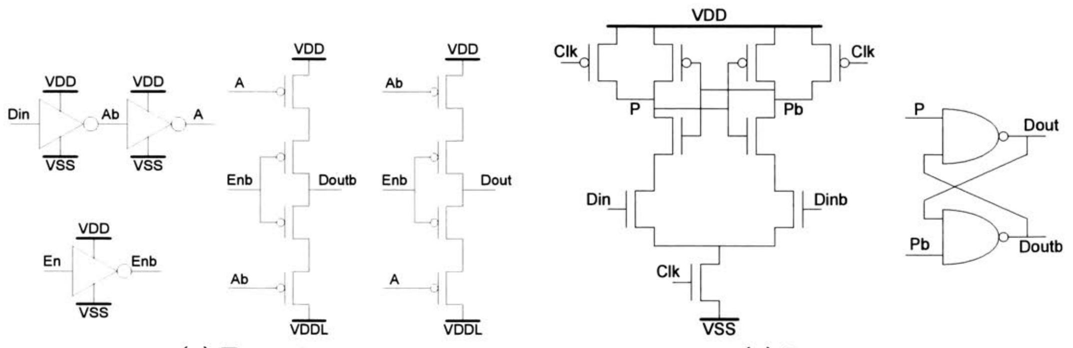

Transmitters and Receivers Layout
Tech Files
Architectural Parameters
User Preferences
Building block characterization
Layout generation
Verification & extraction
Extraction
Post-characterization for delay, power, area
Library Generation
Design selection
Selection
.gds, .sp, .lib, .lef, .v
Can be directly fed into synthesis flow

Figure 4-5: Proposed Datapath Generator's Tool Flow

to how we use a memory compiler. Figure 4-5 shows the proposed datapath generation flow. The generation involves two phases, library generation and selection. In the library generation phase, the program takes a suite of custom-designed transmitters and receivers, architectural parameters that users are interested in, and technology files as inputs; then, it pre-characterizes the custom circuits. Next, the tool generates the layout of all possible combinations and simulates them to get post-layout timing, power, and area. This forms the library of components for the selection phase. In the selection phase, the generator takes architectural parameters and user preferences as inputs to find the most suitable design from the results generated in the library generation phase, and outputs the files needed for the synthesis flow.

In the following subsections, we walk through a detailed example of generating a datapath with a 64-bit 6×6 crossbar, 1 mm links, and receivers in a 45 nm SOI HVT technology.



(a) Transmitter (b) Receiver

Figure 4-6: Schematic of Transmitter and Receiver

Table 4-2: Pre-characterization Results

	Transmitter	Receiver
Average current (μA)	2.6	11.0
Input cap (fF)	1.52 (select)	1.05 (clk)
	2.87 (data)	0.4 (data)

4.3.1 Building Block Pre-characterization

We treat the 1-bit transmitters and receivers as atomic building blocks of the generator, thus giving users the flexibility of using different kinds of transmitter and receiver designs. Given the transmitter and receiver designs, the generator first performs pre-characterization using SPICE-level simulators (we used Cadence UltraSim) to obtain average current and input capacitances. The average current is later used to determine the power wire width, while the input capacitances are used to determine the size of the buffers that drive these building blocks. For example, Figure 4-6 depicts the schematic of a low-swing transmitter design and a receiver design we chose as inputs to the generator [91]. The experiments in both this section and Section 4.4 are performed using the IBM 45 nm SOI HVT technology, and the pre-characterization results are shown in Table 4-2.

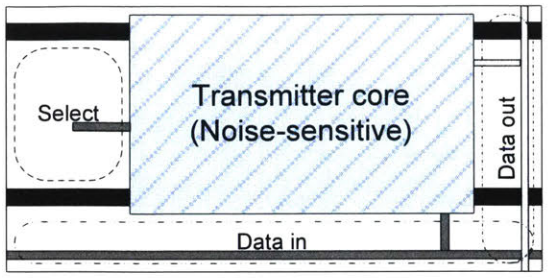


Figure 4-7: Transmitter Abstract Layout

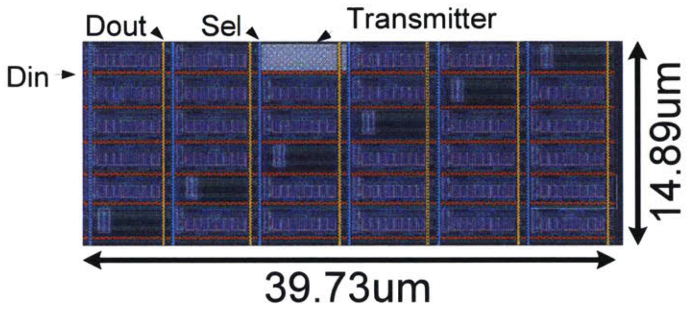


Figure 4-8: Example Single-bit Crossbar Layout with 6 Inputs and 6 Outputs

4.3.2 Layout Generation

In this step, the generator tiles the transmitters and receivers to form the datapath, taking various aspects into consideration, such as building block restrictions, floorplanning, routing, and link design. This section details each of these aspects.

Building block restrictions: We applied constraints to the transmitters' and receivers' pin locations. The reason is twofold. First, the gates of the transistors for low-swing operations are more sensitive to coupling from full-swing wires. Therefore, some constraints on transmitters' and receivers' pin location are helpful to avoid routing low-layer full-swing signal wires over these transistors. Second, constraints on pin locations make the transmitter/receiver cells more easily tile-able. Without loss of generality, we chose one specific pin layout, restricted as shown in Figure 4-7. The power and ground pins' locations are chosen to be the same as the corresponding pins in standard cells. All other pins are placed relative to the transmitter's core, which contains noise-sensitive transistors. For example, the Select pin is on the left of the core, the Data-in pin is at the bottom, and the Data-out pin is on the right. Similar constraints are also applied to the receiver cell design.

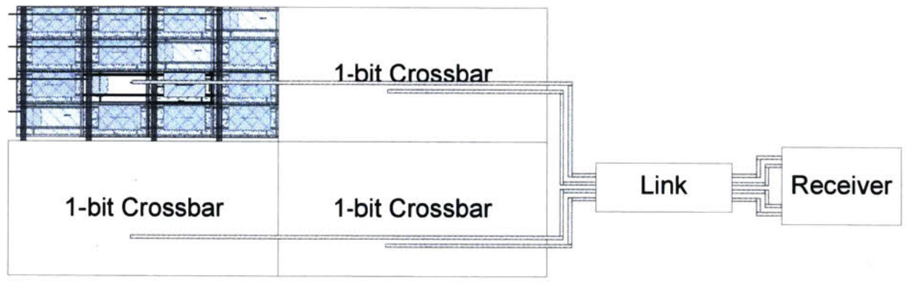


Figure 4-9: 4-bit Crossbar Abstract Layout with 1 Port Connecting to the Link

Floorplanning: To achieve higher transmitter cell area density, we chose the bit-sliced organization, which was shown earlier in Figure 4-1b. The tool first generates a 1-bit $N\times M$ crossbar as shown in Figure 4-8. The transmitters are placed at the cross-points of input horizontal wires and output vertical wires. The tool then places W 1-bit crossbars in a 2-dimensional array to form a W-bit $N\times M$ crossbar, as shown in Figure 4-9. The number of 1-bit crossbars on each side is calculated to square the crossbar layout area so as to minimize the average length of the wires each bit needs to traverse. Receivers are placed so that the routing area from the links to the receiver inputs is minimal.

Although a port-sliced organization is also effective, it requires a more sophisticated wire routing algorithm to achieve the same cell area density as a bit-sliced organization. A naive approach, as shown in Figure 4-1a, would result in low-transistor density and a W^2 bit-to-area relationship, instead of W which can be readily achieved by using the bit-sliced organization.

Routing: For each 1-bit crossbar, the number of metal layers needed to route the power and signals is kept minimal, to maximize the number of available metal layers for output wire routing. No wiring is allowed above noise-sensitive transistors in lower metal layers. While this increases the total crossbar area, it lowers the wiring complexity for Data-out wires from each 1-bit crossbar to crossbar outputs. Since we employed the bit-sliced organization, the Data-out wires are distributed across the entire crossbar. Two metal layers are used to route the Data-out wires to the edge of the crossbar: one is used for outputs in horizontal direction, while the other is used for the vertical direction. Since the same metal layer is used to route all wires in a particular direction, the crossbar area is limited by the wire pitch if the transmitter's cell area is small. Otherwise, it is limited

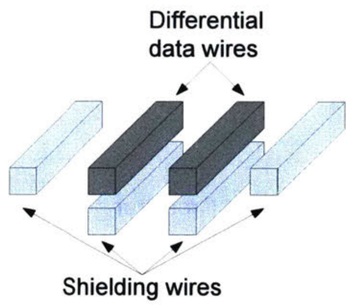


Figure 4-10: Selected Wire Shielding Topology

by transmitter cell area. As shown in Figure 4-9, Data-out wires coming out from the edge of the 1-bit crossbar array are routed to the inputs of links. We carefully designed the routing algorithm so that it takes minimal wiring area to connect the outputs of a crossbar to the links.

A structured layout of the power distribution network is applied. A power ring that surrounds the whole crossbar, one that surrounds the whole receiver block, and power stripes, are all automatically generated. The widths of the power wires are calculated based on the average current so that the current density is less than 1 mA/µm to avoid electromigration. Using the results from the pre-characterization, we used both 0.8 µm-wide and 0.7 µm-wide power wire for crossbar and receiver respectively.

Link Design: Link parameters such as link wire length, width, and spacing are specified as the inputs of the generator. Since the links are running at low-swing, they are more vulnerable to noise. We thus add shielding wires to improve the noise immunity at the cost of increase in link area[1]. We chose the shielding wire organization that is shown in Figure 4-10, where a shielding wire is placed on the same layer as link between two different bits and two shielding wires are placed right below the differential wires. This is chosen as it minimizes low-swing noise from other links and full-swing logic from lower metal layers.

Typically the wire length is set based on the distance between the crossbar and the components this crossbar is connected to. Different choices of wire width and spacing would affect the timing and energy consumption of transmitting a signal. For example,

[1]The area cost is around 1.5× on the same layer and 1× on the layer below.

Table 4-3: Performance of 1 mm Link of Two Organizations

Wire width	Wire spacing	Delay (ps)	Energy/bit (fJ)	Link area (mm^2)
1	2	70.0	35.0	0.093
2	4	33.7	30.5	0.176

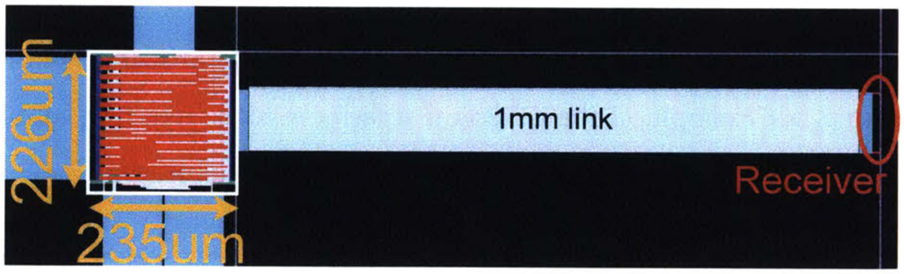


Figure 4-11: Example 6 × 6 64-bit Datapath Layout with One Link Shown

one could reduce the delay by doubling the wire pitch but it requires larger wiring area. Table 4-3 shows this trade-offs between link area and link performance, where the wire width is normalized to the minimum wire width and the wire spacing is normalized to the minimum spacing. The performance was simulated by transmitting a full-swing signal on the link.

A layout of the example datapath generated is shown in Figure 4-11.

4.3.3 Verification and Extraction

We use Calibre from Mentor Graphics to check if the generated circuit obeys the design rules, and to perform layout versus schematic (LVS) verification. A schematic netlist is generated for LVS. In order to get a more accurate delay of the circuit, RC extraction is done for the post-characterization of the generated design.

4.3.4 Post-characterization and Selection

Post-characterization is performed to determine the actual frequency, power, and area the crossbar can achieve. The selection step chooses the suitable datapath design based on the results from the post-characterization step, and outputs the files needed for the standard synthesis flow.

Table 4-4: Example Generated Datapaths

Link wire width	Link wire spacing	Max freq (GHz)	Crossbar area (mm^2)	Energy/bit (fJ)
1	2	2.5	0.053	46.4
2	4	2.7	0.084	48.3

The Table 4-4 shows the simulation results for the walk-through examples. At the selection step, for example, if the criteria is to achieve high frequency and have little constraint on the area, the design with doubled link pitch is returned.

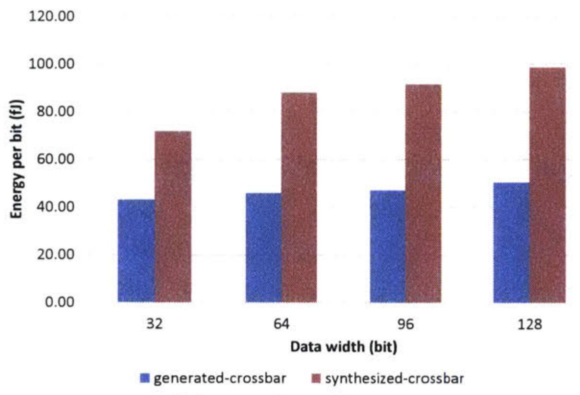
4.4 Evaluation

In this section, we first evaluate the crossbars generated by our proposed tool, against the synthesized crossbars. We then present a case study of a 5-port NoC virtual channel router that is integrated through the standard synthesis flow with the low-swing datapath generated by our tool.

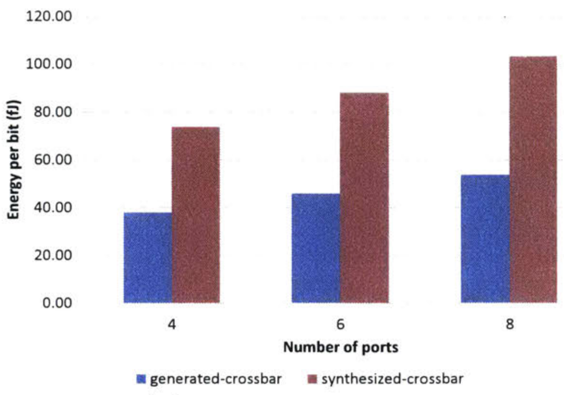
In all our experiments, we used Cadence Ultrasim to evaluate the performance and power consumption of the RC extracted netlists.

4.4.1 Generated vs. Synthesized Datapath

Using the transmitter and the receiver design we describe in Section 4.3, we generated low-swing datapaths across a range of architectural parameters and compared the simulation results with datapaths generated by standard CAD tools using only standard cells. We will refer to the crossbar/datapaths generated by our tool as *generated* crossbars/datapaths, and those generated by standard CAD tools using standard cells as *synthesized* crossbars/datapaths. Evaluating generated datapaths with different transmitter and receiver designs can be done but is equivalent to evaluating the effectiveness of different low-swing techniques, which is beyond the scope of this work. In our experiments, we assumed a link length of 1 mm and specified a delay constraint of 0.6 ns from the input of the crossbar to the output of the link for synthesized datapaths.



(a) Vary Data Widths



(b) Vary Number of Ports

Figure 4-12: Energy per bit Sent of 64-bit Datapaths

Energy per bit: We simulated the datapaths (crossbar and link) at 1.5 GHz and report the results for varying data widths and varying number of ports in Figure 4-12a and Figure 4-12b, respectively. As shown in Figure 4-12a, for both crossbars, as the data width increases, the energy per bit sent also increases because an increase in the data width leads to an increase in the area of the crossbar. This increase results in longer distance (on average) for a bit to travel from an input port to an output port. Longer distance translates to higher energy consumption. The energy per bit sent also increases with the number of ports, because a bit needs to drive more transmitters. Overall, our simulations showed that a generated datapath, as in our design, results in 50 % energy savings (on average per bit sent) compared to a synthesized datapath.

Area: Figure 4-13 shows the area of the generated vs. synthesized crossbars. Due to the bit-sliced organization and larger transmitter size, the generated crossbar area is dominated by the transmitter area. Using this organization results in its crossbar area growing linearly with the data width and quadratically with the number of ports, as captured in Figure 4-13. On the other hand, as Figure 4-13 indicates, a synthesized crossbar has a smaller area footprint because the transmitter design we are simulating is differential, and our wire routing is conservative to achieve high immunity to noise. Both of these factors result in increased area footprint.

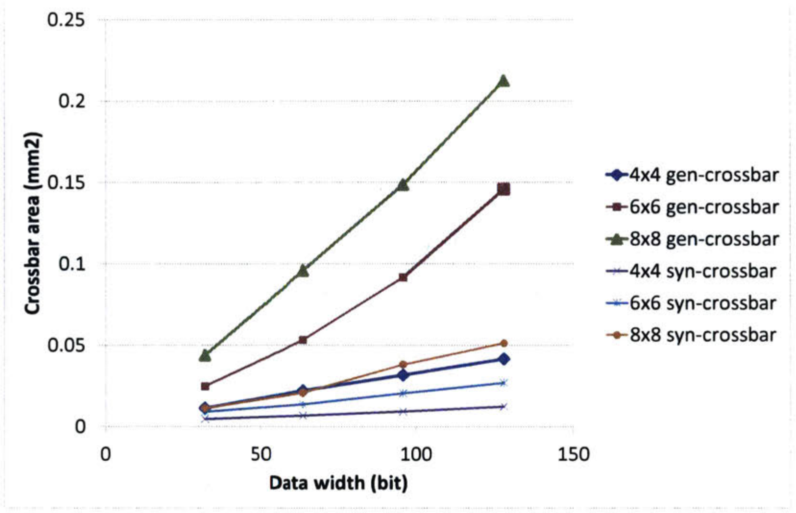


Figure 4-13: Crossbar Area with Various Architectural Parameters

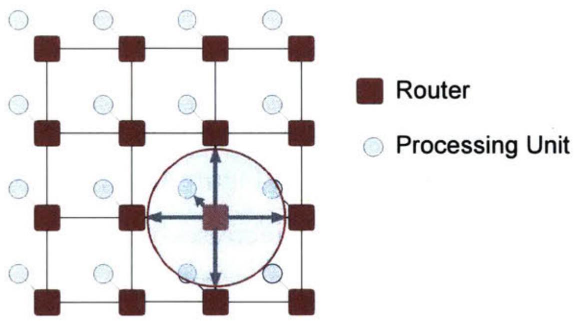


Figure 4-14: Five-port Router in a Mesh Network

4.4.2 Case Study

We synthesized a typical NoC router of a mesh topology integrated with a low-swing datapath using the files generated by our tool. The router is a 3-stage pipelined input

Table 4-5: Router Specifications

Parameter	Value
# of input ports	5
# of output ports	5
Data width	64
# of buffers per port	16 (1k bits)
Flow control	Wormhole with VC
Buffer management	On/Off
Working frequency	1 GHz

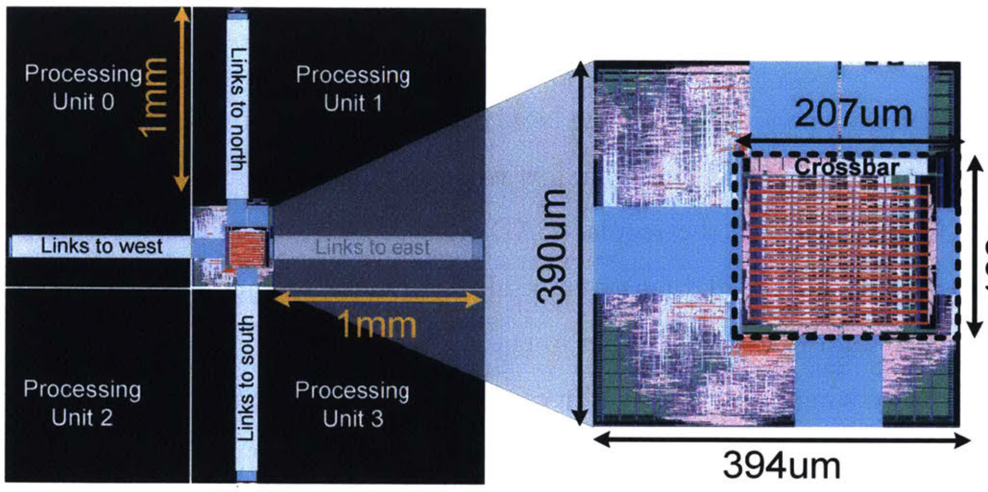


Figure 4-15: Synthesized Router with Generated Low-swing Datapath

buffered virtual channel (VC) router with five inputs and five outputs [27], and with a 64-bit data path. As shown in Figure 4-14, one input and one output port are connected to the local processing unit, while the remaining ports are connected to neighboring routers. We assumed that the local processing unit resides next to the router, the distance between routers is 1 mm, and the target working frequency is 1 GHz. Table 4-5 shows the detailed router specifications.

We used the same synthesis flow shown in Figure 4-4 to realize the router design from RTL to layout. Figure 4-15 shows the final layout of the router with the generated datapath. The black region in the figure is assumed to be occupied by processing units. The low-swing crossbar occupies about 30 % of the total router area. The delay of the low-swing datapath is 630 ps. The power consumed in the generated datapath is 18 % of the total power consumed by the router[2]. The power consumption was obtained from UltraSim simulations by feeding a traffic trace through all the ports of the router. The traffic trace was generated from RTL simulations of a 4×4 NoC; every node injects one message every cycle destined to a random node. The final synthesized router with the generated low-swing crossbar and links consists of 286,079 transistors.

[2]It should be pointed out that this is a textbook NoC router. With a bypassing NoC router, such as that in [62], the NoC power will be largely that of the datapath, since most packets need not be buffered and can go straight from the input port through the crossbar to the output port and link.

4.5 Summary

In this chapter, we present a low-swing NoC datapath generator that automatically creates layouts of crossbar and link circuits at low voltage swings, enabling the ready integration of such interconnects in the regular CAD flow of manycore chips. Our case study demonstrates our generated datapath embedded within the synthesis flow of a 5-port NoC mesh router, leading to 50 % savings in energy-per-bit. While our case study leverages a specific low-swing transmitter and receiver circuit, our generator can work with any TX/RX building block.

5 SMART - Low-Latency Network Generator Tool for SoC Applications

5.1 Motivation

Systems-on-Chip (SoCs) have started adding more and more general-purpose/application-specific IP cores with the emergence of diverse compute intensive applications over the past few years [35, 52], and this has intensified with the proliferation of smart phones [123]. Networks-on-chip (NoCs) are used to connect these cores together, and routers are used at crosspoints of shared links to perform multiplexing of different messages flows on the links.

To reduce on-chip packet delivery latency, one proposed approach is to tailor the NoC topology to match application communication patterns at *design time*. Examples include Fat Tree [1], Star-Ring [56], Octegon [52] and high-radix crossbar [92], etc. If coupled with sophisticated link designs such as [41, 55, 60, 77], these NoCs can realize a single cycle transmission between distant cores. However, this requires knowledge of all applications and their communication graphs at design time to be able to pin these dedicated express links to specific pairs of dedicated cores, and assumes sufficient wiring density to support dedicated links between all communicating cores.

The alternate approach is proposed to use a scalable topology at design time, such as a 2D Mesh connecting a collection of generic IPs (such as ARM processors), then

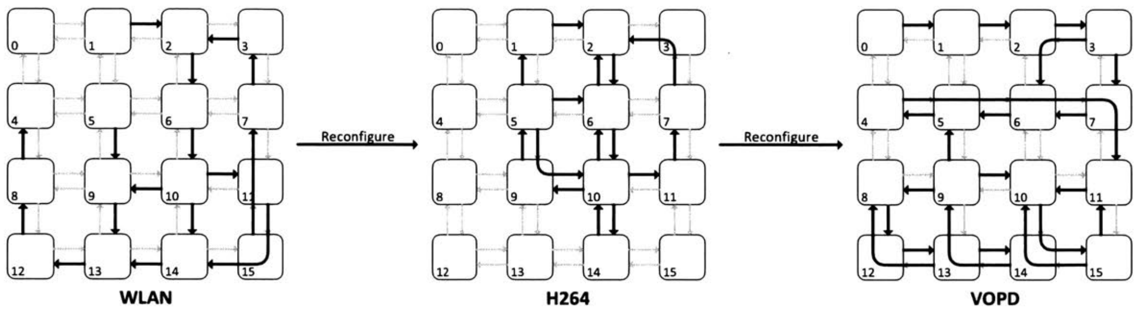


Figure 5-1: Mesh Reconfiguration for Three Applications. All links in bold take one-cycle.

reconfigure it at *run time* to match application traffic. Since router delays can vary depending on congestion [27, 35], some prior research [48, 79, 80, 105, 107] has proposed pre-reservation of (parts of) the route to provide predictable and bounded delays. These works perform an *offline* computation of contention free routes, allowing flits to bypass queues and arbiters at routers where there is no conflict between the routes of different flows.

In this chapter, we propose SMART, Single-cycle Multi-hop Asynchronous Traversal, to enable flits to potentially incur a single-cycle delay all the way from the source to destination, thus providing a virtually tailored topology within a shared mesh. In addition, we present a tool flow that (1) generates the RTL netlist of SMART network and brings it to layout, and (2) takes applications' task graphs with communication flows and generates configurations for tailored topologies. Figure 5-1 shows the goal of our design, where a network reconfigures into 3 different topologies for 3 different applications.

We make the following contributions:

- We propose SMART network that allows flits traversing multiple hops within a single cycle, breaking the on-chip latency barrier (i.e., one cycle per hop).
- We present a tool flow that takes SoC application task graphs, maps each application onto the multi-core fabric, reconfigures the underlying SMART NoC so that it is customized for the SoC application. The tool also provides parameterized RTL netlist for SMART network that can be synthesized and place-and-routed into layout.
- We use the tool to implement a 4 × 4 SMART mesh network and evaluate the impact on multiple SoC applications, showing that it is only 1.5 cycles off in performance

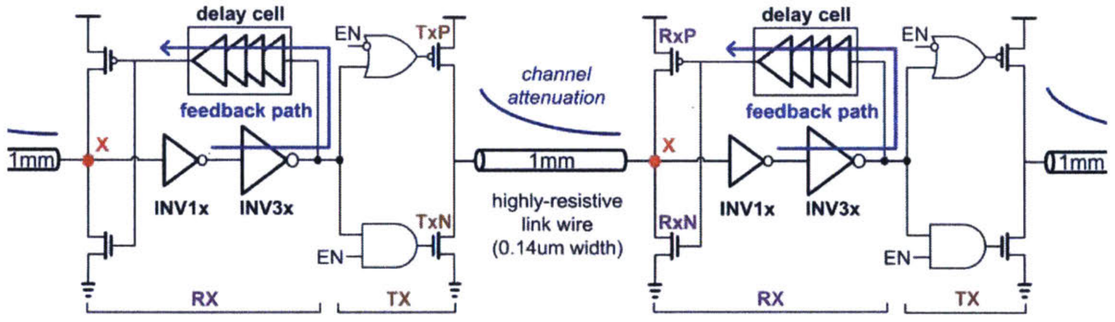


Figure 5-2: VLR Schematic

from a *dedicated* topology for that application. Compared to a mesh network with 3-cycle router, we observed 60 % saving in packet delivery latency and 2.2× reduction in power consumption.

The chapter is organized as follows: we first show the feasibility of traversing multiple within a single cycle in Section 5.2. Then, we present the SMART network architecture in Section 5.3. In Section 5.4 describes the tool flow we develop to power the SMART network. And case studies on a 4 × 4 SMART network are shown in Section 5.5. We summarize the chapter in Section 5.6.

5.2 Background - Clockless Repeated Links

As discussed in Section 2.5, most prior works explore single-cycle-per-hop, which can be viewed as a long link connecting the source and destination router with a clocked repeater inserted each router on the route. However, the actual wire delay of a link between adjacent routers is much shorter than a typical router cycle time (0.5 to 1 ns), which means that it is possible to replace the clocked repeaters with clockless repeaters and allow flits or packets to traverse multiple hops in a single cycle.

We explore the low-swing signaling techniques that can be used to lower both energy consumption and propagation delay, where lower propagation delay implies higher number of hops that can be traversed in a cycle. However, typical low-swing signaling techniques described in Section 2.2 require a clocked receiver and hence are not suitable for our purpose, which requires an asynchronous repeater.

Park proposed the voltage lock repeater (VLR) [90], a low-swing clockless repeater that stretches the maximum distance a full-swing repeated link can span in a cycle at lower energy. In this section, we briefly introduce the mechanism and measurement results of the VLR, and re-optimize the circuit to meet our need.

Figure 5-2 shows the schematic of VLR. A single-ended design is chosen over double-ended design because of lower-wire capacitance per bit and higher data density. The low-swing property is achieved by locking the node X to swing near the threshold voltage of INV1x without decreasing the driving current, enabling lower delay of the next symbol propagation delay in link. The voltage swing level is determined by transistor sizes and link wire impedance[1]. The delay cell in the feedback path generates transient overshoots at the node X, resulting in lower repeater intrinsic delay and larger noise margin without significant energy overhead. Careful transistor sizing and extracted simulations are required to prevent oscillation and static current through the RxP-RxN path in all possible process corners.

Even though VLR does not require clocking power and differential signaling, it has static current paths between two consecutive repeaters, TxP-wire-RxN for logic high and TxN-wire-RxP for logic low. It should be noted, however, that the static energy is much less than a conventional continuous-time comparator since the static current paths include a highly-resistive link wire. Also, switching off the enable signal (EN) when the link is not used helps eliminate unnecessary static power.

[90] shows that the VLR can achieve the maximum data rate of 6.8 Gb/s with 4.14 mW power consumption (i.e., 608 fJ/b energy efficiency for 10-hop (10 mm) link traversal, maintaining bit error rate (BER) below 1×10^{-9}. On the other hand, the equivalent full-swing repeaters can transmit 5.5 Gb/s data at most with BER less than 1×10^{-9}, consuming 4.21 mW (i.e., 765 fJ/b), whereas VLR consumes 3.78 mW (i.e., 687 fJ/b) at the same data rate. Latency wise, Park shows that the delay of a link with VLRs is around 60 ps/mm, whereas the delay of a link with full-swing repeaters is around 100 ps/mm.

[1] V_{high} is given by link wire resistance, TxP's on-state resistance and RxN's on-state resistance, while V_{low} is determined by link wire resistance, TxN's on-state resistance and RxP's on-state resistance.

Table 5-1: Simulation Results of Max Number of Hops per Cycle

(a) Resized/Optimized Circuit for Low-frequency (2 GHz) with Wider Wire Spacing

Data Rate	**1 Gb/s**	**2 Gb/s**	**3 Gb/s**
Full-swing	13 (103 fJ/b/mm)	6 (95 fJ/b/mm)	4 (84 fJ/b/mm)
Low-swing	16 (128 fJ/b/mm)	8 (104 fJ/b/mm)	6 (87 fJ/b/mm)

(b) Sizing used in [90] with Wider Wire Spacing

Data Rate	**4 Gb/s**	**5 Gb/s**	**5.5 Gb/s**
Full-swing	4 (98 fJ/b/mm)	3 (89 fJ/b/mm)	3 (85 fJ/b/mm)
Low-swing	7 (132 fJ/b/mm)	6 (107 fJ/b/mm)	5 (96 fJ/b/mm)

However, in a SoC, the maximum clock frequency is usually limited by the core and router critical path rather than the link. We thus re-optimize the transistor sizes and wire spacing of VLR for a lower clock frequency of 2 GHz, instead of 6.8 GHz, to meet our system-level design goal of *single-cycle multiple-hop link traversal* without unnecessary energy consumption and the simulation results are shown in Table 5-1[2]. At 2 GHz, 8-hop (8 mm) link can be traversed in a cycle at 104 fJ/b/mm.

5.3 SMART Network Architecture

In this section, we present the SMART network architecture that can be tailored at runtime for different applications to enable near single-cycle traversal for flits between communicating cores. We first describe how we modify the router microarchitecture, followed by its routing algorithm and flow control mechanism.

5.3.1 Router Microarchitecture

As shown in Figure 5-3, in addition to the input buffers of the router, the crossbar is also fed by the incoming links to enable a combinational path directly from a router input to a router output. For each direction, an extra multiplexer is added to multiplex the crossbar input port between the input buffer and the incoming link. If the multiplexer is preset to

[2]Smaller transistor sizes and 2× wider wire spacing than the spacing used in [90].

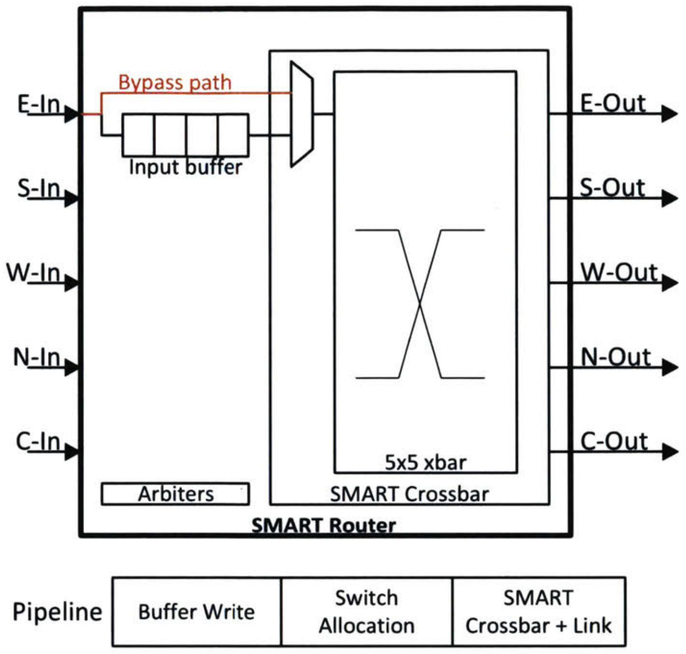


Figure 5-3: SMART Router Microarchitecture and Pipeline

connect the incoming link to the crossbar[3], a bypass path is enabled: incoming flits move directly to the crossbar, traverse it to the outgoing link, and do not get buffered/latched in the router. On the other hand, if the multiplexer is set to connect the input port buffer, the bypass path is disabled, which happens when the output link is shared across communication flows from different input ports. In this case, an incoming flit enters the router and goes through the three pipeline stages described below:

Stage 1: The incoming flit gets buffered and generates an output port request based on the preset route in its header.

Stage 2: All buffered flits arbitrate for the access to the crossbar.

Stage 3: Flits that win arbitration traverse the crossbar and output links to the next routers.

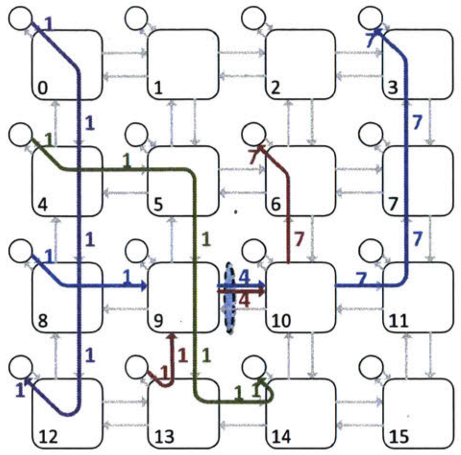


Figure 5-4: SMART NoC in Action with Four Flows (The number next to each arrow indicates the traversal time of that flow.)

5.3.2 Routing

Given an application communication graph, one can use NoC synthesis algorithms like NMAP [83] (see Section 5.5) to map tasks to physical cores and communication flows to static routes on a mesh. Figure 5-4 shows an example SMART NoC with preset routes for four arbitrary flows. In this example, the green and purple flows do not overlap with any other flow, and thus traverse through a series of SMART crossbars and links, incurring just a single-cycle delay from the source NIC to the destination NIC, without entering any of the intermediate routers. The red and blue flows, on the other hand, overlap over the link between routers 9 and 10, and thus need to be stopped at the routers before and after this link to arbitrate for the shared crossbar ports[4]. The rest of the traversal takes a single-cycle. It should be noted that before the application is run, all the crossbar selection lines are preset such that they either always receive a flit from one of the incoming links or from a router buffer.

Since the routes are static, we adopt source routing and encode the route in 2 bits for each router. At the source router, the 2-bit corresponds to East, South, West and North output ports, while at all other routers, the bits correspond to Left, Right, Straight and

[3]The crossbar signals also need to be preset to connect this input port to another output port.

[4]If flits from the red and blue flow arrive at router 9 at *exactly* the same time, they will be sent out serially from the crossbar's East output port.

Core. The direction Left, Right and Straight are relative to the input port of the flit. In this design, we avoid network deadlocks by enforcing a deadlock-free turn model across the routes for all flows.[5]

5.3.3 Flow Control

In a conventional hop-by-hop traversal model, a flit gets buffered at each hop. Thus, a router only needs to keep track of the free VCs/buffers at its neighbors before sending a flit out. Without loss of generality, we adopt the virtual cut-through flow control to simplify the design. A queue is maintained at each output port to track the available free VCs at the downstream router connected to that output port. A free VC is dequeued from this queue before a head flit is sent out of the corresponding output port. Once a VC becomes free at the downstream router, the router sends a credit signal (VCid) back to the upstream router which enqueues this VCid into the queue.

In the SMART NoC, a flit could traverse multiple hops and get buffered, bringing up challenging flow control issues. A router needs to keep track of free VCs at the endpoint of an arbitrary SMART route, though it does not know the SMART route till runtime. We solve this problem by using a reverse credit mesh network, similar to the forward data mesh network that delivers flits. The only overhead of the credit mesh network is a [log(# VCs) + 1(valid)]-bit crossbar added at each router. For example, if the number of VCs is 2, the overhead of the credit network is 2-bit wide crossbars. If a forward route is preset, the reverse credit route is preset as well. A credit that traverses multiple hops does not enter the intermediate routers and goes directly to the crossbar which redirects it along the correct direction.

For example, in Figure 5-4, for the blue flow, credits from NIC 3 are forwarded by preset credit crossbars at routers 3, 7 and 11 to router 10's East output port in a single-cycle without going into intermediate routers; credits from router 10's West input port are sent to router 9's East output port and credits from router 9's West input port are sent to NIC 8.

[5]Deadlock can also be avoided by marking one of the VCs as an escape VC [27] and enforcing a deadlock-free route within that. The exact deadlock-avoidance mechanism is orthogonal to this work.

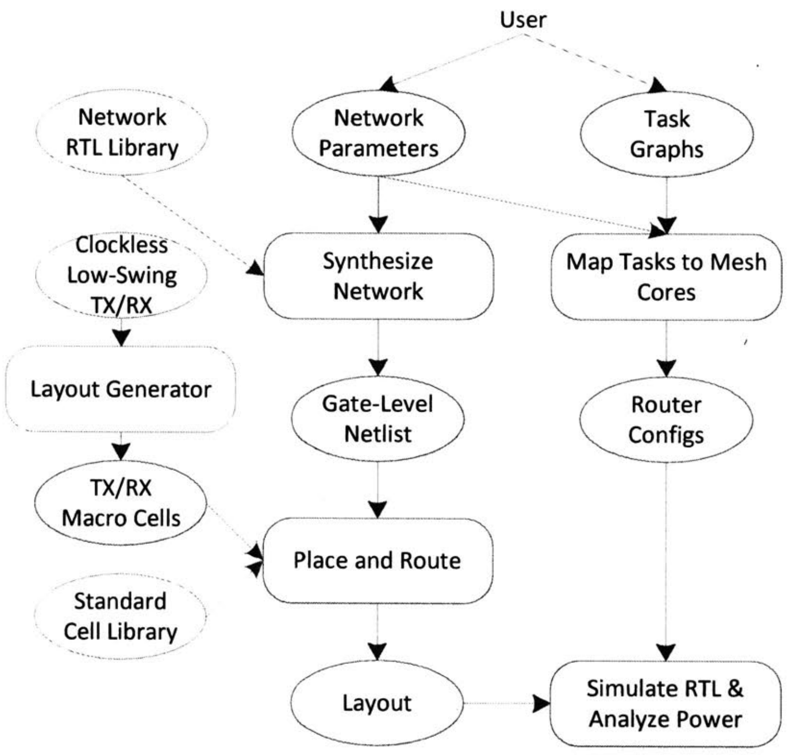


Figure 5-5: Tool Flow

The beauty of this design is that the router does not need to be aware of the reconfiguration and compute whether to buffer/forward credits. Since the credits crossbars act as a wrapper around the router, and are preset before the application starts, the credits automatically get sent to the correct routers/NICs. Thus, if a router receives a credit, it simply enqueues the VCid into its free VC queue. This free VC queue might actually be tracking the VCs at an input port of a router multiple hops away, and not the neighbor, as explained above.

5.4 Tool Flow

In this section, we describe the tool flow, shown in Figure 5-5, that we develop to power the SMART network. The tool flow can be divided into two parts: physical implementation and application mapping.

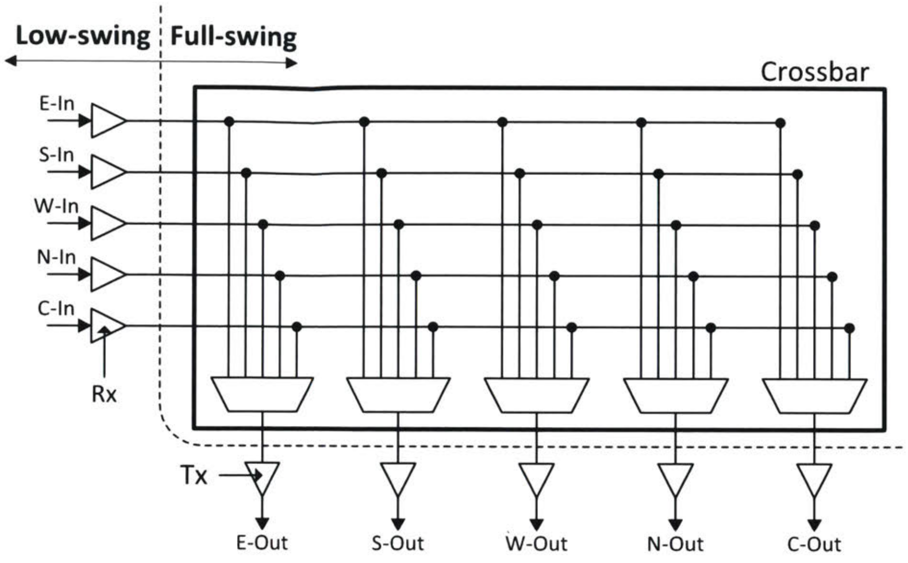


Figure 5-6: One-bit SMART Crossbar

Figure 5-7: 32-bit Tx Block Layout

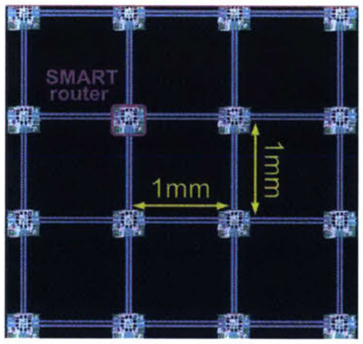


Figure 5-8: Generated 4x4 NoC Layout

5.4.1 Physical Implementation

VLR-Integrated Crossbar: To leverage the benefit from the VLR described in Section 5.2, we integrate it into the crossbar, as shown in Figure 5-6. The idea is to insert a crossbar between the Rx and Tx components of each repeater. The data received from the link will first be converted to full-swing (Rx), traverse the full-swing crossbar, and then be converted back to low-swing (Tx) again before it is forwarded to the next hop. To implement the crossbar, we develop a SKILL script to take 1-bit Tx/Rx layout and data with as input and place-and-route them regularly to multi-bit Tx/Rx blocks. Figure 5-7 shows an example of a 32-bit Tx block. We do not embed the VLRs in the crossbar as discussed in Chapter 4, because that leads to high area overhead. Also, we do not use existing commercial place-and-route tools, because these tools are often designed for general circuit blocks and cannot leverage the regularity property, adding unnecessary overhead. In addition, the script also generates the timing liberty format (.lib) and the library exchange format (.lef) files to allow the generated layout to be place-and-routed with the router.

Other Router Components: We develop a parameterized library of various router components in Verilog, and a tool that generates the RTL description of the SMART router and network with given network parameters. The input and output ports are clock-gated to reduce unnecessary dynamic power consumption based on the preset signals, which are set before each application runs. We provide scripts to help synthesis and place-and-route the router with the VLR-integrated crossbar, bringing the SMART router to layout. Furthermore, due to the limitation of the general routing tool that introduces unnecessary wiring overhead, we develop TCL scripts to control the tool to generate links between routers.

Reconfiguration Registers: To support SMART path reconfiguration for different applications, we encode the preset signals for crossbars and input/output ports into a double-word configuration register for each router. These registers are memory mapped such that these can be set by performing a few memory store operations. Before each application runs, these registers need to be set properly to suit the application's traffic characteristic. The network needs to be emptied while setting the registers. The values of the registers are determined based on the mapped flows on the mesh. Application developers need to prepend the application with memory store instructions to set the registers properly and the reconfiguration cost at runtime is just the amount of time to execute these instructions. For example, for a 16-node SMART NoC, there are 16 registers to be set which correspond to 16 instructions. If there is only 1 core that can perform the reconfiguration, a separate network (e.g., ring) is required to set these registers.

5.4.2 Application Mapping

The purpose of this part is to determine the preset signals for the application that will be run. We assume that the applications are already mapped to core and we take the

Table 5-2: 4x4 NoC Configuration

Name	Value
Technology	45 nm SOI
VDD, Freq	0.9 V, 2 GHz
Topology	4×4 mesh
Channel width	32 bits
Credit width	2 bits
Router ports	5
VCs per port	2, 10-flit deep
Packet size	256 bits
Flit size	32 bits
Header width	20 bits (Head), 4 bits (Body, Tail)

resulting task graphs including tasks to be mapped to physical cores and communication demands (flows) between them as the input to our tool. We adopt a modified NMAP algorithm [80] to map the tasks to physical cores in the mesh. We first map the task with highest communication demand to the core with the most number of neighbors (i.e., middle of the mesh). Then, we pick a task that communicates the most with the mapped tasks and find an unmapped core that minimizes the chance of of getting buffered at intermediate cores. This process is iterated to map all tasks to physical cores.

As the tasks are mapped to the physical cores, the flows between tasks are also mapped to routes with minimum number of hops between cores. Note that since the reconfiguration process only involves a few memory stores, the overhead of the reconfiguration can be omitted.

5.5 Case Study

5.5.1 Configurations

We use the tool flow present in Section 5.4 to implement a 4×4 SMART NoC and evaluate it with a suite of SoC applications. The configuration of the network is shown in Table 5-2, and the final layout is shown in Figure 5-8. It should be noted that the routers

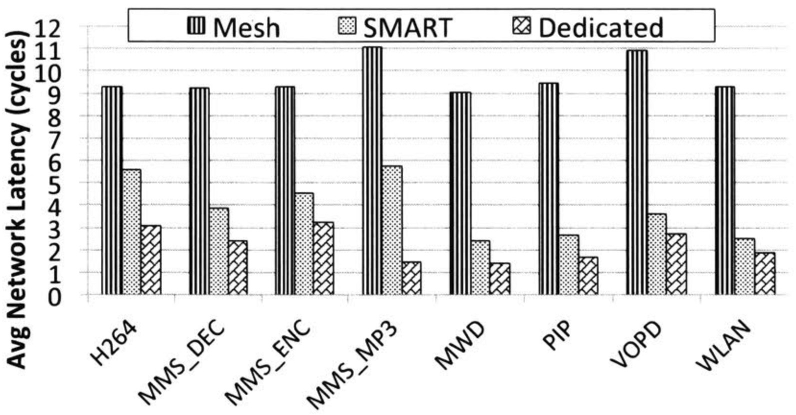


Figure 5-9: Performance

are assumed to be 1 mm spaced and the black regions shown are reserved for the cores. We refer to this design as *SMART*.

We evaluate SMART against two baselines: *Mesh* and *Dedicated*. Mesh is a state-of-the-art NoC topology without reconfiguration support [27], where each hop takes 3 cycles in router and 1 cycle in link. Dedicated is a NoC with 1-cycle dedicated links between all communicating cores tailored to each application. While this has area overheads, we use this design as an ideal yardstick for SMART. All designs use the VLR links.

We generate synthetic traffic from 8 SoC task graphs, modeling a uniform random injection rate to meet the specified bandwidth for each flow[6]. We feed this traffic through post-layout simulation of the SMART NoC to get average network latency. We also use the VCD files from these simulations to estimate power using Synopsys Prime Power.

5.5.2 Performance Evaluation

Figure 5-9 shows the average network latency across the applications for the baseline and SMART NoCs. Compared to the Mesh, SMART reduces network latency by 60.1 % on average due to the bypassing of the complete router pipelines[7]. On average, SMART reduces the network latency to 3.8 cycles, which is only 1.5 cycles higher than that of the Dedicated 1-cycle topology. For PIP, VOPD and WLAN, the latencies achieved by

[6]The bandwidth requirements of the three MMS benchmarks are scaled up 100× to allow reasonable on-chip traffic in our 2 GHz design. All other benchmarks' bandwidth remain unchanged.

[7]In the worst case, if all flows contend, SMART and Mesh will have the same network latency.

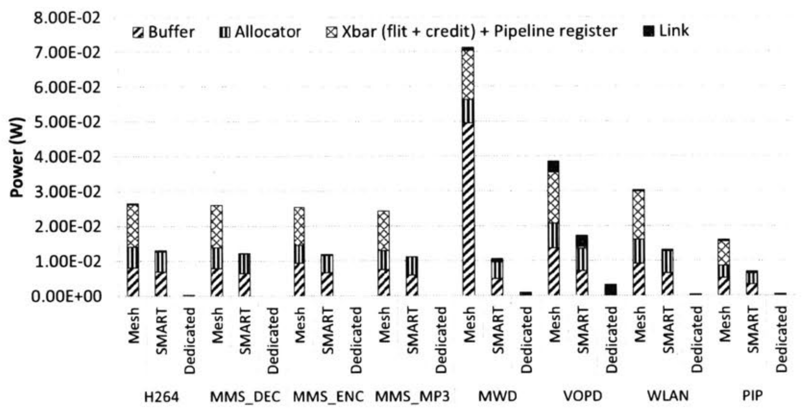


Figure 5-10: Power Breakdown

SMART and Dedicated are almost identical. If there are multiple traffic flows to the same destination, they need to stop at a router at the destination to go up serially into the NIC, both in SMART and Dedicated. However, SMART is limited by the available link bandwidth in a mesh to multiplex all flows, while Dedicated has no bandwidth limitation. This allows Dedicated to have 2 to 4 cycles lower latency than SMART in H264 and MMS_MP3 where one core acts as a sink for most flows, while another acts as the source for most flows, thus resulting in heavy contention and multiplexing. This can be ameliorated by splitting the 32-bit wide SMART channels into two 16-bit narrower channels (or more)[8], then clocking them at twice or thrice the rate, leveraging the high frequency of SMART links to mitigate conflicts. SMART can also enable non-minimal routes for higher path diversity without any delay penalty. We leave these as future work.

In an actual SoC, the task to core mapping may not be able to change drastically across applications as cores are often heterogeneous, and certain tasks are tied to specific cores. This will result in longer paths, magnifying the benefits of SMART.

5.5.3 Power Analysis:

Figure 5-10 shows the post-layout dynamic power breakdown across the applications for all three designs. All designs send the same traffic through the network, and hence have similar link power. Compared with Mesh, where flits need to stop at every router,

[8]Essentially, this increases the radix of the router and the path diversity.

SMART reduces power by 2.2× on average both due to bypassing of buffers, and due to clock gating at routers where there is no traffic. The total power for Dedicated is much lower than SMART because only link power is plotted, which is negligible due to low network activity. A Dedicated topology also has high-radix routers at destinations (if it acts as a sink for multiple flows), pipeline registers and muxes at the source (if multiple flows originate from it), which we ignored in the power estimates, though these will not be negligible.

5.6 Summary

In this chapter, we proposed SMART NoCs and demonstrated how scalable NoCs such as meshes can realize single-cycle, intra-chip communication while delivering high bandwidth by dynamically reconfiguring its switches to match application traffic. In the past, SoC architectures, compilers and applications have been aggressively optimizing for locality. As we drive towards more and more sophisticated SMART NoCs, we hope that will pave the way towards locality-oblivious SoC design, easing the move towards many-core SoCs.

6

SMART Network Chip

6.1 Motivation

In Chapter 5, we propose the SMART, a network architecture that allows flits or packets to traverse multiple hops within a single cycle. Even though we only present in Chapter 5 the SMART network targeting SoC applications, we also develop another version of SMART network that targets manycore system applications, which we will go through in detail in Appendix A. For the rest of the thesis, we will refer to the SMART for SoC applications as $\text{SMART}_{\text{app}}$, and the SMART for many core system applications as $\text{SMART}_{\text{cycle}}$. The main difference between these two flavors of SMART network is that $\text{SMART}_{\text{app}}$ is suitable for applications with predictable traffic patterns and limited communication flows, whereas $\text{SMART}_{\text{cycle}}$ is suitable for applications with unpredictable traffic patterns or near all-to-all communication flows.

The key idea behind the SMART network is to dramatically reduce the packet delivery latency by reducing the number of times that a packet needs to be stopped at intermediate routers, instead of retiming[1] the pipeline stages within a router to achieve higher clock frequency or lower number of pipeline stages.

[1]Retiming is a technique used in digital circuits to move the structural location of latches or flip-flops to improve the performance, area and/or power, while preserving the same functional behavior at the outputs.

The equation of packet delivery latency (T) in cycles is then effectively reduced from:

$$T = HT_r + HT_w + T_c + L/b \tag{6.1}$$

to

$$T = \lceil H/\mathrm{HPC} \rceil T_r + \lceil H/\mathrm{HPC} \rceil T_w + T_c + L/b \tag{6.2}$$

where H is the number of hops, T_r is the router pipeline delay, T_w is the wire (between two routers) delay, T_c is the contention delay at routers, L/b is the serialization delay for the body and tail flits, (i.e., the amount of time for a packet of length L to cross a channel with bandwidth b), and HPC stands for number of hops that can be traversed in a cycle. The higher the HPC allowed, the lower the packet delivery latency can be achieved. For example, as shown in Chapter 5, VLR circuit allows data to traverse 16 mm (i.e., 16 hops with a 1 mm separation) in 1 ns, indicating that a maximum HPC of 16 is feasible. However, the actual data path of the SMART network is more complex than a chain of repeaters and links; it is composed of crossbars and links. Therefore, the actual maximum HPC is less than 16, which needs to be further examined.

Unlike typical NoC designs where the metrics (e.g., timing, area and power) solely depend on their router designs, the SMART network's metrics depend on not only the router design but also the maximum HPC allowed. In this chapter, we investigate the tradeoffs that SMART network's low-latency benefit comes with between the maximum HPC and critical metrics (e.g., timing, area and power). We first review the repeated link and then examine the essential components that are necessary to either $\mathrm{SMART}_{\mathrm{app}}$ or $\mathrm{SMART}_{\mathrm{cycle}}$. Furthermore, since the maximum HPC allowed is affected by the link performance, which is hard to characterize accurately even through post-layout simulations. Therefore, in addition to the analyses on essential components, we present a case study of a 64-node SMART network chip, fabricated using a 32 nm SOI CMOS technology, and demonstrate thorough timing and power analyses with measurement results.

The rest of the chapter is organized as follows. Section 6.2 demonstrates the feasibility of the SMART network through preliminary timing analyses on repeated link and critical components. Section 6.3 shows the architecture of the chip prototype, where its implementation consideration is presented in Section 6.4. Section 6.5 evaluates the chip's area, timing and power through simulations and measurements. Section 6.6 summarizes the chapter.

6.2 Design Analyses of SMART on Process Limitation

The ultra-low latency of SMART network comes with a price. If we use the same circuit and transistor sizing, the higher HPC_{max} requires a higher cycle period (i.e., a lower clock frequency). On the other hand, we can size up the circuit to improve its timing to achieve higher HPC_{max}; however, it comes with a cost of higher area footprint and energy consumption. In this section, we focus on evaluating the tradeoff between area/energy and HPC_{max} for critical components of SMART network: repeated link, data path, as well as control path for $SMART_{cycle}$, at a clock frequency of 1 GHz (i.e., 1 ns clock period).

Even though we discussed the benefit of using VLR in Section 5.2 with a tool flow to ease the integration, evaluating the tradeoffs for those critical components requires both re-designs of the VLR cells for each HPC_{max} and detailed SPICE-level simulations for correct behavior, which dramatically increase the complexity and time required. Therefore, we implement these components with complete full-swing circuits in RTL, and obtain the energy and area numbers from post-layout circuits.

6.2.1 Repeated Link

In addition to the discussion in Section 5.2, we revisit the performance of full-swing repeated link under looser constraints. We use Cadence Encounter to place-and-route a 128-bit repeated link in 45 nm SOI CMOS technology. The wire spacing is 3× of the minimum spacing instead of 2× used in Section 5.2, resulting in lower coupling capacitance (a decrease in overall capacitance by approximately 13 % with an increase in area by 33 %), and hence lower delay as well as energy consumption.

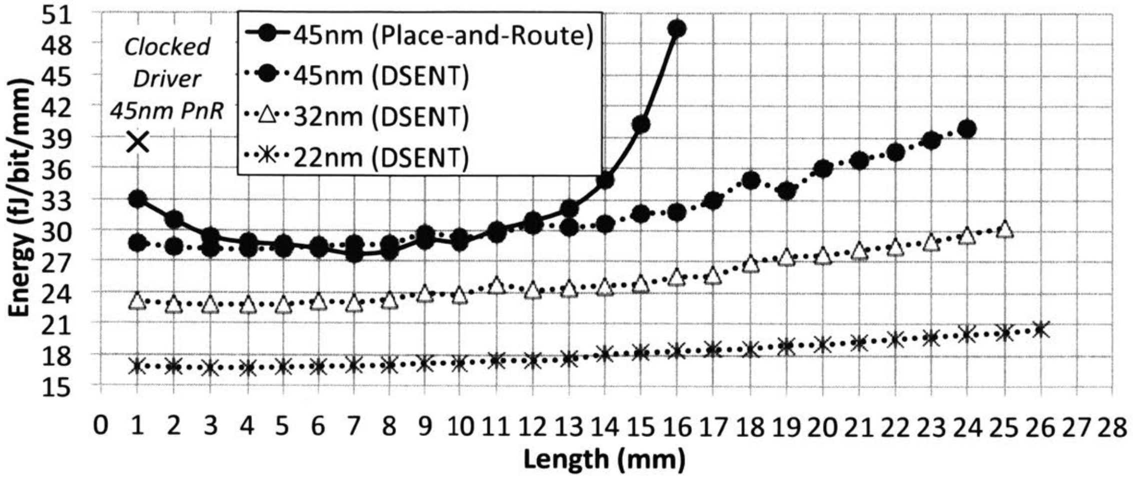


Figure 6-1: Achievable HPC_{max} for Repeated Links at 1 GHz.

We keep increasing the length of the wire, letting the tool size the repeaters appropriately, till it fails timing closure at 1 ns (i.e., 1 GHz).[2] Figure 6-1 shows that a place-and-routed repeated wire in 45 nm can go up to 16 mm in a ns. The sharp rise in energy per bit is the cost of having HPC_{max} higher than 12, contributed by larger repeater sizes and poor wire layout for long links[3]. Figure 6-1 shows a similar trend for 32 and 22 nm technology, with energy going down by 19 and 42 % respectively, using the timing-driven NoC power modeling tool DSENT[4] described in Chapter 3.

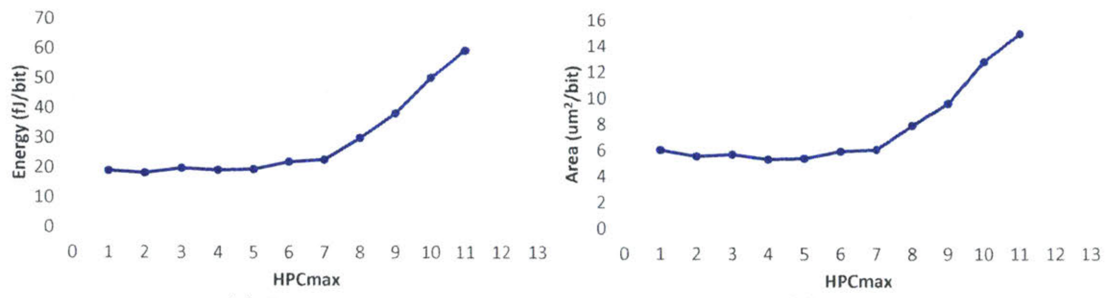
6.2.2 Data Path

The data path of the SMART network consists of a chain of crossbars and links, and is modeled as a series of a 128-bit 2:1 multiplexer (for buffer bypass), a 4:1 multiplexer (for crossbar) followed by a 128-bit 1 mm link.

[2]Wire Width: DRC_{min}, Wire Spacing: $3\times$ DRC_{min}, Metal Layer: M6. Repeater Spacing: 1 mm

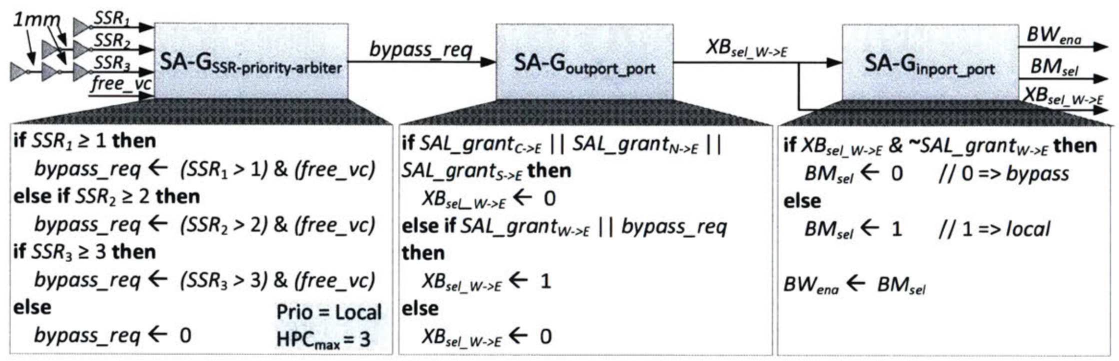
[3]It is an artifact of using Cadence Encounter, an automatic place-and-route tool, which zig-zags wires to fit to a fixed global grid that is unfortunately not a multiple of M6 DRC_{min} width. This artifact adds unnecessary wire lengths, leading to higher energy cost. A custom design may go farther and with flatter energy profile.

[4]DSENT's projections on maximum length a repeated link can achieve in 1 ns are slightly overestimated because it does not model inter-layer via parasitics (needed to access the repeater transistors on M1 from the link on M6), which become significant when there are many repeaters.



(a) Energy (b) Area

Figure 6-2: Energy and Area versus HPC_{max} for Crossbar



```
if $SSR_1 \geq 1$ then
    bypass_req ← ($SSR_1$ > 1) & (free_vc)
else if $SSR_2 \geq 2$ then
    bypass_req ← ($SSR_2$ > 2) & (free_vc)
if $SSR_3 \geq 3$ then
    bypass_req ← ($SSR_3$ > 3) & (free_vc)
else                                   Prio = Local
    bypass_req ← 0                     $HPC_{max}$ = 3
```

```
if $SAL\_grant_{C->E}$ || $SAL\_grant_{N->E}$ ||
$SAL\_grant_{S->E}$ then
    $XB_{sel\_W->E}$ ← 0
else if $SAL\_grant_{W->E}$ || bypass_req
then
    $XB_{sel\_W->E}$ ← 1
else
    $XB_{sel\_W->E}$ ← 0
```

```
if $XB_{sel\_W->E}$ & ~$SAL\_grant_{W->E}$ then
    $BM_{sel}$ ← 0     // 0 => bypass
else
    $BM_{sel}$ ← 1     // 1 => local

$BW_{ena}$ ← $BM_{sel}$
```

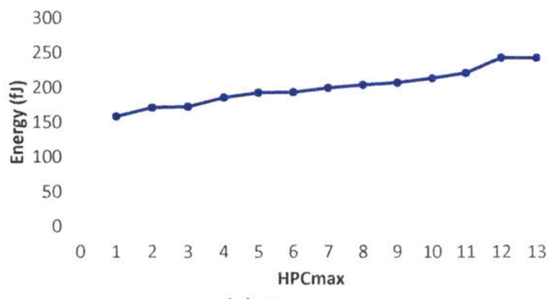
Figure 6-3: Implementation of SA-G at W_{in} and E_{out} for 1D version of $SMART_{cycle}$

Figure 6-2 shows the energy-per-bit and area-per-bit of the modeled data path (without link). Both energy and area profiles stay flat when HPC_{max} is less than 7 because the total path delay is still within the 1 ns constraint with the same cell sizes. After HPC_{max} of 7, larger cell sizes are used to reduce the per-hop data path delay, leading to increased energy and area profile. The data path can go up to 11 hops in 1 ns clock period.

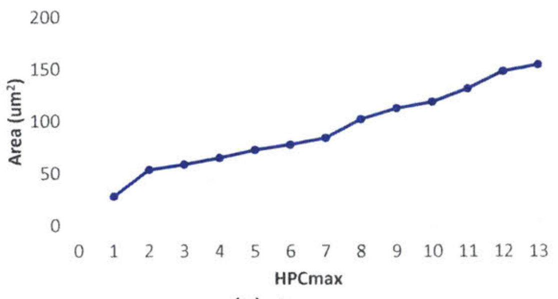
6.2.3 Control Path

The control path consists of HPC_{max}-hops repeated link and SA-G logic used in $SMART_{cycle}$. Detailed description of how each component works can be found in Appendix A.

In 1D version of $SMART_{cycle}$, each input port receives one SSR from every router up to HPC_{max}-hops away in that direction. The input, output and internal signals correspond to the ones shown in the router in Figure A-1. The logic for SA-G for Prio=Local is in a 1D version of $SMART_{cycle}$ design at the W_{in} and E_{out} ports of the router is shown in

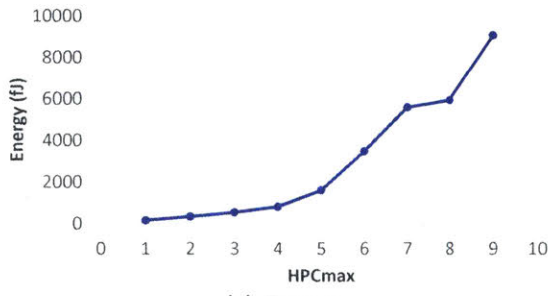


(a) Energy

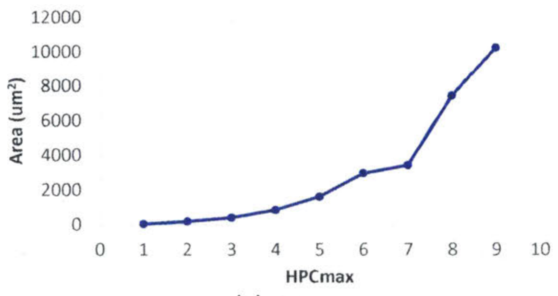


(b) Area

Figure 6-4: Energy and Area versus HPC_{max} for 1D version of SA-G



(a) Energy



(b) Area

Figure 6-5: Energy and Area versus HPC_{max} for 2D version of SA-G

Figure 6-3[5]. To reduce the critical path delay, BW_{ena} is relaxed such that it is 0 only when there are bypassing flits (since the flit's valid-bit is also used to determine when to buffer), and BM_{sel} is relaxed to always pick local if there is no bypass. XB_{sel} is strict and does not connect an input port to an output port unless there is a local or SSR request for it.

In 2D version of $SMART_{cycle}$, all routers that are H-hops away, $H \in [1, HPC_{max}]$, together send a total of $(2 \times HPC_{max} - 1)$ SSRs to every input port. $SA\text{-}G_{SSR\_priority\_arbiter}$ is similar to Figure 6-3 in this case and choose a set of winners based on distance, while $SA\text{-}G_{output\_port}$ disambiguates between them based on turns, as discussed earlier in Figure A.4.2.

For 1D version of SA-G, the energy and area increase linearly with HPC_{max}, as shown in Figure 6-4. And it is able to achieve an HPC_{max} of 13 with 890 ps SSR and 90 ps SA-G.

As for 2D version of SA-G, since it needs to arbitrate the SSRs from all the routers HPC_{max}-hop away, the energy and area grow quadratically with HPC_{max}. 2D version of SA-G can only achieve an HPC_{max} of 9 with 620 ps SSR and 360 ps SA-G.

[5]The implementation of Prio=Bypass is not discussed but is similar.

6.2.4 Summary

To implement the $\text{SMART}_{\text{app}}$ with a clock frequency of 1 GHz, it is possible to design HPC_{max} to be 11, based on the data path performance. However, it comes with extra area footprint and energy consumption. To avoid this overhead, 7 is the maximum HPC_{max} that can be set at design time; otherwise a lower clock frequency is required to achieve higher HPC_{max} without inducing area/energy overhead.

As for the $\text{SMART}_{\text{cycle}}$, compared to the data path, the control path performance is better, and the increases in its area and energy consumption are mild. In addition, even though the 2D version offers a lower low-load latency (compared to the 1D version), the high energy and area overhead as well as the number of SSRs required make it unfavorable. Therefore, the maximum HPC_{max} that can be set is also 7.

These analyses guided my design choice as follows:

- Complete full-swing circuits to reduce the design complexity and time.
- A maximum HPC_{max} of 7 with a clock frequency of 1 GHz at design time for both $\text{SMART}_{\text{app}}$ and $\text{SMART}_{\text{cycle}}$ to avoid area and energy consumption overhead.
- 1D version of $\text{SMART}_{\text{cycle}}$ to avoid excessive overhead of the 2D version.

6.3 Chip Architecture

In the rest of this chapter, we present a case study of a chip prototype of a 64-node (8×8) SMART network. The design target is to achieve HPC_{max} of 7 at a clock frequency of 1 GHz. In addition, we make HPC_{max} a parameter that we can configure at runtime to evaluate the tradeoff between the achievable clock frequency and HPC_{max} at the same design point.

The chip is 3×3 mm in size, as shown in Figure 6-6. Each node constists of a router, a network interface controller (NIC) and a tester, as illustrated in Figure 6-7. A PLL is used with a synchronous clock style to clock all the nodes. Detailed specification of the chip is shown in Table 6-1.

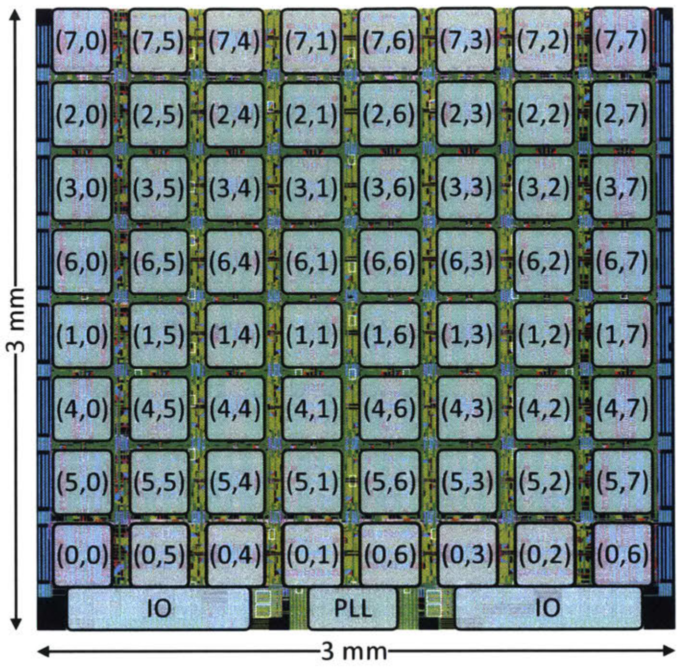


Figure 6-6: Chip Layout

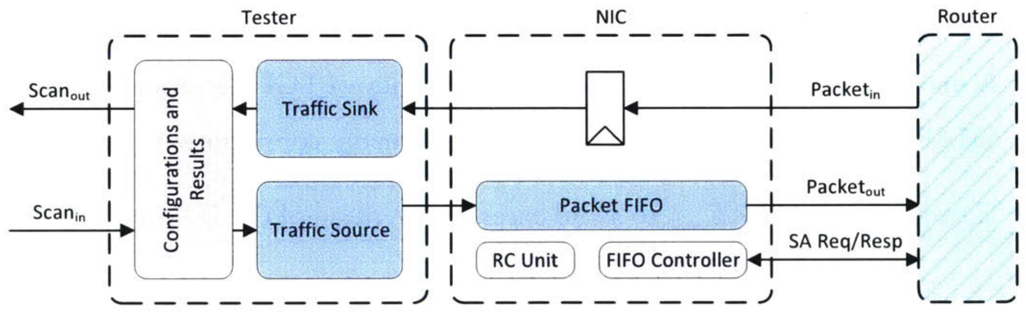


Figure 6-7: Node Microarchitecture

Due to the die area limitation, several design decisions are made based on multiple iterations of performance simulation and synthesis. We show those decisions as follows:

- Even though the SMART network, especially SMART$_{\text{cycle}}$, works better with larger network sizes, we choose a network of 64 nodes so that the number of nodes is just enough for design target (i.e., HPC$_{\text{max}}$ of 7 at a clock frequency of 1 GHz).
- We do not include processor cores into our chip. Instead, we place a tester at each node to generate synthetic traffic to help evaluate the network performance and power consumption.

Table 6-1: Chip specification

Name	Value
Chip dimension	$3 \times 3\,\text{mm}^2$
Technology	32 nm SOI
Gate count	9.19 M
Power supply	0.9 V
Clock frequency	Target: 1 GHz, Actual: 548 to 817.1 MHz
Network size	8×8
Router pitch	1 mm on average
Flit width	64 bits
# VCs	8
# Buffers	1 per VC
Routing algorithm	X-Y + User-defined
Flow control	$\text{SMART}_{\text{cycle}}$ + $\text{SMART}_{\text{app}}$
	Configurable from 1 to 7 for $\text{SMART}_{\text{cycle}}$
HPC_{max}	No restriction for $\text{SMART}_{\text{cycle}}$

- We assume that each synthetic packet consists of only one flit, and do not implement the support for multi-flit packets to avoid high amount of buffers required for desired performance. As a result, a flit width of 64-bit is chosen, one buffer per VC is sufficient, and a total of 8 buffers is used to achieve decent performance without too much area overhead.

6.3.1 NIC and Tester Microarchitecture

The tester consists of a traffic source and a traffic sink. The traffic source generates synthetic packets based on runtime configurations such as traffic type, injection rates, etc. We use several multi-bit linear feedback shift registers (LFSR) to control the packet generation, packet destinations, as well as packet payloads. The traffic sink consumes a packet upon its arrival, and checks if it contains error bits using the parity bit information tagged with the packet. A custom scan chain is used to transfer the configurations and collected results.

The NIC serves as an interface between the tester and router. It receives the generated packets from the traffic source and stores them in a FIFO, and joins the switch arbitration

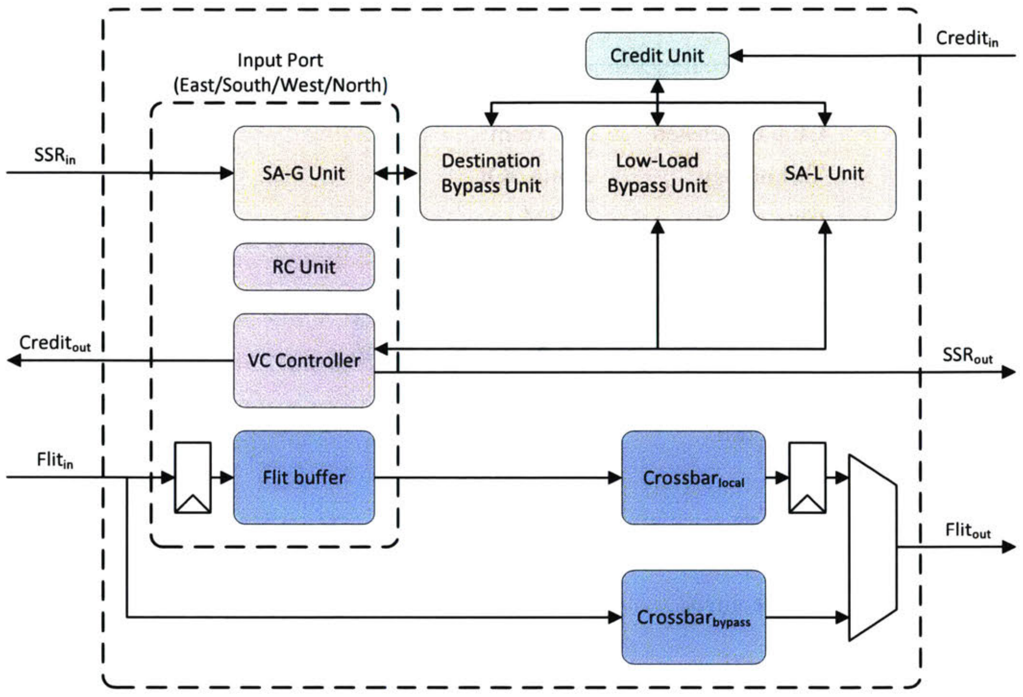


Figure 6-8: Router Microarchitecture

to win the access to the crossbar. It also receives packets from the router and forwards them to the traffic sink. Since the traffic sink is designed to consume packets upon arrival, a pipeline register is used instead of a FIFO.

6.3.2 Router Microarchitecture

Figure 6-8 shows the microarchitecture of the router. The design supports SMART$_{app}$ as well as one-dimensional SMART$_{cycle}$. All the routers need to be configured to run in the same mode for correct behavior. Instead of the 2D version of SMART$_{cycle}$, the 1D version is chosen to avoid high area and energy overhead when HPC$_{max}$ of 7 at design time.

SMART$_{app}$: We implement a two-cycle router where its pipeline is shown in Figure 6-9. When an input port is configured to block flows, it first buffers incoming packets in the input pipeline register, and then performs VC allocation and joins switch arbitration. If a packet wins the switch arbitration, it traverses the crossbar$_{local}$ and gets buffered in the output pipeline register, and in the next cycle, it traverses the links and crossbar$_{bypass}$ and gets stopped at another input port or NIC. Since there is no computing unit on the chip,

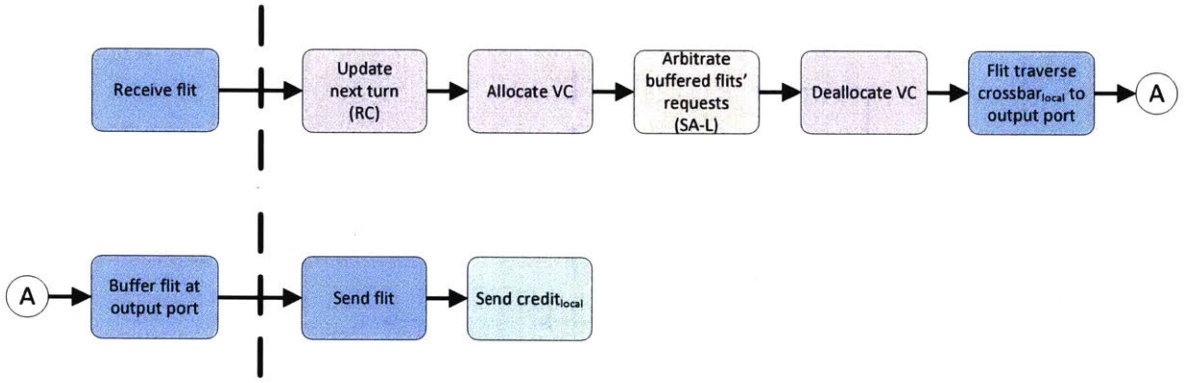

Receive flit
Update next turn (RC)
Allocate VC
Arbitrate buffered flits' requests (SA-L)
Deallocate VC
Flit traverse crossbar_local to output port
A
A
Buffer flit at output port
Send flit
Send credit_local

Figure 6-9: Router Pipeline

we scan in the configurations for each router, such as the crossbar control bits, turning information for flows.[6]

SMART$_{\text{cycle}}$: We implement the 1D version of SMART, which allows routers to be bypassed only along one dimension, as well as the support for bypassing the ejection router and bypassing SA-L at low load. To increase the maximum achievable clock frequency, we move the crossbar traversal stage for buffered flits one cycle earlier, which requires an additional crossbar and an additional credit port to ensure correct functionality without degrading throughput. We use $crossbar_{local}$ and $crossbar_{bypass}$ to refer to the two crossbars, and $credit_{local}$ and $credit_{bypass}$ for the credit ports.

In Figure 6-10, we present the modified pipeline in detail that a flit may go through after its SSR is received. The number of pipeline stages varies from 0 to 3 cycles depending on the scenarios. For simplicity, we assume that the flit arrives at the west input port and may request to depart to all other ports except west port. We also assume that the received flit is valid; otherwise, nothing needs to be done. When one or more SSR(s) arrives in cycle 0, the router first picks the SSR that comes from the closest router and discards the other SSRs. The router uses the *num_hops* and *is_dest* information carried by the SSR to determine how to handle the flit arriving in cycle 1, as shown below:

- **Bypass flit to opposite (east):**

[6]However, a mistake was made in the design such that only one flow per router can be configured. It limits the evaluation since most applications require at least two flows from some nodes.

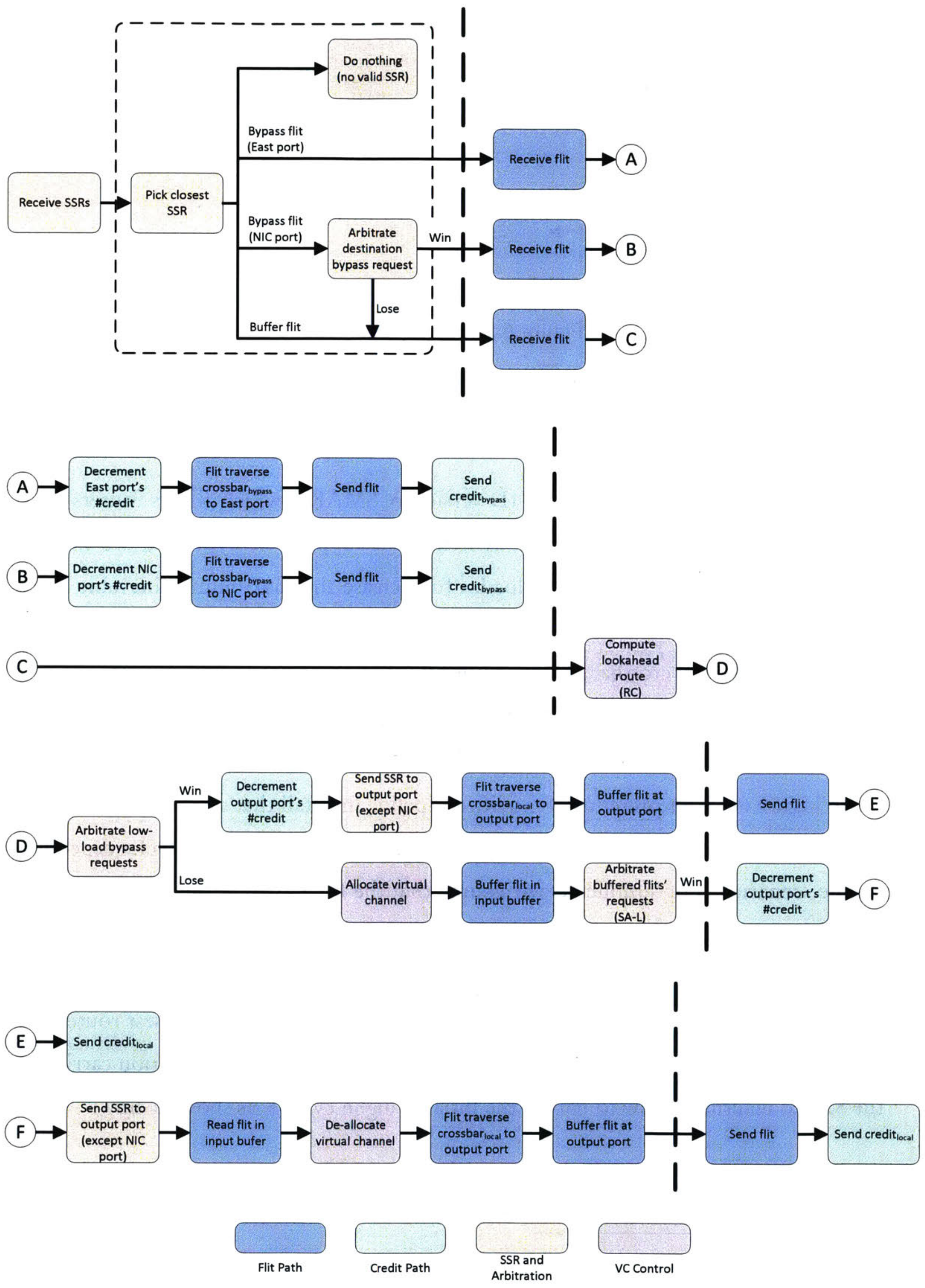

Receive SSRs
Pick closest SSR
Do nothing (no valid SSR)
Bypass flit (East port)
Bypass flit (NIC port)
Arbitrate destination bypass request
Win
Lose
Buffer flit
Receive flit
A
B
C
Decrement East port's #credit
Flit traverse crossbar$_{bypass}$ to East port
Send flit
Send credit$_{bypass}$
Decrement NIC port's #credit
Flit traverse crossbar$_{bypass}$ to NIC port
Compute lookahead route (RC)
D
Arbitrate low-load bypass requests
Decrement output port's #credit
Send SSR to output port (except NIC port)
Flit traverse crossbar$_{local}$ to output port
Buffer flit at output port
E
Allocate virtual channel
Buffer flit in input buffer
Arbitrate buffered flits' requests (SA-L)
F
Send credit$_{local}$
Read flit in input bufer
De-allocate virtual channel
Flit Path
Credit Path
SSR and Arbitration
VC Control

Figure 6-10: Router Pipeline

Cycle 1: The flit directly traverses the crossbar$_{\text{bypass}}$ to the east output port as well as the link to the next router. The router decrements the east port's credit and sends a credit$_{\text{bypass}}$ back to the router on the west.

- **Bypass flit to NIC:**

  Cycle 0: Since multiple input ports may request to bypass to NIC in the same cycle, the router arbitrates these requests using a fixed priority arbiter. If the SSR from the west input port loses the arbitration, the flit will follow the steps in *Buffer flit*.

  Cycle 1: The flit traverses the crossbar$_{\text{bypass}}$ to the NIC port. The router decrements the NIC port's credit and sends back a credit$_{\text{bypass}}$ to the router on the west.

- **Buffer flit:**

  Cycle 1: The flit arrives and is buffered in the input pipeline register.

  Cycle 2: The router updates the flit's lookahead route information. The flit joins the low-load bypass arbitration if there was no SA-L winner in cycle 1.

  Cycle 2a: The flit wins the arbitration. The router decrements the output port's credit, and the flit traverses the crossbar$_{\text{local}}$ to the output port and gets buffered at the output pipeline register. An SSR for this flit is sent out to the output port (except the NIC port).

  Cycle 3a: The flit traverses to the next router or NIC. A credit$_{\text{local}}$ is sent back to the router on the west.

  Cycle 2b: The flit loses the low-load bypass arbitration. The input port allocates a VC, stores the flit in the flit buffer, and the router performs SA-L for all buffered flits. If this flit loses the arbitration, it will re-attempt the SA-L in the next cycle.

  Cycle 3b: The flit wins the SA-L, traverses crossbar$_{\text{local}}$ from the flit buffer to the output port, and gets buffered at the output pipeline register. The credit of the output port is decremented.

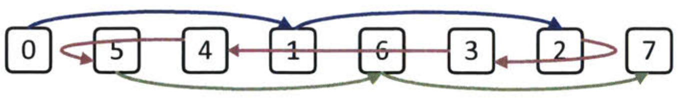


Figure 6-11: Folded network with router pitch of 1 mm

Cycle 4b: The router sends the flit to the next router and a credit$_{\text{local}}$ back to the router on the west.

6.4 Implementation Consideration

We implement the chip using a two-level bottom-up hierarchical method; we first make the router into a hard block and then use it as a black box for chip/network level implementation. Since SMART allows traversing multiple hops within a single cycle, it indicates that potentially there are excessive amount of combinational loops formed by links and crossbars. Thus, at the chip level, we remove the timing checks on these paths to avoid exposing the combinational loops to the tools, and only use the tools to implement the global clock tree as well as reset and scan chain connections.

Metal Layers: The process that we use to fabricate the chip provides 11 metal layers to use: 5 for local, 4 for intermediate and 2 for global. We use the global layers to route the global power network and the top-level clock network, the top 2 intermediate layers for routing links, while the rest are for router internal wires. It should be noted that even though the intermediate layers require a wider minimum width constraint compared to the local layers, the low-resistance property of the intermediate layers due to taller wires make it suitable for long-distance data transportation.

Link: If we naively squeeze the 8×8 network into a $3 \times 3\,\text{mm}^2$ chip, the router pitch (distance between routers) would be approximately only 0.35 mm, which is much shorter than the typical pitch (distance between cores/tiles, typically 1 to 2 mm) used in NoC research proposals. Therefore, we space out the routers and fold the network twice (see Figure 6-11 to increase the pitch to 1 mm on average, which increases the link length to 0.75 mm on average. We then explore the link design space by varying repeater types and sizes as well as wire spacing, and chose the parameters that allow traversing the most number of hops within the same period without violating design rules[7]. The repeater spacing is fixed at 350 mm, which is the router layout pitch. The chosen parameters allow traversal of a link of length 10 mm with 1 ns.

Router: We implement the router with a target clock frequency of 1 GHz. This constraint is only applied to ensure that the router can be run at 1 GHz regardless of the actual HPC_{max},which is specified at runtime. While setting the timing constraints for input and output ports, the goal is to implement a router with as low delay as possible for all paths going through these ports. The timing of all possible paths are discussed in Section 6.5.3.

6.5 Evaluation

6.5.1 Setup

To evaluate the timing and power consumption of the chip, we perform both post-layout evaluations and chip measurements. For post-layout evaluations, we run static timing analysis (STA) on the post-layout netlist to analyze the timing and run simulations to obtain power breakdown for various scenarios. The post-layout evaluations are performed using the TT corner library at VDD of 0.9 V and temperature of 50 °C. To increase the accuracy, we annotate wires' parasitic resistance and capacitance.

For the measurements, we use 3 power supplies to measure the power consumption; one for IO pads, one for testers and PLL, and one for the rest of the network. Because of the high current consumption, the resistance of the cables connecting the power supplies to the board is not negligible and leads to 0.05 to 0.1 V voltage drop. Therefore, we configure the power supplies to operate in 4-wire sensing mode to resolve this issue.[8] A function generator is used to provide the 10 MHz reference clock for the PLL. In addition, we also use a heat sink with a fan to dissipate the excessive heat generated by the chip. We

[7]We can potentially have a large wire spacing to lower the coupling capacitance between wires. However, an over-sized wire spacing would violate the minimum metal density rule.

[8]While operating in 4-wire sensing mode, a power supply dynamically senses the actual voltage level across the design under test, and adjusts the output voltage level to maintain the specified level across the design under test.

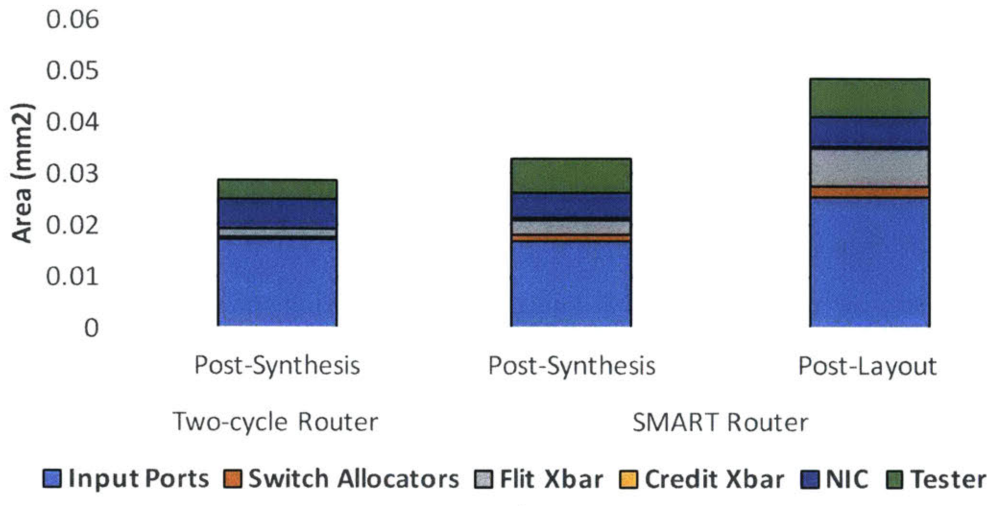


Figure 6-12: Area Breakdown

set the VDD to 0.9 V for both timing and power measurements. The ambient temperature of the measuring environment is approximately 22 °C.

6.5.2 Area

The area of a SMART router is 75,675.6 μm^2 with a 64 % cell density. We compared the area breakdown of standard cells in Figure 6-12. The post-layout area is larger than the post-synthesis area by 48 %. This is because that standard cells are sized up and extra buffers are added to meet the stringent timing constraint. In comparison to the post-synthesis area of a two-cycle router[9], the overhead of SMART-specific logics is 14 % due to a larger tester with additional statistic gathering logics.

Among all the components, the input ports contribute to 50 % of the total cell area, and both the flit crossbar and tester contribute to 15 %, respectively. In contrast to the flit crossbar of two-cycle router designed to traverse one hop, the flit crossbar of SMART router is implemented to achieve high HPC_{max}, leading to much larger area.[10]

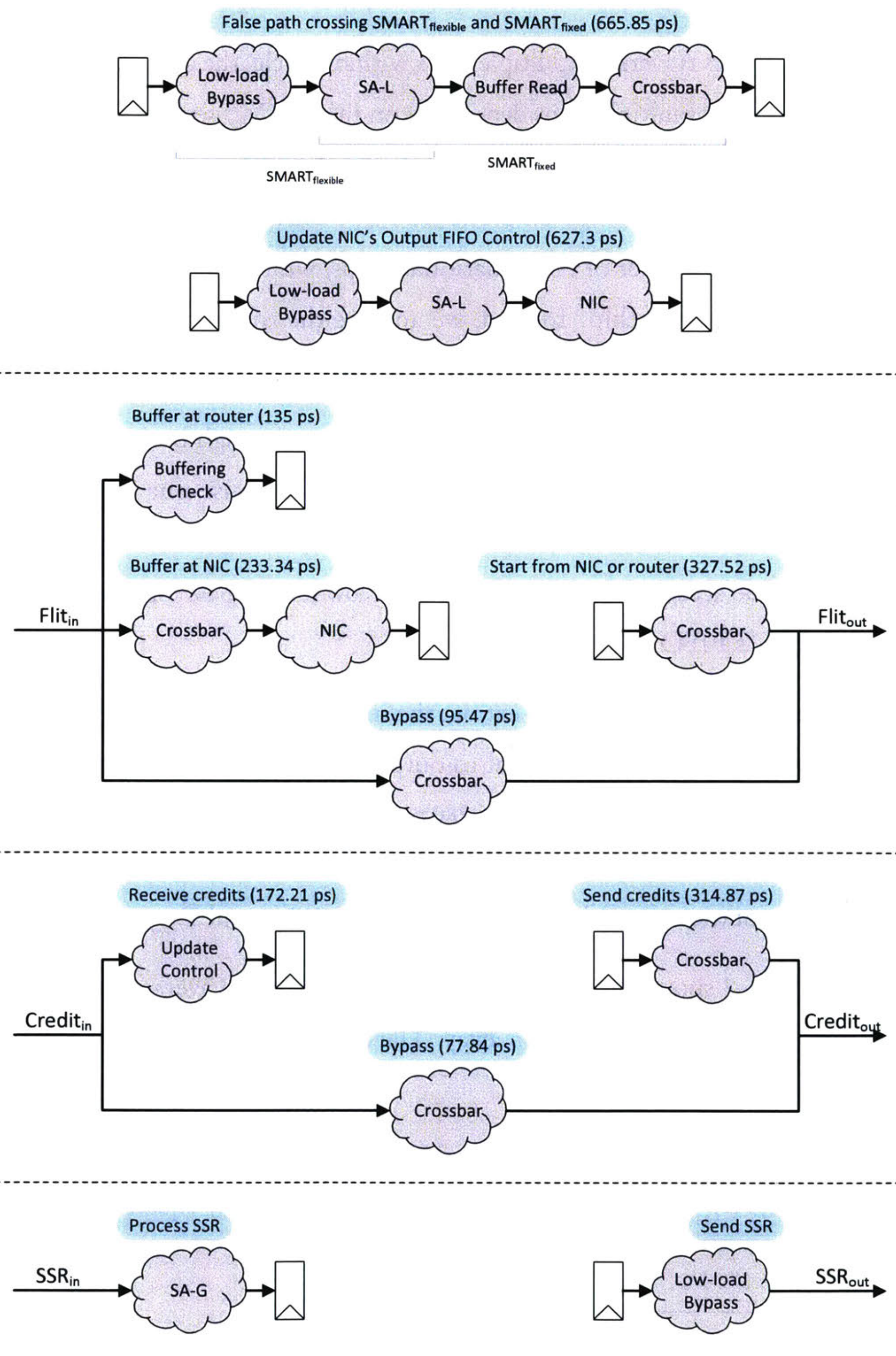


Figure 6-13: Router Critical Paths

6.5.3 Timing – Static Timing Analysis (STA)

Since SMART allows traversing multiple hops within a cycle, the actual critical path of the chip may span across multiple hops depending on the HPC_{max} setting. To understand the maximum achievable frequency of the chip, we perform STA using Synopsys PrimeTime to identify the critical paths for different HPC_{max}. However, at architecture level, the chip has enormous number of combinational loops, which increases the complexity to perform STA on the full-chip. Instead, we analyze the router and construct the delay estimation of the critical path on the chip. The results are shown in Figure 6-13.

Intra-Router: The critical path of the router goes through both $SMART_{cycle}$'s and $SMART_{app}$'s logic blocks. The path is a false path[11] because one mode can only be operated at a time. Nonetheless, it prevents other internal paths from further timing optimization. The actual critical path starts from low-load bypass logic followed by SA-L, and ends at NIC's FIFO update logic.

For router's boundary paths, we extract the input-to-register delay (T_{in2reg}), register-to-output delay ($T_{reg2out}$), as well as input-to-output delay (T_{in2out}) for flit, credit, and SSR signals. In general, SSR's T_{in2reg} is 140 to 280 ps and $T_{reg2out}$ is 148 to 160 ps. However, the input delay and output delay were incorrectly applied to the SSR ports on the north side during implementation, resulting in T_{in2reg} of 528 to 722 ps and $T_{reg2out}$ of 241 ps.

Link: Table 6-2 shows the link length and delay for flit. In average, it takes 53.7 ps to travel to an adjacent router 0.743 mm away. The delay of SSR and credit links are similar to flit links.

Inter-Router: We identify potential critical paths across multiple hops for flit, credit and SSR signals, respectively, and construct corresponding delay equations as functions of

[9]The router is designed to have one cycle for buffering and arbitration, and the other one for crossbar and link traversal. The same buffer size is used.

[10]4.2× for post-layout, and 1.7× for post-synthesis.

[11]A false path is a timing path that will never be exercised in a design.

Table 6-2: Flit Link Length and Delay

(a) Horizontal

Segment No.	Length (mm)	Delay (ps)
1	0.815	69.03
2	0.815	68.97
3	0.520	34.85
4	0.877	59.65
5	0.518	33.44
6	0.878	67.93
7	0.878	67.48

(b) Vertical

Segment No.	Length (mm)	Delay (ps)
1	0.780	56.85
2	0.780	57.50
3	0.504	31.43
4	0.845	57.33
5	0.504	30.51
6	0.843	58.11
7	0.843	58.68

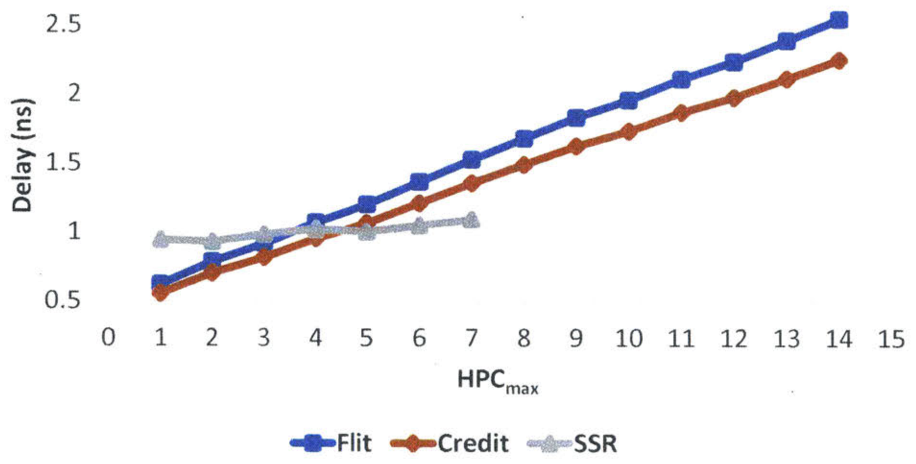


Figure 6-14: Chip Critical Path

HPC$_{\text{max}}$ as shown below.

$$
\begin{aligned}
T_{\text{reg2reg}}(\text{Flit}, \text{HPC}_{\text{max}}) &= T_{\text{reg2out}}(\text{Flit}) + T_{\text{in2reg}}(\text{Flit}) \\
&\quad + T_{\text{link}}(\text{Flit}) \times \text{HPC}_{\text{max}} + T_{\text{in2out}}(\text{Flit}) \times (\text{HPC}_{\text{max}} - 1) \\
T_{\text{reg2reg}}(\text{Credit}, \text{HPC}_{\text{max}}) &= T_{\text{reg2out}}(\text{Credit}) + T_{\text{in2reg}}(\text{Credit}) \\
&\quad + T_{\text{link}}(\text{Credit}) \times \text{HPC}_{\text{max}} + T_{\text{in2out}}(\text{Credit}) \times (\text{HPC}_{\text{max}} - 1) \\
T_{\text{reg2reg}}(\text{SSR}, \text{HPC}_{\text{max}}) &= T_{\text{reg2out}}(\text{SSR}) + T_{\text{link}}(\text{SSR}) \times \text{HPC}_{\text{max}} + T_{\text{in2reg}}(\text{SSR})
\end{aligned}
$$

We visualize these equations with HPC$_{\text{max}}$ from 1 to 14 for flit and credit, and from 1 to 7 for SSR in Figure 6-14. With HPC$_{\text{max}}$ from 1 to 3, the SSR path is the critical path

Table 6-3: Clock Skew (ns)

(a) Column

0	1	2	3	4	5	6	7
47.22	60.74	70.10	55.82	47.50	75.44	49.33	81.97

(b) Row

0	1	2	3	4	5	6	7
17.82	26.40	25.65	106.43	31.54	20.12	71.69	29.66

because of the high T_{in2reg}(SSR). From HPC$_{\text{max}}$ of 4 and above, the flit path starts to dominate, and lengthens the critical path by 149 ps per additional hop on average.

Clock Skew: Typical mesh network designs only allow flits and credits to be sent to adjacent routers. Therefore, only the clock skew between the adjacent routers needs to be considered and its tolerance is high since the link traversal is not on the critical path. However, since a SMART router may receive data from another router multiple hops away, the clock skew between any pair of routers on the same row or column may lengthen the critical path. Table 6-3 shows the clock skew for each column and row. The maximum clock skew is 106.43 ps.

6.5.4 Timing - Measurement

To determine the maximum achievable frequency of the chip for various HPC$_{\text{max}}$, we increment the clock frequency until faulty or missing packets are observed. We conduct each experiment 10 times. Each time is run with different seeds for 4 billion cycles to ensure a sufficient amount of packets is sent from each router so that most of the paths are covered. The critical path delay is computed to be the multiplicative inverse of the observed frequency.

Flit/Credit Only Path: We run the chip in SMART$_{\text{app}}$ mode and setup a single route from a router to another router multiple hops away. Since only one route is setup, the bypass control logic is determined beforehand, the SSR paths and paths through switch allocation are not used, and hence the critical path would be the flit path. Figure 6-15

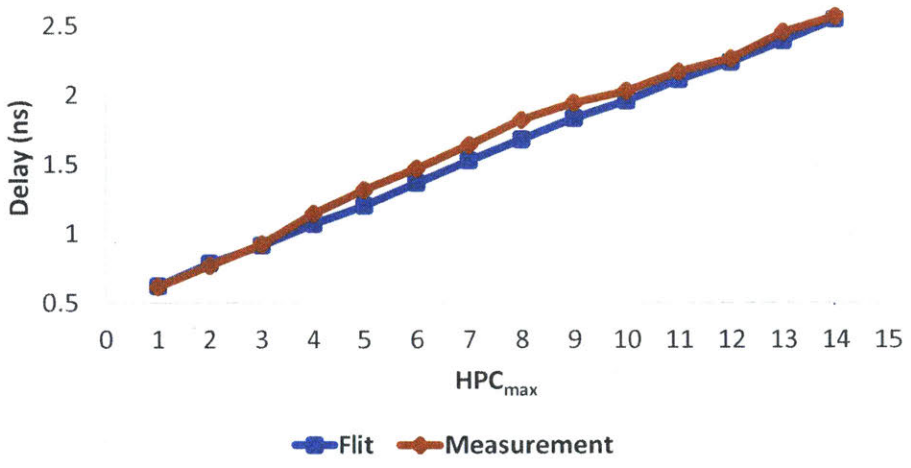


Figure 6-15: Flit/Credit Only Path Delay

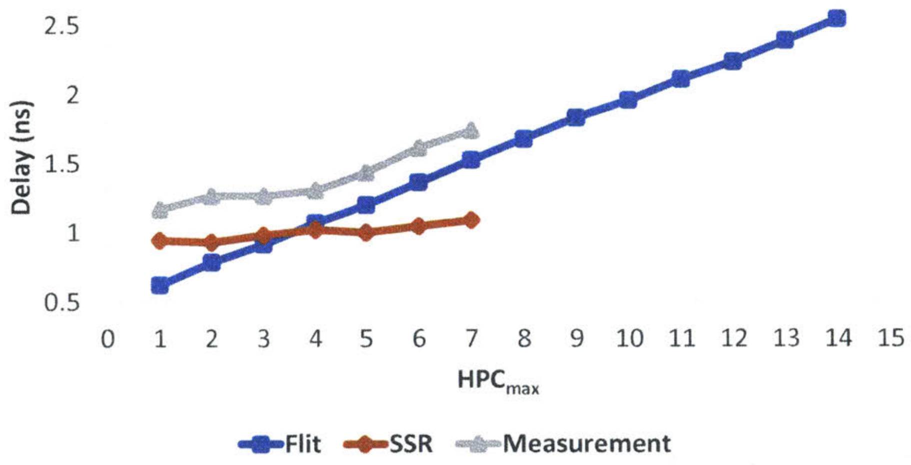
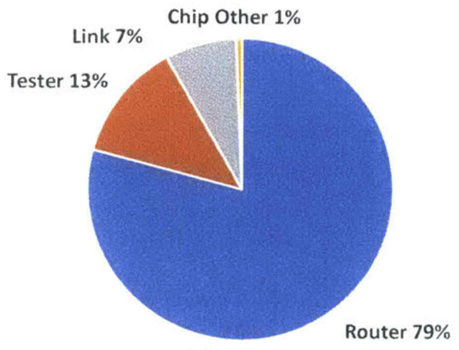


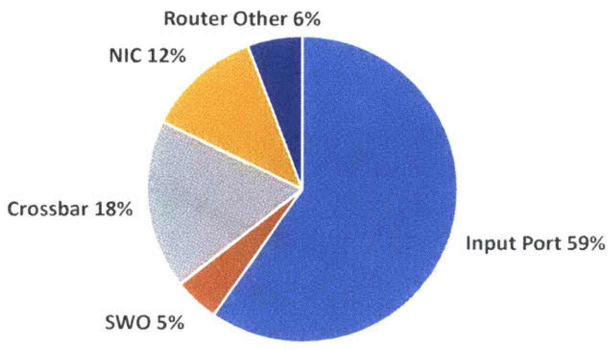
Figure 6-16: Flit/Credit + SSR Path Delay

shows that the measurement results match the estimation from STA, and thus confirms this hypothesis.

Flit/Credit + SSR Path: To take the effect of SSR into consideration, we configure the chip in SMART$_{cycle}$ mode and run traffic patterns such as uniform random, transpose, and bit-complement with various packet injection rates. Figure 6-16 shows that the measurement results follow the estimation from STA with a 250 ps gap. It is because that higher current is drawn at high injection rate, leading to higher IR-drop in power network and hence performance degradation.



(a) Chip



(b) Router

Figure 6-17: Leakage Power Breakdown

6.5.5 Power - Simulation

To estimate chip power consumption, we use Synopsys PrimeTime PX along with signal activities derived from logic simulations on post-layout netlists. All configurations are run at the same clock frequency of 500 MHz. We report the average power consumed for both leakage and dynamic power.

Leakage Power: Figure 6-17 shows the breakdown of the leakage power. In total, the chip consumes 1.6 W, and 86 % of it is consumed by the routers and links. Each router consumes 19.8 mW. Since the input port consists of flit buffers, and most of the routing and flow control logics, it contributes to 59 % of the router leakage power. The reason why the leakage power is high is because that we use regular V_t cells to implement the chip for maximum performance; using high V_t cells would largely reduce the leakage current.

Dynamic Power: We perform simulations with uniform random traffic and various injection rates[12] and HPC_{max} for 20,000 cycles[13], and show the dynamic power breakdown in Figure 6-18. The dynamic power is approximately the same for all HPC_{max}. At injection rate of 0.001, the dynamic power is 0.54 W and 99 % of it is contributed by the clock

[12]Injection rates of 0.001, 0.1, 0.2, 0.3 and 0.4 packets/cycle/router are simulated, where 0.4 is close to the saturation point.

[13]The number of cycles is limited by several constraints, such as simulation time, memory usage, as well as switching activity file size.

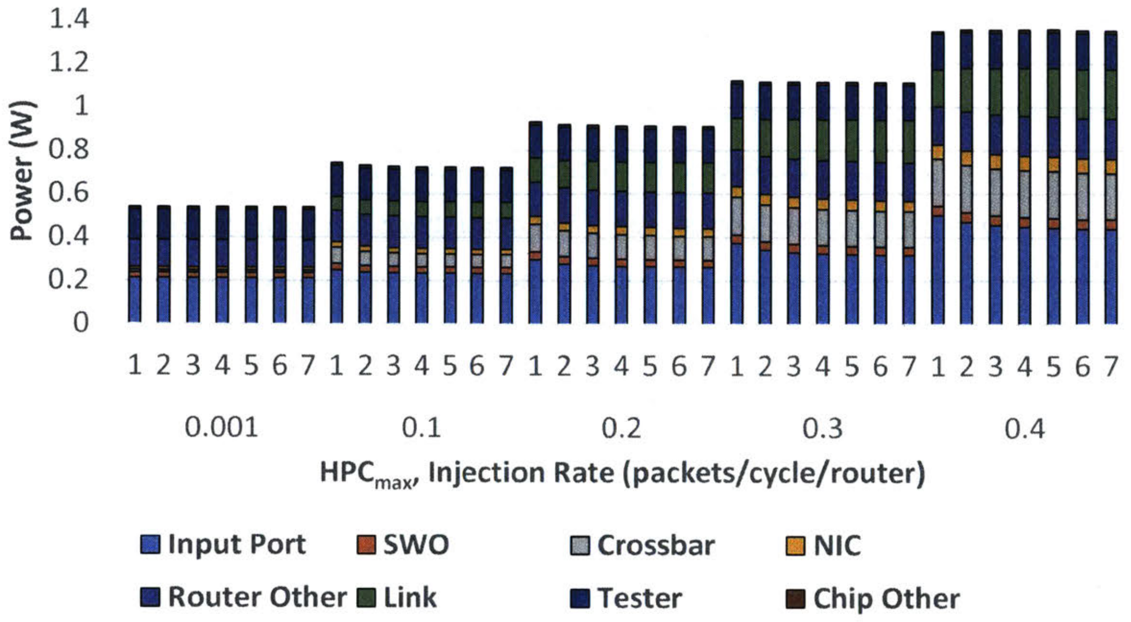


Figure 6-18: Dynamic Power Breakdown

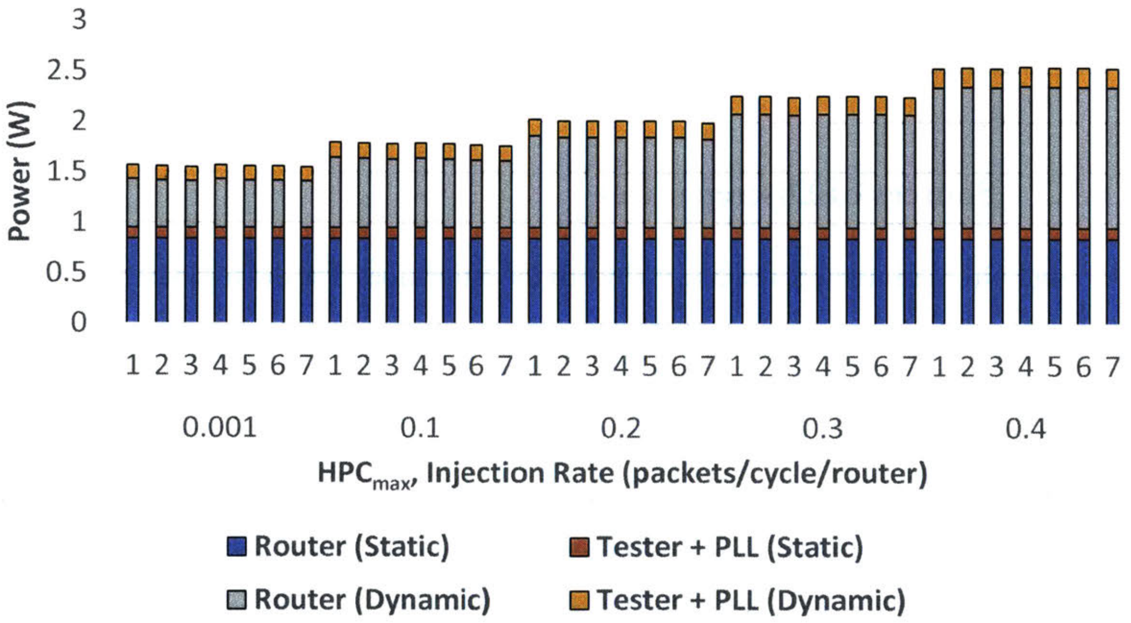


Figure 6-19: Measured Power

network that is not gated. For an injection rate increase of 0.1, the dynamic power is increased by 0.2 W.

6.5.6 Power – Measurement

Similar to the measurement for timing, we also perform the experiments with various seeds and run for 4,000,000,000 cycles. The power measurement is obtained by observing the current drawn from the power supply and multiplying that with the voltage level. We show the results in Figure 6-19. Overall, the measured power is lower than the simulated power by 0.62 W.

Static Power: To measure the static power, we run an experiment in reset mode without the clock reference to ensure zero switching activity. The measured static power is 0.95 W, which is lower than the simulation result. It is possible if the actual chip temperature is lower than simulated temperature of 50 °C. For simplicity, in this example, we assume the static power is the same across all configurations, even though it may increase with the increased temperature induced by higher traffic loads.

Dynamic Power: Similar to the simulation results, the measured power is approximately the same across all HPC_{max}. At zero load, the dynamic power is 0.62 W and is increased by 0.24 W for an injection rate increase of 0.1.

6.5.7 Sources of Discrepancies

Overall, the measurement results are close to the estimation. The discrepancies are mainly contributed by the factors listed below.

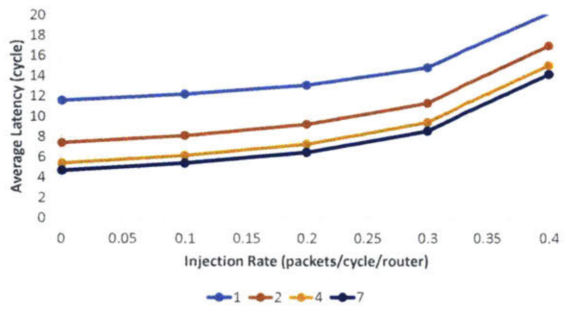
- **Clock skew:** Since a SMART bypass path is across routers multiple hopes away, the clock skew between the *start* and *stop* routers reduces the effective amount of time for flits to travel, and hence reduces the HPC_{max} at a certain clock frequency. In the delay estimation of various paths across multiple hops, we only perform static timing analysis on the paths of a single router without taking the clock skew between routers. To improve the performance of 1D version of $\text{SMART}_{\text{cycle}}$, we need to design a clock network that minimize the clock skew between the routers on the same row/column. And as for 2D version of $\text{SMART}_{\text{cycle}}$, a minimized global clock skew between all pairs of routers is required, which makes it harder to minimize.

- **IR-drop:** The power estimation shows that the higher the injection rate is applied, the higher power (i.e., current) is consumed. However, this higher current induces higher IR-drop and affects the transistor performance, leading to an increase in the critical path delay. As a result, the difference in measured clock period between zero load and highest load is approximately 110 ps. Since the leakage current (nearly 1 A) contributes to a high portion of the total current consumed, to alleviate the IR-drop issue, one way is to replace the cells not on the critical path (i.e., flit path) with high V_t cells to lower the leakage current at no performance cost.
- **Temperature:** Since the total amount of current drawn from the chip is high (i.e., high power consumption), the chip temperature depends on the effectiveness of the cooling system. However, because the cooling system is not taken into consideration while designing the board, the empty space around the package is small, limiting the size and structure of the heat sink as well as the fan. We have tried several heat sinks and chose the one that leads to the least leakage current for our measurement.

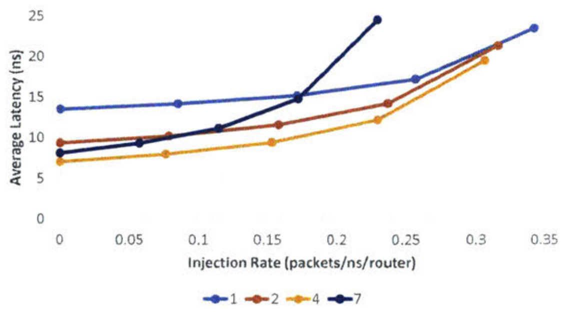
However, the estimation is far off from the design target; only HPC_{max} of 4 can be achieved at a clock frequency of 1 GHz, instead of 7. The design target is set based on the preliminary analyses present in Section 6.2. While the SSR path can nearly achieve HPC_{max} of 7 at the clock frequency of 1 GHz, the difference is mainly because of the complicated timing relationship between the crossbar selection, flit input and output signals of the router. As a result, coarse grain timing constraints applied on these paths lead to a high register-to-output delay (327.52 ps) and high input-to-register delay (233.34 ps) than assumed, which is the through path delay (95.47 ps). To close the gap, finer grain timing constraints, which tightly bound the paths for various scenarios, are required.

6.5.8 Insights

While running with the same clock frequency, a higher HPC_{max} leads to a lower average low-load latency (see Figure 6-20a); i.e., HPC_{max} of 7 yields the lowest low-load latency. Figure 6-20b shows the same figure when we swap the *cycle* with the measured minimum clock period (i.e., inverse of the measured clock frequency). It should be noted that in



(a) Same Frequency



(b) Different Frequencies (Measurement)

Figure 6-20: Average Latency versus Injection Rate

Figure 6-20a, the x-axis is in flits/ns/router and y-axis is in ns, instead of flits/cycle/router and cycle.

Even though HPC_{max} of 7 allows traversing more hops in a single cycle, the performance increase is marginal and thus the slow clock frequency makes it unfavorable. HPC_{max} of 4 now presents the lowest network latency at low-load. And HPC_{max} of 1 provides the highest throughput since it can be run at the highest clock frequency.

It should be noted that the performance of the SMART network with HPC_{max} of 1 is equivalent to the performance of a network with conventional 2-cycle routers; that is, the clock frequency of the conventional 2-cycle router needs to be twice as fast as the SMART network with HPC_{max} of 4 to beat the SMART network in average latency.

The key takeaway is that the SMART network achieves low-latency by sacrificing clock frequency (i.e., lower throughput), and is suitable for applications that are sensitive to average latency but not throughput. The downside of the SMART network is that its clock frequency may need to be different from the clock frequency for cores which is typically 1 to 2 GHz to achieve the lowest average latency. As for the area and power, even though it takes more area to implement the SMART network, the lower frequency may lead to a lower dynamic power consumption compared to conventional 2-cycle router.

6.6 Summary

In this chapter, we present preliminary analyses to show the tradeoff between hardware cost and HPC_{max}. Then, we present a case study of a 64-node SMART network to demonstrate the feasibility of the SMART network, and to further study its timing and power through simulations and measurements. Our measured results show that the chip works at 817.1 MHz with $HPC_{max} = 1$, and at 548 MHz with $HPC_{max} = 7$. The chip consumes 1.57 to 2.53 W across various runtime configurations. We also point out the critical issues that can be addressed to close the gap between the measurement results and the design target, and hence to further improve the performance.

7

SCORPIO - A 36-core Shared Memory Processor Demonstrating Snoopy Coherence on a Mesh Interconnect

This is joint work with Bhavya Daya, Woo Cheol Kwon, Suvinay Subramanian, Sunghyun Park and Tushar Krishna [28]. I co-led the SCORPIO project with Bhavya Daya, with her as the architecture lead, while I was the chip RTL and design lead.

7.1 Motivation

Shared memory, a dominant communication paradigm in mainstream multicore processors today, achieves inter-core communication using simple loads and stores to a shared address space, but requires mechanisms for ensuring cache coherence. Over the past few decades, research in cache coherence has led to solutions in the form of either snoopy or directory-based variants. However, a critical concern is whether hardware-based coherence will scale with the increasing core counts of chip multiprocessors [49, 66]. Existing coherence schemes can provide accurate functionality for up to hundreds of cores, but area, power, and bandwidth overheads affect their practicality. Two main scalability concerns are (1) directory storage overhead, and (2) uncore (caches+interconnect) scaling.

For scalable directory-based coherence, the directory storage overhead has to be kept minimal while maintaining accurate sharer information. Full bit-vector directories encode the set of sharers of a specific address. For a few tens of cores, it is very efficient, but requires storage that linearly grows with the number of cores; limiting its use for larger systems. Alternatives, such as coarse-grain sharer bit-vectors and limited pointer schemes contain inaccurate sharing information, essentially trading performance for scalability. Research in scalable directory coherence is attempting to tackle the storage overhead while maintaining accurate sharer information, but at the cost of increased directory evictions and corresponding network traffic as a result of the invalidations.

Snoopy coherence is not impacted by directory storage overhead, but intrinsically requires an ordered network to ensure all cores see requests in the same order to maintain memory consistency semantics. Snoopy compatible interconnects comprise buses or crossbars (with arbiters to order requests), or bufferless rings (which guarantee in-order delivery to all cores from an ordering point). However, existing on-chip ordered interconnects scale poorly. The Achilles heel of buses lie in limited bandwidth, while that of rings is delay, and for crossbars, it is area. Higher-dimension NoCs such as meshes provide scalable bandwidth and is the subject of a plethora of research on low-power and low-latency routers, including several chip prototypes [36, 43, 91, 110]. However, meshes are unordered and cannot natively support snoopy protocols.

Snoopy **CO**herent **R**esearch **P**rocessor with **I**nterconnect **O**rdering (SCORPIO) incorporates global ordering support within the mesh network by decoupling message delivery from the ordering. This allows flits to be injected into the NoC and reach destinations in any order, at any time, and still maintain a consistent global order; as a result, SCORPIO enjoys both the low-area benefit from snoopy coherence and low-latency/high-bandwidth benefit from the mesh network. The SCORPIO architecture was included in an $11 \times 13\,\text{mm}^2$ chip prototype in IBM 45 nm SOI, to interconnect 36 commercial Power Architecture cores, comprising private L1 and L2 caches, and two Cadence on-chip DDR controllers. The SCORPIO NoC is designed to comply with the

ARM AMBA interface [7] to be compatible with existing SoC IP originally designed for AMBA buses.

Section 7.2 delves into the overview and microarchitecture of the globally ordered mesh network. Section 7.3 describes the designed and fabricated 36-core chip with the SCORPIO NoC. Section 7.4 presents the evaluations and design exploration of the SCORPIO architecture with software models. Section 7.5 demonstrates the evaluations of the chip with the implemented RTL, and area/power results. Section 7.6 shows the lessons learned from the SCORPIO development. Section 7.7 discusses related multicore chips and NoC prototypes, and Section 7.8 summarizes.

7.2 Globally Ordered Mesh Network

Traditionally, global message ordering on interconnects relies on a centralized ordering point, which imposes greater *indirection*[1] and *serialization latency*[2] as the number of network nodes increases. The dependence on the centralized ordering point prevents architects from providing global message ordering guarantee on scalable but unordered networks.

To tackle the problem above, we propose the SCORPIO network architecture. We eliminate the dependence on the centralized ordering point by decoupling message ordering from message delivery using two physical networks, as shown in Figure 7-1; we use the *main network* to deliver the messages and *notification network* to help determine the global order of messages. The key idea is to send messages over a high-performance unordered network and ensure the messages are consumed in the same global order at all nodes. We next describe the mechanism of the two networks as well as the interaction between them, followed by a walkthrough example for better understanding.

Main network: The main network is an unordered network and is responsible for broadcasting actual coherence requests to all other nodes and delivering the responses to the requesting nodes. Since the network is unordered, the broadcast coherence requests

[1]Network latency of a message from the source node to ordering point.

[2]Latency of a message waiting at the ordering point before it is ordered and forwarded to other nodes.

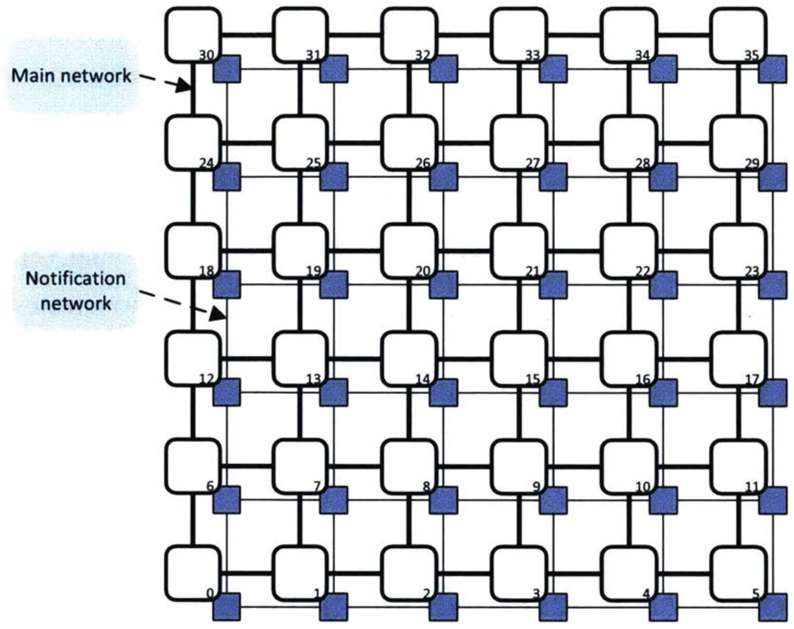


Figure 7-1: Proposed SCORPIO Network

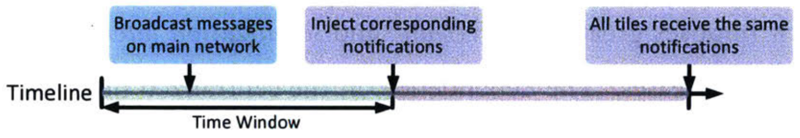

Broadcast messages on main network
Inject corresponding notifications
All tiles receive the same notifications
Timeline
Time Window

Figure 7-2: Time Window for Notification Network

from different source nodes may arrive at the network interface controllers (NIC) of each node in any order. The NICs of the main network are then responsible for forwarding requests in global order to the cache controller, assisted by the notification network.

Notification network: For every coherence request sent on the main network, a notification message encoding the source node's ID (SID) is broadcast on the notification network to notify all nodes that a coherence request from this source node is in-flight and needs to be ordered. Then, the goal is transformed to ensure all nodes receive the notification messages, instead of the corresponding coherence messages, in the same order. To achieve this, we maintain synchronous time windows, send notification messages only at the beginning of each time window, and design the notification network so that all nodes receive the same set of notifications at the end of that time window, as shown in Figure 7-2. By processing the received notification messages in accordance with a *consistent* ordering rule, all NICs determine *locally* the *global* order for the actual coherence request in the main network. To fulfill the requirements of the notification network, we define the notification message to be a bit vector with a length of the number of nodes, where each bit corresponds to a coherence request from a source node, so that the notification messages can be merged by OR-ing without information loss. As a result, the notification network is contention-less and has a fixed maximum network latency bound, which we can use to determine the size of the time window.

Network interface controller: Each node in the system consists of a main network router, a notification router, as well as a network interface controller or logic interfacing the core/cache and the two routers. The NIC encapsulates the coherence requests/responses from the core/cache and injects them into the appropriate virtual networks in the main network. On the receive end, it forwards the received coherence requests to the core/cache in accordance with the global order, which is determined using the received notification messages at the end of each time window. The NIC uses an *Expected Source ID* (ESID) register to keep track of and informs the main network router which coherence request it is waiting for. For example, if the ESID stores a value of 3, it means that the NIC is waiting for a coherence request from node 3 and would not forward coherence requests from other nodes to the core/cache. Upon receiving the request from node 3, the NIC updates the ESID and waits for the next request based on the global order determined using the received notification messages. The NIC forwards coherence responses to the core/cache in any order.

7.2.1 Walkthrough Example

Next, we walkthrough an example to demonstrate how two messages are ordered.

1. As shown in Figure 7-3, at times T1 and T2, the cache controllers inject cache miss messages M1, M2 to the NIC at core 11, 1 respectively. The NICs encapsulate these coherence requests into single flit packets, tag them with the SID of their source (11, 1 respectively), and broadcast them to all nodes in the main network.
2. At time T3, the start of the time window, notification messages N1 and N2 are generated corresponding to M1 and M2, and sent into the notification network.

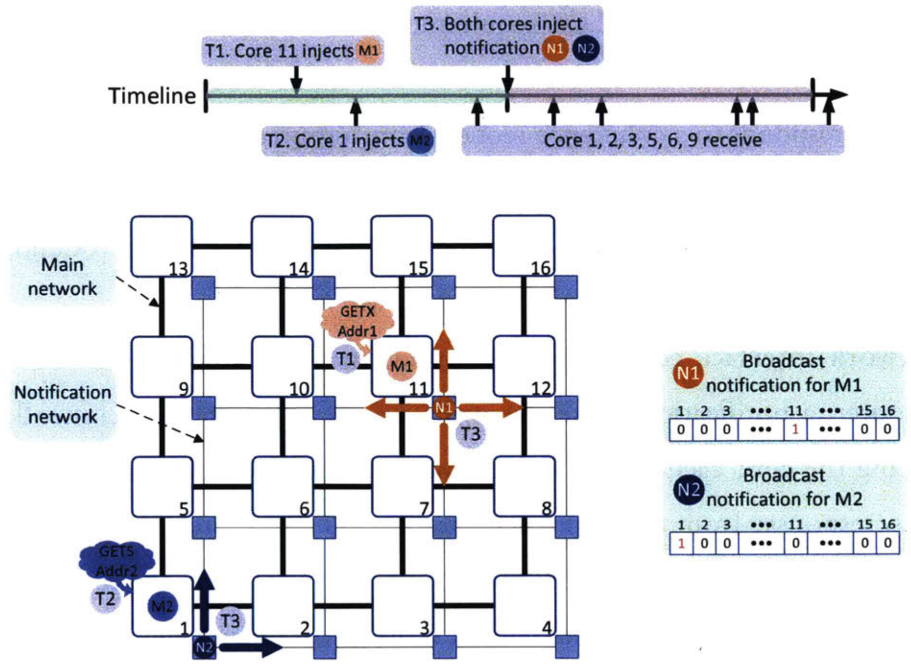


Figure 7-3: Walkthrough Example (from T1 to T3)

3. As shown in Figure 7-4, notification messages broadcast at the start of a time window are guaranteed to be delivered to all nodes by the end of the time window (T4). At this stage, all nodes process the notification messages received and perform a local but consistent decision to order these messages. In SCORPIO, we use a rotating priority arbiter to order messages according to increasing SID – the priority is updated each time window ensuring fairness. In this example, all nodes decide to process M2 before M1.

4. The decided global order is captured in the ESID register in the NIC. In this example, ESID is currently 1 – the NICs are waiting for the message from core 1 (i.e., M2).

5. At time T5, when a coherence request arrives at a NIC, the NIC performs a check of its source ID (SID). If the SID matches the ESID then the coherence request is processed (i.e., dequeued, parsed and handed to the cache controller) else it is held in the NIC buffers. Once the coherence request with the SID equal to ESID is processed, the ESID is updated to the next value (based on the notification messages received). In this example, the NIC has to forward M2 before M1 to the cache controller. If

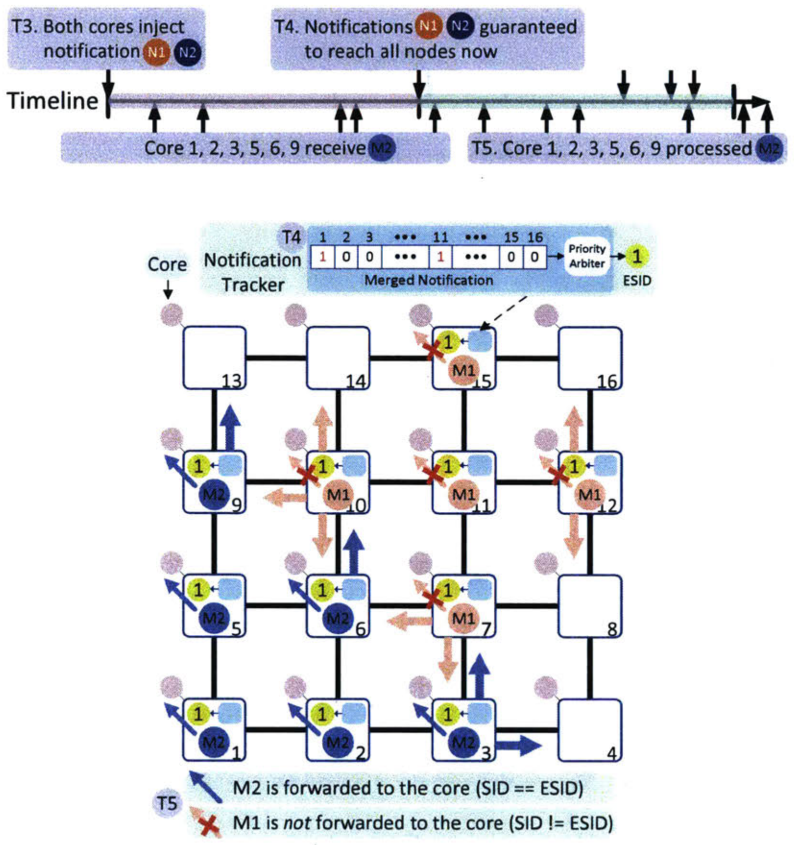

T3. Both cores inject notification N1 N2
T4. Notifications N1 N2 guaranteed to reach all nodes now
Timeline
Core 1, 2, 3, 5, 6, 9 receive M2
T5. Core 1, 2, 3, 5, 6, 9 processed M2
T4 Notification Tracker
Core
Merged Notification
Priority Arbiter
ESID
M2 is forwarded to the core (SID == ESID)
M1 is not forwarded to the core (SID != ESID)
T5

Figure 7-4: Walkthrough Example Cont. (from T4 to T5)

M1 arrives first, it will be buffered in the NIC (or router, depending on the buffer availability at NIC) and wait for M2 to arrive.

6. As shown in Figure 7-5, core 6 and 13 respond to M1 (at T7) and M2 (at T6) respectively. All cores thus process all messages in the same order (i.e., M2 followed by M1).

7.2.2 Main Network Microarchitecture

Figure 7-6 shows the microarchitecture of the three-stage main network router. During the first pipeline stage, the incoming flit is buffered (BW), and in parallel arbitrates with the other virtual channels (VCs) at that input port for access to the crossbar's input port (SA-I). In the second stage, the winners of SA-I from each input port arbitrate for the crossbar's output ports (SA-O), and in parallel obtain a free VC at the next router if

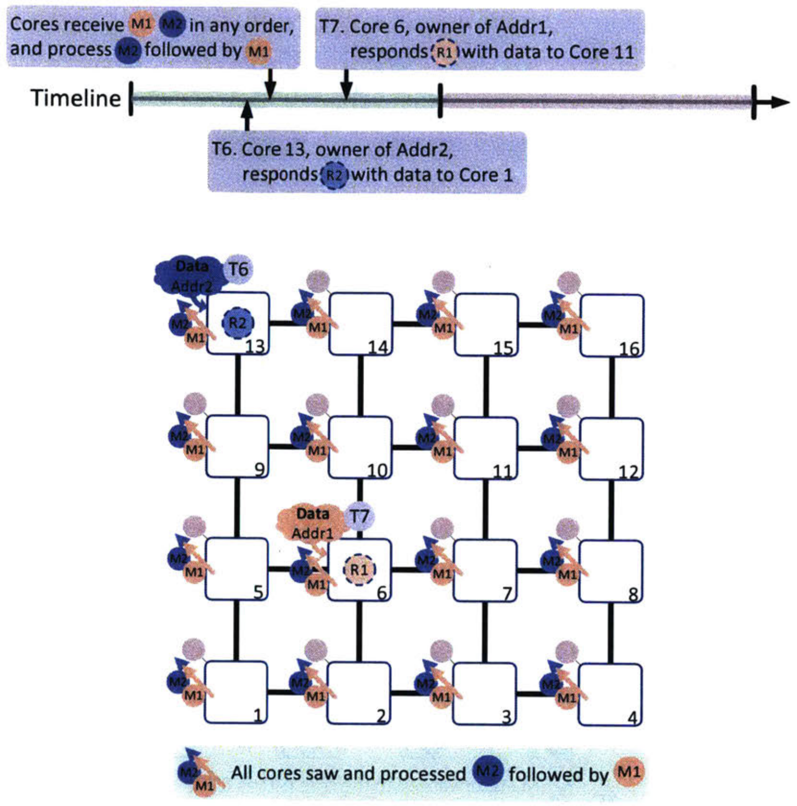

Cores receive M1 M2 in any order, and process M2 followed by M1
T7. Core 6, owner of Addr1, responds R1 with data to Core 11
Timeline
T6. Core 13, owner of Addr2, responds R2 with data to Core 1
Data Addr2
T6
R2
Data Addr1
T7
R1
M2
M1
13
14
15
16
9
10
11
12
5
6
7
8
1
2
3
4
All cores saw and processed M2 followed by M1

Figure 7-5: Walkthrough Example Cont. (from T6 to T7)

possible (VA). In the final stage, the winners of SA-O traverse the crossbar (ST). Next, the flits traverse the link to the adjacent router in the following cycle.

Single-cycle pipeline optimization: To reduce the network latency and buffer read/write power, we implement lookahead (LA) bypassing [62, 91]; a lookahead containing control information for a flit is sent to the next router during that flit's ST stage. At the next router, the lookahead performs route-computation and tries to pre-allocate the crossbar for the approaching flit. Lookaheads are prioritized over buffered flits[3] – they attempt to win SA-I and SA-O, obtain a free VC at the next router, and setup the crossbar for the approaching flits, which then bypass the first two stages and move to ST stage directly. Conflicts between lookaheads from different input ports are resolved using a static, rotating priority scheme. If a lookahead is unable to setup the crossbar, or obtain a free VC at the next router, the incoming flit is buffered and goes through all three stages. The control

[3]Only buffered flits in the reserved VCs, used for deadlock avoidance, are an exception, prioritized over lookaheads.

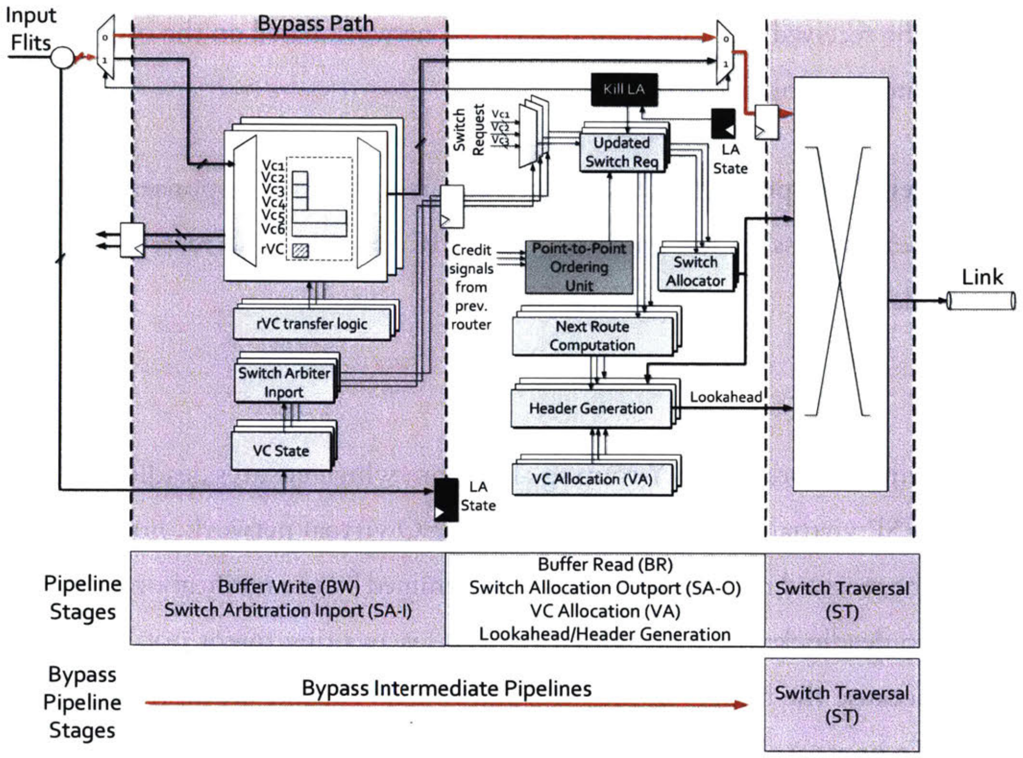


Figure 7-6: Router Microarchitecture

information carried by lookaheads is already included in the header field of conventional NoCs – destination coordinates, VC ID and the output port ID – and hence does not impose any wiring overhead.

Single-cycle broadcast optimization: To alleviate the overhead imposed by the coherence broadcast requests, routers are equipped with single-cycle multicast support [91]. Instead of sending the same requests for each node one by one into the main network, we allow requests to fork through multiple router output ports in the same cycle, thus providing efficient hardware broadcast support.

Deadlock avoidance: The snoopy coherence protocol messages can be grouped into network requests and responses. Thus, we use two message classes or virtual networks to avoid protocol-level deadlocks:

- **Globally Ordered Request (GO-REQ):** Delivers coherence requests, and provides global ordering, lookahead-bypassing and hardware broadcast support. The NIC processes the received requests from this virtual network based on the order determined by the notification network.

- **Unordered Response (UO-RESP):** Delivers coherence responses, and supports lookahead-bypassing for unicasts. The NIC processes the received responses in any order.

The main network uses XY-routing algorithm which ensures deadlock-freedom for the UO-RESP virtual network. For the GO-REQ virtual network, however, the NIC processes the received requests in the order determined by the notification network which may lead to deadlock; the request that the NIC is awaiting might not be able to enter the NIC because the buffers in the NIC and routers enroute are all occupied by other requests. To prevent the deadlock scenario, we add one reserved virtual channel (rVC) to each router and NIC, reserved for the coherence request with SID equal to ESID that the NIC at that router is waiting for. Thus, we can ensure that the requests can always proceed toward the destinations.

Point-to-point ordering for GO-REQ: In addition to enforcing a global order, requests from the *same* source also need to be ordered with respect to each other. Since requests are identified by source ID alone, the main network must ensure that a later request does not overtake an earlier request from the same source. To enforce this in SCORPIO, the following property must hold: *Two requests at a particular input port of a router, or at the NIC input queue cannot have the same SID.* At each output port, a SID tracker table keeps track of the SID of the request in each VC at the next router.

Suppose a flit with SID = 5 wins the north port during SA-O and is allotted VC 1 at the next router in the north direction. An entry in the table for the north port is added, mapping (VC 1) $\rightarrow$ (SID = 5). At the next router, when flit with SID = 5 wins all its required output ports and leaves the router, a credit signal is sent back to this router and then the entry is cleared in the SID tracker. Prior to the clearance of the SID tracker entry, any request with SID = 5 is prevented from placing a switch allocation request.

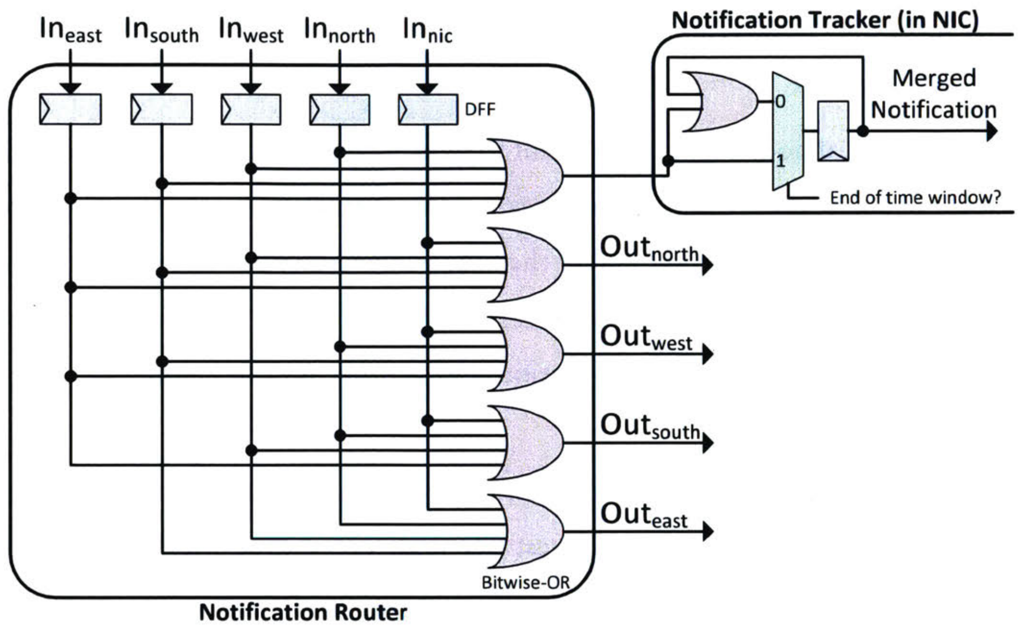


Figure 7-7: Notification Router Microarchitecture

7.2.3 Notification Network Microarchitecture

The notification network is an ultra-lightweight bufferless mesh network consisting of 5 N-bit bitwise-OR gates and 5 N-bit latches at each *router* as well as N-bit links connecting these *routers*, as shown in Figure 7-7, where N is the number of cores. A notification message is encoded as a N-bit vector where each bit indicates whether a core has sent a coherence request that needs to be ordered. With this encoding, the notification router can merge two notification messages via a bitwise-OR of two messages then forward the merged message to the next router. At the beginning of a time window, a core that wants to send a notification message asserts its associated bit in the bit-vector and sends the bit-vector to its notification router. Every cycle, each notification router merges received notification messages and forwards the updated message to all its neighbor routers in the same cycle. Since messages are merged upon contention, messages can always proceed through the network without being stopped, and hence, no buffer is required and network latency is bounded. At the end of that time window, it is guaranteed that all nodes in the network receive the same merged message, and this message is then sent to the NIC for further processing to determine the global order of the corresponding coherence requests in the main network.

For example, if node 0 and node 6 want to send notification messages, at the beginning of a time window, they send the messages with bit 0 and bit 6 asserted, respectively, to their notification routers. At the end of the time window, all nodes receive a final message with both bits 0 and 6 asserted. In a 6×6 mesh notification network, the maximum latency is 6 cycles along the X dimension and another 6 cycles along Y, so the time window is set to 13 cycles.

Multiple requests per notification message: Thus far, the notification message described handles one coherence request per node every time window, i.e. only one coherence request from each core can be ordered within a time window. However, this is inefficient for more aggressive cores that have more outstanding misses. For example, when the aggressive core generates 6 requests at around the same time, the last request can only be ordered at the end of the 6th time window, incurring latency overhead. To resolve this, instead of using only 1 bit per core, we dedicate multiple bits per core to encode the number of coherence requests that a core wants to order in this time window, at a cost of larger notification message size. For example, if we allocate two bits instead of 1 per core in the notification message, the maximum number of coherence requests can be ordered in this time window can be increased to 3[4]. Now, the core sets the associated bits to the number of coherence requests to be ordered and leaves other bits as zero. This allows us to continue using the bitwise-OR to merge the notification messages from other nodes.

7.2.4 Network Interface Controller Microarchitecture

Figure 7-8 shows the microarchitecture of the NIC, which interfaces between the core/cache and the main and notification network routers.

[4]The number of coherence requests is encoded in binary, where a value of 0 means no request to be ordered, 1 implies 1 request, while 3 indicates 3 requests to be ordered (maximum value that a 2-bit number can represent).

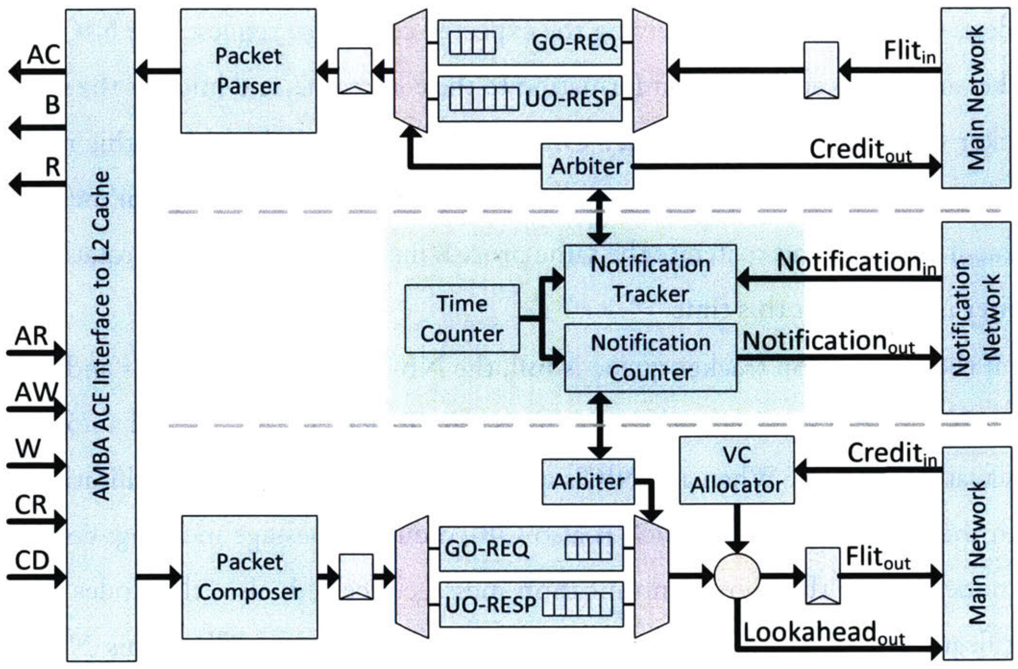


Figure 7-8: Network Interface Controller Microarchitecture

Sending notifications: On receiving a message from core/cache, the NIC encapsulates the message into a packet and sends it to the appropriate virtual network. If the message is a coherence request, the NIC needs to send a notification message so that the coherence request can be ordered. Since the purpose of the notification network is to decouple the coherence request ordering from the request delivery, the NIC can always send the coherence requests to the main network whenever possible and send the corresponding notification messages at the beginning of later time windows. We use a counter to keep track of how many pending notification messages still remain to be sent. The counter can be sized arbitrarily for expected bursts; when the maximum number of pending notification messages, represented by this counter, is reached, the NIC blocks new coherence requests from injecting into the main network.

Receiving notifications: At the end of every time window, the NIC pushes the received merged notification message into the notification tracker queue. When the notification tracker queue is not empty and there is no previously read notification message being processed, the head of the queue is read and passed through a rotating priority arbiter to determine the order of processing the incoming coherence requests (i.e.

to determine ESIDs). On receiving the expected coherence request, the NIC parses the packet and passes appropriate information to the core/cache, and informs the notification tracker to update the ESID value. Once all the requests indicated by this notification message are processed, the notification tracker reads the next notification message in the queue if available and re-iterate the same process mentioned above. The rotating priority arbiter is updated at this time.

If the notification tracker queue is full, the NIC informs other NICs and suppresses other NICs from sending notification messages. To achieve this, we add a *stop* bit to the notification message. When any NIC's queue is full, that NIC sends a notification message with the *stop* bit asserted, which is also OR-ed during message merging; consequently all nodes ignore the merged notification message received; also, the nodes that sent a notification message this time window will resend it later. When this NIC's queue becomes non-full, the NIC sends the notification message with the *stop* bit de-asserted. All NICs are enabled again to (re-)send pending notification messages when the *stop* bit of the received merged notification message is de-asserted.

7.3 36-Core Processor with SCORPIO NoC

The 36-core fabricated multicore processor is arranged in a grid of 6×6 tiles, as seen in Figure 7-9 and 7-10. Within each tile is an in-order core, split L1 I/D caches, private L2 cache with MOSI snoopy coherence protocol, L2 region tracker for destination filtering [81], and SCORPIO NoC (see Table 7-1 for a full summary of the chip features). The commercial Power Architecture core simply assumes a bus is connected to the AMBA AHB data and instruction ports, cleanly isolating the core from the details of the network and snoopy coherence support. Between the network and the processor core IP is the L2 cache with AMBA AHB processor-side and AMBA ACE network-side interfaces. Two Cadence DDR2 memory controllers attach to four unique routers along the chip edge, with the Cadence IP complying with the AMBA AXI interface, interfacing with Cadence PHY to off-chip DIMM modules. All other IO connections go through an external FPGA board with the connectors for RS-232, Ethernet, and flash memory.

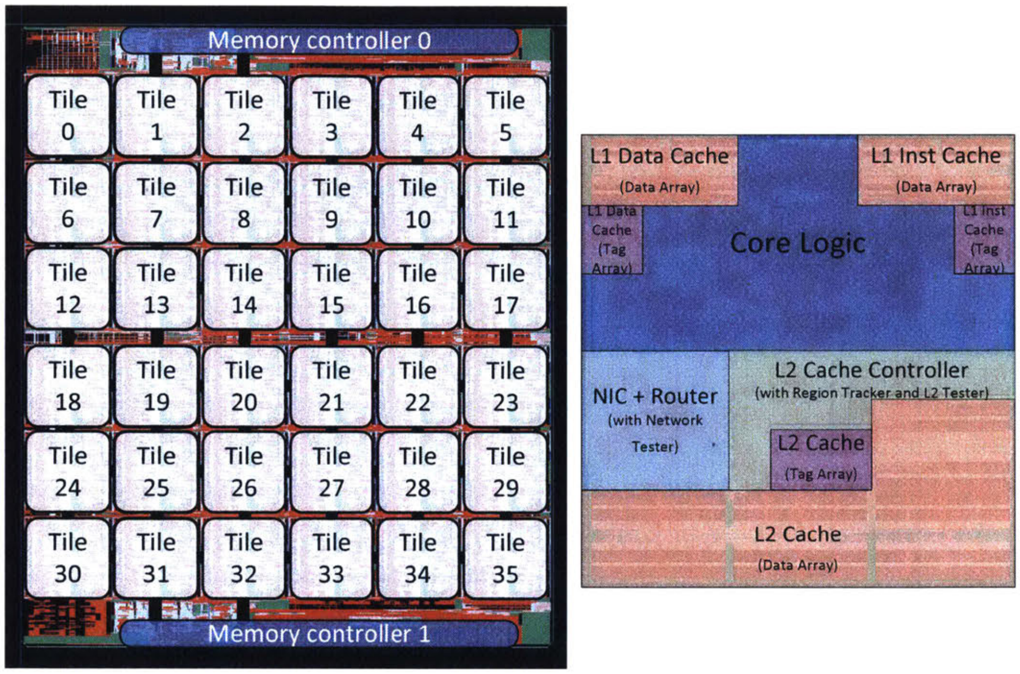


Figure 7-9: 36-core Chip Layout with SCORPIO NoC

7.3.1 Processor Core and Cache Hierarchy Interface

While the ordered SCORPIO NoC can plug-and-play with existing ACE coherence protocol controllers, we were unable to obtain such IP and hence designed our own. The cache subsystem comprises L1 and L2 caches and the interaction between a self-designed L2 cache and the processor core's L1 caches is mostly subject to the core's and AHB's constraints.

The core has a split instruction and data 16 KB L1 cache with independent AHB ports. The ports connect to the multiple master split-transaction AHB bus with two AHB masters (L1 caches) and one AHB slave (L2 cache). The protocol supports a single read or write transaction at a time, hence there is a simple request or address phase, followed by a response or data phase. Transactions, between pending requests from the same AHB port, are not permitted thereby restricting the number of outstanding misses to two, one data cache miss and one instruction cache miss, per core. For multilevel caches, snooping hardware has to be present at both L1 and L2 caches. However, the core was not originally designed for hardware coherency. Thus, we added an invalidation

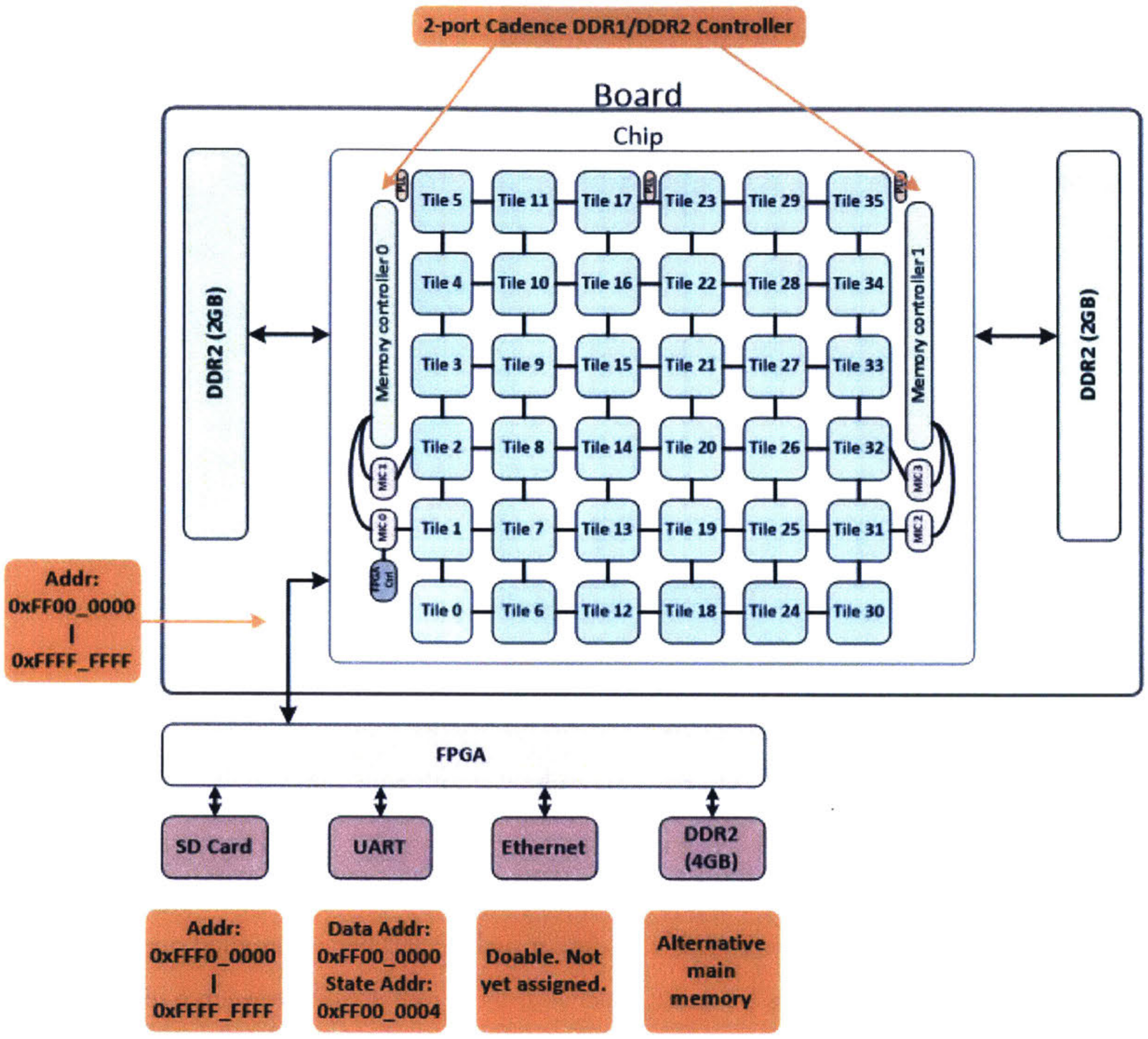


Figure 7-10: 36-core Chip Schematic

port to the core allowing L1 cachelines to be invalidated by external input signals. This method places the inclusion requirement on the caches. With the L1 cache operating in write-through mode, the L2 cache will only need to inform the L1 during invalidations and evictions of a line.

7.3.2 Coherence Protocol

The standard MOSI protocol is adapted to reduce the writeback frequency and to disallow the blocking of incoming snoop requests. Writebacks cause subsequent cacheline accesses

Table 7-1: SCORPIO chip features

Name	Value
Process	IBM 45 nm SOI
Dimension	$11 \times 13\,\text{mm}^2$
Transistor count	600 M
Gate count	88.9 M
Frequency	833 MHz
Power	28.8 W
Core	Dual-issue, in-order, 10-stage pipeline
ISA	32-bit Power Architecture™
L1 cache	Private split 4-way set associative write-through 16 KB I/D
L2 cache	Private inclusive 4-way set associative 128 KB
L2 replacement policy	Pseudo LRU
Line Size	32 B
Coherence protocol	MOSI (O: forward state)
Directory cache	128 KB (1 owner bit, 1 dirty bit)
Snoop filter	Region tracker (4KB regions, 128 entries)
NoC Topology	6×6 mesh
Channel width	137 bits (Ctrl packets - 1 flit, data packets - 3 flits)
Virtual networks	1. Globally ordered - 4 VCs, 1 buffers each 2. Unordered - 2 VCs, 3 buffers each
Router	XY routing, cut-through, multicast, lookahead bypassing
Pipeline	3-stage router (1-stage with bypassing), 1-stage link
Notification network	36-bits wide, bufferless, 13 cycles time window, max 4 pending messages
Memory controller	2× Dual port Cadence DDR2 memory controller + PHY
FPGA controller	1× Packet-switched flexible data-rate controller

to go off-chip to retrieve the data, degrading performance, hence we retain the data on-chip for as long as possible. To achieve this, an additional O_D state instead of a dirty bit per line is added to permit on-chip sharing of dirty data. For example, if another core wants to write to the same cacheline, the request is broadcast to all cores resulting in invalidations, while the owner of the dirty data (in M or O_D state) will respond with the dirty data and change itself to the Invalid state. If another cores wants to read the same cacheline, the request is broadcast to all cores. The owner of the dirty data (now in M state), responds with the data and transitions to the O_D state, and the requester goes to the Shared state. This ensures the data is only written to memory when an eviction occurs, without any overhead because the O_D state does not require any additional state bits.

When a cacheline is in a transient state due to a pending write request, snoop requests to the same cacheline are stalled until the data is received and the write request is completed. This causes the blocking of other snoop requests even if they can be serviced right away. We service all snoop requests without blocking by maintaining a forwarding IDs (FID) list that tracks subsequent snoop requests that match a pending write request. The FID consists of the SID and the request entry ID or the ID that matches a response to an outstanding request at the source. With this information, a completed write request can send updated data to all SIDs on the list. The core IP has a maximum of 2 outstanding messages at a time, hence only two sets of forwarding IDs are maintained per core. The SIDs are tracked using a N bit-vector, and the request entry IDs are maintained using $2N$ bits. For larger core counts and more outstanding messages, this overhead can be reduced by tracking a smaller subset of the total core count. Since the number of sharers of a line is usually low, this will perform as well as being able to track all cores. Once the FID list fills up, subsequent snoop requests will then be stalled.

The different messages types are matched with appropriate ACE channels and types. The network interface retains its general mapping from ACE messages to packet type encoding and virtual network identification resulting in a seamless integration. The L2 cache was thus designed to comply with the AMBA ACE specification. It has five outgoing channels and three incoming channels (see Figure 7-8), separating the address and data among different channels. ACE is able to support snoop requests through its Address Coherent (AC) channel, allowing us to send other requests to the L2 cache.

7.3.3 Functional Verification

Besides the unit tests to ensure the correct functionality of each component, Table 7-2 lists the regression tests we used to verify the entire chip. Since the core is a verified commercial IP, our regression tests focus on verifying integration of various components, which involves the following:

Table 7-2: Regression Tests

Test Name	Description
hello	Performs basic load/store and arithmetic operations on non-overlapped cacheable regions.
mem patterns	Performs load/store operations for different data types on non-overlapped cacheable regions.
config space	Performs load/store operations on non-cacheable regions.
flash copy	Transfers data from the flash memory to the main memory.
sync	Uses flags and performs `msync` operation.
atom smashers	Uses spin locks, ticket locks and ticket barriers, and performs operations on shared data structures.
ctt	Performs a mixture of arithmetic, lock/unlock, load/store operations on overlapped cacheable regions.
intc	Performs store operations on the designate interrupt address which triggers other cores' interrupt handler.

```
#include <support.h>
#include <lib/common/utils.h>

volatile uint32_t A __attribute__ ((section(".syncvars"))) = 0;
volatile uint32_t B __attribute__ ((section(".syncvars"))) = 0;

int main(int argc, char *argv[]) {
    uint32_t core_id = getCoreID();         // Get its own core id
    if (core_id == 0) {
        A = 1;
        asm volatile("sync" : : : "memory");// "A = 1" is seen by other cores
        B = 1;
        asm volatile("sync" : : : "memory");// "B = 1" is seen by other cores
    } else if (core_id == 1) {
        while (B == 0) { }                  // Spin while B is 0
        if (A != 1) {                       // B is set to 1, then A should 1 too
            exit_fail();
        }
    }
    exit_pass();
}
```

Figure 7-11: *sync* Test for 2 Cores

- Load/store operations on both cacheable and non-cacheable regions.
- `lwarx`, `stwcx` and `msync` instructions.
- Coherency between L1s, L2s and main memory.
- Software-triggered interrupt.

For brevity, Figure 7-11 shows the code segment of the shortest *sync* test. The tests are written in assembly and C, and we built a software chain that compiles tests into machine codes for SCORPIO.

7.4 Architecture Analysis

Modeled system: For full-system architectural simulations of SCORPIO, we use Wind River Simics [121] extended with the GEMS toolset [75] and the GARNET [3] network model. The SCORPIO and baseline architectural parameters as shown in Table 7-1 are faithfully mimicked within the limits of the GEMS and GARNET environment:

- GEMS only models in-order SPARC cores, instead of SCORPIO's Power cores.
- L1 and L2 cache latency in GEMS are fixed at 1 cycle and 10 cycles. The prototype L2 cache latency varies with request type and cannot be expressed in GEMS, while the L1 cache latency of the core IP is 2 cycles.
- The directory cache access latency is set to 10 cycles and DRAM to 80 cycles in GEMS. The directory cache access was approximated from the directory cache parameters, but vary depending on request type for the chip.
- The L2 cache, NIC, and directory cache accesses are fully-pipelined in GEMS.
- Maximum of 16 outstanding messages per core in GEMS, unlike our chip prototype which has a maximum of two outstanding messages per core.

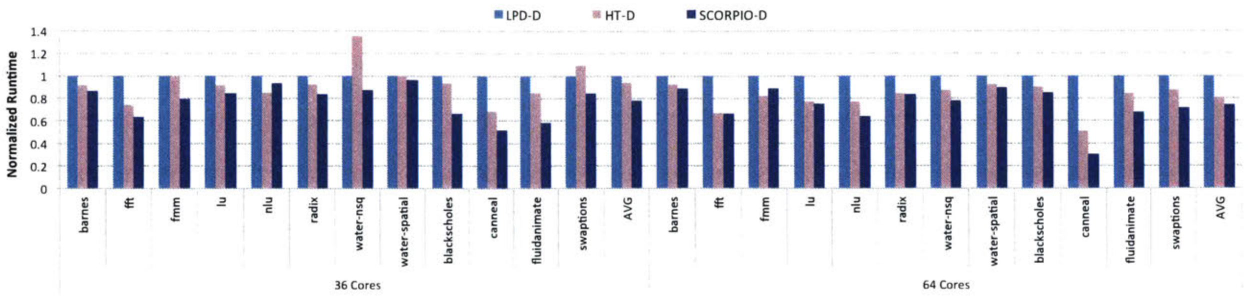
Directory baselines: For directory coherence, all requests are sent as unicasts to a directory, which forwards them to the sharers or reads from main memory if no sharer exists. SCORPIO is compared with two baseline directory protocols. The *Limited-pointer directory* (LPD) [2] baseline tracks when a block is being shared between a small number of processors, using specific pointers. Each directory entry contains 2 state bits, log N bits to record the owner ID, and a set of pointers to track the sharers. We evaluated LPD against full-bit directory in GEMS 36 core full-system simulations and discovered almost identical performance when approximately 3 to 4 sharers were tracked per line as well as the owner ID. Thus, the pointer vector width is chosen to be 24 and 54 bits for 36 and 64 cores, respectively. By tracking fewer sharers, more cachelines are stored within the same directory cache space, resulting in a reduction of directory cache misses. If the number of sharers exceeds the number of pointers in the directory entry, the request is broadcast to all cores. The other baseline is derived from *HyperTransport* (HT) [24]. In

HT, the directory does not record sharer information but rather serves as an ordering point and broadcasts the received requests. As a result, HT does not suffer from high directory storage overhead but still incurs on-chip indirection via the directory. Hence for the analysis only 2 bits (ownership and valid) are necessary. The ownership bit indicates if the main memory has the ownership; that is, none of the L2 caches own the requested line and the data should be read from main memory. The valid bit is used to indicate whether main memory has received the writeback data. This is a property of the network, where the writeback request and data may arrive separately and in any order because they are sent on different virtual networks.

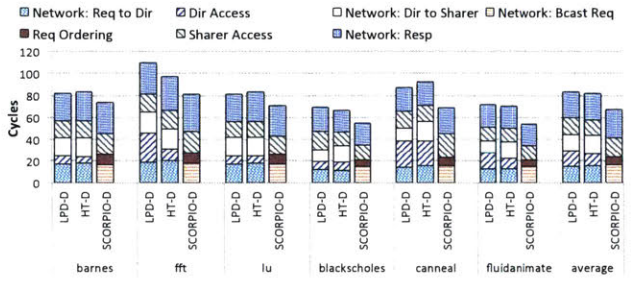
Workloads: We evaluate all configurations with SPLASH-2 [124] and PARSEC [11] benchmarks. Simulating higher than 64 cores in GEMS requires the use of trace-based simulations, which fail to capture dependencies or stalls between instructions, and spinning or busy waiting behavior accurately. Thus, to evaluate SCORPIO's performance scaling to 100 cores, we obtain SPLASH-2 and PARSEC traces from the Graphite [78] simulator and inject them into the SCORPIO RTL testbench.

Evaluation Methodology: For performance comparisons with baseline directory protocols, we use GEMS to see the relative runtime improvement. The centralized directory in HT and LPD adds serialization delay at the single directory. Multiple distributed directories alleviates this but adds on-die network latency between the directories and DDR controllers at the edge of the chip for off-chip memory access, for both baselines. We evaluate the distributed versions of LPD (LPD-D), HT (HT-D), and SCORPIO (SCORPIO-D) to equalize this latency and specifically isolate the effects of indirection and storage overhead. The directory cache is split across all cores, while keeping the total directory size fixed to 256 KB. Our chip prototype uses 128 KB, as seen in Table 7-1, but we changed this value for baseline performance comparisons only such that we do not heavily penalize LPD by choosing a smaller directory cache.

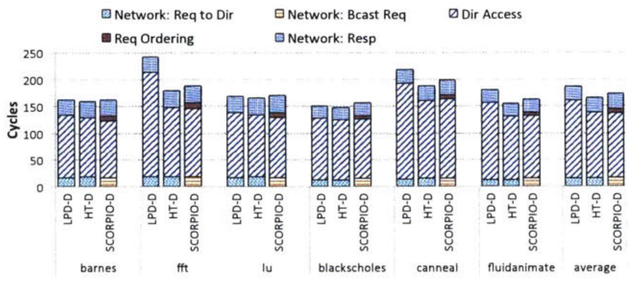
The SCORPIO network design exploration provides insight into the performance impact as certain parameters are varied. The finalized settings from GEMS simulations are used in the fabricated 36-core chip NoC. In addition, we use behavioral RTL simulations on the 36-core SCORPIO RTL, as well as 64 and 100-core variants, to explore the scaling



(a) Normalized runtime for 36 and 64 cores



(b) Served by other caches (36 cores)



(c) Served by directory (36 cores)

Figure 7-12: Normalized Runtime and Latency Breakdown

of the uncore to high core counts. For reasonable simulation time, we replace the Cadence memory controller IP with a functional memory model with fully-pipelined 90-cycle latency. Each core is replaced with a memory trace injector that feeds SPLASH-2 and PARSEC benchmark traces into the L2 cache controller's AHB interface. We run the trace-driven simulations for 400 K cycles (220 K for 10×10 mesh for tractability), omitting the first 20 K cycles for cache warm-up.

7.4.1 Performance

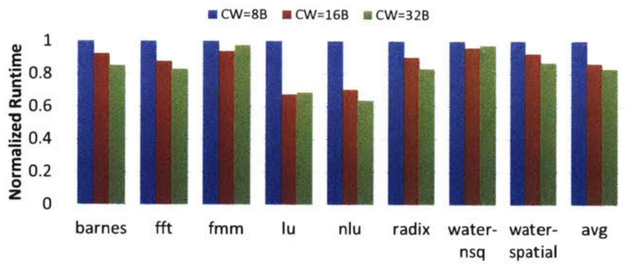
To ensure the effects of indirection and directory storage are captured in the analysis, we keep all other conditions equal. Specifically, all architectures share the same coherence protocol and run on the same NoC (minus the ordered virtual network GO-REQ and notification network).

Figure 7-12 shows the normalized full-system application runtime for SPLASH-2 and PARSEC benchmarks simulated on GEMS. On average, SCORPIO-D shows 24.1 % better performance over LPD-D and 12.9 % over HT-D across all benchmarks. Diving in, we realize that SCORPIO-D experiences average L2 service latency of 78 cycles, which is lower than that of LPD-D (94 cycles) and HT-D (91 cycles). The average L2 service latency is computed over all L2 hit, L2 miss (including off-chip memory access) latencies and it also captures the internal queuing latency between the core and the L2. Since the L2 hit latency and the response latency from other caches or memory controllers are the same across all three configurations, we further breakdown request delivery latency for three SPLASH-2 and three PARSEC benchmarks (see Figure 7-12). When a request is served by other caches, SCORPIO-D's average latency is 67 cycles, which is 19.4 % and 18.3 % lower than LPD-D and HT-D, respectively. Since we equalize the directory cache size for all configurations, the LPD-D caches fewer lines compared to SCORPIO-D and HT-D, leading to a higher directory access latency which includes off-chip latency. SCORPIO provides the most latency benefit for data transfers from other caches on-chip by avoiding the indirection latency.

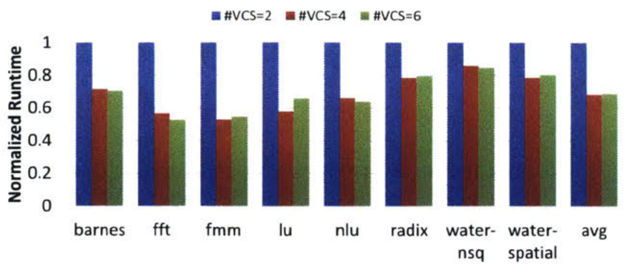
As for requests served by the directory, HT-D performs better than LPD-D due to the lower directory cache miss rate. Also, because the directory protocols need not forward the requests to other caches and can directly serve received requests, the ordering latency overhead makes the SCORPIO delivery latency slightly higher than the HT-D protocol. Since the directory only serves 10 % of the requests, SCORPIO still shows 17 % and 14 % improvement in average request delivery latency over LPD-D and HT-D, respectively, leading to the overall runtime improvement.

7.4.2 NoC Design Exploration for 36-Core Chip

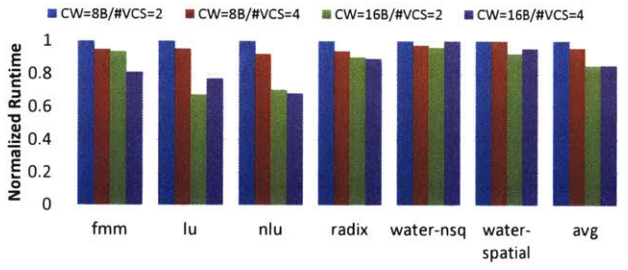
With GEMS, we swept several key SCORPIO network parameters, channel-width, number of VCs, and number of simultaneous notifications, to arrive at the final 36-core fabricated configuration. Channel-width impacts network throughput by directly influencing the number of flits in a multi-flit packet, affecting serialization and essentially packet latency. The number of VCs also affects the throughput of the network and application runtimes, while the number of simultaneous notifications affect ordering delay. Figure 7-13 shows the variation in runtime as the channel-width and number of



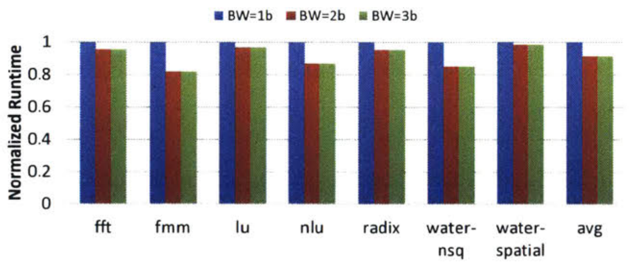
(a) Channel-widths



(b) GO-REQ VCs



(c) UO-RESP VCs



(d) Simultaneous notifications

Figure 7-13: Normalized Runtime with Varying Network Parameters

VCs are varied. All results are normalized against a baseline configuration of 16-byte channel-width and 4 VCs in each virtual network.

Channel-width: While a larger channel-width offers better performance, it also incurs greater overheads - larger buffers, higher link power and larger router area. A channel-width of 16 bytes translates to 3 flits per packet for cache line responses on the UO-RESP virtual network. A channel-width of 8 bytes would require 5 flits per packet for cache line responses, which degrades the runtime for a few applications. While a 32 byte channel offers a marginal improvement in performance, it expands router and NIC area by 46 %. In addition, it leads to low link utilization for the shorter network requests. The 36-core chip contains 16-byte channels due to area constraints and diminishing returns for larger channel-widths.

Number of VCs: Two VCS provide insufficient bandwidth for the GO-REQ virtual network which carries the heavy request broadcast traffic. Besides, one VC is reserved for deadlock avoidance, so low VC configurations would degrade runtime severely. There is a negligible difference in runtime between 4 VCs and 6 VCs. Post-synthesis timing analysis of the router shows negligible impact on the operating frequency as the number of VCs is varied, with the critical path timing hovering around 950 ps. The number of VCs indeed affects the SA-I stage, but it is off the critical path. However, a tradeoff of area, power, and performance still exists. Post-synthesis evaluations show 4 VCs is 15 % more area efficient, and consumes 12 % less power than 6 VCs. Hence, our 36-core chip contains 4 VCs in the GO-REQ virtual network. For the UO-RESP virtual network, the number of VCs does not seem to impact run time greatly once channel-width is fixed. UO-RESP packets are unicast messages, and generally much fewer than the GO-REQ broadcast requests. Hence 2 VCs suffices.

Number of simultaneous notifications: The Power Architecture cores used in our 36-core chip are constrained to two outstanding messages at a time because of the AHB interfaces at its data and instruction cache miss ports. Due to the low injection rates, we choose a 1-bit-per-core (36-bit) notification network which allows 1 notification per core per time window.

We evaluate if a wider notification network that supports more notifications each time window will offer better performance. Supporting 3 notifications per core per time window, will require 2 bits per core, which results in a 72-bit notification network. Figure 7-13d shows 36-core GEMS simulations of SCORPIO achieving 10 % better performance for more than one outstanding message per core with a 2-bit-per-core notification network, indicating that bursts of 3 messages per core occur often enough to result in overall runtime reduction. However, more than 3 notifications per time window (3-bit-per-core notification network) does not reap further benefit, as larger bursts of messages are uncommon. A notification network data width scales as $O(m \times N)$, where m is the number of notifications per core per time window. Our 36-bit notification network has $< 1\%$ area and power overheads; wider data widths only incurs additional wiring which has minimal area and power compared to the main network.

7.4.3 Scaling Uncore Throughput for High Core Counts

As core counts scale, if each core's injection rate (cache miss rate) remains constant, the overall throughput demand on the uncore scales up. We explore the effects of two techniques to optimize SCORPIO's throughput for higher core counts.

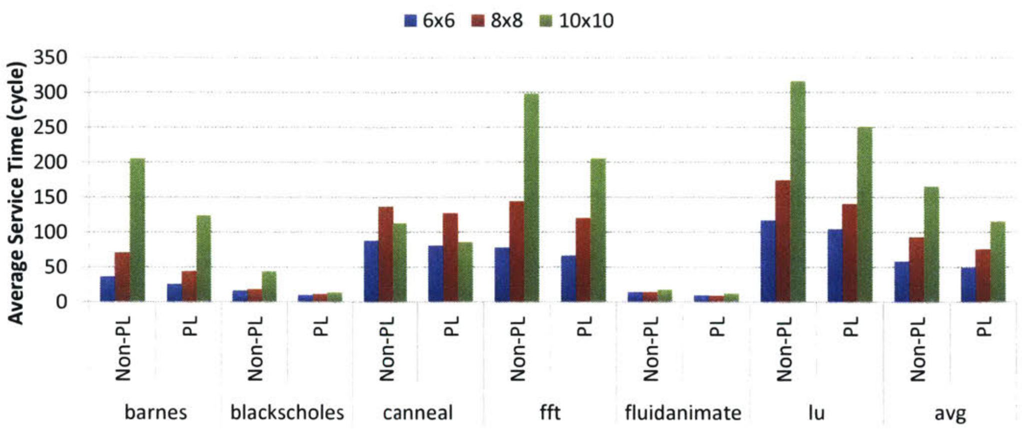


Figure 7-14: Pipelining effect on performance and scalability

Pipelining uncore: Pipelining the L2 caches improves its throughput and reduces the backpressure on the network which may stop the NIC from de-queueing packets. Similarly, pipelining the NIC will relieve network congestion. The performance impact of pipelining the L2 and NIC can be seen in Figure 7-14 in comparison to a non-pipelined version. For 36 and 64 cores, pipelining reduces the average latency by 15 % and 19 %, respectively. Its impact is more pronounced as we increase to 100 cores, with an improvement of 30.4 %. Canneal's 10×10 result is better than 8×8 case because within 220 K cycles, higher latency requests are not captured.

Boosting main network throughput with VCs: For good scalability on any multiprocessor system, the cache hierarchy and network should be co-designed. As core count increases, assuming similar cache miss rates and thus traffic injection rates, the load on the network now increases. The theoretical throughput of a $k \times k$ mesh is $1/k^2$ for broadcasts, reducing from 0.027 flits/node/cycle for 36-cores to 0.01 flits/node/cycle for 100-cores. Even if overall traffic across the entire chip remains constant, say due to less sharing or larger caches, a 100-node mesh will lead to longer latencies than a 36-node mesh. Common ways to boost a mesh throughput include multiple meshes, more VCs/buffers per mesh, or wider channel.

Within the limits of the RTL design, we analyze the scalability of the SCORPIO architecture by varying core count and number of VCs within the network and NIC, while keeping the injection rate constant. The design exploration results show that increasing the UO-RESP virtual channels does not yield much performance benefit. But, the OREQ virtual channels matter since they support the broadcast coherent requests. Thus, we increase only the OREQ VCs from 4 VCs to 16 VCs (64 cores) and 50 VCs (100 cores), with 1 buffer per VC. Increasing VCs will stretch the critical path and affect the operating frequency of the chip. It will also affect area, though with the current NIC+router taking up just 10 % of tile area, this may not be critical. A much lower overhead solution for boosting throughput is to go with multiple main networks, which will double/triple the throughput with no impact on frequency. It is also more efficient area wise as excess wiring is available on-die.

For at least 64 cores in GEMS full-system simulations, SCORPIO performs better than LPD and HT despite the broadcast overhead. The 100-core RTL trace-driven simulation results in Figure 7-14 show that the average network latency increases significantly. Diving in, we realize that the network is very congested due to injection rates close to saturation throughput. Increasing the number of VCs helps push throughput closer to the theoretical, but is ultimately still constrained by the theoretical bandwidth limit of the topology. A possible solution could be to use multiple main networks, which would not affect the correctness because of our decoupling of message delivery from ordering approach. Our trace-driven methodology could have a factor on the results too, as we were only able to run 20 K cycles for warmup to ensure tractable RTL simulation time; we noticed that L2 caches are under-utilized during the entire RTL simulation runtime, implying caches are not warmed up, resulting in higher than average miss rates.

An alternative to boosting throughput is to reduce the bandwidth demand. INCF [4] was proposed to filter redundant snoop requests by embedding small coherence filters within routers in the network.

Table 7-3: Request Categories

Category	Data Location	Sufficient Permission	Trigger Condition
Local	Requester cache	Yes	Load hit and store hit (in `Modify` state)
Local Owner	Requester cache	No	Store hit (in `Owned` state)
Remote	Other cache	No	Load miss and store miss
Memory	Memory	No	Load miss and store miss

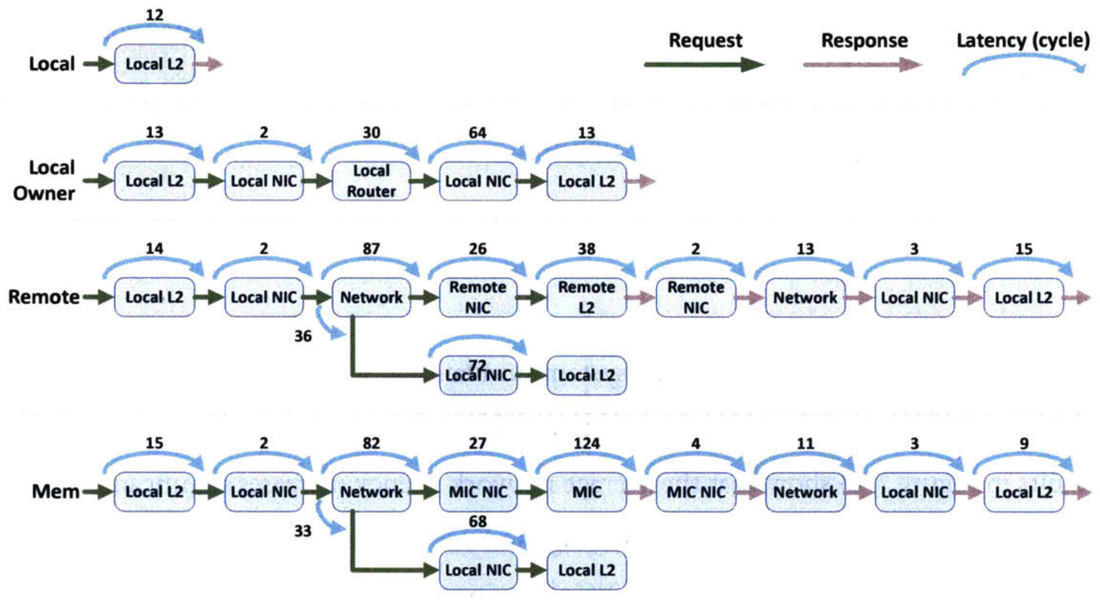


Figure 7-15: L2 Service Time Breakdown (*barnes*)

7.5 Architectural Characterization of SCORPIO Chip

7.5.1 L2 Service Latency

In Section 7.4.1, we show that the L2 service latency plays an important role of the overall system performance. Here, we perform the RTL simulations using the same methodology mentioned in Section 7.4 to quantify the effect of different L2 request types on the average L2 service latency.

We classify L2 requests into 4 categories (see Table 7-3). We first show the latency breakdown of each request category for the *barnes* benchmark traces in Figure 7-15. For *Local* requests, as data resides in the local cache, only local L2 contributes to the round-trip latency with an average latency of 12 cycles, which is the queuing latency and its zero-load latency. For *Local owner* requests, which only occur on a store-hit in `Owned` state, even though the local cache has valid data, it needs to send the request to the network and wait until the request is globally ordered before upgrading to `Modify` state to perform the store operation. The significant delay in the router and NIC is due to this ordering delay. For *Remote* requests, where the valid permission and data is in another cache, the latency involves the time spent at Local L2 and the following:

- The request travel time through the network, and ordering time at the remote cache (Local NIC→Network→Remote NIC).
- The processing time to generate the response (Remote L2).
- The response time through the network (Remote NIC→Network→Local NIC).

Memory requests are similar to *Remote* requests, except that valid permission and data resides in the main memory, so requests are responded by the memory controller instead. In addition to the response, the local L2 needs to see its own requests to complete the transaction which contributes to the forks in the breakdown. For both *Remote* and *Memory* requests, response travel time are faster than that of requests, as requests need to be ordered at the destination and cannot directly be consumed, which introduces backpressure and increases network as well as NIC latency, whereas the responses are unordered and can fully benefit from the low latency network.

Figure 7-16 shows the latency distribution of each request category for *barnes*. The *Memory* requests involve memory access latency and network latency, contributing to the tail of the distribution. Because the L2 access latency is lower than the memory access latency, the overall latency for *Remote* requests is 200 cycles on average. Spatial locality in the memory traces lead to 81 % hits in the requester cache. So even though the latency is relatively high for *Remote* and *Memory* requests, the average service latency is around 51 cycles, still close to the expected zero-load latency of 23 cycles.

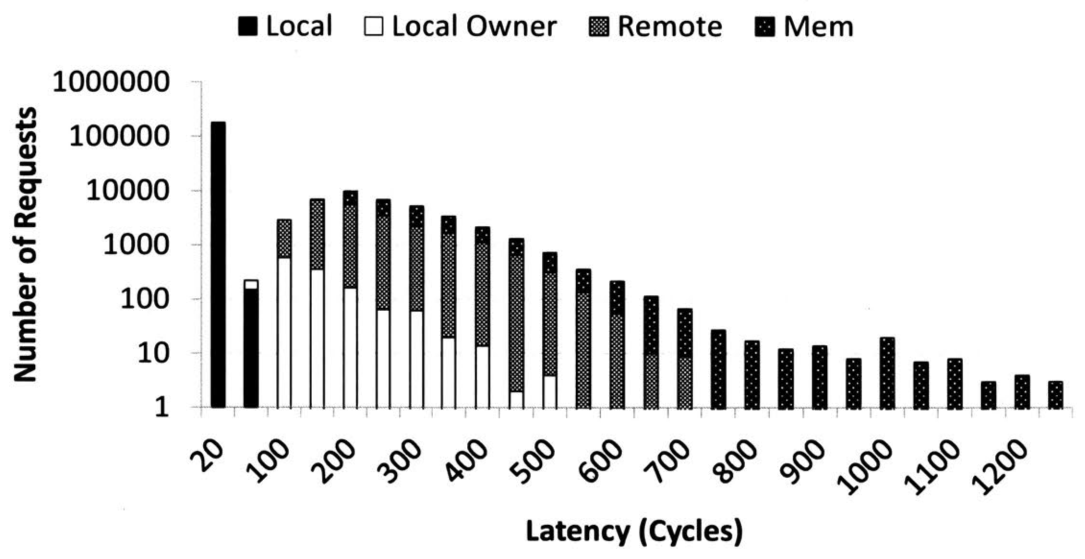


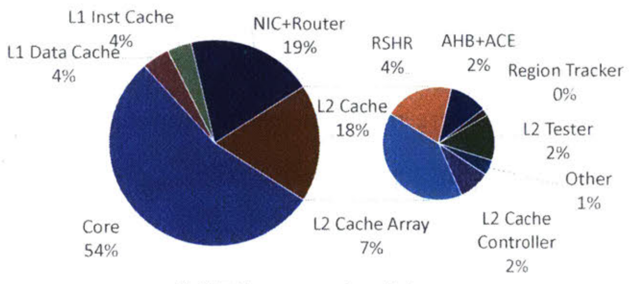
Figure 7-16: L2 Service Time Histogram (*barnes*)

7.5.2 Overheads

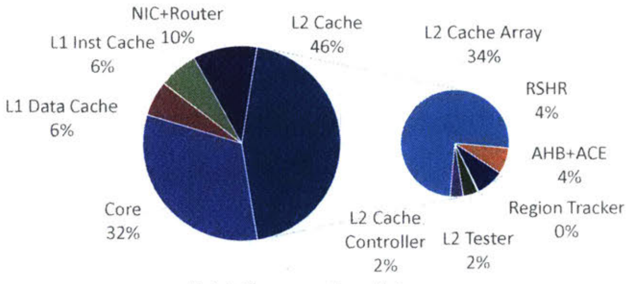
We evaluate the area and power overheads to identify the practicality of the SCORPIO NoC. To obtain the power consumption, we perform gate-level simulation[5] on the post-synthesis netlist and use the generated vector change dump (VCD) files and Synopsys PrimeTime PX. To reduce the simulation time and generated VCD size, we use the trace-driven simulation to obtain the L2 and network power consumption. We attach a mimicked AHB slave that can respond to memory requests in a couple of cycles, to the core and run Dhrystone benchmark to exercise the core for power consumption values. The area overhead breakdown is obtained from layout.

Power: Overall, the aggregated power consumption of SCORPIO is around 28.8 W (around 3.5 W from leakage power) and the detailed power breakdown of a tile is shown in Figure 7-17a. The power consumption of a core with L1 caches is around 62 % of the tile power, whereas the L2 cache consumes 18 % and the NIC and router 19 % of tile power. A notification router costs only a few OR gates; as a result, it consumes less than 1 % of the tile power. Since most of the power is consumed at clocking the pipeline and state-keeping flip-flops for all components, the breakdown is not sensitive to workload.

[5]The simulation is run for 2,000,000 cycles at TT corner, 25 °C and with annotated paracitics.



(a) Tile power breakdown



(b) Tile area breakdown

Figure 7-17: Tile Overheads

Area: The dimension of the fabricated SCORPIO is $11 \times 13\,\text{mm}^2$. Each memory controller and each memory interface controller occupies around $5.7\,\text{mm}^2$ and $0.5\,\text{mm}^2$ respectively. Detailed area breakdown of a tile is shown in Figure 7-17b. Within a tile, L1 and L2 caches are the major area contributors, taking 46 % of the tile area and the network interface controller together with router occupying 10 % of the tile area.

7.6 Chip Measurements and Lessons Learned

Unfortunately, the IO of the chip do not function correctly; the outputs are stuck at either logic 0 or logic 1, and hence the chip functionality cannot be verified. Several checks have been done to identify the source of the issue. We examined the board design, package design, as well as the connections and orientation of the board-package interface and package-chip interface. By using X-ray and IR-imaging, we compare the actual package layout and connections between the package and chip. On the simulation side, even though we couldn't simulate the whole chip due to the high simulation time, we extracted the IO-related portion of the post-layout netlist and simulated in SPICE.

Nevertheless, there are several things that we could have done for improving SCORPIO's performance and for implementing SCORPIO.

Performance: Starting from the L2 cache controller, we opted for simplicity and did not pipeline it. This leads to delays in processing existing requests while backpressure the network, preventing the NIC from consuming packets. At the NIC, we omitted pipelining of the updating of the ESID counter, which throttles its throughput for some scenarios. We could also have increased buffering beyond the current 4 buffers at the NIC, which would not have a significant impact on area/power given the current low overheads. These pipelining and backpressure effects were not captured in our GEMS model, and hence did not crop up until post-fabrication. Finally, the strict sequential consistency ordering that SCORPIO maintains also imposes additional ordering delay. In-network ordering techniques may be incorporated to support relaxed consistency and is not covered in the scope of this dissertation.

Implementation: During implementation, we first built the tile block with the core, L2 controller, NIC and router. Then, we stamped the tile 36 times and connected them together (i.e., two-level hierarchical implementation). However, stamping 36 tiles at once increases the implementation complexity, which dramatically increases the place-and-route time from couple of hours to one whole day. A better way is to implement the chip using hierarchical place-and-route approach with more levels to lower the complexity at each level; for example, first implement a tile, and then a row of 6 tiles, followed by a network of 6 rows.

7.7 Related Work

Multicore processors: Table 7-4 includes a comparison of AMD, Intel, Tilera, SUN multiprocessors with the SCORPIO chip. These relevant efforts were a result of the continuing challenge of scaling performance while simultaneously managing frequency, area, and power. When scaling from multi to many cores, the interconnect is a significant factor. Current industry chips with relatively few cores typically use bus-based, crossbar or ring fabrics to interconnect the last-level cache, but suffers from poor scalability. Bus bandwidth saturates with more than 8 to 16 cores on-chip [25], not to mention the power overhead of signaling across a large die. Crossbars have been adopted as a higher bandwidth alternative in several multicores [20, 87], but it comes at the cost of a large area footprint that scales quadratically with core counts, worsened by layout constraints imposed by long global wires to each core. From the Oracle T5 die photo, the 8-by-9 crossbar has an estimated area of 1.5X core area, hence about 23 mm^2 at 28 nm. Rings are

Table 7-4: Comparison of multicore processors

		Intel Core i7 [31]	**AMD Opteron** [6]	**TILE64** [119]	**Oracle T5** [87]	**Intel Xeon E7** [46]	**SCORPIO**
Clock frequency		2 to 3.3 GHz	2.1 to 3.6 GHz	750 MHz	3.6 GHz	2.1 to 2.7 GHz	1 GHz (833 MHz post-layout)
Power supply		1.0 V	1.0 V	1.0 V		1.0 V	1.1 V
Power consumption		45 to 130 W	115 to 140 W	15 to 22 W		130 W	28.8 W
Lithography		22 nm	32 nm SOI	90 nm	28 nm	32 nm	45 nm SOI
Core count		4 to 8	4 to 16	64	16	6 to 10	36
ISA		x86	x86	MIPS-derived VLIW	SPARC	x86	Power
Cache hierarchy	L1D	32 KB private	16 KB private	8 KB private	16 KB private	32 KB private	16 KB private
	L1I	32 KB private	64 KB shared among 2 cores	8 KB private	16 KB private	32 KB private	16 KB private
	L2	256 KB private	2 MB shared among 2 cores	64 KB private	128 KB private	4 MB shared	128 KB private
	L3	8 MB shared	16 MB shared	N/A	8 MB	18 to 30 MB shared	N/A
Consistency model		Processor	Processor	Relaxed	Relaxed	Processor	Sequential consistency
Coherency		Snoopy	Broadcast-based directory (HT)	Directory	Directory	Snoopy	Snoopy
Interconnect		Point-to-Point (QPI)	Point-to-Point (HyperTransport)	5 8 $\times$ 8 meshes	8 $\times$ 9 crossbar	Ring	6 $\times$ 6 mesh

an alternative that supports ordering, adopted in Intel Xeon E7, with bufferless switches (called stops) at each hop delivering single-cycle latency per hop at high frequencies and low area and power. However, scaling to many cores lead to unnecessary delay when circling many hops around the die.

The Tilera TILE64 [119] is a 64-core chip with 5 packet-switched mesh networks. A successor of the MIT RAW chip which originally did not support shared memory [110], TILE64 added directory-based cache coherence, hinting at market support for shared memory. Compatibility with existing IP is not a concern for startup Tilera, with cache, directory, memory controllers developed from scratch. Details of its directory protocol are not released but news releases suggest directory cache overhead and indirection latency are tackled via trading off sharer tracking fidelity. Intel Single-chip Cloud Computer (SCC) processor [43] is a 48-core research chip with a mesh network that does not support shared memory. Each router has a four stage pipeline running at 2 GHz. In comparison, SCORPIO supports in-network ordering with a single-cycle pipeline leveraging virtual lookahead bypassing, at 1 GHz.

NoC-only chip prototypes: Swizzle [100] is a self-arbitrating high-radix crossbar that embeds arbitration within the crossbar to achieve single cycle arbitration. Prior crossbars require high speedup (crossbar frequency at multiple times core frequency) to boost bandwidth in the face of poor arbiter matching, leading to high power overhead. Area remains a problem though, with the 64-by-32 Swizzle crossbar taking up 6.65 mm^2 in 32 nm process [100]. Swizzle acknowledged scalability issues and proposed stopping at 64-port crossbars, and leveraging these as high-radix routers within NoCs. There are several other stand-alone NoC prototypes that also explored practical implementations with timing, power and area consideration, such as the 1 GHz Broadcast NoC [91] that optimizes for energy, latency and throughput using virtual bypassing and low-swing signaling for unicast, multicast, and broadcast traffic. Virtual bypassing is leveraged in the SCORPIO NoC.

7.8 Summary

The SCORPIO architecture supports global ordering of requests on a mesh network by decoupling the message delivery from the ordering. With this we are able to address key coherence scalability concerns. While our 36-core SCORPIO chip is an academic chip design that can be better optimized in many aspects, we learnt significantly through this exercise about the intricate interactions between processor, cache, interconnect and memory design, as well as the practical implementation overheads of the SCORPIO architecture.

8 Conclusion

With the advance in CMOS technology, more and more general-purpose and/or application-specific cores have been added to the same chip. On-chip networks are adopted to support the communication between these cores. As the number of cores increases, the on-chip network latency and power become critical for system performance. In this dissertation, I tackle both the latency and power issues in large NoC. Particularly, I focus on two key challenges in the realization of low-latency and low-power NoCs:

- The development of NoC design toolchains that can ease and automate the design of large-scale NoCs integrated with advanced ultra-low-power and ultra-low-latency techniques to be embedded within many-core chips.
- The design and implementation of chip prototypes with ultra-low-latency and low-power NoCs for thorough analysis and understanding of the design tradeoffs.

In this chapter, I summarize the main contributions of this dissertation in Section 8.1 and provide future research directions in Section 8.2.

8.1 Dissertation Summary

8.1.1 Development of NoC Design Toolchains

The dissertation begins with DSENT, a NoC timing, power and area evaluation tool, that enables rapid cross-hierarchical evaluation of opto-electronic NoCs. DSENT is based on development of a technology-portable standard cell library so designs can be flexibly modeled while maintaining accuracy. It has been validated against SPICE simulations and shown to be within 20 % accuracy. DSENT provides not only models for electrical digital circuits but also sophisticated models for emerging attractive integrated photonic interconnects. Through DSENT, we demonstrate case studies and show that due to non-data-dependent laser and tuning power, a photonic NoC has poor energy-efficiency at low traffic load, and how it can be improved by using tuning models provided in DSENT. In addition, since photonic technology is still in its infancy, DSENT also serves as a useful tool that can help determine the importance of various parameters. We release DSENT open-source [30] and DSENT is downloaded over 600 times and cited 200 times till now.

We next identify that a datapath consisting of crossbar and link is a major source of NoC energy consumption. Low-swing signaling circuits have been demonstrated to significantly reduce datapath power, but has required custom circuit design in the past. Here, I propose a low-swing NoC crossbar generator toolchain that enables the embedding of low-swing TX/RX cells automatically within NoC RTL [17]. Our case study shows a 50 % energy-per-bit savings for a 5-port mesh router with the generated datapath.

To tackle the latency issue in large networks, clockless repeated links have been shown to be able to obviate the need for latching at routers, thus enabling virtual bypass paths that allow packets to zoom from source to destination cores/NICs without stopping at intermediate routers. This allows a NoC topology to be customized for each SoC application so virtual direct connections can be made between communicating nodes. I propose a NoC synthesis tool flow that takes as input a SoC application with its communication flows, then synthesizes a NoC configured for the application, and generates RTL to layout of the NoC [18]. Our results show that, as compared to an all-to-all topology where every communicating core has a 1-cycle direct link to each other, the synthesized NoC delivers the average network latency that is slightly higher by 1.5 cycles.

8.1.2 Design and Implementation of Chip Prototypes

I led the design and implementation of two chips to rigorously investigate the practical design tradeoffs. The SMART NoC chip was fabricated on 32 nm SOI technology, and measurements show that it works at 817.1 MHz with HPC_{max} of 1 and at 548 MHz with HPC_{max} of 7, consuming 1.57 to 2.53 W, respectively.

The SCORPIO 36-core processor chip was implemented on 45 nm SOI technology, and the RTL analysis showed that the chip can attain 1 GHz (833 MHz post-layout) at 28.8 W with the NoC taking up just 10 % of tile area and 19 % of tile power, demonstrating that low-latency, low-power mesh NoCs can support mainstream snoopy coherence many-core systems.

8.2 Future Research Directions

The dissertation tackles three aspects of building low-latency and low-power NoCs. However, the design of SoC or manycore systems is still a rich topic of research. In this section, we focus on some future research directions that are related to the topics in this dissertation.

Modeling: Even though DSENT lays out the framework for electrical circuits, it only provides models for NoC components, which essentially consist of muxes, buffers, and wires. However, the scope of computer architecture is large and it cannot be expressed only by these components, which calls for a need of more models for basic building blocks and methodologies that can precisely translate high-level architectural design concepts into these building blocks to allow fast evaluation of many more upcoming architecture proposals.

On-chip Photonics: Optical signaling is attractive due to its potential for light-speed latency, high bandwidth and ultra-low power. However, limited materials that can be used on chip constrains the efficiency and performance of optical links, leading to limited on-chip applications. In addition, using WDM implies the use of ring modulators tied to specific frequencies, which is highly sensitive to temperature and process variation. How to effectively resolve or bypass the frequency issue along with reducing the losses of optical components still require future researches to make WDM links more favorable. Solutions such as introducing new elements to the commercial processes to allow devices with better efficiency and using wafer-level integration where optical active components are placed onto a separate plane can be considered while designing future optical interconnects [128]. Furthermore, while design automation is common for digital circuits, in addition to research in basic components, high level design automation for optical link design and optimization is essential for system level integration.

NoC: SMART breaks the on-chip latency barrier imposed by topologies, and shows ultra-low network latency to deliver packets. However, the design relies on the assumption of synchronous clocking. Modern manycore systems often incorporate dynamic voltage and frequency scaling (DVFS) techniques to improve power efficiency, which destroys the notion of *cycle* between different cores. A separate frequency and voltage domain can be dedicated to the network to avoid the problem, but it may not be energy efficient. Furthermore, systems with heterogeneous cores have gained in importance as a way to leverage the wealth of transistors on chip. These cores may be irregular in size, resulting in the need of irregular topologies. How to systematically design a network and router with SMART support for irregular topologies is an avenue for future research.

A SMART Network Architecture Targeting Many-core System Applications

This is joint work with Tushar Krishna [59]. Tushar Krishna and I co-designed the SMART$_{cycle}$ architecture. I performed physical implementation and evaluation, while Tushar Krishna performed system-level performance evaluation.

A.1 Motivation

In this chapter, we present SMART$_{\text{cycle}}$, a generalized version of SMART network that can reconfigure 1-cycle virtual bypass paths on a cycle-to-cycle basis. For simplicity, all the SMART mentioned in this chapter refer to SMART$_{\text{cycle}}$.

The chapter is organized as follows. Section A.2 defines the router microarchitecture that SMART$_{\text{cycle}}$ is built upon and terminology for the rest of the chatper. Section A.3 presents SMART for a k-ary 1-Mesh, and Section A.4 extends it to a k-ary 2-Mesh. Section A.5 summarizes the chapter.

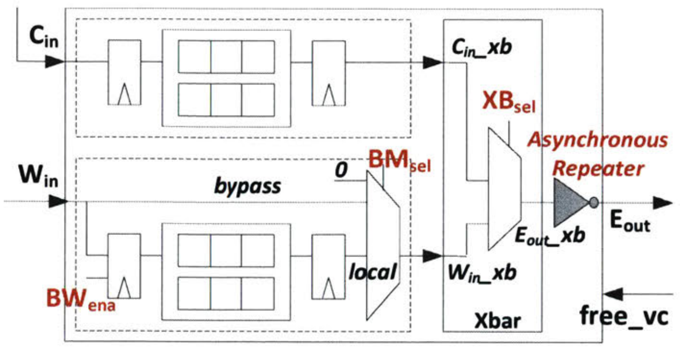


Figure A-1: SMART Router Microarchitecture

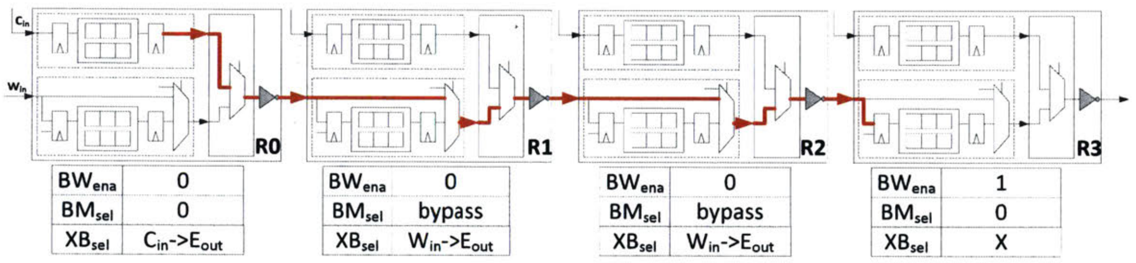


Figure A-2: Example of Single-cycle Multi-hop Traversal

A.2 SMART Router and Terminology

For better understanding of this chapter, we show again a SMART router in Figure A-1, similar to the one described in Chapter 5 except that we construct the SMART router on top of an 1-cycle router instead of 3-cycle. For simplicity, we only show Core$_{in}$ (C$_{in}$)[1], West$_{in}$ (W$_{in}$) and East$_{out}$ (E$_{out}$) ports. All other input ports are identical to W$_{in}$, and all other output ports are identical to E$_{out}$. Each repeater has to be sized to drive not just the link, but also the muxes (2:1 bypass and 4:1 Xbar) at the next router, before a new repeater is encountered.

The three primary components of the design is shown in Figure A-1: (1) Buffer Write enable (BW$_{ena}$) at the input flip flop which determines if the input signal is latched or not, (2) Bypass Mux select (BM$_{sel}$) at the input of the crossbar to choose between the local buffered flit, and the bypassing flit on the link, and (3) Crossbar select (XB$_{sel}$). Figure A-2 shows an example of a multi-hop traversal: a flit from Router R0 traverses 3-hops within

[1]C$_{in}$ does not have a bypass path like the other ports because all flits from the NIC have to get buffered at the first router, before they can create SMART paths, which will be explained later in Section A.3.

Table 1-1: Terminology

Term	Meaning
HPC	Hops Per Cycle. The number of hops traversed in a cycle by any flit.
HPC_{max}	Maximum number of hops that can be traversed in a cycle by a flit. This is fixed at design time.
SMART-hop	*Multi-hop* path traversed in a *Single-cycle* via a SMART link. It could be straight, or have turns. The length of a SMART-hop can vary anywhere from 1-hop to HPC_{max}.
injection router	First router on the route. The source NIC injects a flit into the C_{in} port of this router.
ejection router	Last router on the route. This router ejects a flit out of the C_{out} port to the destination NIC.
start router	Router from which any SMART-hop starts. This could be the injection router, or any router along the route.
inter router	Any intermediate router on a SMART-hop.
stop router	Router at which any SMART-hop ends. This could be the ejection router or any router along the route.
turn router	Router at a turn (W_{in}/E_{in} to N_{out}/S_{out}, or N_{in}/S_{in} to W_{out}/E_{out}) along the route.
local flits	Flits buffered at any start router.
bypass flits	Flits which are bypassing inter routers.
SMART-hop Setup Request (SSR)	Length (in hops) for a requested SMART-hop. For example, SSR=H indicates a request to stop H-hops away. Optimization: Additional *ejection*-bit if requested stop router is ejection router.
premature stop	A flit is forced to stop before its requested SSR length.
Prio=Local	Local flits have higher priority over bypass flits, i.e. Priority α 1/(hops_from_start_router).
Prio=Bypass	Bypass flits have higher priority over local flits, i.e. Priority α (hops_from_start_router).
SMART_1D	Design where routers along the dimension (both X and Y) can be bypassed. Flits need to stop at the turn router.
SMART_2D	Design where routers along the dimension and one turn can be bypassed.

a cycle, till it is latched at R3. The crossbars at R1 and R2 are preset to connect the W_{in} to E_{out}, with their BM_{sel} preset to choose bypass over local. A SMART path can thus be created by appropriately setting BW_{ena}, BM_{sel}, and XB_{sel} at intermediate routers. In the next two sections, we describe the flow control to preset these signals.

Throughout the rest of the chapter, we will use the terminolgy defined in Table 1-1.

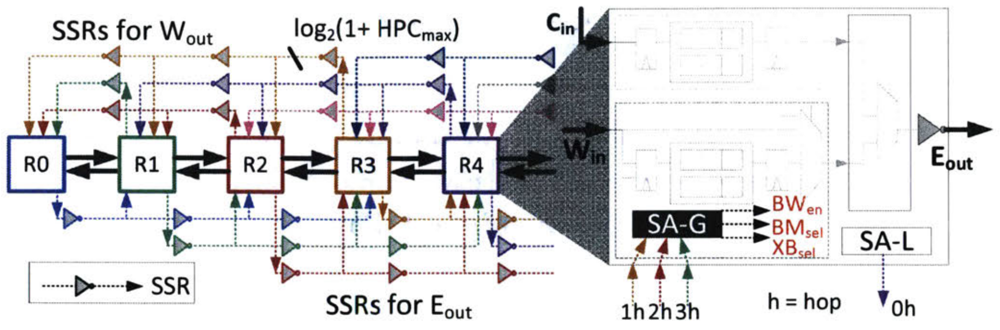


Figure A-3: k-ary 1-Mesh with dedicated SSR links.

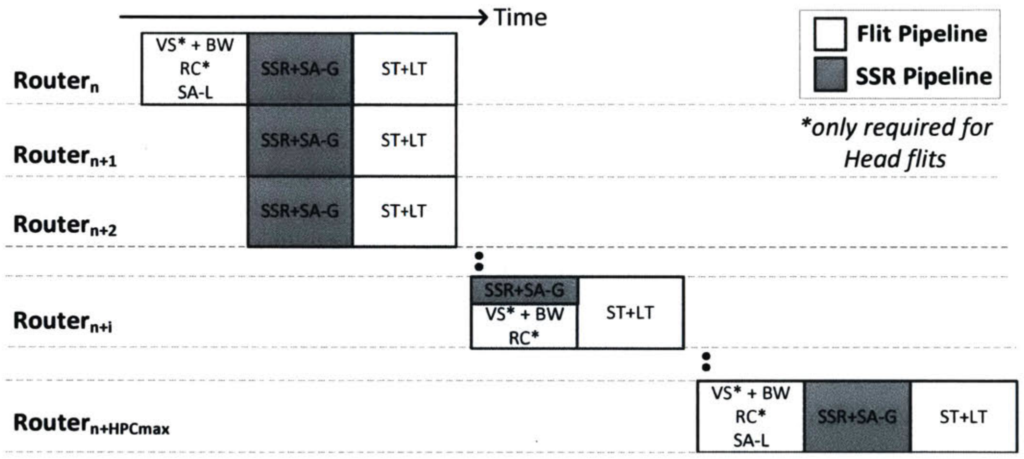


Figure A-4: SMART Pipeline

A.3 SMART in a k-ary 1-Mesh

We start by demonstrating how SMART works in a k-ary 1-Mesh, shown in Figure A-3. Each router has 3 ports: West, East and Core[2]. As shown earlier in Figure A-1, E_{out}_xb can be connected either to C_{in}_xb or W_{in}_xb. W_{in}_xb can be driven either by *bypass*, *local* or *0*, depending on BM_{sel}.

The design is called **SMART_1D** (since routers can be bypassed only along one dimension). The design will be extended to a k-ary 2-Mesh to incorporate turns, in Section A.4. For purposes of illustration, we will assume HPC_{max} to be 3.

A.3.1 SMART-hop Setup Request (SSR)

The SMART router pipeline is shown in Figure A-4. A SMART-hop starts from a start router, where flits are buffered. Unlike the baseline router, Switch Allocation in SMART occurs over two stages: Switch Allocation Local (SA-L) and Switch Allocation Global (SA-G). SA-L is identical to the SA stage in the conventional pipeline (described in Section 2.1.4): every start router chooses a winner for each output port from among its buffered (local) flits. In the next cycle, instead of the winners directly traversing the crossbar (ST), they broadcast a SMART-hop setup request (SSR) via *dedicated* repeated wires (which are inherently multi-drop[3]) up to HPC_{max}. These dedicated SSR wires are shown in Figure A-3. These are $log_2(1 + \text{HPC}_{\text{max}})$-bits wide, and are part of the control path. The SSR carries the length (in hops) up to which the flit winner wishes to go. For instance, SSR = 2 indicates a 2-hop path request. Each flit tries to go as close as possible to its ejection router, hence SSR $= \min(\text{HPC}_{\text{max}}, H_{\text{remaining}})$.

During SA-G, all inter routers arbitrate among the SSRs they receive to set the BW_{ena}, BM_{sel} and XB_{sel} signals. The arbiters guarantee that only *one* flit will be allowed access to any particular input/output port of the crossbar. In the next cycle (ST+LT), SA-L winners that also won SA-G at their start routers traverse the crossbar and links up to multiple hops till they are stopped by BW_{ena} at some router. Thus flits spend at least 2 cycles (SA-L and SA-G) at a start router before they can use the switch. Flits can end up getting prematurely stopped (i.e. before their SSR length) depending on the SA-G results at different routers.

We illustrate all these with examples. In Figure A-5, Router R2 has Flit_A and Flit_B buffered at C_{in}, and Flit_C and Flit_D buffered at W_{in}, all requesting E_{out}. Suppose Flit_D wins SA-L during Cycle-0. In Cycle-1, it sends out $\text{SSR}_\text{D} = 2$ (i.e. request to stop at R4) out of E_{out} to Routers R3, R4 and R5. SA-G is performed at each router as the following.

- **R2**: 0-hop away ($< \text{SSR}_\text{D}$), BM_{sel} = local, $\text{XB}_{\text{sel}} = \text{W}_{\text{in}}\text{\_xb} \rightarrow \text{E}_{\text{out}}\text{\_xb}$.

[2]For illustration purposes, we only show C_{in}, W_{in} and E_{out} in the figures.

[3]Wire cap is an order of magnitude higher than gate cap, adding no overhead if all nodes connected to the wire receive.

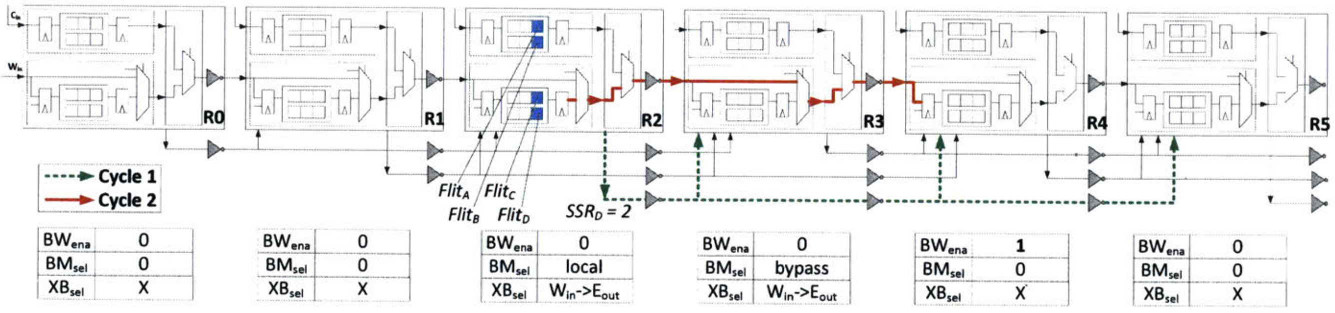

R0
R1
R2
R3
R4
R5
Cycle 1
Cycle 2
Flit$_A$
Flit$_B$
Flit$_C$
Flit$_D$
SSR$_D$ = 2
BW$_{ena}$ 0, BM$_{sel}$ 0, XB$_{sel}$ X
BW$_{ena}$ 0, BM$_{sel}$ 0, XB$_{sel}$ X
BW$_{ena}$ 0, BM$_{sel}$ local, XB$_{sel}$ W$_{in}$->E$_{out}$
BW$_{ena}$ 0, BM$_{sel}$ bypass, XB$_{sel}$ W$_{in}$->E$_{out}$
BW$_{ena}$ 1, BM$_{sel}$ 0, XB$_{sel}$ X
BW$_{ena}$ 0, BM$_{sel}$ 0, XB$_{sel}$ X

Figure A-5: SMART Example: No SSR Conflict

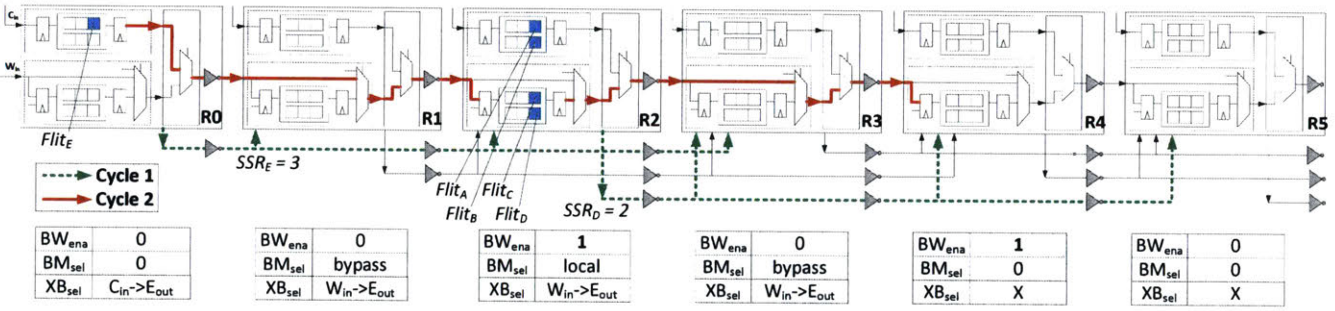

R0
R1
R2
R3
R4
R5
Flit$_E$
SSR$_E$ = 3
Cycle 1
Cycle 2
Flit$_A$
Flit$_B$
Flit$_C$
Flit$_D$
SSR$_D$ = 2
BW$_{ena}$ 0, BM$_{sel}$ 0, XB$_{sel}$ C$_{in}$->E$_{out}$
BW$_{ena}$ 0, BM$_{sel}$ bypass, XB$_{sel}$ W$_{in}$->E$_{out}$
BW$_{ena}$ 1, BM$_{sel}$ local, XB$_{sel}$ W$_{in}$->E$_{out}$
BW$_{ena}$ 0, BM$_{sel}$ bypass, XB$_{sel}$ W$_{in}$->E$_{out}$
BW$_{ena}$ 1, BM$_{sel}$ 0, XB$_{sel}$ X
BW$_{ena}$ 0, BM$_{sel}$ 0, XB$_{sel}$ X

Figure A-6: SMART Example: SSR Conflict with Prio=Local

- **R3**: 1-hop away ($<$ SSR$_D$), BM$_{sel}$ = bypass, XB$_{sel}$ = W$_{in}$_xb→E$_{out}$_xb.
- **R4**: 2-hops away (= SSR$_D$), BW$_{ena}$ = high.
- **R5**: 3-hops away ($>$ SSR$_D$), SSR$_D$ is ignored.

In Cycle-2, Flit$_D$ traverses the crossbars and links at R2 and R3, and is stopped and buffered at R4.

What happens if there are competing SSRs? In the same example, suppose R0 also wants to send Flit$_E$ 3-hops away to R3, as shown in Figure A-6. In Cycle-1, R2 sends out SSR$_D$ as before, and in addition R0 sends SSR$_E$ = 3 out of E$_{out}$ to R1, R2 and R3. Now at R2 there is a conflict between SSR$_D$ and SSR$_E$ for the W$_{in}$_xb and E$_{out}$_xb ports of the crossbar. *SA-G priority* decides which SSR wins the crossbar. More details about priority will be discussed later in Section A.3.2. For now, let us assume Prio=Local (which is defined in Table 1-1) so Flit$_E$ loses to Flit$_D$. The values of BW$_{ena}$, BM$_{sel}$ and XB$_{sel}$ at each router for this priority are shown in Figure A-6. In Cycle-2, Flit$_E$ traverses the crossbar and link at R0 and R1, but is stopped and buffered at R2. Flit$_D$ traverses the crossbars

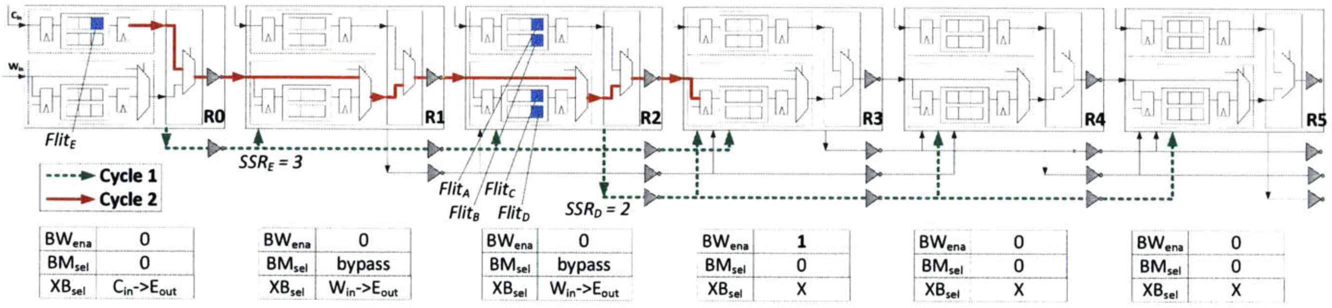


Figure A-7: SMART Example: SSR Conflict with Prio=Bypass

and links at R2 and R3 and is stopped and buffered at R4. Flit$_E$ now goes through BW and SA-L at R2 before it can send a new SSR and continue its network traversal. A free VC/buffer is guaranteed to exist whenever a flit is made to stop (see Section A.3.4).

A.3.2 Switch Allocation Global: Priority

Figure A-7 shows the previous example with Prio=Bypass instead of Prio=Local. This time, in Cycle-2, Flit$_E$ traverses all the way from R0 to R3, while Flit$_D$ is stalled.

Do all routers need to enforce the same priority? Yes. This guarantees that all routers will arrive at the same consensus about which SSRs win and lose. This is required for correctness. In the example discussed earlier in Figure A-6 and A-7, BW$_{ena}$ at R3 was low with Prio=Local, and high with Prio=Bypass. Suppose R2 performs Prio=Bypass, but R3 performs Prio=Local, Flit$_E$ will end up going from R0 to R4, instead of stopping at R3. This is not just a misrouting issue, but also a signal integrity issue because HPC$_{max}$ is 3, but the flit was forced to go up to 4 hops in a cycle, and will not be able to reach the clock edge in time. Note that enforcing the same priority is only necessary for SA-G, which corresponds to the global arbitration among SA-L winners at every router. During SA-L, however, different routers/ports can still choose to use different arbiters (round robin, queueing, priority) depending on the desired QoS/ordering mechanism.

Can a flit arrive at a router, even though the router is not expecting it (i.e. false positive[4])? No. All flits that arrive at a router are expected, and will stop/bypass based

[4]The result of SA-G (BW$_{ena}$, BM$_{sel}$ and XB$_{sel}$) at a router is a prediction for the null hypothesis: a flit will arrive the next cycle, and stop/bypass.

on the success of their SSR in the previous cycle. This is guaranteed since all routers enforce the same SA-G priority.

Can a flit not arrive at a router, even though the router is expecting it (i.e. false negative)? Yes. It is possible for the router to be setup for stop/bypass for some flit, but no flit arrives. This can happen if that flit is forced to prematurely stop earlier due to some SSR interaction at prior inter routers that the current router is not aware of. For example, suppose a local flit at W_{in} at R1 wants to eject out of C_{out}. A flit from R0 will prematurely stop at R1's W_{in} port if Prio=Local is implemented. However, R2 will still be expecting the flit from R0 to arrive[5]. Unlike false positives, this is not a correctness issue but just a performance (throughput) issue, since some links go idle which could have potentially been used by other flits if more global information were available.

A.3.3 Ordering

In SMART, any flit can be prematurely stopped based on the interaction of SSRs that cycle. We need to ensure that this does not result in re-ordering between (a) flits of the same packet, or (b) flits from the same source (if point-to-point ordering is required in the coherence protocol).

The first constraint is in routing (relevant to 2D topologies). Multi-flit packets, and point-to-point ordered virtual networks should only use deterministic routes, to ensure that prematurely buffered flits do not end up choosing alternate routes, while bypassing flits continue on the old route.

The second constraint is in SA-G priority. Every input port has a bit to track if there is a prematurely stopped flit among its buffered flits. When an SSR is received at an input port, and there is either (a) a prematurely buffered Head/Body flit, or (b) a prematurely buffered flit within a point-to-point ordered virtual network, the incoming flit is stopped.

[5]The *valid*-bit from the flit is thus used in addition to BW_{cna} when deciding whether to buffer.

A.3.4 Guaranteeing Free VC/buffers at Stop Routers

In a conventional network, a router's output port tracks the IDs of all free VCs at the neighbor's input port. A buffered Head flit chooses a free VCid for its *next* router (neighbor), before it leaves the router. The neighbor signals back when that VCid becomes free. In a SMART network, the challenge is that the *next* router could be any router that can be reached within a cycle. A flit at a start router choosing the VCid before it leaves will not work because (a) it is not guaranteed to reach its presumed next router, and (b) multiple flits at different start routers might end up choosing the same VCid. Instead, we let the VC selection occur at the stop router. Every SMART router receives 1-bit from each neighbor to signal if *at least one* VC is free[6]. During SA-G, if an SSR requests an output port where there is no free VC, BW_{ena} is made high and the corresponding flit is buffered. This solution does not add any extra multi-hop wires for VC signaling. The signaling is still between neighbors. Moreover, it ensures that a Head flit comes into a router's input port only if that input port has free VCs, else the flit is stopped at the previous router.

However, this solution is conservative because a flit will be stopped prematurely if the neighbor's input port does not have free VCs, even if there was no competing SSR at the neighbor and the flit would have bypassed it without having to stop.

How do Body/Tail flits identify which VC to go to at the stop router? Using their injection router id. Every input port maintains a table to map a VCid to an injection router id[7]. Whenever the Head flit is allocated a VC, this table is updated. The injection router id entry is cleared when the Tail arrives. The VC is freed when the Tail leaves. We implement private buffers per VC, with depth equal to the maximum number of flits in the packet (i.e. virtual cut-through), to ensure that the Body/Tail will always have a free buffer in its VC[8].

[6]If the router has multiple virtual networks (vnets) for the coherence protocol, we need a 1-bit free VC signal from the neighbors for each vnet. The SSR also needs to carry the vnet number, so that the inter routers can know which vnet's free VC signal to look at.

[7]The table size equals the number of multi-flit VCs at that input port.

[8]Extending this design to fewer buffers than the number of flits in a packet would involve more signaling, and is left for future work.

What if two Body/Tail flits with same injection router id arrive at a router? We guarantee that this will never occur by forcing *all* flits of a packet to leave from an output port of a router, before flits from another packet can leave from that output port (i.e. virtual cut-through). This guarantees a unique mapping from injection router id to VCid in the table at every router's input port.

What if a Head bypasses, but Body/Tail is prematurely stopped? The Body/Tail still needs to identify a VCid to get buffered in. To ensure that it does have a VC, we make the Head flit reserve a VC not just at its stop router, but also at all its inter routers, even though it does not stop there. This is done from the *valid*, *type* and *injection router* fields of the bypassing flit. The Tail flit frees the VCs at all the inter routers. Thus, for multi-flit packets, VCs are reserved at all routers, just like the baseline. But the advantage of SMART is that VCs are reserved and freed at *multiple routers* within the *same cycle*, thus reducing the buffer turnaround time.

A.3.5 Additional Optimizations

We add additional optimizations to SMART to push it towards an ideal 1-cycle network (or Dedicated network described in Section 5.5).

Bypassing the ejection router: So far we have assumed that a flit starting at an injection router traverses one (or more) SMART-hops till the ejection router, where it gets buffered and requests for the C_{out} port. We add an extra ejection-bit in the SSR to indicate if the requested stop router corresponds to the ejection router for the packet, and not any intermediate router on the route. If a router receives an SSR from H-hops away with value H (i.e. request to stop there), $H < HPC_{max}$, and the ejection-bit is high, it arbitrates for C_{out} port during SA-G. If it loses, BW_{ena} is made high.

Bypassing SA-L at low load: We add low-load bypassing [27] to the SMART router. If a flit comes into a router with an empty input port and no SA-L winner for its output port for that cycle, it sends SSRs directly, in parallel to getting buffered, without having to go through SA-L. This reduces T_r at lightly-loaded start routers to 2, instead of 3, as shown in Figure A-4 for Router$_{n+i}$. Multi-hop traversals within a single-cycle meanwhile happen at all loads.

A.3.6 Summary

In summary, a SMART NoC works as follows:

- Buffered flits at injection/start routers arbitrate locally to choose input/output port winners during SA-L.
- SA-L winners broadcast SSRs along their chosen routes, and each router arbitrates among these SSRs during SA-G.
- SA-G winners traverse multiple crossbars and links asynchronously within a cycle, till they are explicitly stopped and buffered at some router along their route.

In a **SMART_1D** design with both ejection and no-load bypass enabled, if HPC_{max} is larger than the maximum hops in any route, a flit will only spend 2 cycles in the entire network in the best case (1-cycle for SSR and 1-cycle for ST+LT all the way to the destination NIC).

A.4 SMART in a *k*-ary 2-Mesh

We demonstrate how SMART works in a k-ary 2-Mesh. Each router has 5 ports: West, East, North, South and Core.

A.4.1 Bypassing routers along dimension

We start with a design where we do not allow bypass at turns, i.e. all flits have to stop at their turn routers. We re-use *SMART_1D* described for a k-ary 1-Mesh in a k-ary 2-Mesh. The extra router ports only increase the complexity of the SA-L stage, since there are multiple local contenders for each output port. Once each router chooses SA-L winners, SA-G remains identical to the description in Section A.3.1. The E_{out}, W_{out}, N_{out} and S_{out} ports have dedicated SSR wires going out till HPC_{max} along that dimension. Each input port of the router can receive only one SSR from a router that is H-hops away. The SSR requests a stop, or a bypass along that dimension. Flits with turning routes perform their traversal one-dimension at a time, trying to bypass as many routers as possible, and stopping at the turn routers.

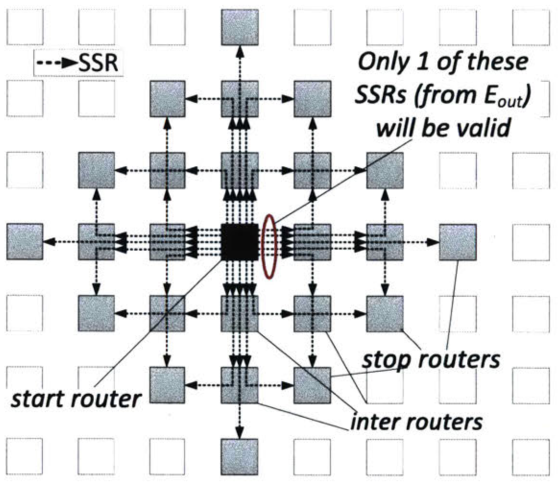


Figure A-8: k-ary 2-Mesh with SSR Wires From Shaded Start Router

A.4.2 Bypassing routers at turns

In a k-ary 2-Mesh topology, all routers within a HPC_{max} neighborhood can be reached within a cycle, as shown in Figure A-8 by the shaded diamond. We now describe **SMART_2D** which allows flits to bypass both the routers along a dimension and the turn router(s). We add dedicated SSR links for each possible XY/YX path from every router to its HPC_{max} neighbors. Figure A-8 shows that the E_{out} port has 5 SSR links, in comparison to only one in the SMART_1D design. During the routing stage, the flit chooses one of these possible paths. During the SA-G stage, the router broadcasts *one* SSR out of each output port, on one of these possible paths. We allow only one turn within each HPC_{max} quadrant to simplify the SSR signaling.

SA-G Priority:

In the SMART_2D design, there can be more than one SSR from H-hops away, as shown in the example in Figure A-9 for router R_j. R_j needs a specific policy to choose between these requests, to avoid sending false positives on the way forward to R_k. Section A.3.2 discussed that false positives can result in misrouted flits or flits trying to bypass beyond HPC_{max}, thus breaking the system. To arbitrate between SSRs from routers that are the same number of hops away, we choose Straight > Left Turn > Right Turn. For the inter router R_j in Figure A-9, the SSR from R_m will have higher priority (1_0) over the one from R_n (1_1) for the N_{out} port, as it is going straight, based

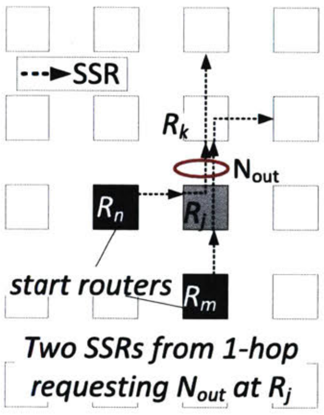
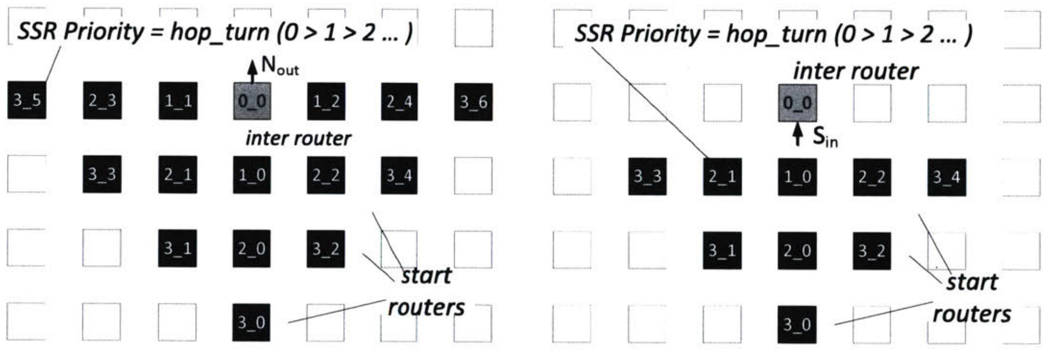


Figure A-9: Conflict Between Two SSRs for N_{out} Port



(a) Fixed Priority at N_{out} port of inter router. (b) Fixed Priority at S_{in} port of inter router.

Figure A-10: SMART_2D SA-G priorities

on Figure A-10a. Similarly at R_k, the SSR from R_m will have higher priority (2_0) over the one from R_n (2_1) for the S_{in} port, based on Figure A-10b. Thus both routers R_j and R_k will unambiguously prioritize the flit from R_m to use the links, while the flit from R_n will stop at Router R_j. Any priority scheme will work as long as every router enforces the same priority.

A.5 Summary

In this chapter, we present $\text{SMART}_{\text{cycle}}$, a flavor of SMART network that is able to reconfigure virtual bypass paths every cycle to lower the network latency for applications with unpredictable traffic or near all-to-all traffic flows.

Bibliography

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2003 (cit. on p. 61).

[2] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz. "An evaluation of directory schemes for cache coherence". In: *Int'l Symp. on Computer Architecture (ISCA)*. 1988 (cit. on p. 124).

[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: *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*. 2009 (cit. on pp. 20, 34, 124).

[4] N. Agarwal, L.-S. Peh, and N. K. Jha. "In-Network Coherence Filtering: Snoopy Coherence without Broadcasts". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2009 (cit. on p. 131).

[5] N. Agarwal, L.-S. Peh, and N. K. Jha. "In-Network Snoop Ordering (INSO): Snoopy Coherence on Unordered Interconnects". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2009 (cit. on p. 17).

[6] *AMD Opteron 6200 Series Processors*. URL: `https://www.amd.com/Documents/Opteron_6000_QRG.pdf` (cit. on p. 137).

[7] *ARM AMBA*. URL: `https://www.arm.com/products/system-ip/amba-specifications.php` (cit. on pp. 44, 107).

[8] J. Balfour and W. J. Dally. "Design Tradeoffs for Tiled CMP On-Chip Networks". In: *Int'l Conf. on Supercomputing (ICS)*. 2006 (cit. on p. 21).

[9] N. Banerjee, P. Vellanki, and K. S. Chatha. "A Power and Performance Model for Network-on-Chip Architectures". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2004 (cit. on p. 21).

[10] S. Beamer, C. Sun, Y.-J. Kwon, A. Joshi, C. Batten, V. Stojanović, and K. Asanović. "Re-architecting DRAM memory systems with monolithically integrated silicon photonics". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2010 (cit. on pp. 14, 19).

[11] C. Bienia, S. Kumar, J. P. Singh, and K. Li. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *Int'l Conf. on Parallel Architecture Compilation Techniques (PACT)*. 2008 (cit. on p. 125).

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. "The gem5 simulator". In: *Computer Architecture News* 39 (2 2011), pp. 1–7 (cit. on pp. 34, 41).

[13] N. Binkert, A. Davis, N. P. Jouppi, M. McLaren, N. Muralimanohar, R. Schreiber, and J. H. Ahn. "The role of optics in future high radix switch design". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2011 (cit. on p. 35).

[14] W. Bogaerts, D. V. Thourhout, and R. Baets. "Fabrication of uniform photonic devices using 193nm optical lithography in silicon-on-insulator". In: *European Conf. on Integrated Optics (ECIO)*. 2008 (cit. on p. 31).

[15] *CACTI6.5*. URL: `http://www.hpl.hp.com/research/cacti` (cit. on p. 27).

[16] J. Chan, G. Hendry, A. Biberman, K. Bergman, and L. P. Carloni. "PhoenixSim: a simulator for physical-layer analysis of chip-scale photonic interconnection networks". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2010 (cit. on p. 21).

[17] C.-H. O. Chen, S. Park, T. Krishna, and L.-S. Peh. "A Low-Swing Crossbar and Link Generator for Low-Power Networks-on-Chip". In: *Int'l Conf. on Computer Aided Design (ICCAD)*. 2011 (cit. on pp. iii, 4, 45, 142).

[18] C.-H. O. Chen, S. Park, T. Krishna, S. Subramanian, A. Chandrakasan, and L.-S. Peh. "SMART: A Single-Cycle Reconfigurable NoC for SoC Applications". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2013 (cit. on pp. iii, 4, 142).

[19] C.-H. O. Chen, S. Park, S. Subramanian, T. Krishna, W.-C. K. Bhavya K. Daya, B. Wilkerson, J. Arends, A. P. Chandrakasan, and L.-S. Peh. "SCORPIO: 36-core Shared Memory Processor Demonstrating Snoopy Coherence on a Mesh Interconnect". In: *Symp. on High Performance Chips*. 2014 (cit. on pp. iv, 5).

[20] D. Chen, N. A. Eisley, P. Heidelberger, R. M. Sneger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker. "The IBM Blue Gene/Q Interconnection Fabric". In: *IEEE Micro* 32.1 (2012), pp. 32–43 (cit. on p. 136).

[21] L. Chen, L. Zhao, R. Wang, and T. M. Pinkston. "MP3: Minimizing performance penalty for power-gating of Clos network-on-chip". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2014 (cit. on p. 41).

[22] A. A. Chien. "A Cost and Speed Model for k-any n-cube Wormhole Routers". In: *Symp. on High Performance Interconnects.* 1993 (cit. on p. 20).

[23] M. J. Cianchetti, J. C. Kerekes, and D. H. Albonesi. "Phastlane: a rapid transit optical routing network". In: *Int'l Symp. on Computer Architecture (ISCA).* 2009 (cit. on p. 15).

[24] P. Conway and B. Hughes. "The AMD Opteron Northbridge Architecture". In: *IEEE Micro* 27 (2007), pp. 10–21 (cit. on p. 124).

[25] D. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach.* Morgan Kaufmann, 1999 (cit. on p. 136).

[26] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. "xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs". In: *Int'l Conf. on Computer Design (ICCD).* 2003 (cit. on p. 44).

[27] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers, 2004 (cit. on pp. 9, 15, 59, 62, 68, 73, 154).

[28] B. K. Daya, C.-H. O. Chen, S. Subramanian, W.-C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L.-S. Peh. "SCORPIO: A 36-Core Research Chip Demonstrating Snoopy Coherence on a Scalable Mesh NoC with In-Network Ordering". In: *Int'l Symp. on Computer Architecture (ISCA).* 2014 (cit. on pp. iv, 5, 105).

[29] P. Dong, S. Liao, D. Feng, H. Liang, D. Zheng, R. Shafiiha, X. Zheng, G. Li, K. Raj, A. V. Krishnamoorthy, and M. Asghari. "High Speed Silicon Microring Modulator Based on Carrier Depletion". In: *National Fiber Optic Engineers Conference (NFOEC).* 2010 (cit. on p. 33).

[30] *DSENT Download Link.* URL: `http://www.rle.mit.edu/isg/technology.htm` (cit. on pp. 41, 142).

[31] *First the tick, now the tock: Next generation Intel microarchitecture (Nehalem).* URL: `http://www.intel.com/content/dam/doc/white-paper/intel-microarchitecture-white-paper.pdf` (cit. on p. 137).

[32] M. Georgas, J. Orcutt, R. J. Ram, and V. Stojanović. "A Monolithically-Integrated Optical Receiver in Standard 45-nm SOI". In: *European Solid-State Circuits Conference (ISSCIRC).* 2011 (cit. on pp. 15, 33).

[33] M. Georgas, J. Leu, B. Moss, C. Sun, and V. Stojanović. "Addressing Link-Level Design Tradeoffs for Integrated Photonic Interconnects". In: *Custom Integrated Circuits Conference (CICC).* 2011 (cit. on pp. 14, 29–31, 35).

[34] R. Golshan and B. Haroun. "A novel reduced swing CMOS BUS interface circuit for high speed low power VLSI systems". In: *Int'l Symp. on Circuits and Systems (ISCAS).* 1994 (cit. on p. 13).

[35] K. Goossens, J. Dielissen, and A. Radulescu. "Æthereal Network on Chip: Concepts, Architectures, and Implementations". In: *Design & Test of Computers* 22.5 (2005), pp. 414–421 (cit. on pp. 61, 62).

[36] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler, and D. Burger. "On-chip interconnection networks of the TRIPS chip". In: *IEEE Micro* 27.5 (2007), pp. 41–50 (cit. on pp. 10, 106).

[37] R. Gupta, B. Tutuianu, and L. T. Pileggi. "The Elmore delay as a bound for RC trees with generalized input signals". In: *Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 16.1 (1997), pp. 95–104 (cit. on p. 26).

[38] H. Hatamkhani, K.-L. J. Wong, R. Drost, and C.-K. K. Yang. "A 10-mW 3.6-Gbps I/O transmitter". In: *Symp. on VLSI Circuits.* 2003 (cit. on p. 30).

[39] G. Hendry, E. Robinson, V. Gleyzer, J. Chan, L. Carloni, N. Bliss, and K. Bergman. "Circuit-Switched Memory Access in Photonic Interconnection Networks for High-Performance Embedded Computing". In: *Int'l Conf. on Supercomputing (ICS).* 2010 (cit. on p. 15).

[40] M. Hiraki, H. Kojima, H. Misawa, T. Akazawa, and Y. Hatano. "Data-Dependent Logic Swing Internal Bus Architecture for Ultralow-Power LSIs". In: *Journal of Solid-State Circuits (JSSC)* (1995), pp. 397–402 (cit. on p. 13).

[41] R. Ho, T. Ono, F. Liu, R. Hopkins, A. Chow, J. Schauer, and R. Drost. "High-Speed and Low-Energy Capacitive-Driven On-Chip Wires". In: *Int'l Solid-State Circuits Conference (ISSCC)* (2007) (cit. on pp. 13, 61).

[42] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. "A 5-GHz mesh interconnect for a teraflops processor". In: *IEEE Micro* 27.5 (2007), pp. 51–61 (cit. on pp. 10, 16, 43).

[43] J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. V. D. Wijngaart, and T. Mattson. "A 48-Core IA-32 message-passing processor with DVFS in 45 nm CMOS". In: *Int'l Solid-State Circuits Conference (ISSCC).* 2010 (cit. on pp. 10, 16, 106, 138).

[44] *IBM CoreConnect.* URL: http://www.xilinx.com/products/intellectual-property/dr_pcentral_coreconnect.html (cit. on p. 44).

[45] *Intel Hybrid Silicon Laser.* URL: http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-labs-hybrid-silicon-laser-uses-paper.pdf (cit. on p. 14).

[46] *Intel Xeon Processor E7 Family.* URL: http://www.intel.com/content/www/us/en/processors/xeon/xeon-processor-e7-family.html (cit. on p. 137).

[47] *International Technology Roadmap for Semiconductors (ITRS).* URL: http://www.itrs2.net (cit. on p. 25).

[48] C. Jackson and S. J. Hollis. "Skip-links: A Dynamically Reconfiguring Topology for Energy-efficient NoCs". In: *Int'l Symp. on System on Chip (SoC).* 2010 (cit. on pp. 16, 62).

[49] D. R. Johnson, M. R. Johnson, J. H. Kelm, W. Tuohy, S. S. Lumetta, and S. J. Patel. "Rigel: A 1,024-Core Single-Chip Accelerator Architecture". In: *IEEE Micro* 31.4 (2011), pp. 30–41 (cit. on p. 105).

[50] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanović. "Silicon-Photonic Clos Networks for Global On-Chip Communication". In: *Int'l Symp. on Networks-on-Chip (NOCS)*. 2009 (cit. on pp. 15, 19, 31, 34).

[51] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2009 (cit. on pp. 21, 32).

[52] F. Karim, A. Nguyen, and S. Dey. "An Interconnect Architecture for Networking Systems on Chips". In: *IEEE Micro* 22.5 (2002), pp. 36–45 (cit. on p. 61).

[53] A. Khakifirooz and D. A. Antoniadis. "MOSFET Performance Scaling - Part II: Future Directions". In: *Trans. on Electron Devices* 55.6 (2008), pp. 1401–1408 (cit. on p. 25).

[54] A. Khakifirooz, O. M. Nayfeh, and D. Antoniadis. "A Simple Semiempirical Short-Channel MOSFET Current-Voltage Model Continuous Across All Regions of Operation and Employing Only Physical Parameters". In: *Trans. on Electron Devices* 56.8 (2009), pp. 1674–1680 (cit. on p. 25).

[55] B. Kim and V. Stojanović. "A 4Gb/s/ch 356fJ/b 10mm Equalized On-chip Interconnect with Nonlinear Charge-Injecting Transmit Filter and Transimpedance Receiver in 90nm CMOS". In: *Int'l Solid-State Circuits Conference (ISSCC)*. 2009 (cit. on pp. 13, 44, 61).

[56] J. Y. Kim, J. Park, S. Lee, M. Kim, J. Oh, and H. J. Yoo. "A 118.4 GB/s Multi-Casting Network-on-Chip With Hierarchical Star-Ring Combined Topology for Real-Time Object Recognition". In: *Journal of Solid-State Circuits (JSSC)* 45.7 (2010), pp. 1399–1409 (cit. on p. 61).

[57] P. Koka, M. O. McCracken, H. Schwetman, C.-H. O. Chen, X. Zheng, R. Ho, K. Raj, and A. V. Krishnamoorthy. "A micro-architectural analysis of switched photonic multi-chip interconnects". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2012 (cit. on p. 41).

[58] T. Krishna, A. Kumar, L. S. Peh, J. Postman, P. Chiang, and M. Erez. "Express Virtual Channels with Capacitively Driven Global Links". In: *IEEE Micro* 29.4 (2009), pp. 48–61 (cit. on p. 15).

[59] T. Krishna, C.-H. O. Chen, W. C. Kwon, and L.-S. Peh. "Breaking the On-Chip Latency Barrier Using SMART". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2013 (cit. on pp. 41, 145).

[60] T. Krishna, A. Kumar, P. Chiang, M. Erez, and L. S. Peh. "NoC with Near-Ideal Express Virtual Channels Using Global-Line Communication". In: *Symp. on High Performance Interconnects*. 2008 (cit. on p. 61).

[61] T. Krishna, L.-S. Peh, B. M. Beckmann, and S. K. Reinhardt. "Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2011 (cit. on p. 15).

[62] T. Krishna, J. Postman, C. Edmonds, L.-S. Peh, and P. Chiang. "SWIFT: A SWing-reduced Interconnect For a Token-based Network-on-Chip in 90nm CMOS". In: *Int'l Conf. on Computer Design (ICCD)*. 2010 (cit. on pp. 15, 16, 44, 59, 112).

[63] A. Kumar, P. Kunduz, A. P. Singhx, L. S. Pehy, and N. K. Jhay. "A 4.6Tbits/s 3.6GHz Single-cycle NoC Router with a Novel Switch Allocator in 65nm CMOS". In: *Int'l Conf. on Computer Design (ICCD)*. 2007 (cit. on pp. 12, 15).

[64] A. Kumar, L. S. Peh, and N. K. Jha. "Token Flow Control". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2008 (cit. on p. 15).

[65] G. Kurian, O. Khan, and S. Devadas. "The locality-aware adaptive cache coherence protocol". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2013 (cit. on p. 41).

[66] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal. "ATAC: A 1000-Core Cache-Coherent Processor with On-Chip Optical Network". In: *Int'l Conf. on Parallel Architecture Compilation Techniques (PACT)*. 2010 (cit. on pp. 14, 15, 19, 105).

[67] G. Kurian, C. Sun, C. H. O. Chen, J. E. Miller, J. Michel, L. Wei, D. A. Antoniadis, L. S. Peh, L. Kimerling, V. Stojanovic, and A. Agarwal. "Cross-layer Energy and Performance Evaluation of a Nanophotonic Manycore Processor System using Real Application Workloads". In: *Int'l Parallel & Distributed Processing Symposium*. 2012 (cit. on pp. 20, 41).

[68] E. Kyriakis-Bitzaros and S. S. Nikolaidis. "Design of low power CMOS drivers based on charge recycling". In: *Int'l Symp. on Circuits and Systems (ISCAS)*. 1997 (cit. on p. 13).

[69] K. Lee, S.-J. Lee, S.-E. Kim, H.-M. Choi, D. Kim, S. Kim, M.-W. Lee, and H.-J. Yoo. "A 51mW 1.6GHz On-Chip Network for Low-Power Heterogeneous SoC Platform". In: *Int'l Solid-State Circuits Conference (ISSCC)*. 2004 (cit. on p. 43).

[70] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2009 (cit. on pp. 6, 27).

[71] J. Liu, X. Sun, R. Camacho-Aguilera, L. C. Kimerling, and J. Michel. "Ge-on-Si laser operating at room temperature". In: *Optics Letters* 35.5 (2010), pp. 679–681 (cit. on p. 14).

[72] R. Marculescu, D. Marculescu, and M. Pedram. "Probabilistic modeling of dependencies during switching activity analysis". In: *Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 17.2 (1998), pp. 73–83 (cit. on p. 28).

[73] M. M. Martin, M. D. Hill, and D. A. Wood. "Timestamp Snooping: An Approach for Extending SMPs". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2000 (cit. on p. 17).

[74] M. M. Martin, M. D. Hill, and D. A. Wood. "Token Coherence: Decoupling Performance and Correctness". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2003 (cit. on p. 16).

[75] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset". In: *Computer Architecture News* (2005) (cit. on p. 124).

[76] H. Matsutani, M. Koibuchi, H. Amano, and T. Yoshinaga. "Prediction router: Yet another low latency on-chip router architecture". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2009 (cit. on p. 15).

[77] E. Mensink, E. Mensink, D. Schinkel, E. Klumperink, E. van Tuijl, and B. Nauta. "A 0.28pJ/b 2Gb/s/ch Transceiver in 90nm CMOS for 10mm On-chip interconnects". In: *Int'l Solid-State Circuits Conference (ISSCC)* (2007) (cit. on pp. 13, 61).

[78] J. E. Miller, H. Kasture, G. Kurian, C. G. III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. "Graphite: A Distributed Parallel Simulator for Multicores". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2010 (cit. on pp. 20, 125).

[79] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. "Application-Aware Topology Reconfiguration for On-Chip Networks". In: *Trans. on Very Large Scale Integration (VLSI) Systems* 19.11 (2011), pp. 2010–2022 (cit. on pp. 16, 62).

[80] M. Modarressi, A. Tavakkol, and H. Sarbazi-Azad. "Virtual Point-to-Point Connections for NoCs". In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 29.6 (2010), pp. 855–868 (cit. on pp. 16, 62, 72).

[81] A. Moshovos. "RegionScout: Exploiting Coarse Grain Sharing in Snoop-Based Coherence". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2005 (cit. on p. 118).

[82] R. Mullins, A. West, and S. Moore. "Low-Latency Virtual-Channel Routers for On-Chip Networks". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2004 (cit. on p. 15).

[83] S. Murali and G. De Micheli. "Bandwidth-constrained mapping of cores onto NoC architectures". In: *Conf. on Design, Automation and Test in Europe (DATE)*. 2004 (cit. on p. 67).

[84] M. H. Na, E. J. Nowak, W. Haensch, and J. Cai. "The effective drive current in CMOS inverters". In: *Int'l Electron Devices Meeting (IEDM)*. 2002 (cit. on p. 24).

[85] *NCSU FreePDK45*. URL: `http://www.eda.ncsu.edu/wiki/FreePDK` (cit. on p. 25).

[86] C. Nitta, M. Farrens, and V. Akella. "Addressing System-Level Trimming Issues in On-Chip Nanophotonic Networks". In: *Int'l Symp. on High Performance Computer Architecture (HPCA)*. 2011 (cit. on p. 31).

[87] *Oracle's SPARC T5-2, SPARC T5-4, SPARC T5-8, and SPARC T5-1B Server Architecture.* URL: http://www.oracle.com/technetwork/server-storage/sun-sparc-enterprise/documentation/o13-024-sparc-t5-architecture-1920540.pdf (cit. on pp. 136, 137).

[88] J. S. Orcutt, A. Khilo, C. W. Holzwarth, M. A. Popović, H. Li, J. Sun, T. Bonifield, R. Hollingsworth, F. X. Kärtner, H. I. Smith, V. Stojanović, and R. J. Ram. "Nanophotonic integration in state-of-the-art CMOS foundries". In: *Optical Express* 19.3 (2011), pp. 2335–2346 (cit. on p. 31).

[89] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary. "Firefly: Illuminating On-Chip Networks with Nanophotonics". In: *Int'l Symp. on Computer Architecture (ISCA).* 2009 (cit. on pp. 14, 15, 19).

[90] S. Park. "Towards Low-Power yet High-Performance Networks-on-Chip". PhD thesis. Massachusetts Institute of Technology (cit. on pp. 64, 65).

[91] S. Park, T. Krishna, C.-H. O. Chen, B. K. Daya, A. P. Chandrakasan, and L.-S. Peh. "Approaching the theoretical limits of a mesh NoC with a 16-node chip prototype in 45nm SOI". In: *Design Automation Conference (DAC).* 2012 (cit. on pp. 15, 16, 51, 106, 112, 113, 138).

[92] G. Passas, M. Katevenis, and D. Pnevmatikatos. "A 128 × 128 × 24 Gb/s Crossbar Interconnecting 128 Tiles in a Single Hop and Occupying 6% of Their Area". In: *Int'l Symp. on Networks-on-Chip (NOCS).* 2010 (cit. on p. 61).

[93] L.-S. Peh and W. J. Dally. "A Delay Model and Speculative Architecture for Pipelined Routers". In: *Int'l Symp. on High Performance Computer Architecture (HPCA).* 2001 (cit. on pp. 20, 27).

[94] L.-S. Peh and N. E. Jerger. *On-Chip Networks.* Morgan and Claypool, 2009 (cit. on p. 9).

[95] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. "Overview of the Architecture, Circuit Design, and Physical Implementation of a First-Generation Cell Processor". In: *Journal of Solid-State Circuits (JSSC)* 41.1 (2006), pp. 179–196 (cit. on p. 25).

[96] C. Pollock and M. Lipson. *Integrated Optics.* Springer, 2003 (cit. on p. 14).

[97] J. M. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective, second edition.* Prentice Hall, 2003 (cit. on pp. 13, 26).

[98] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. W. Keckler, and C. R. Moore. "Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture". In: *Int'l Symp. on Computer Architecture (ISCA).* June 2003 (cit. on p. 43).

[99] D. Schinkel, E. Mensink, E. Klumperink, A. van Tuijl, and B. Nauta. "Low-Power, High-Speed Transceivers for Network-on-Chip Communication". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.1 (Jan. 2009), pp. 12–21 (cit. on p. 44).

[100] K. Sewell. "Scaling High-Performance Interconnect Architectures to Many-Core Systems". PhD thesis. University of Michigan (cit. on p. 138).

[101] M. A. Shalan, E. S. Shin, and V. J. M. III. "DX-GT: Memory Management and Crossbar Switch Generator for Multiprocessor System-on-a-Chip". In: *Workshop on Synthesis And System Integration of Mixed Information technologies*. 2003 (cit. on p. 44).

[102] M. Sinha and W. Burleson. "Current-sensing for crossbars". In: *Int'l ASIC/SOC Conference*. 2001 (cit. on p. 44).

[103] R. Sredojević and V. Stojanović. "Optimization-based framework for simultaneous circuit-and-system design-space exploration: A high-speed link example". In: 2008 (cit. on p. 44).

[104] *STBus Communication System: Concepts And Definitions*. URL: `http://www.st.com/content/ccc/resource/technical/document/user_manual/39/81/fa/c8/2e/4d/41/f5/CD00176920.pdf/files/CD00176920.pdf/jcr:content/translations/en.CD00176920.pdf` (cit. on p. 44).

[105] M. B. Stensgaard and J. Sparsø. "ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology". In: *Int'l Symp. on Networks-on-Chip (NOCS)*. 2008 (cit. on pp. 16, 62).

[106] K. Strauss, X. Shen, and J. Torrellas. "Uncorq: Unconstrained Snoop Request Delivery in Embedded-Ring Multiprocessors". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2007 (cit. on p. 16).

[107] M. B. Stuart, M. B. Stensgaard, and J. Sparsø. "Synthesis of Topology Configurations and Deadlock Free Routing Algorithms for ReNoC-based Systems-on-Chip". In: *Int'l Conf. on Hardware/Software Codesign and System*. 2009 (cit. on pp. 16, 62).

[108] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanović. "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling". In: *Int'l Symp. on Networks-on-Chip (NOCS)*. 2012 (cit. on pp. iii, 4, 19).

[109] D. Taillaert, P. Bienstman, and R. Baets. "Compact efficient broadband grating coupler for silicon-on-insulator waveguides". In: *Optics Letters* 29.23 (2004), pp. 2749–2751 (cit. on p. 14).

[110] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. "The RAW microprocessor: A computational fabric for software circuits and general-purpose programs". In: *IEEE Micro* 22.2 (2002), pp. 25–35 (cit. on pp. 10, 106, 138).

[111] M. B. Taylor, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, J. Kim, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. "Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2004 (cit. on p. 43).

[112] S. Vangal, N. Borkar, and A. Alvandpour. "A Six-Port 57 GB/s Double-Pumped Nonblocking Router Core". In: *Symp. on VLSI Circuits*. 2005 (cit. on p. 43).

[113] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn. "Corona: System implications of emerging nano-photonic technology". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2008 (cit. on pp. 14, 15, 19).

[114] H. Wang, L.-S. Peh, and S. Malik. "Power-driven Design of Router Microarchitectures in On-chip Networks". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2003 (cit. on p. 43).

[115] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. "Orion: A Power-Performance Simulator for Interconnection Networks". In: *Int'l Symp. on Microarchitecture (MICRO)*. 2002 (cit. on p. 21).

[116] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, P. Xu, and S. Yalamanchili. "Manifold: A parallel simulation framework for multicore systems". In: *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*. 2014 (cit. on p. 41).

[117] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood. "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip". In: *Int'l Symp. on Computer Architecture (ISCA)*. 2013 (cit. on p. 41).

[118] L. Wei, F. Boeuf, T. Skotnicki, and H. .-.-S. P. Wong. "Parasitic Capacitances: Analytical Models and Impact on Circuit-Level Performance". In: *Trans. on Electron Devices* 58.5 (2011), pp. 1361–1370 (cit. on p. 25).

[119] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal. "On-Chip Interconnection Architecture of the Tile Processor". In: *IEEE Micro* 27.5 (2007), pp. 15–31 (cit. on pp. 10, 137, 138).

[120] P. Wijetunga. "High-performance crossbar design for system-on-chip". In: *Int'l System-on-Chip for Real-Time Application*. 2003 (cit. on p. 44).

[121] *Wind River Simics*. URL: `http://www.windriver.com/products/simics` (cit. on p. 124).

[122] D. Wingard. "MicroNetwork-Based Integration for SoCs". In: *Design Automation Conference (DAC)*. 2001 (cit. on p. 44).

[123] N.-S. Woo. "High Performance SOC for mobile applications". In: *Asian Solid-State Circuits Conference (ASSCC)*. 2010 (cit. on p. 61).

[124] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". In: *Int'l Symp. on Computer Architecture (ISCA)*. 1995 (cit. on p. 125).

[125] H. Yamauchi, H. Akamatsu, and T. Fujita. "An Asymptotically Zero Power Charge-Recycling Bus Architecture for Battery-Operated Ultrahigh Data Rate ULSIs". In: *Journal of Solid-State Circuits (JSSC)* 30 (1995), pp. 423–431 (cit. on p. 13).

[126] B.-D. Yang and L.-S. Kim. "High-Speed and Low-Swing On-Chip Bus Interface Using Threshold Voltage Swing Driver and Dual Sense Amplifier Receiver". In: *European Solid-State Circuits Conference (ISSCIRC)*. 2000 (cit. on p. 13).

[127] H. Zhang, V. George, and J. M. Rabaey. "Low-Swing On-Chip Signaling Techniques: Effectiveness and Robustness". In: *Trans. on Very Large Scale Integration (VLSI) Systems* 8.3 (2000), pp. 264–272 (cit. on p. 13).

[128] W. Zhang, Zhang, Li, W. Bing, Z. Zhu, K. Lee, J. Michel, S.-J. Chua, and L.-S. Peh. "Ultralow Power Light-Emitting Diode Enabled On-Chip Optical Communication Designed in the III-Nitride and Silicon CMOS Process Integrated Platform". In: *Design & Test of Computers* 31.5 (2014), pp. 36–45 (cit. on p. 144).