

HopliteBuf FPGA Network-on-Chip: Architecture and Analysis

by

Tushar Garg

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

© Tushar Garg 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

We can prove occupancy bounds of stall-free FIFOs used in deflection-free, low-cost, and high-speed FPGA overlay Network-on-chips (NoCs). In our work, we build on top of the HopliteRT livelock-free overlay NoC with an FPGA-friendly 2D unidirectional torus topology to propose the novel HopliteBuf NoC. In our new NoC, we strategically introduce stall-free FIFOs in the network and support these FIFOs with static analysis based on network calculus to compute FIFO occupancy, latency, and bandwidth bounds. The microarchitecture of HopliteBuf combines the performance benefits of conventional buffered NoCs (high throughput, low latency) with the cost advantages of deflection-routed NoCs (low FPGA area, high clock frequencies).

Specifically, we look at two design variants of the HopliteBuf NoC: (1) Single corner-turn FIFO ($W \rightarrow S$), and (2) Dual corner-turn FIFO ($W \rightarrow S+N$). The single corner-turn ($W \rightarrow S$) design is simpler and only introduces a buffering requirement for packets changing dimension from X ring to the downhill Y ring (or West to South). The dual corner-turn variant requires two FIFOs for turning packets going downhill ($W \rightarrow S$) as well as uphill ($W \rightarrow N$). The dual corner-turn design overcomes the mathematical analysis challenges associated with single corner-turn designs for communication workloads with cyclic dependencies between flow traversal paths at the expense of small increase in resource cost. Essentially, we resolve an analysis challenge with extra hardware resources.

Across a range of 100 synthetically-generated workloads on a 5×5 NoC, HopliteBuf outperforms HopliteRT by $1.2 - 2 \times$ in terms of latency, 10% in terms of injection rate, and 30-60% in terms of flowset feasibility. These advantages come at the cost of $3 - 4 \times$ higher FPGA resource requirement for buffers and muxes. Our analysis also delivers latency bounds that are not only better than HopliteRT in absolute terms but also tighter by $2 - 3 \times$ allowing us to provision less hardware to meet our specifications.

Acknowledgements

I would like to thank my supervisor, Nachiket Kapre, for all the support and help throughout my Masters. I also like to thank Professor Rodolfo Pellizzoni for his help with this work.

Dedication

This is dedicated to my sister, Stuti.

Table of Contents

| | |
|---|----------|
| List of Tables | ix |
| List of Figures | xi |
| Abbreviations | xv |
| Nomenclature | xvii |
| 1 Introduction | 1 |
| 1.1 Main Contributions | 4 |
| 1.2 Thesis Organization | 5 |
| 2 Background and Literature Review | 6 |
| 2.1 Idea of Deflection Routing | 6 |
| 2.1.1 Hoplite | 6 |
| 2.1.2 HopliteRT | 9 |
| 2.2 Survey of FPGA-overlay NoCs | 12 |
| 2.2.1 CMU CONNECT NoC | 13 |
| 2.2.2 Penn Split-Merge NoC | 14 |
| 2.2.3 PaterNoster NoC | 15 |
| 2.2.4 Kim NoC router | 16 |
| 2.3 Survey of Real-time NoCs | 17 |

| | | |
|----------|--|-----------|
| 2.4 | Emergence of Hard NoCs | 18 |
| 2.4.1 | Xilinx Versal | 18 |
| 2.4.2 | Embedded NoC | 19 |
| 2.5 | LUT organization in Intel and Xilinx devices | 20 |
| 2.5.1 | Xilinx LUT organization | 20 |
| 2.5.2 | Intel LUT organization | 21 |
| 2.6 | Basics of Network Regulation | 24 |
| 2.6.1 | Need for Traffic Regulation | 24 |
| 2.6.2 | Token-bucket Regulator | 25 |
| 2.6.3 | Traffic Modelling with Token bucket regulator | 26 |
| 3 | HopliteBuf Microarchitecture | 28 |
| 3.1 | Stall-free Buffers | 28 |
| 3.2 | $W \rightarrow S$ buffer design | 31 |
| 3.3 | $W \rightarrow S + N$ buffer design | 32 |
| 3.4 | FPGA Implementation | 32 |
| 3.5 | Routing Policy | 33 |
| 3.6 | $W \rightarrow S$ Buffer Design with Backpressure | 39 |
| 3.7 | $N \rightarrow S$ Buffer Design with Deflections | 40 |
| 4 | Network Calculus and Analysis | 41 |
| 4.1 | Client Traffic Regulation | 41 |
| 4.2 | Traffic Model | 42 |
| 4.3 | Injection Latency | 45 |
| 4.4 | Vertical Ring Analysis $W \rightarrow S$ Design | 46 |
| 4.5 | Cyclic Dependencies in HopliteBuf $W \rightarrow S$ design | 51 |
| 4.6 | Linearized Analysis: $W \rightarrow S + N$ Design | 54 |



| | | |
|----------|---|-----------|
| 5 | Evaluation | 58 |
| 5.1 | RTL Simulation Results | 59 |
| 5.1.1 | Flowset Feasibility | 59 |
| 5.1.2 | Worst-Case Latency Trends | 61 |
| 5.1.3 | Worst-Case Latency Breakdown | 62 |
| 5.1.4 | Latency Distribution | 64 |
| 5.1.5 | FIFO Sizing | 65 |
| 5.2 | FPGA Implementation Results | 68 |
| 5.3 | Analysis Results | 70 |
| 5.3.1 | Feasible Flowsets | 70 |
| 5.3.2 | Worst-Case Latency: Analysis vs. Simulation | 70 |
| 5.3.3 | FIFO Sizing: Analysis vs. Simulation | 72 |
| 6 | Conclusion and Future Research | 73 |
| 6.1 | Future Research | 74 |
| | References | 75 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Routing Function Table to support single 6-LUT implementation of Hoplite. N has highest priority, followed by W port and PE has the least priority. PE and W cannot inject simultaneously if W has a packet even if the flows are non-overlapping. | 9 |
| 2.2 | Routing Function Table to support two 6-LUT implementation of Hoplite. This supports all routing possibilities. Allows PE and W to inject simultaneously when there is no overlap. | 9 |
| 2.3 | FPGA costs for 64b router (4×4 NoC) with Vivado 2016.4 (Default settings) + Virtex-7 485T FPGA (adapted from [57]) | 11 |
| 2.4 | Routing Function Table (adapted from [57]) for HopliteRT. PE injection has lowest priority and will stall on conflict. PE→E + W→S is not supported to avoid an extra select signal driving the multiplexers and doubling LUT cost by preventing fracturing a 6-LUT into 2×5-LUTs. | 11 |
| 3.1 | Resource utilization on a Xilinx Virtex-7 FPGA for different Data Width and FIFO sizes. | 34 |
| 3.2 | Resource utilization on a Intel Arria-10 FPGA for different Data Width and FIFO sizes. | 34 |
| 3.3 | DOR Routing Policy for FIFO- W router. PE_i always has the least priority. S exit is shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port. Extra combination of $W \rightarrow E$ and $W' \rightarrow S$ also supported. | 35 |
| 3.4 | DOR Routing Policy for FIFO- $W \rightarrow S + N$ router. PE_i again has the least priority. S_o exit shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port, $W'' \rightarrow N$ uses NFIFO read port. | 37 |

| | | |
|-----|---|----|
| 4.1 | Flow parameters for the example NoC. Γ^C are the conflicting flows used in Section 4.3; $f_{W \rightarrow S}$ and $f_{N \rightarrow S}$ are the $W \rightarrow S$ and $N \rightarrow S$ interfering flows used in Section 4.4. ‘-’ denotes not applicable. | 44 |
| 4.2 | Conflicting and interfering flows for the $W \rightarrow S + N$ design. ‘-’ denotes not applicable, as the flow is not buffered in that direction. | 55 |

List of Figures

| | | |
|-----|--|----|
| 1.1 | 2×2 HopliteBuf NoC Topology with Stall-Free Corner-Turn buffers (). Stall-Free buffers are sized to never go full avoiding the need for backward flow control. The dual buffer ($W \rightarrow S + N$) topology disconnects vertical rings () and introduces an extra uphill multiplexer in each router. | 3 |
| 2.1 | An example 3×3 torus with horizontal and vertical wire rings. The processing element (PE) can be an FPGA datapath that injects packet in the network, while the Switches (SW) arbitrate the packets based on network status and arbitration scheme. | 7 |
| 2.2 | Implementation choices for the Hoplite FPGA NoC Router. A LUT-economical version (left) is able to exploit fracturable Xilinx 6-LUTs to fit both 2:1 muxes into a single 6-LUT. The larger, higher-bandwidth version (right) needs two 6-LUTs instead as the number of common inputs is lower than required to allow fracturing. | 8 |
| 2.3 | Endless deflection scenario (adapted from [57]) where red packets from PE (0,0) \rightarrow PE (3,3) are perpetually deflected by blue packets from PE (3,3) \rightarrow PE (3,1). The red spaghetti is the flight path of one packet that gets trapped in the top-most ring of the NoC and never gets a chance to exit due to the green packets. | 10 |
| 2.4 | High-level design sketch of HopliteRT [57] FPGA NoC Routers. HopliteRT adds a N \rightarrow E turn to Hoplite to eliminate livelock. This design also fit one bit of crossbar in one Xilinx 6-LUT. | 11 |
| 2.5 | Worst-Case path on Hoplite-RT for packet traversing from top-left PE (0,0) to bottom-right PE (3,3). The red packets will deflect N \rightarrow E in each ring due to a conflicting flow (not shown). The blue packets previously had priority are now deflected in the top-most ring before delivery. (Adapted from [57]) | 13 |

| | | |
|------|---|----|
| 2.6 | Xilinx LUT-6 sturcture. A fracturable Xilinx 6-LUT can fit two 5-LUTs with common inputs | 21 |
| 2.7 | (a) A LUT-6 based 32-bit Shift Register on Xilinx FPGA (b)Two LUT-6 cascaded to implement a 64-bit Shift register. The cascading is done with the help of F7AMUX which is available in SLICEM of a Combinational Logic Block in Xilinx FPGAs | 22 |
| 2.8 | Various Intel ALUT configurations | 23 |
| 2.9 | The example shows an inundated client at (1,0) that has flooded the NoC with packets at full link bandwidth (one packet per cycle) | 25 |
| 2.10 | A cartoon diagram representing the Token Bucket regulation. A regulator is used at the injection port of each NoC client. Number of tokens represent the number of packets allowed to inject in the network by a client. | 26 |
| 2.11 | Example traffic curve for $\lambda_{b=3,\rho=1/4}$ | 27 |
| 3.1 | Two design alternatives for adding buffers to the Hoplite NoC router. $W \rightarrow S$ adds a buffer on the corner turn, while $W \rightarrow S + N$ adds an extra uphill buffer. | 29 |
| 3.2 | Routing scenario to show the number of buffers required to support a flow of packets from PE (2,2) \rightarrow PE (3,2). For $N \rightarrow S$ buffering case, the packet is required to be stored at multiple times in each buffer along column-3. For single-buffering case, the packet is stored only once at PE (3,2) to reach its destination. | 31 |
| 3.3 | Flow of traffic in the 2-D torus with HopliteBuf $W \rightarrow S$ switches. In case of contention between W and N packets, W goes in the buffer and N gets access to S . There is no backpressuring among the switches; and therefore, buffer sizing is done by static analysis of flows. | 36 |
| 3.4 | Flow of traffic in the network with HopliteBuf $W \rightarrow S + N$ switches. The torus does not have vertical links going from bottom to top. Each switch has a dedicated upward path. A packet trying to exit at a node above it will have to go all the way and then come down to exit as shown in blue flow from switch (3,3) \rightarrow (3,1). | 38 |
| 3.5 | Buffered NoC designs with backpressuring and deflections | 39 |
| 4.1 | Example traffic curves for $\lambda_{b=3,\rho=1/4}$ and $\lambda_{b=2,\rho=1/4}$ along with an arrival curve for $\gamma_{b=7/4,\rho=1/4}$ | 42 |

| | | |
|-----|---|----|
| 4.2 | Example $W \rightarrow S$ NoC design with five flows $f_{1..5}$ | 44 |
| 4.3 | Flows through a $W \rightarrow S$ router. | 48 |
| 4.4 | Arrival (α) and Service ($\beta_{R,T}$) curve that shows the delay (d) and rate (R) for the aggregate flow on W port when obstructed by the high-priority $N \rightarrow S$ flow. x denotes the backlog (amount of buffering) on $W \rightarrow S$ | 49 |
| 4.5 | Service curve ($\beta_{R',T'}$) that extracts flow f from the aggregate flow ($f+f_{W \rightarrow S}$), $\beta_{R,T}$ shown in Figure 4.4. d is the queueing delay for turning flow, f | 50 |
| 4.6 | Cyclic dependency flow example in $W \rightarrow S$ HopliteBuf: rightmost column. | 52 |
| 4.7 | $W \rightarrow S + N$ design example: rightmost column. | 55 |
| 4.8 | Solving cyclic dependency with linearization in $W \rightarrow S + N$ design example: rightmost column. | 56 |
| | | |
| 5.1 | Feasible flowsets for RANDOM traffic with $b=1$ at 5×5 system size with 128-deep FIFOs in the NoC routers. | 59 |
| 5.2 | Feasible flowsets for RANDOM traffic with $b=1$, FIFO depth=128 with different system sizes and injection rates. | 60 |
| 5.3 | Worst-case latency trends for ALL-TO-ONE, ALL-TO-ROW, and ALL-TO-COLUMN traffic patterns on NoCs with 5×5 system sizes and $b=1$. HopliteBuf designs offers no improvements for these traffic patterns. | 61 |
| 5.4 | Worst-case latency trends for RANDOM traffic pattern on NoCs with 5×5 system sizes and $b=1$. HopliteBuf performs better than other designs for this workload. | 62 |
| 5.5 | Breakdown of Source-Queueing and In-Flight NoC latencies for RANDOM workload with $b=1$, FIFO depth=128 at 5×5 system size. Both metrics improved due to buffering. | 63 |
| 5.6 | Source-Queueing and In-Flight NoC latencies for RANDOM workload with $b=1$, FIFO depth=32 at 5×5 system size. | 64 |
| 5.7 | Distribution of worst-case packet latencies for RANDOM workload with $b=1$, $\rho=7.5\%$ at 5×5 system size. HopliteRT has a wider spread due to the unpredictable nature of the deflections. HopliteBuf has narrower spreads. | 65 |
| 5.8 | Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-ONE pattern | 66 |

| | | |
|------|---|----|
| 5.9 | Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-ROW pattern | 66 |
| 5.10 | Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-COLUMN pattern | 67 |
| 5.11 | Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for RANDOM pattern | 67 |
| 5.12 | LUT utilization for logic and memory across various Hoplite routers on Xilinx and Intel FPGAs with Payload=64b and FIFO=64 deep. | 68 |
| 5.13 | Maximum Area usage at different injection rates for RANDOM pattern with 5×5 system size. Each node in the system uses carefully picked FIFO sizes through static analysis in case of $W \rightarrow S$ and $W \rightarrow S + N$ HopliteBuf designs. | 69 |
| 5.14 | Feasible flowsets predicted by static analysis. Analysis is more conservative than simulation for HopliteRT, but much tighter for HopliteBuf. | 71 |
| 5.15 | Worst-Case Latency Prediction-vs-Simulation, RANDOM traffic, $b=1$, 5×5 system size, 128-deep FIFOs. | 71 |
| 5.16 | Worst-Case FIFO size Prediction-vs-Simulation for RANDOM traffic at 5×5 system size with 128-deep FIFOs and burstiness of 8. | 72 |

Abbreviations

- ADAS** Advanced drive-assistance systems [2](#)
- ALUT** Adaptive Look-up Table [21](#)
- ASICs** Application-Specific Integrated Circuits [1](#)
- BE** Best-effort [18](#)
- BRAM** Block Random Access Memory [1](#)
- DOR** Dimension-ordered routing [7](#), [8](#), [13](#), [29](#), [34](#), [36](#)
- DSP** Digital Signal Processing [1](#)
- FFs** Flip-Flops [1](#)
- FPGAs** Field Programmable Gate Arrays [1](#)
- HBM** High-Bandwidth memory [1](#), [2](#)
- ISOC** Isochronous [18](#)
- LAB** Logic Array Block [24](#)
- LL** Low-latency [18](#)
- LUTs** Look-up Tables [1](#)
- MLAB** Memory Logic Array Block [24](#)

NoC Network-on-chip [1](#)

QoS Quality of Service [18](#)

SRL Shift-Register LUT [15](#)

TDM Time division multiplexed [17](#)

UAV Unmanned air vehicle [2](#)

VC Virtual-channel [14](#), [19](#)

WSF West-side first [15](#)

Nomenclature

credit-based flow control A backpressure mechanism in which each NoC router keeps track of the number of available buffer spaces or credits. By tracking credits, a router only sends packet downstream if it has available credits. [19](#)

Dimension-Ordered Routing A routing algorithm in which packets first move in the X-direction only until they reach the right column and then they turn in Y-direction to reach their destination. Once the packet is travelling in Y-direction, it is not allowed to turn back to X-direction. [34](#)

head-of-line The situation when unrelated packets cannot move forward in the network because of the backpressure caused by some other flow is called head-of-line blocking. [19](#)

in-flight latency The amount of time a packet takes between exiting the source node and reaching the destination client. [9](#), [62](#)

livelock A condition due to which packets wander across the network forever. The situation can occur when the channel required to deliver a packet to its destination is occupied by some other packet. [2](#), [8](#), [29](#), [34](#), [40](#)

source-queueing latency The amount of time a packet has to wait at the source client node before it is allowed to enter the network. [9](#), [24](#), [62](#)

Chapter 1

Introduction

Field Programmable Gate Arrays (FPGAs) are integrated circuits that can be electrically configured to implement **any** digital system. They are built around a matrix of programmable logic blocks, called **Look-up Tables (LUTs)**, connected together with a network of programmable interconnects to support data movement. Unlike **Application-Specific Integrated Circuits (ASICs)**, FPGAs can be reprogrammed, thus allowing configurable computing. Over the years, FPGAs have evolved to support more than just programmable LUT blocks. They now include an on-chip CPU, large memories, high-speed I/O links and have now become first-class citizens in data-center applications for workloads such as Machine learning, Graph acceleration, Search engine optimization. Modern FPGAs can pack millions of **LUTs** and **Flip-Flops (FFs)**, thousands of **Digital Signal Processing (DSP)** blocks and **Block Random Access Memory (BRAM)**. Such dense devices need a rich, and capable interconnect network-on-chip to satisfy the communication requirements of FPGA workloads. Latest memory interfaces, such as **High-Bandwidth memory (HBM)**, for instance, requires FPGAs to be able to move data on upto 32×256 -bit wide high-speed links that supports upto 460 GB/s bandwidth across the entire chip. Furthermore, wide variety of IO protocols and interface specifications, like, 8-lanes PCIe Gen4, 100G Ethernet, 58Gb/s Transceivers, are all pushing the need of an efficient system level interface to move data to various portions of the chip.

FPGA logic can be configured to implement packet-switched **Network-on-chip (NoC)** to allow IP blocks to exchange data with each other. This is easy to do, but comes at the cost of stealing logic and routing resources away from the developer. Prior work in soft-NoCs is rich and extensive. CMU Connect [49], Penn Split-Merge [27], Hoplite [31], and Hoplite-RT [57] are a few state-of-the-art soft NoCs. The programmable and wiring rich

architecture of an FPGA makes it a suitable choice to implement NoC designs to match user requirements.

However, a careful consideration is required when choosing features and topologies to support on an FPGA. For example, a 32bit CMU Connect [49] router utilizes around 1.5K LUTs and runs at clock frequency of 9.6 ns while the Penn Split-Merge [27] router takes up 1.7K LUTs and runs at 4.5 ns. For context, a light weight RISC-V CPU takes ≈ 200 –500 LUTs on an FPGA. The reason for such a bloated and slow design is that these routers support exotic features in a NoC, such as flow-control, extensive buffering, arbitration logic, virtual-channels which are not favourable for FPGA realization.

Deflection-routed NoCs, such as, Hoplite [31], and HopliteRT [57], on the other hand, take advantage of FPGA-specific features to implement a bufferless, low-cost and high-speed (90 LUTs and runs at a clock frequency of 2 ns for 32-bit payload) communication network on an FPGA. However, due to deflection, packets may suffer long delays in the interconnect (Hoplite, HopliteRT) or they may even deflect endlessly and suffer from **live-lock** (Hoplite). With these deflection-routed NoCs, the packet ordering also becomes the responsibility of the destination node with reassembly buffers which adds to the logic cost. Large or endless deflections become a problem when employing these NoC designs into Real-time FPGA applications. Mission critical applications, such as **Unmanned air vehicle (UAV)**, **Advanced drive-assistance systems (ADAS)**, medical devices, self-driving automotive require stringent timing guarantees for assessing and handling worst-case behavior. Furthermore, the state-of-the-art analysis of NoC buffer bounds is pessimistic due to the complexity of modeling pipelining effects and merging of various flows in more conventional buffered mesh networks with backpressure.

Configurable interconnect was adequate for circuit communication needs until recently, but not any longer. System-level IO bandwidth requirements due to HBM memory channels, and high-speed serial IO transceivers are pushing FPGA vendors to implement hardened NoCs, such as the Xilinx Versal NoC [55] for data transport, without using any soft fabric logic. Intel uses a hardened configuration NoC design on Stratix 10 [39]. This hardened NoC allows programmers to support high-performance transactions on FPGA using a flexible interconnect that uses separate command and response networks for higher parallelism and lower resource utilization. Xilinx uses a hardened AXI memory-mapped NoC architecture in their ACAP/Versal FPGA that supports various full-duplex 128-bit links providing a throughput of over 16GB/s. It creates a high-speed interconnect to move data between system-level I/O like **HBM**, and compute accelerators to various parts of the FPGA fabric. An embedded hard-NoC along with custom interfaces was presented by University of Toronto [4]. This NoC can transport 150 bits of data at 1.2 Ghz clock speed and support an overall bandwidth of 22.5GB/s.

Regardless of the choice of soft or hard NoC technology, real-time system developers wishing to use FPGAs need tool support to analyze the timing properties of their FPGA mappings to ensure they are able to meet relevant scheduling deadlines. Real time systems are characterized by a need to rigorously prove timing requirements of various computing and communicating blocks. For instance, the ISO 26262 standard [29] requires performance isolation between communicating components on a chip. Livelocks in NoCs violate this isolation requirement. While timing analysis of statically-scheduled FPGA datapaths is simple, the analysis of communicating components using a shared dynamically-scheduled resource like a NoC is not so. NoC performance analysis is notoriously hard; it is often pessimistic and leads to over-provisioning of resources. In this work, we aim to build analysis-friendly, low-cost FPGA NoC architectures and develop accompanying analysis tools to prove worst-case NoC packet routing latencies.

In this paper, we propose the HopliteBuf NoC, shown in Figure 1.1, derived from the low-cost Hoplite NoC. HopliteBuf introduces (1) small stall-free buffers for certain router functions to simplify flow control, to (2) eliminate deflections with any associated livelocks, (3) provides optional linearization of vertical NoC rings to enhance analysis that, (4) bounds buffer sizes to distributed-RAM friendly implementation sizes.

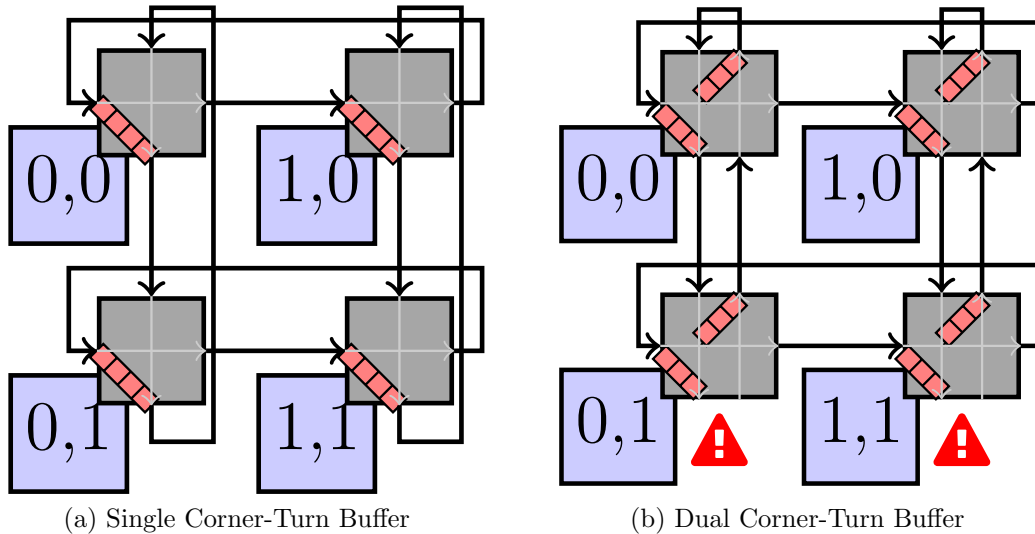


Figure 1.1: 2×2 HopliteBuf NoC Topology with Stall-Free Corner-Turn buffers (▬▬▬). Stall-Free buffers are sized to never go full avoiding the need for backward flow control. The dual buffer ($W \rightarrow S + N$) topology disconnects vertical rings (⚠) and introduces an extra uphill multiplexer in each router.

The key contributions of our work is the microarchitecture of analysis-friendly HopliteBuf NoC (including routers, and topology modifications) coupled with a buffer sizing algorithm that is able to determine the worst-case occupancies for all the FIFOs in the NoC. In particular, the proposed topology in Figure 1.1b with two corner-turn buffers and no vertical loopback simplifies static analysis of buffer sizes. This also improves provable wire utilization while preserving wiring cost and requiring a modest increase in LUT count over Figure 1.1a. For our workloads we observe that these occupancies are small enough to be realizable in LUT-based FIFOs (Xilinx SRL32s, and Intel MLABs). On an average, HopliteBuf NoC is more expensive than Hoplite or HopliteRT by 3–4× due to buffering (5× system size), but still cheaper than conventional buffered NoCs. HopliteBuf incorporates stall-free FIFOs that guarantees in-order packet delivery and uses a static-analysis tool that not only size these FIFOs but also provide detailed information, such as per-flow latency, and bandwidth availability at each switch in the network.

1.1 Main Contributions

We summarize the main contributions of our work here:

- Design of an FPGA NoC router microarchitecture design with stall-free FIFOs to eliminate deflections and provide in-order packet delivery. Optimization and customization of the NoC router RTL to match Xilinx and Intel FPGA LUT organization.
- Development of a buffer sizing algorithm using network calculus to compute the worst-case bounds on FIFO occupancies. Static analysis tools compute upper bounds on size of FIFOs required for stall-free operation, source queueing delay, in-flight routing latency under various conditions.
- Use of vertical NoC link linearization to improve provable link utilization.
- Engineering of a robust simulation infrastructure to compute cycle counts of packet traversals in the NoC. Resource and performance analysis of the NoC under various synthetic workloads.
- The proposed design reduces the latency by 1.2 – 2×, achieves 10% higher injection rate, supports 30-60% more feasibility rate while paying 3–4× extra cost of buffering than Hoplite(RT).

1.2 Thesis Organization

The remainder of the thesis is organized as follows.

- Chapter 2 presents a detailed literature review on Soft and Hard NoCs implemented on FPGAs and ASICs, their implementation, limitations and advantages. It then presents a detailed background on Xilinx and Intel FPGA architectures, their LUT design and memory implementations. And then it reviews existing Traffic regulations techniques and basic Network Calculus.
- Chapter 3 presents the main idea of this work including the architecture of the two variants of NoCs. It then reviews the FPGA implementation of these NoC designs and explains the details of the arbitration scheme.
- Chapter 4 explains, in detail, the mathematics behind static analysis to implement the buffer sizing algorithm including concepts of Network Calculus and Traffic regulation. It then explains the linearization model that gives the dual-buffer design an upper hand over the single buffer design.
- In Chapter 5, we demonstrate the quantitative benefits of this work over the previous designs. We explain the detailed results of various synthetic traffic models.
- Finally, Chapter 6 concludes the work presented in this thesis and outlines future research directions.

Chapter 2

Background and Literature Review

In this chapter, we review existing literature on deflection-routed FPGA overlay NoCs, buffered FPGA overlay NoCs and understand their limitations and advantages. Next, we review real-time NoCs and identify the features that makes them suitable for mission-critical applications. We then provide a brief tutorial on LUT architecture for Intel and Xilinx FPGAs. Finally, we look at the concept of traffic regulation and network calculus to lay the groundwork for understanding analysis of NoC flow.

2.1 Idea of Deflection Routing

Deflection Routing is key to building high-performance and low-cost NoC designs on ASICs as well as FPGAs. Deflection routing eliminates buffers and misroutes packets to keep cost low. Surprisingly, this does not lead to dramatic performance losses for real world applications. We have seen some state-of-the-art deflection routed NoCs like Hoplite [31], HopliteRT [57], CHIPPER (Cheap-Interconnect Partially Permuting Router) [19] and MinBD (Minimally-Buffered Deflection) [20]. While Hoplite and HopliteRT are FPGA-overlay NoCs, CHIPPER and MinBD target ASICs. Let's take a closer look at their underlying architecture and understand their limitations and advantages:

2.1.1 Hoplite

Hoplite [31] is an FPGA-friendly NoC router that uses deflection routing on a torus topology 2.1. It eliminates buffering and flow control to provide a low-cost implementation

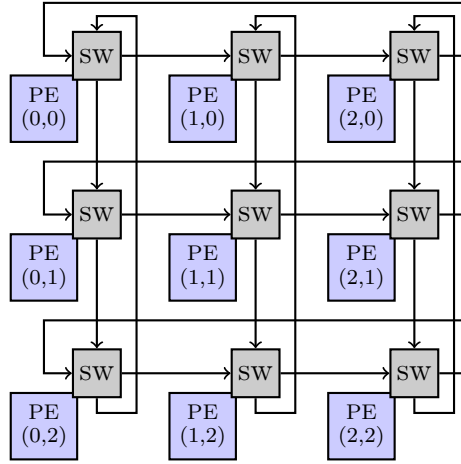


Figure 2.1: An example 3×3 torus with horizontal and vertical wire rings. The processing element (PE) can be an FPGA datapath that injects packet in the network, while the Switches (SW) arbitrate the packets based on network status and arbitration scheme.

on modern Xilinx and Intel FPGAs. Packets traverse using [Dimension-ordered routing \(DOR\)](#) policy in the X-dimension (horizontally, $W \rightarrow E$) first before turning ($W \rightarrow S$) and then routing in the Y-dimension (vertically, $N \rightarrow S$). While DOR is not strictly required for deflection-routed switches, Hoplite includes this routing scheme to reduce switching cost by eliminating certain turns in the router. The internal microarchitecture of the router is shown in [Figure 2.2](#). This simple design requires a pair of 2:1 muxes and DOR logic that defines the arbitration scheme with three inputs N (North), W (West) and P E (processing element/client) and two outputs S (South + Switch exit, shared) and E (East). The DOR control logic is very simple and consumes very few LUTs as it can be constructed directly on valid signals of the incoming packets alone. Packets exiting the NoC must do so over the S port to allow the mux and wires to be shared by both south-bound and exiting packets. A separate output valid signal helps determine the nature of the packet.

When packets at two input ports contend to access the same output port, one of them is intentionally mis-routed/deflected adhering to the concept of DOR along an undesirable direction to avoid the need for buffering. The fractured implementation [2.2a](#) of the NoC serializes the multiplexing decisions to enable a compact single Xilinx 6-LUT realization of the switching crossbar per bit. However, this sacrifices the routing freedom to achieve this low cost. A larger design [2.2b](#) that needs two 6-LUTs per bit (2 more cost) permits a greater bandwidth through the switches. The larger cost is due to the inability to share enough

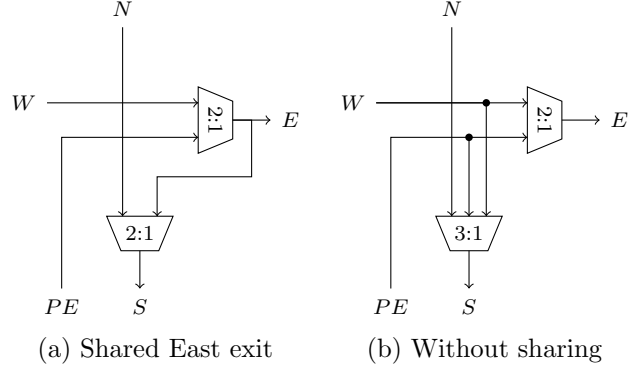


Figure 2.2: Implementation choices for the Hoplite FPGA NoC Router. A LUT-economical version (left) is able to exploit fracturable Xilinx 6-LUTs to fit both 2:1 muxes into a single 6-LUT. The larger, higher-bandwidth version (right) needs two 6-LUTs instead as the number of common inputs is lower than required to allow fracturing.

inputs that would have allowed fracturing the Xilinx 6-LUT into dual 5-LUTs. Details of FPGA LUT organization is explained later in Section 2.5. The arbitration scheme is illustrated in Table 2.1 for the shared input fracturable design of Hoplite and in Table 2.2 for the non-shared two 6-LUT implementation of the same design.

Livelock in Hoplite

A key limitation of Hoplite is the inability of the NoC to avoid livelock. This is because a W packet continues to get deflected to the E port as long as a packet on N port wants to travel S . Furthermore, a series of packets sent from a source client to a destination client may take different paths through the network and need not deflect in identical manner. An example flow to create a livelock situation has been shown in Figure 2.3 where PE (0,0) wants to communicate with PE (3,3) and follows the red-path in the figure and PE (3,3) wants to communicate with PE (3,1) and follows the blue path. Following DOR, there is a contention among red and blue packets to access the *South* port. The arbitration scheme of Hoplite prioritizes $N \rightarrow S$ packets; and therefore, blue packet gets access to the *South* port while red gets mis-routed back in the horizontal lane. When this situation continues on the following cycles as well, the red packet gets deflection over and over again and creates a situation of **livelock**. It is only allowed to reach it's destination once there is no contention and hence the worst-case latency is outrageously high. Due to this, the

Table 2.1: Routing Function Table to support single 6-LUT implementation of Hoplite. N has highest priority, followed by W port and PE has the least priority. PE and W cannot inject simultaneously if W has a packet even if the flows are non-overlapping.

| Mux select | | Routes | Explanation |
|------------|-------|--------------|-----------------------|
| s_sel | e_sel | | |
| 0 | 0 | PE→E or PE→S | No W or N packet |
| 0 | 1 | W→S or W→E | Even if PE has packet |
| 1 | 0 | N→S + PE→E | No W packet |
| 1 | 1 | N→S + W→E | No N packet |

Table 2.2: Routing Function Table to support two 6-LUT implementation of Hoplite. This supports all routing possibilities. Allows PE and W to inject simultaneously when there is no overlap.

| Mux select | | | Routes | Explanation |
|------------|--------|-------|------------|-----------------|
| s_sel1 | s_sel0 | e_sel | | |
| 0 | 0 | 1 | W→E + PE→S | No N packet |
| 0 | 1 | 0 | W→S + PE→E | No N packet |
| 1 | 0 | 0 | N→S + PE→E | No W packet |
| 1 | 0 | 1 | N→S + W→E | Non overlapping |

amount of time a packet has to wait at the source client node before it is allowed to enter the network, called the [source-queueing latency](#), and the amount of time a packet takes to reach its destination, called [in-flight latency](#), for Hoplite are both ∞ .

2.1.2 HopliteRT

HopliteRT [57] is a refinement over Hoplite that inverts the priorities and deflects N packet to the E (hence the new N→E turn in Figure 2.4). By doing this, HopliteRT achieves livelock freedom and bounds the worst-case latency as a function of system size. HopliteRT requires two 3:1 muxes but still requires the same number of LUTs as Hoplite. This is possible because the multiplexer inputs are shared and it requires a 5-input, 2-output function which can be implemented using careful mapping using a LUT-6 (explained in Section 2.5). HopliteRT overcomes the livelock limitation of Hoplite by forcing the low-

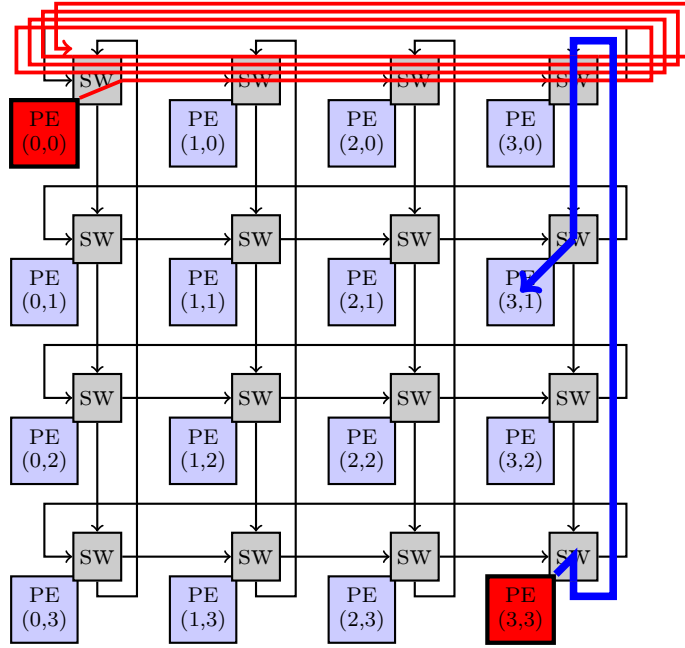


Figure 2.3: Endless deflection scenario (adapted from [57]) where red packets from PE (0,0) \rightarrow PE (3,3) are perpetually deflected by blue packets from PE (3,3) \rightarrow PE (3,1). The red spaghetti is the flight path of one packet that gets trapped in the top-most ring of the NoC and never gets a chance to exit due to the green packets.

priority N packet to deflect E in case of contention cause by high-priority $W \rightarrow S$ flow. The N packet then reappear as a W packet with higher priority. This simple modification means that a packet will only suffer a single deflection at a given switch as it descends down the NoC. The adaptation not only avoids livelock but puts an upper bound on in-flight NoC latency to $\Delta X + \Delta Y \times m + 2$ for an $m \times m$ NoC, where ΔX and ΔY is the number of steps or nodes a packet has traversed in the X and Y direction respectively. This indicates that, in the worst-case (when $\Delta X = m$ and $\Delta Y = m$), the torus NoC could reduce down to a ring $O(m^2)$.

HopliteRT limits the number of deflections by inverting the priorities of the router to always prefer W traffic over N traffic (opposite to Hoplite) and an extra turn from $N \rightarrow E$. This modified policy makes the X-ring (West) completely conflict free, while the Y-ring traffic (North) suffers deflections. Once the Y-ring traffic is deflected onto the X-ring, it gains a higher priority over any other Y-ring traffic. This ensures that the packet will always make forward progress in HopliteRT unlike Hoplite where packet can stuck in the X-ring. Note that the source-queueing and in-flight latency were unbounded in case of

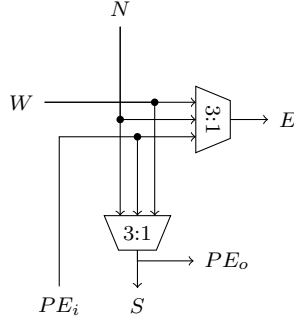


Figure 2.4: High-level design sketch of HopliteRT [57] FPGA NoC Routers. HopliteRT adds a N→E turn to Hoplite to eliminate livelock. This design also fit one bit of crossbar in one Xilinx 6-LUT.

Hoplite, but due to the modified arbitration scheme and architecture of HopliteRT – we can get an upper bound on in-flight latency. To bound source-queueing latency, HopliteRT uses the concept of Traffic regulation explained later in Section 2.6.1.

Table 2.3: FPGA costs for 64b router (4×4 NoC) with Vivado 2016.4 (Default settings) + Virtex-7 485T FPGA (adapted from [57])

| Router | LUTs | FFs | Period (ns) |
|-----------|------|-----|-------------|
| Hoplite | 89 | 149 | 1.29 ns |
| HopliteRT | 86 | 146 | 1.22 ns |

Table 2.4: Routing Function Table (adapted from [57]) for HopliteRT. PE injection has lowest priority and will stall on conflict. PE→E + W→S is not supported to avoid an extra select signal driving the multiplexers and doubling LUT cost by preventing fracturing a 6-LUT into 2×5-LUTs.

| Mux select | | Routes | Explanation |
|------------|------|------------|--|
| sel1 | sel0 | | |
| 0 | 0 | W→E + N→S | Non-interfering |
| 0 | 1 | W→S + N→E | Conflict over S (Not supported in Hoplite) |
| 1 | 0 | PE→E + N→S | No W packet |
| 1 | 1 | PE→S + W→E | No N packet (Not possible in Hoplite) |

The routing policy for HopliteRT is shown in Table 2.4. The routing policy is built

such that it uses the same select bits for both 3:1 multiplexers. Hence, the design can exploit fracturability of the 6-LUT by supplying identical 5 inputs to the 3:1 mux. The interpretation of the select bits for each mux is different thus ensuring desired routing decision. However, the routing policy does not support, the PE→E with W→S turns happening in the same cycle even though the mux bandwidth is rich enough to support this condition. This is done to avoid creating a third mux select signal that would prevent the fractured 6-LUT mapping. The bandwidth capability of the HopliteRT switch is in-between Figure 2.2a and Figure 2.2b. Thus, HopliteRT is equally expensive (shown in Table 2.3) to a less-capable switch in Figure 2.2a while it offers extra bandwidth and latency guarantees.

1-D Ring Scenario in HopliteRT

While HopliteRT costs the same as Hoplite and removes livelocks, it still suffers from an unusually high deflection in worst cases while doing nothing to eliminate out-of-order delivery.

Lets take a look at an example of how HopliteRT reduces a torus NoC to a 1-D ring, as we mentioned above. In Figure 2.5, we show a red path taken by a packet from PE (0,0)→(3,3) and a blue path taken by a packet from PE (3,3)→(3,1) same as the example earlier in Figure 2.3 . The arbitration scheme of HopliteRT prioritizes the $W \rightarrow S$ over $N \rightarrow S$ packets. In this scenario, we assume there are $W \rightarrow S$ flows in each X-ring 0, 1, 2, and 3 that will interfere with the red packets as it descends down in the network. Due to the presence of high-priority ($W \rightarrow S$) flows, the low-priority red packets are forced to deflect at each X-ring. Hence, in the worst case the 2-D torus becomes a 1-D ring with $O(m^2)$ deflections.

Reducing a bandwidth-rich torus to a ring is neither an efficient, nor a scalable use of FPGA resources. Also, HopliteRT reorders the packets in the NoC and hence reordering of packets now becomes the responsibility of the NoC client and can add extra memory resources at the endpoints.

2.2 Survey of FPGA-overlay NoCs

In this section, we will review existing FPGA-overlay NoCs, such as, **CMU CONNECT** [49], **Penn Split-Merge** [27], **PaterNoster NoC** [42], and **Kim NoC router** [35]. We will try to understand the benefits and features supported on this NoC designs. We are

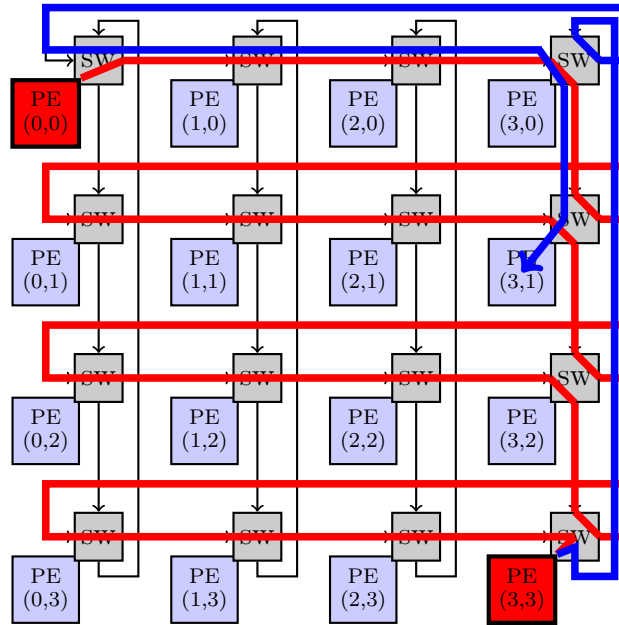


Figure 2.5: Worst-Case path on Hoplite-RT for packet traversing from top-left PE (0,0) to bottom-right PE (3,3). The red packets will deflect N→E in each ring due to a conflicting flow (not shown). The blue packets previously had priority are now deflected in the top-most ring before delivery. (Adapted from [57])

also going to take a look at their limitations to generate a low-cost and high-performance design on FPGAs.

2.2.1 CMU CONNECT NoC

CMU CONNECT [49] is a FPGA-overlay NoC with flexible construction methodologies such as topology, buffer, and virtual channels. It supports various routing algorithms, including DOR, implements optional virtual channels, and supports variety of flow-control options. CONNECT was implemented using Bluespec System-Verilog (BSV) [1] which makes it possible to maintain a flexible design supporting a wide-range of features, such as variable IO ports, flit support, variable flit width, virtual channel support, pipelining in Virtual channels, and flexible routing. Each IO port in the router consists of two channels: one for data transmission and the other for handling flow-control. CONNECT supports wormhole switching by attaching a header to each flit of the message.

Problems with CONNECT design

Unlike traditional ASIC like NoCs, support for [Virtual-channel \(VC\)](#), large buffering, and other exotic NoC features are too expensive to implement on FPGA. This is because ASICs can support custom SRAM buffers, and crossbar arrays can be implemented directly on silicon without any intermediate configuration layer. CONNECT tried to replicate those features on an FPGA which resulted in a bloated and slow implementation of this NoC. We list some of the exotic features of this NoC here:

- **Flow Control:** The flow-control logic in this NoC used a credit-based system. In credit-based flow control a sending router keeps track of credits from its downstream receiving routers. Implementing such counters on FPGA fabric will use multi-bit adders which are inherently slow. Moreover, this design implemented a separate counters for each Virtual-Channel.
- **Optional Virtual Channels:** The way CONNECT implements virtual-channels on FPGAs increases the cost massively as each VC-buffer and its associated control adds to circuit complexity. While CONNECT allows a VC-free design, they still need FIFO-based flow control.
- **Routing strategy:** This NoC uses routing-tables implemented using LUTs to store the output port information for every possible destination in the network. This creates a major problem while scaling the NoC architecture to support large number of IOs. As large number of bits and location will be required to store the routing information in LUTs.
- **Use of Flit-Buffers:** This NoC implements a flit buffer for every input port. The NoC builds a Circular-FIFO using Distributed RAMs (LUTs) for each virtual channel in the network. Again, this requires a lot of memory which is not efficient to be built using Distributed RAMs.

In summary, due to inefficient buffer implementation this architecture is not scalable and use a lot of resources on the FPGA, around 1,500 LUTs for a 32-bit payload and runs at a very low speed of 9.6 ns.

2.2.2 Penn Split-Merge NoC

Split-Merge [\[27\]](#) is another FPGA-overlay NoC that fixes the architectural issues associated with CMU CONNECT by decoupling routing into composable split and merge blocks. The

NoC architecture implements a VC-free, handshake-based NoC. This NoC design takes advantage of highly-pipelined FPGA-friendly design units that results in building a high-speed solution and runs upto $3\times$ faster than CONNECT.

This NoC reduces the logic complexity by a large margin by employing a decentralized implementation of routing and arbitration functionality. It has programmable input and output queues that can be deeply pipelined. It uses FPGA’s inbuilt [Shift-Register LUT \(SRL\)](#) logic to implement FIFOs. The NoC can support DOR, and [West-side first \(WSF\)](#) routing algorithms and also supports Wormhole switching. The dataflow path in the NoC is streamlined where the packets pass through input queues to reach the Split/Routing stage. The Split logic looks at the header of each flit that enters the NoC and sends them to appropriate output port. The packets then goes through the output queues and reach the Arbitration/Merge stage where the merge logic interleaves messages from different input ports to the same output port while maintaining packet completeness.

Drawbacks of implementing Split-Merge on FPGA fabric

The Split-Merge NoC use deep pipelines to achieve higher clock speed and hence increase the worst case latency. Although this architecture is more FPGA-friendly as it does not use Virtual-channels and implements a hand-shake based flow-control, it still requires a large amount of buffering. Split-Merge NoC spends about 40% of its resources in buffering.

To summarize, Split-Merge NoC creates an FPGA-friendly architecture and provides a high-speed design than CONNECT. However, due to other design overheads and deep pipelining it ends up using more resources than CONNECT, 1,700 LUTs for 32-bit payload and 2-stage pipelining and runs at clock speed of 4.5 ns.

2.2.3 PaterNoster NoC

PaterNoster NoC [42] exploits the benefits of a 2-D torus topology just like Hoplite, HopliteRT and uses DOR routing policy. The switch architecture has similar logic blocks like Hoplite, HopliteRT as shown in Figure 2 in [42]. The architecture of the NoC has 3-inputs (West In, Local In, and South In) and 3-outputs (North Out, East Out, and Local Out) with an additional Corner-Turn buffer for the turning packets. It supports wormhole switching with multiple flits, however, all the flits along with the head flit are assumed to have same header that contains information about the source and destination of a flit which adds a little extra overhead but also reduces the complexity of the router of implementing flow control logic and router information storage.

Arbitration and Switching

The arbitration scheme of this NoC prioritizes *SouthIn* port over *WestIn* and *LocalIn* ports. Following DOR routing, a packet is first routed in the X-ring and when it reaches the right column it is turned to Y-ring through the Corner-Turn buffer. In case of a contention between *SouthIn* and turning *WestIn* packets to access the *NorthOut* port, *SouthIn* packet wins and *WestIn* packet gets in the Corner-Turn buffer. A packet only leaves the buffer once the *NorthOut* port is free. Also, in a case when *WestIn* and *SouthIn* packets contend to exit the switch, due to higher priority *SouthIn* wins and *WestIn* is deflected towards *EastOut* to take another round and try again.

In a situation, when the Corner-Turn buffer is full and a *WestIn* packet wants to turn, it is deflected on the *EastOut* port back to the X-ring to take another round. Additionally, the router makes sure that a new turning flit is not added to the buffer before the previously deflected flit comes back and try again to get into the buffer. This also ensures that flits from the same source leave the ring in the same order.

The downside of this NoC is that it is not optimized to exploit FPGA-architectures and uses many LUTs to implement the large amount of multiplexing logic. Also, to preserve the order of packets, each flit that takes an extra round (deflected) is marked to make sure that it is injected in the corner-turn buffer first before any other flit from source is added to the buffer. To implement the marking scheme each node uses a counter equal to the length of a round.

A 4×4 implementation of this NoC takes ≈ 750 – 1200 ALMs and supports 64-bit wide datapath with 8-deep corner-turn FIFOs whereas HopliteBuf supports a 5×5 implementation with 64-bit datapath and 64-deep FIFOs with just 400 ALMs. Also, since a counter is needed at each node within a ring, the hardware costs for PaterNoster NoC increases dramatically.

2.2.4 Kim NoC router

The **Kim NoC** router [35], uses a bi-directional 2-D torus topology with two ports at each direction (North, West, East, and South) for input and output. As shown in Figure 5 in [35], it creates separate X and Y rings like Hoplite. The router microarchitecture uses prioritized switch allocation, and crossbar partitioning to achieve high-throughput.

Arbitration and Switching

The microarchitecture uses two separate routers (sliced) - one for each dimension, X and Y to create a dimension-sliced router. The slicing of the router into two dimensions create two small crossbar switches: the x router (R_x) and the y router (R_y). The switch follows Dimension-Ordered Routing policy and hence a packet has to traverse in both dimensions (X and Y) to reach its destination and switch dimensions exactly once. Even if the source and destination share the same row or column, the packet will still need to switch as the local injection port is at R_x router and local ejection port is at R_y router. The switch microarchitecture uses buffers to decouple the two-sliced routers. It uses backpressure flow control to manage full buffers and avoid misrouting of packets. Just like HopliteBuf, it also implements turn-buffers when a packet switches from X to Y dimension and buffers a packet only once.

The arbitration scheme prioritizes *North* packets over *west* and locally injected packets. It uses a simple 2-deep turn buffer for the packets turning from X to Y and uses classic backpressure when the buffer is full. To avoid client starvation, where a client connected to local input port does not get a chance to inject packets into the network due to the presence of other high-priority packets, the switch uses a handshake based strategy.

However, the switch pays in terms of large resource costs and design complexity to implement the handshake based strategy to avoid client starvation. This topology also uses multiplexers abundantly as it is targeted as ASICs, and would not match the LUT fabric of the FPGA as well as Hoplite or HopliteBuf. Furthermore, HopliteBuf does not require backpressure flow control either as FIFOs are not allowed to go full.

2.3 Survey of Real-time NoCs

In this section, we are going to take a look at some of the existing real-time NoCs and what features make them suitable to implement mission-critical applications. Real-time NoC designs has mostly focused in two directions:

- **Time division multiplexed (TDM) NoCs** [24, 30, 43]: These NoCs require complete knowledge of the communication flows in advance and create a routing decision table upfront in each switch of the network. By doing so, they achieve a low-latency, fast packet delivery solution without any deflection or waiting time in the buffer. However, to build these static TDM scheduling tables both packet injection and routing, these NoCs require a lot of expensive storage for routing decisions and

tends to be unsuitable for NoCs that need to support both real-time flows requiring worst case guarantees and best effort flows where average case delay matters.

- **Prioritized NoCs** [34, 53]: Priority-aware network-on-chip designs incorporate virtual-channels for each priority flow in the network. Here, packets are arbitrated based on the relative priority of the constituent flow instead of requiring complete knowledge of flows. While this approach is more favorable to effectively supports clients of different criticality, it requires expensive virtual channels: at each port, every priority flow needs a separate buffer. Since a lot of buffering space is required to build these NoCs, they are unsuitable to be effectively overlay on FPGA devices.

2.4 Emergence of Hard NoCs

So far we have reviewed some of the best soft and statically analyzed network-on-chip designs. In the section, we are going to review state-of-the-art hardened NoC designs that can gain performance benefits and can extract more information from our analysis approach to enhance their NoC.

2.4.1 Xilinx Versal

In 2018, Xilinx released their next generation Versal architecture, where they added a hard NoC to FPGAs [55]. In the proposed architecture they added several components, such as, an on-chip dual-core ARM processor, multiple DDR4 memory controllers, 16-lanes PCIe Gen4, 32G SerDes, and large amounts of high-speed on-chip memory. And to satisfy the intra-chip communication requirement among these high-speed logic blocks, they added a highly configurable memory mapped network-on-chip. They also added a buffered AXI-stream based NoC inside the AI Engine sub-block to transfer the intermediate computation data among different compute units.

The proposed NoC architecture supports dataflows with different **Quality of Service (QoS)** traffic classes: **Low-latency (LL)**, **Isochronous (ISOC)**, and **Best-effort (BE)**. The different traffic classes assert a different level of priority for transporting packets from source to destination. For instance, LL traffic is treated as highest-priority and is typically requires variable bandwidth and burstiness. Cache line refills from CPU is a typical example of LL traffic class. ISOC has the same priority as LL, but packets with ISOC class have a deadline after which they are redundant. BE has the least priority and can be serviced whenever bandwidth is available.

Limited information is available about the NoC at the time this thesis was written. The paper [55] mentions that the hard NoC implements a deterministic routing with wormhole switching and it uses a packetized network. It also supports multiple VCs to avoid deadlock and head-of-line blocking. It also provide offline routing, configurable paths and weighted arbitration.

Packet communication

The packet communication in the Versal NoC architecture was supported with AXI4 [59] protocol. The paper [55] refers the source/master port as NoC Master Unit (NMU) and the destination/slave port as NoC Slave Unit (NSU). As shown in Figure 5 in [55], the master and slave interfaces show the channels required for AXI memory mapped operation. The NoC design uses a separate VC for each of these interface signals and the same process is followed to capture the response signals. It uses Request, Reply buffers in side the NoC architecture that runs at a decoupled clock frequency from the source and destination logic frequency.

Topology

As the paper explains, the topology is not mesh. However, as shown in Figure 6 in [55] there are Horizontal NoCs (HNoC) and Vertical NoCs (VNoC) which are irregularly provisioned across the entire chip to provide high-speed communication interconnect. The HNoCs are used to connect accelerators and memory controllers with the FPGA fabric while the VNoCs are used to connect Programmable FPGA fabric with each other. The NoC uses a programmable routing table that is programmed with a dedicated on-chip processor. It also implements a tree-structured peripheral bus that provides a deterministic routing path before the NoC is configured. The NoC routers support a datawidth of upto 128 and can run at a frequency of 900Mhz - 1Ghz and can achieve a raw throughput of 16 GB/sec.

2.4.2 Embedded NoC

In 2016, Mohamed et al. [4] presented an embedded hard NoC design on FPGAs to support system-level communications. They implement an embedded FPGA NoC and custom interfaces to connect this NoC design to the FPGA and I/Os as well. The NoC architecture uses a credit-based flow control and Virtual-channels to support wormhole routing. It

supports data widths of upto 150 bits and can route DDR3 data using a single port at a speed of 1.2 Ghz. This NoC supports a bandwidth of 22.5 GB/s on each link.

As shown in Figure 3 in [4], the overall network-on-chip architecture includes soft/fabric logic to append/strip metadata to/from the incoming packets and there is a custom fabric and NoC design which is hardened in the middle. The clock speed at which soft and hard-logic operates is also decoupled. The incoming translator unit takes the incoming data and appends header and other information to it and then it stores upto 4 flits of data to send to the hard custom fabric. The custom fabric serializes these flits of data at a higher clock speed and pass it to the virtual channels in the NoC. When the flits are received at the other end, the output translator strips the control headers from the data and pass it to the output nodes.

In this work, the authors mention that they over-provision the resources based on the the highest-bandwidth interface, DDR3, on an FPGA. We believe that this could lead to problems such as (1) not supporting newer applications that require even more bandwidth, DDR4 or the upcoming DDR5, (2) waste of on-chip resources for applications that do not require such a high bandwidth.

2.5 LUT organization in Intel and Xilinx devices

In this section, we cover the basic details about the architecture, features, and functionality of Look-up-tables in Xilinx and Intel FPGAs. LUTs are the basic building blocks to assemble a complex system on FPGAs. Lets first take a look at Xilinx’s LUT structure:

2.5.1 Xilinx LUT organization

With modern Xilinx FPGA devices, you can build a maximum of 6-input and 1-output function by using one 6-LUT [58]. A 6-LUT is fracturable into two 5-LUTs with five common inputs across both LUTs. This allows you to implement one function of 5-inputs (any function) and one function of 6-inputs (if it overlaps with the 5-input function) in the same LUT. Figure 2.6 shows the LUT organization of Xilinx FPGA device.

The shared East exit version of Hoplite NoC (Figure 2.2a) and HopliteRT (Figure 2.4) uses the fracturability of Xilinx’s 6-LUT architecture. In both of these designs, 5-inputs are shared and the 6th input is used to separate O5 from O6 in figure 2.6. Hoplite shares N , W , PE , and 1-bit select line each for $Emux$ and $Smux$, while HopliteRT shares N , W , PE , and 2-bit common select lines for $Emux$ and $Smux$.

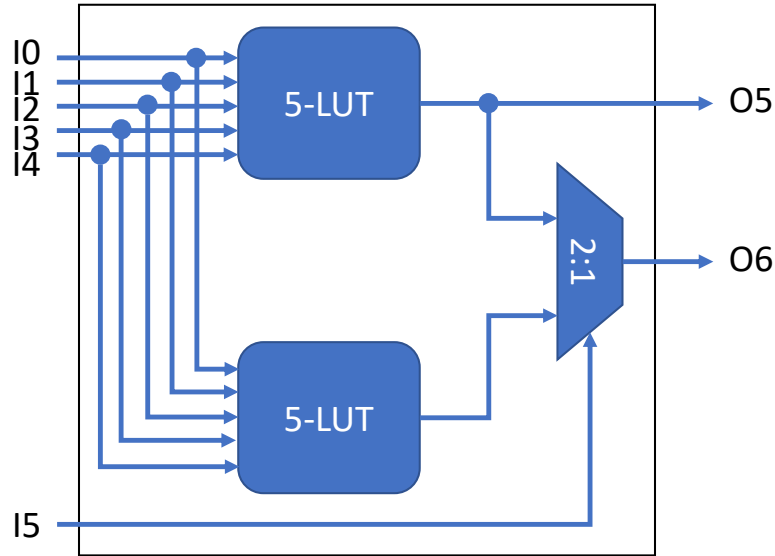


Figure 2.6: Xilinx LUT-6 structure. A fracturable Xilinx 6-LUT can fit two 5-LUTs with common inputs

Xilinx’s CLB structure is extremely flexible and allows to be configured as Storage elements, such as, Distributed RAM, and Shift-Registers [58]. Figure 2.7a shows how a LUT can be configured to work as 32-bit SRL. The 32:1 mux selects one of the 32 inputs coming from the shift-register based on the address port A. By adding a slight amount of logic on top of this module can make it work like a 32-deep 1-bit FIFO.

Multiple LUTs can be cascaded to increase the *depth* and *width* of the FIFO implemented using SRLs. One such example is shown in Figure 2.7b where two LUTs are cascaded such that the FIFO depth is 64. In a similar manner, within a SLICE, a maximum of four 32-bit SRL32 can be cascaded with the help of F7AMUX and F8AMUX to implement a 128-bit Shift-Register or FIFO.

2.5.2 Intel LUT organization

Now that we know how LUTs are organized in the Xilinx FPGA devices, lets take a look at how Intel FPGAs implement them. An Intel Adaptive Look-up Table (ALUT) is organized differently and has 8-inputs shared across two 6-LUTs [7]. Intel ALUTs are much more flexible when it comes to implementing different input size functions, Figure 2.8 shows various configurations that you can choose on a modern device like Arria-10.

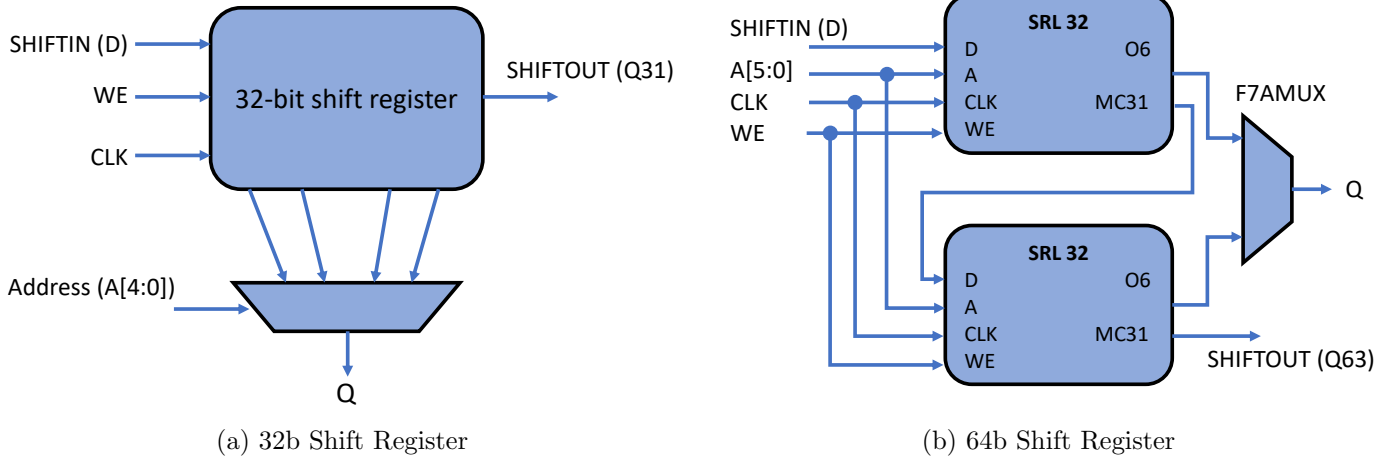
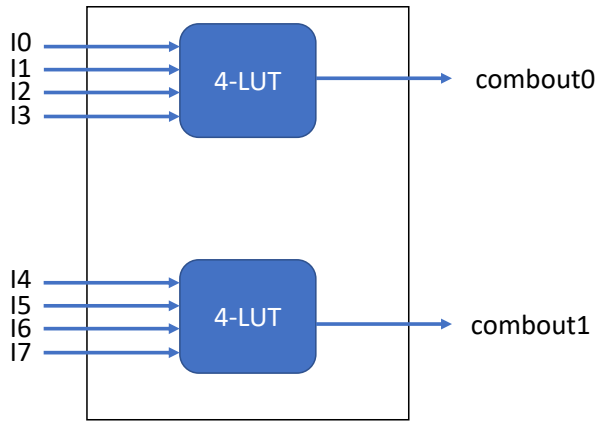


Figure 2.7: (a) A LUT-6 based 32-bit Shift Register on Xilinx FPGA (b) Two LUT-6 cascaded to implement a 64-bit Shift register. The cascading is done with the help of F7AMUX which is available in SLICEM of a Combinational Logic Block in Xilinx FPGAs

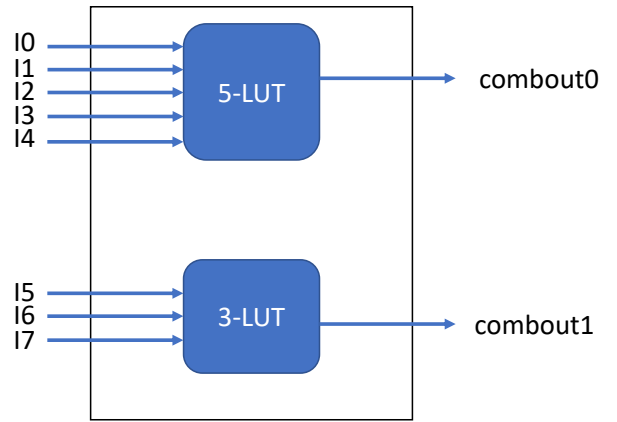
ALUTs can be configured to work in the following modes:

1. to implement a 6-input 1-output function. (only one 6-ALUT)
2. to implement two independent 4-input functions. Shown in Figure 2.8a
3. to implement one 5-input function and a separate 3-input function. Shown in Figure 2.8b
4. to implement one 5-input function and one 4-input function with one input shared. Shown in Figure 2.8c
5. to implement two 6-input functions with four common inputs and two distinct inputs each. Shown in Figure 2.8d

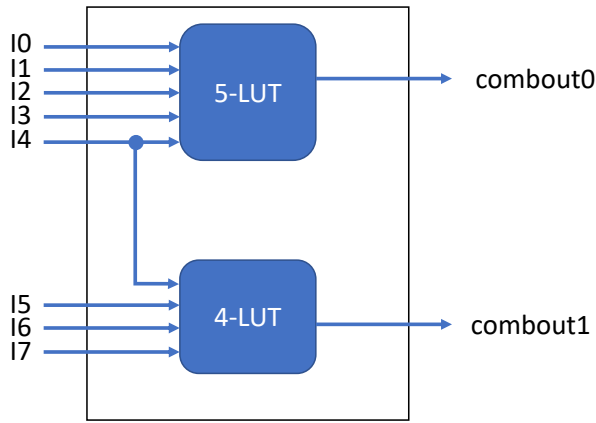
Hoplite needs one 5-input function and a separate 3-input function to implement the multiplexing logic and hence uses the Intel LUT architecture shown in figure 2.8b. While, HopliteRT needs two 5-input functions with 3 common inputs and hence it uses the Intel LUT architecture with 6-inputs shown in figure 2.8d.



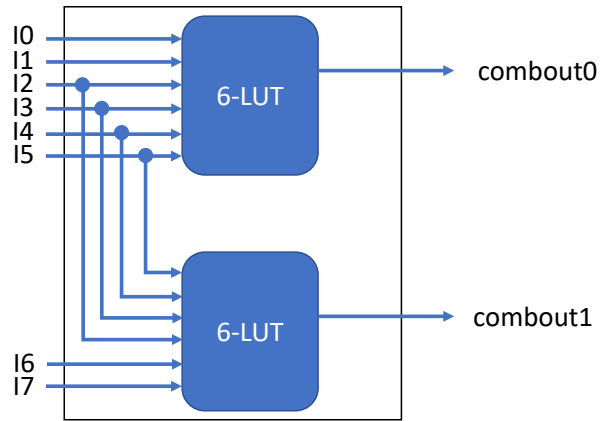
(a) Two 4-ALUTs



(b) One 5-ALUT and One 3-ALUT



(c) One 5-ALUT and One 4-ALUT with a shared input



(d) Two 6-ALUTs with four shared inputs

Figure 2.8: Various Intel ALUT configurations

Similar to what Xilinx LUTs can do to implement shift-registers and memories, Intel implements this functionality with their [Memory Logic Array Block \(MLAB\)](#) resources. MLABs are a superset of [Logic Array Block \(LAB\)](#) and support all the features that LABs can support MLABs have a slightly different architecture to build the memory configuration. Each MLAB supports a maximum of 640bits of simple dual-port SRAM. MLABs can configured to implement 32-deep and 20bit wide simple dual-port SRAMs. For more information, please refer to [\[7\]](#).

The stall-free buffer component of our new NoC designs are implemented using cheap resources like LUTs, and MLABs.

2.6 Basics of Network Regulation

We now turn our attention to some basic concepts of traffic regulation and network calculus needed to understand the static analysis section of the HopliteBuf NoC. Let us take a look at Traffic regulation first.

2.6.1 Need for Traffic Regulation

In Hoplite, *PE* port always gets the least priority than network ports *N* and *W*. What it means is that, a client waiting to inject packets into the network may be dominated profusely by other client's packets which are already travelling in the network. Hence, in the worst case [source-queueing latency](#) is also unbounded for Hoplite.

Consider an example shown in [Figure 2.9](#) to visualize this situation– We have two flows: the blue flow from PE (0,0)→ PE (3,0) is already in the network and sending packets at a 100% rate (back-to-back on every clock cycle) while the unlucky red flow at switch (1,0) is trying to enter into the network but it will never get a chance to do so because of the least priority and no open window to inject. So we can say that for Hoplite, both in-flight latency and source queueing latency is ∞ .

We can correct this by asserting a discipline on traffic injection. HopliteRT uses one such mechanism to bound the source-queueing latency by regulating traffic injection into the switch which is explained next.

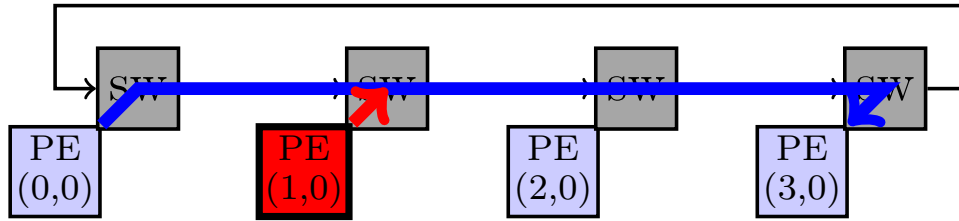


Figure 2.9: The example shows an inundated client at (1,0) that has flooded the NoC with packets at full link bandwidth (one packet per cycle)

2.6.2 Token-bucket Regulator

In networking, traffic control is the process of managing, controlling, and reducing the network traffic through network schedulers. Without effective traffic control, networks are vulnerable to congestion when the injected traffic exceeds the network capacity, leading to serious deterioration of network performance. The goal of traffic control is to achieve fairness in traffic injection and guaranteed service to each node. One such traffic control algorithm is the Token bucket regulation [38]. The regulator ensures that the data injected in the network conform to defined limits on *rate* and *burstiness*. This regulator is highly efficient and cheap to implement with just two counters. Figure 2.10 shows a high-level cartoon diagram of a Token Bucket Regulator inserted on the client \rightarrow NoC interface. The regulator restricts the amount of traffic injected into the NoC and also bounds the amount of time a client has to wait to inject a packet into the network. The regulator is user programmable with two-parameters: b , burst and ρ , rate of injection.

Two counters: rate counter, and a token counter, are required to implement token bucket regulation. Rate counter, is a free-running counter that is programmed to add a token into the token bucket, shown in Figure 2.10, when it overflows after a period of every $\frac{1}{\rho}$ cycles. Token counter, keeps track of tokens consumed by the client/user logic and also makes sure that the token bucket never exceeds its maximum size, b . The token bucket is assumed to initially full with, b tokens (user programmable) in it. Whenever a client wishes to inject a packet in the network, it checks if the NoC is ready to accept packets and client has enough tokens in the token counter. For each packet sent in the network, token counter is decremented by one which also represents that we have one empty slot in the token bucket.

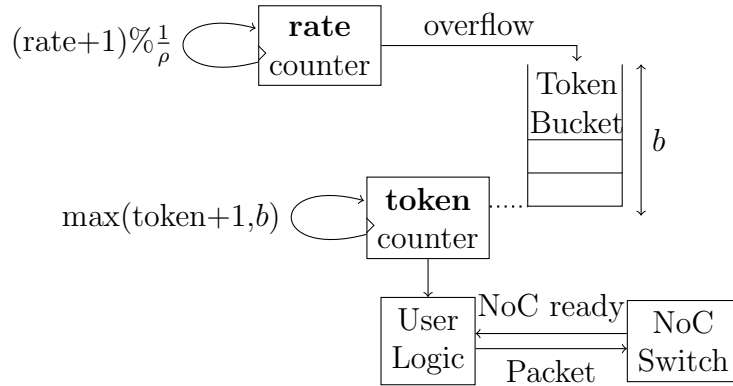


Figure 2.10: A cartoon diagram representing the Token Bucket regulation. A regulator is used at the injection port of each NoC client. Number of tokens represent the number of packets allowed to inject in the network by a client.

2.6.3 Traffic Modelling with Token bucket regulator

Lets understand Token bucket regulation with the help of an example as we use the same regulation scheme for the new HopliteBuf NoC.

Traffic curve: To analyze the traffic characteristics, we introduce a traffic curve $\lambda_{b,\rho}(t)$ to denote the maximum number of packets sent on a NoC link in any interval of t cycles. By definition, a token bucket regulator with parameters b, ρ provides a traffic curve:

$$\lambda_{b,\rho}(t) = \min(t, b + \lfloor \rho \cdot (t - 1) \rfloor)$$

Example: Traffic curve with one regulator with $b = 3, \rho = 1/4$ are depicted in Figure 2.11. The traffic curve derivation assumes that the bucket is initially full. Hence, $b = 3$ packets can be sent consecutively at times $t = 1, 2, 3$. After the first packet is sent at time $t = 1$, the regulator starts generating a new packet, which is then added to the bucket at time $1 + 1/\rho = 5$; this corresponds to the fourth transmitted packet. Afterwards, new packets are sent every $1/\rho$ cycles.

So a regulator gives all the clients a fair chance to inject packets by asserting a discipline as shown in Figure 2.11.

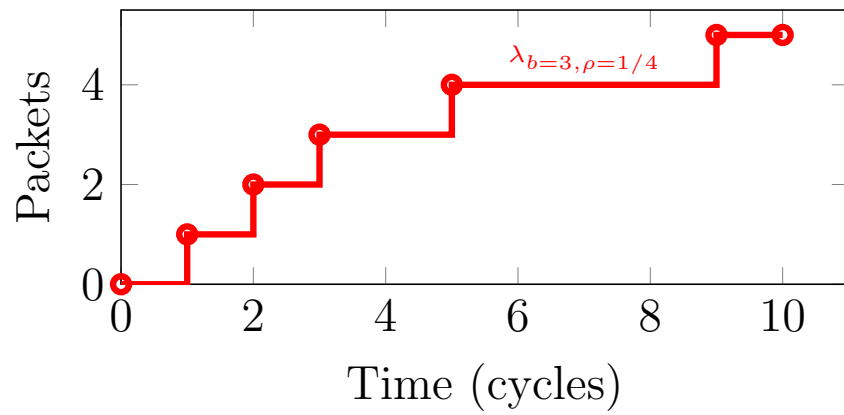


Figure 2.11: Example traffic curve for $\lambda_{b=3, \rho=1/4}$.

Chapter 3

HopliteBuf Microarchitecture

We now present the hardware design of the proposed FPGA-overlay NoC with stall-free buffering, HopliteBuf. We present two variants of HopliteBuf: $W \rightarrow S$, $W \rightarrow S + N$ and discuss the benefits and limitations of each architecture and specifically, we highlight how the $W \rightarrow S + N$ enhances the static analysis to produce concrete bounds on the FIFO sizes. We also studied the behavior of two more variants of buffer NoC designs: $W \rightarrow S$ buffer design with back-pressuring and $N \rightarrow S$ buffer design with deflections beyond those covered in the original HopliteBuf proposal 1.1. We will look at the micro-architecture of all these designs in this chapter and then we will see how they compare with each other in the Evaluation chapter. Let us start with the idea of stall-free buffering.

3.1 Stall-free Buffers

This work is targeted for real-time applications that are characterized by a need to rigorously prove timing requirements of various computing and communication blocks. We saw deflection routing NoCs implemented on FPGAs in Sections 2.1.1 and 2.1.2 which are cheap to implement (≈ 90 LUTs) and runs at a high clock speed (2.9 ns). However, the penalties these NoCs pay in terms of latency, out-of-order delivery, and throughput are due to mis-routing and deflections. HopliteRT 2.1.2 improves and bounds the worst-case latency but again, due to large number of deflections, the latency is still unacceptably large. In this work, we eliminate deflections entirely by implementing corner-turn buffers in the NoC. However, we have already seen in Sections 2.2.1 and 2.2.2 that adding buffering can lead to a bloated design with other complexities of flow control. Hence, we want to add the benefits of buffering to low-cost deflection routing with excessive resource overheads.

We use the architecture of Hoplite NoC (Figure 2.2) as our starting point and add buffers for corner turns only. Earlier in Figure 1.1, we sketched two variants of the proposed HopliteBuf topologies with buffers for turning packets. In Figure 3.1, we show the switch microarchitectures of these variants.

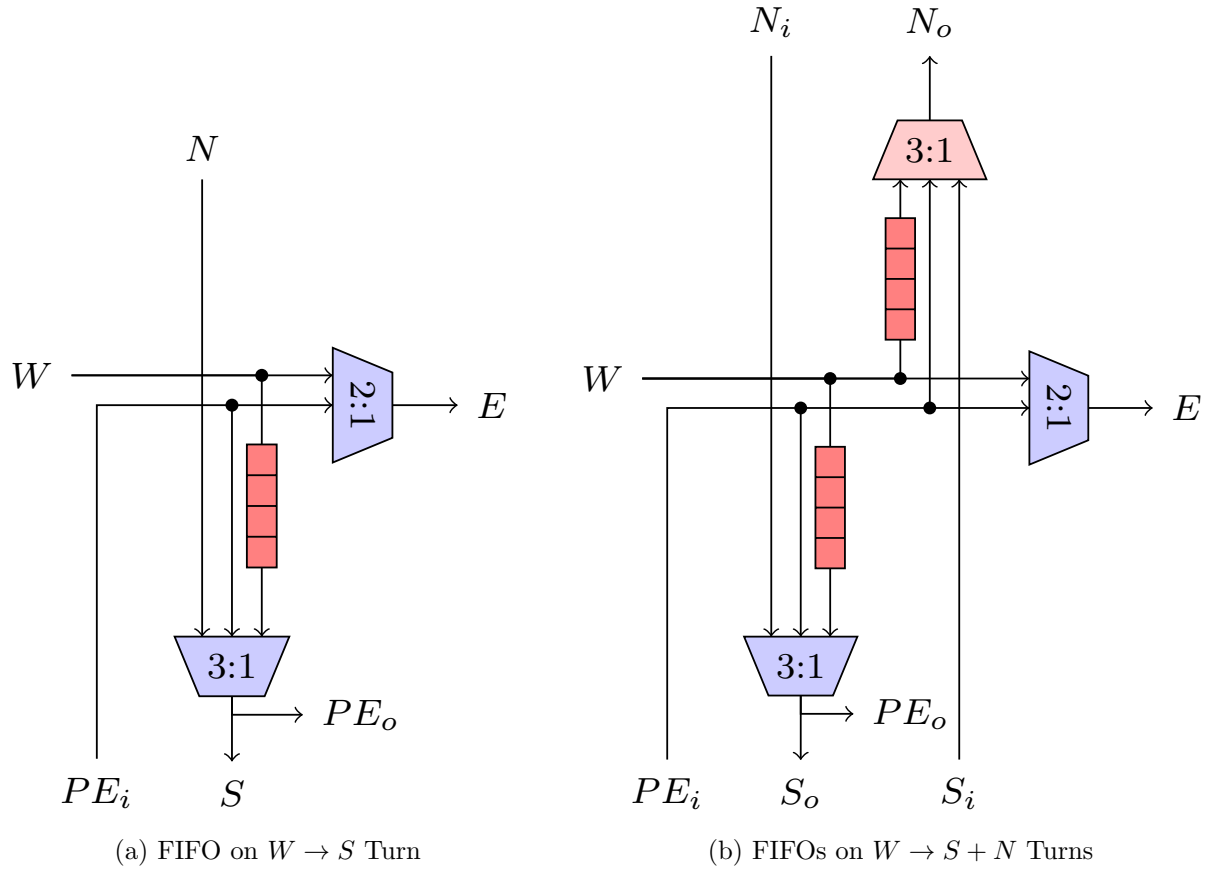


Figure 3.1: Two design alternatives for adding buffers to the Hoplite NoC router. $W \rightarrow S$ adds a buffer on the corner turn, while $W \rightarrow S + N$ adds an extra uphill buffer.

The basic multiplexing functionality implements turns to support **DOR** routing scheme. What this means is that, unlike the HopliteRT design, our proposed microarchitecture does not support the $N \rightarrow E$ turn. Now, recall that contention in Hoplite arises from packets wanting the same South (S) resource either for turns ($W \rightarrow S$) or for vertical descent ($N \rightarrow S$) and during contention, the arbitration scheme of Hoplite gave priority to ($N \rightarrow S$) while packets from ($W \rightarrow S$) gets deflected back creating potential **livelock** situation. In this NoC, we avoid livelocks by having buffers for low-priority ($W \rightarrow S$) turn

packets and do not deflect them.

Why buffer $W \rightarrow S$ packets?

Now one might argue that, why don't we put buffers on the $(N \rightarrow S)$ and give high-priority to $(W \rightarrow S)$ turn packets? Well, we can choose to make either or both of these conflicting parties wait in buffers, but the $W \rightarrow S$ option is preferred as it limits buffering penalty for a given flow to a **single** buffer. Buffering the $N \rightarrow S$ path will force packets descending vertically to wait at each hop prolonging their stay in the network. This would make both end-to-end routing latency as well as FIFO size larger than needed. And, along with that this architecture also complicates the flow analysis.

Let us take a look at an example to understand this:

- **If $N \rightarrow S$ is buffered:** Figure 3.2a shows the torus structure with an example flow from switch $(2,2) \rightarrow (1,3)$. The arbitration scheme in this example gives priority to $W \rightarrow S$ packets and $N \rightarrow S$ packets are going to be stored in the buffer during contention. If you consider the worst case assumption that at each switch in column-3 there is a contention caused by $W \rightarrow S$ packets, the example flow packet is going to take the red-path and it has to go through all the buffers in the vertical ring. This design not only makes the FIFO sizes larger, to store the same packet **multiple** times, but also adds to the latency of waiting in the buffer.
- **If $W \rightarrow S$ is buffered (advocated by this work):** Figure 3.2b shows the torus structure with the same example flow that we saw earlier from switch $(2,2) \rightarrow (1,3)$. Only this time the arbitration scheme is similar to Hoplite and gives priority to $N \rightarrow S$ packets and $W \rightarrow S$ packets are going to be stored in the buffer during contention. In this case, even in the worst case assumption that at each switch in column-3 there is a contention caused by $N \rightarrow S$ packets, the example flow packet is going to take the red-path and it needs to be stored only in a **single** buffer at switch $(2,3)$. This design is more favorable to achieve lower-latency and lower FIFO occupancy bounds. Hence, we focus only on W packets for buffering.

We now elaborate on the two design options that only buffer W packets in two ways:

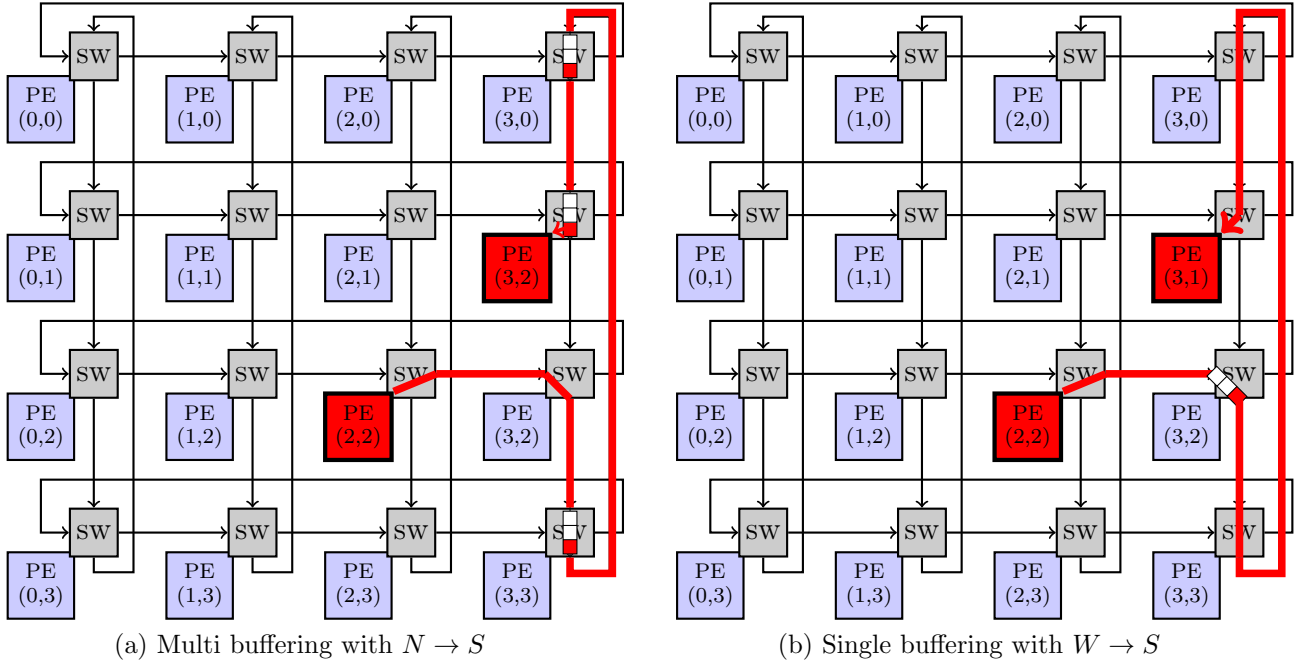


Figure 3.2: Routing scenario to show the number of buffers required to support a flow of packets from PE (2,2) \rightarrow PE (3,2). For $N \rightarrow S$ buffering case, the packet is required to be stored at multiple times in each buffer along column-3. For single-buffering case, the packet is stored only once at PE (3,2) to reach its destination.

3.2 $W \rightarrow S$ buffer design

Refer to Figure 3.1a. In this scenario, packets turning from W port to S will be buffered if a $N \rightarrow S$ packet arrives at that router in the same cycle. The routing policy now prioritizes N packets over W packets as there is no longer the option of deflecting to E like in HopliteRT. The East mux now sees W , and PE as input, and South mux sees W' (FIFO output), N , and PE as inputs. The multiplexer select lines also need to be distinct as the routing combinations prevent sharing. We discuss how this may fit on the FPGA fracturable LUT organization in the Section 3.4 and the restrictions of the routing combinations with an example in Section 3.5. From the perspective of the NoC, the packet will have to wait in a buffer **only** at the point of turn. The PE_o exit shares the same wires as the S just like in the original Hoplite and HopliteRT routers to avoid paying the extra cost of exit multiplexers. Empirical evaluation has shown negligible performance hit from this cost-saving transformation.

3.3 $W \rightarrow S + N$ buffer design

Refer to Figure 3.1b. In this second scheme, the routing policy introduces a $S \rightarrow N$ link and allows a new $W \rightarrow N$ turn. At first glance, this may seem like an unlikely design choice as inserting an entirely new routing path will increase LUT resource costs. While this is true, this scheme does **not** increase wiring requirements as seen in Figure 1.1b. The vertical wrap-around link in the original Hoplite ring is now forced to traverse through the switch on the way uphill. Thus total wirelength stays unchanged. Furthermore, as we will see later, this organization enhances the static analysis pass by removing the loopback and allows a higher provable link utilization on the vertical ring. You may notice we retain the shared single exit to the client PE_o that shares wires with the S_o port. As the vertical ring is disconnected, traffic is delivered to destination PEs only on the downhill traversal. This is another cost-saving measure that eliminates introducing an exit multiplexer along with an accompanying FIFO for packets on the uphill $S_i \rightarrow N_o$ that may wish to exit sooner.

3.4 FPGA Implementation

We saw in Section 2.5 the details about implementing deep memories and shift registers using LUTs in Xilinx devices and using MLABs in Intel devices. In this section, we are going to see how muxes and the stall-free component of our new designs are implemented using these cheap resources.

Original Hoplite Mapping: When implementing the original Hoplite router shown in Figure 2.2 on a Xilinx FPGA, we can easily fit the two 2:1 muxes in two Xilinx 5-LUTs to allow a compact 1 6-LUT mapping per bit of switching shown in Figure 2.6. This is possible as the East 2:1 mux can use a 5-LUT (requiring 3 inputs), while the South 2:1 mux can be mapped to the embedded 2:1 mux that drives the O6 output (needing two more inputs, only one of which needs to be unique). When implementing the original Hoplite router on an Intel FPGA, it is trivially possible to fit this in a single ALM with two 6-LUTs without even forcing the East mux serialization shown in Figure 2.8d. This is because, the 3:1 mux (South mux if implemented fully) needs 5 inputs while the 2:1 mux only needs 3. Two of these inputs are shared by both muxes (W and PE_i), while the distinct mux select inputs can be supplied to the two 6-LUTs independently without violating the common input restriction.

Mapping $W \rightarrow S$ buffer Design: This design requires the switching crossbar to consume four packet inputs: N , W , W' (FIFO output) and PE_i inputs along with 3

mux-select inputs. This already exceeds the 6-input LUT capacity of the Xilinx FPGA and cannot use fracturing (shown in Figure 2.6. On the Intel FPGA, however, we require four common inputs to each 6-LUT (shown in Figure 2.8d; this constraint is satisfied by our design thereby enabling a compact fit. Additionally, we can supply two unique inputs to each 6-LUT which is adequate to support the mux-select signals. When it comes to implementing deep FIFOs using this cheap resources, both Xilinx and Intel FPGAs provide configuration to use LUT and ALMs as Memory registers. To implement deeper FIFOs with Xilinx, we use the cascading strategy explained in Section 2.5.1 where multiple LUTs can be cascaded with each other by using F7AMUX and F8AMUX. With Intel devices, deeper memories can be built much easily by just instantiating multiple MLAB primitives explained in [7].

Mapping $W \rightarrow S + N$ buffer Design This design requires the switching crossbar to consume six packet inputs: N_i , S_i , W , W' ($W \rightarrow S$ FIFO), W'' ($W \rightarrow N$ FIFO) and PE_i and 5 mux-select inputs. We choose to split the turning packets into separate FIFOs to prevent mutual interference between traversing flows. The total distributed RAM capacity stays same as it just split into two SRL32 or MLAB instantiations instead of a longer single distributed RAM block. Here, with limited opportunity for input sharing, the resulting design is larger in LUT cost, but as we will see later, this allows efficient use of the NoC links. In this design as well, we implement the memory components using LUTs/MLABs as in the $W \rightarrow S$ buffer design.

Table 3.1 and 3.2 shows the resource costs of implementing different variants of HopliteBuf on Xilinx and Intel FPGAs. Maximum FIFO depth analyzed in this work is capped at 128, to reasonably implement them using LUTs and ALMs. As show in the table, the $W \rightarrow S$ design is slightly more expensive than HopliteRT on Xilinx FPGAs due to their LUT architecture. However, $W \rightarrow S$ design uses same number of ALMs to implement the logic as HopliteRT but is a little expensive in terms of buffer implementation using MLABs. In the dual-buffer variant, $W \rightarrow S+N$, the memory costs are identical to $W \rightarrow S$ but it is slightly more expensive due to an added 3:1 mux to implement the upward logic. On an average, HopliteBuf is around 3 – 4× more expensive than HopliteRT but as you will see in the evaluation section that this added cost is going to pay in terms of performance gain over HopliteRT.

3.5 Routing Policy

Now that we understand the architecture and implementation details of the two variants of HopliteBuf, $W \rightarrow S$ and $W \rightarrow S + N$, now we are going to look at the arbitration scheme

Table 3.1: Resource utilization on a Xilinx Virtex-7 FPGA for different Data Width and FIFO sizes.

| Xilinx | | | | | | |
|------------------------|------------------|-----|--------------|-----|----------------|-----|
| | HopliteRT | | W → S | | W → S+N | |
| | LUTs | FFs | LUTs | FFs | LUTs | FFs |
| DW=32, FIFO=64 | 59 | 86 | 197 | 95 | 262 | 142 |
| DW=64, FIFO=64 | 91 | 150 | 325 | 159 | 413 | 238 |
| DW=32, FIFO=128 | 59 | 86 | 281 | 96 | 346 | 144 |
| DW=64, FIFO=128 | 91 | 150 | 482 | 160 | 562 | 240 |

Table 3.2: Resource utilization on a Intel Arria-10 FPGA for different Data Width and FIFO sizes.

| Intel | | | | | | |
|------------------------|------------------|-----|--------------|-----|----------------|-----|
| | HopliteRT | | W → S | | W → S+N | |
| | LUTs | FFs | LUTs | FFs | LUTs | FFs |
| DW=32, FIFO=64 | 92 | 102 | 140 | 248 | 156 | 278 |
| DW=64, FIFO=64 | 156 | 166 | 221 | 408 | 246 | 438 |
| DW=32, FIFO=128 | 92 | 102 | 224 | 331 | 261 | 445 |
| DW=64, FIFO=128 | 156 | 166 | 360 | 555 | 409 | 733 |

and routing policies to implement these structures.

The original Hoplite and HopliteRT routers implemented bufferless deflection routing rooted in [Dimension-Ordered Routing](#) (DOR) policy. The policy ensured that arriving packets from W and N ports were sent to E and S ports respectively. For turning packets, Hoplite prioritizes N port over the W port thereby introducing the possibility of [livelock](#), while HopliteRT prioritizes W over N to ensure bounded NoC routing delays. Thus, HopliteRT deviates from DOR by allowing a $N \rightarrow E$ deflection that is not permitted under standard DOR implementation.

HopliteBuf: $W \rightarrow S$ design: For $W \rightarrow S$ design we restore [DOR](#) routing policy as we move back to the same architecture (with an added buffer on W to S link) as Hoplite. But in order to service FIFO packets, the routing policy has been modified. We list the routing combinations for $W \rightarrow S$ NoC design in [Table 3.3](#).

Table 3.3: DOR Routing Policy for FIFO-W router. PE_i always has the least priority. S exit is shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port. Extra combination of $W \rightarrow E$ and $W' \rightarrow S$ also supported.

| Packet paths | | | | | Muxsel | | FIFO |
|-------------------|--------------------|-------------------|--------------------|--------------------|--------|-----|------|
| $W \rightarrow E$ | $W' \rightarrow S$ | $N \rightarrow S$ | $PE \rightarrow E$ | $PE \rightarrow S$ | Smx | Emx | read |
| | | x | | | 00 | - | |
| | x | | | | 01 | - | 1 |
| x | | | | | - | 0 | |
| | | | x | | - | 1 | |
| | | | | x | 10 | - | |
| x | | x | | | 00 | 0 | |
| | | x | x | | 00 | 1 | |
| x | | | | x | 10 | 0 | |
| | x | x | | | 00 | - | 0 |
| | x | | x | | 01 | 1 | 1 |
| x | x | | | | 01 | 0 | |

The upper half of the routing table shows the possible flow combinations when only one-port, either W , W' , N or PE_i , has packets to transmit. The lower half shows more complicated routing decision cases when multiple ports are willing to transmit simultaneously.

For South Mux, the arbitration scheme gives the highest priority to N port, as it is not buffered and cannot hold the packets. The second-highest priority is given to W' (West FIFO output) while PE_i gets the least priority. For East mux, W port has higher priority than PE_i packets. The decision logic also takes of a unique and complex routing case of simultaneous data transfer between $W \rightarrow E$ and $W' \rightarrow S$.

The example in Figure 3.3 illustrates how the HopliteBuf $W \rightarrow S$ design fits in the torus. The example also shows how two flows from PE (0,0) \rightarrow PE (3,2) and PE (3,3) \rightarrow (3,1) contend to access the S port at switch (3,0). The arbitration scheme gives priority to N packets (blue flow) and hence red flow goes in the buffer. Note that there is no flow-control logic implemented in the switch and hence, static analysis of all the flows in the network is needed to get accurate bounds on the buffer sizes so that they never overflow and become stall-free. In this work, we provide the tools to analyze these flows and get the buffer bounds on each switch in the network.

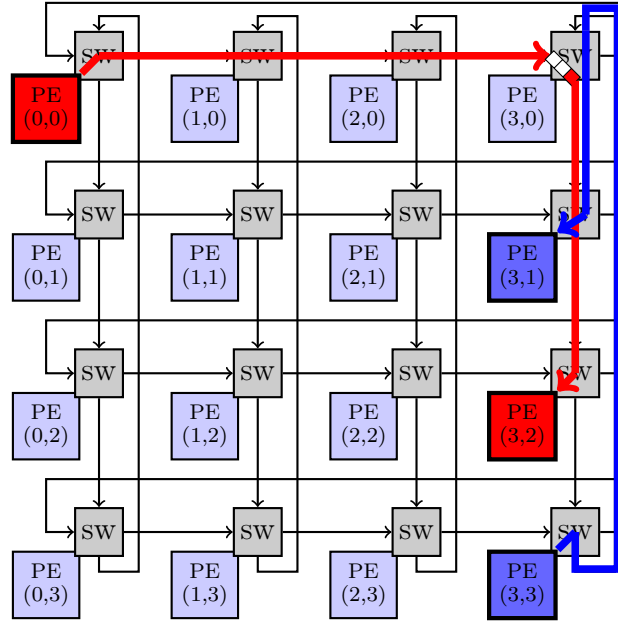


Figure 3.3: Flow of traffic in the 2-D torus with HopliteBuf $W \rightarrow S$ switches. In case of contention between W and N packets, W goes in the buffer and N gets access to S . There is no backpressuring among the switches; and therefore, buffer sizing is done by static analysis of flows.

HopliteBuf: $W \rightarrow S + N$ design: For $W \rightarrow S + N$ design as well we use [DOR](#) routing policy. With buffering, W packets are forced to wait in the while thereby transferring priority to N_i and S_i packets. This design still accept PE_i packets with the least priority. We list the routing combinations for $W \rightarrow S + N$ NoC design in [Table 3.4](#).

For HopliteBuf: $W \rightarrow S + N$ design, the routing logic is much complex in comparison to $W \rightarrow S$ design as it has 8 possible flow combinations and all have different priority structures. The table summarizes all the possible combinations along with their priority.

There is another example in [Figure 3.4](#) illustrates how the HopliteBuf $W \rightarrow S + N$ design fits in the torus. The example also shows how two flows from $PE (0,0) \rightarrow PE (3,2)$ and $PE (3,3) \rightarrow PE (3,1)$ travel in the network with a new upward path. The exit port PE_o is retained at S_o and hence if a packet wishes to exit at a switch above it, then it has to travel all the way up take the upper turn-around link and then enter the downward chain to exit at S_o port. The arbitration scheme gives priority to N_i and S_i packets (blue flow)

Table 3.4: DOR Routing Policy for FIFO- $W \rightarrow S+N$ router. PE_i again has the least priority. S_o exit shared with port PE_o exit. $W' \rightarrow S$ uses WFIFO read port, $W'' \rightarrow N$ uses NFIFO read port.

| | | | | | | | | | | | | Muxsel | | | | FIFO | |
|-------------------|--------------------|---------------------|-------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|-------|-------|--------|-------|-------|-------|-------|--|
| Packet paths | | | | | | | | | | | | Smx | Emx | Nmx | readS | readN | |
| $W \rightarrow E$ | $W' \rightarrow S$ | $W'' \rightarrow N$ | $N \rightarrow S$ | $Si \rightarrow N$ | $PE \rightarrow E$ | $PE \rightarrow S$ | $PE \rightarrow N$ | $PE \rightarrow S$ | $PE \rightarrow N$ | Smx | Emx | Nmx | readS | readN | | | |
| | | | x | | | | | | | 00 | - | - | | | | | |
| | x | | | | | | | | | 01 | - | - | 1 | | | | |
| | | | | x | | | | | | - | - | 00 | | | | | |
| | | x | | | | | | | | - | - | 01 | | 1 | | | |
| x | | | | | | | | | | - | 0 | - | | | | | |
| | | | | | | | | | x | - | - | 10 | | | | | |
| | | | | | | | | | | 10 | - | - | | | | | |
| | | | | | | | | x | | - | 1 | - | | | | | |
| x | | | x | | | | | | | 00 | 0 | 00 | | | | | |
| x | | x | | | | | | | | 00 | 0 | 01 | | 1 | | | |
| x | | | x | | | | | | x | 00 | 0 | 10 | | | | | |
| | | | x | | | | | | | 00 | 1 | 00 | | | | | |
| | | x | | | | | | | | 00 | 1 | 01 | | 1 | | | |
| x | x | | | x | | | | | | 01 | 0 | 00 | | 1 | | | |
| x | x | | | | | | | | | 01 | 0 | 01 | | 1 | | | |
| x | | | | | | | | | | 01 | 0 | 10 | | 1 | | | |
| | | | | | | | | | | 01 | 1 | 00 | | 1 | | | |
| | | | | | | | | | | 01 | 1 | 01 | | 1 | | | |
| x | | | | | | | | | | 10 | 0 | 00 | | | | | |
| x | | | | | | | | | | 10 | 0 | 01 | | 1 | | | |

and hence red flow goes in the buffer. Note that there is no flow-control logic implemented in the switch and hence, static analysis of all the flows in the network is needed to get accurate bounds on the buffer sizes so that they never overflow and become stall-free. We will show in the next chapter how this design linearizes the analysis and helps achieving more concrete buffer bounds.

To study the effect of backpressuring and deflections on the buffered NoC designs, we created two more variants of HopliteBuf: $W \rightarrow S$ buffer design with backpressuring, and $N \rightarrow S$ buffer design with deflections. Let us take a look at their architecture and arbitration scheme.

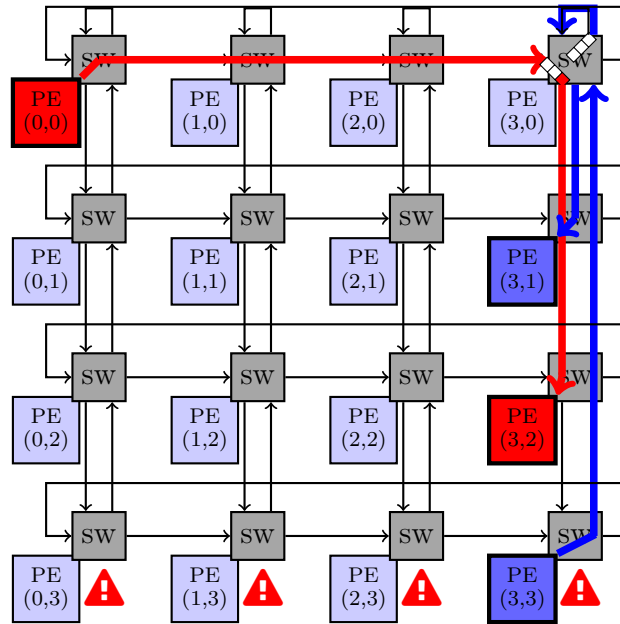


Figure 3.4: Flow of traffic in the network with HopliteBuf $W \rightarrow S + N$ switches. The torus does not have vertical links going from bottom to top. Each switch has a dedicated upward path. A packet trying to exit at a node above it will have to go all the way and then come down to exit as shown in blue flow from switch $(3,3) \rightarrow (3,1)$.

3.6 $W \rightarrow S$ Buffer Design with Backpressure

The idea of this design is similar in concept to Kim NoC Router [35] where backpressure is generated when the corner-turn buffer becomes full.

The microarchitecture of this NoC is similar to HopliteBuf $W \rightarrow S$ design with an added *Backpressure* unit as shown in Figure 3.5a. The backpressure logic ensures that when the FIFO is full or there is an input backpressure coming from the next switch in the horizontal ring, then it generates an output backpressure to hold the packets in the horizontal chain or/and at PE port. The backpressure logic is highly optimized and uses only one register to hold the data in the case when backpressure is generated and the data from the previous node has already entered the switch.

The priority is still at the N port, same as $W \rightarrow S$ HopliteBuf design, and W port has higher priority than PE . As the NoC is simply backpressuring the switches or clients in the horizontal ring, it doesn't affect the flit order delivery and we still get in-order packet delivery. We will evaluate and compare the latency and logic cost of this router with the proposed NoC design in the evaluation section.

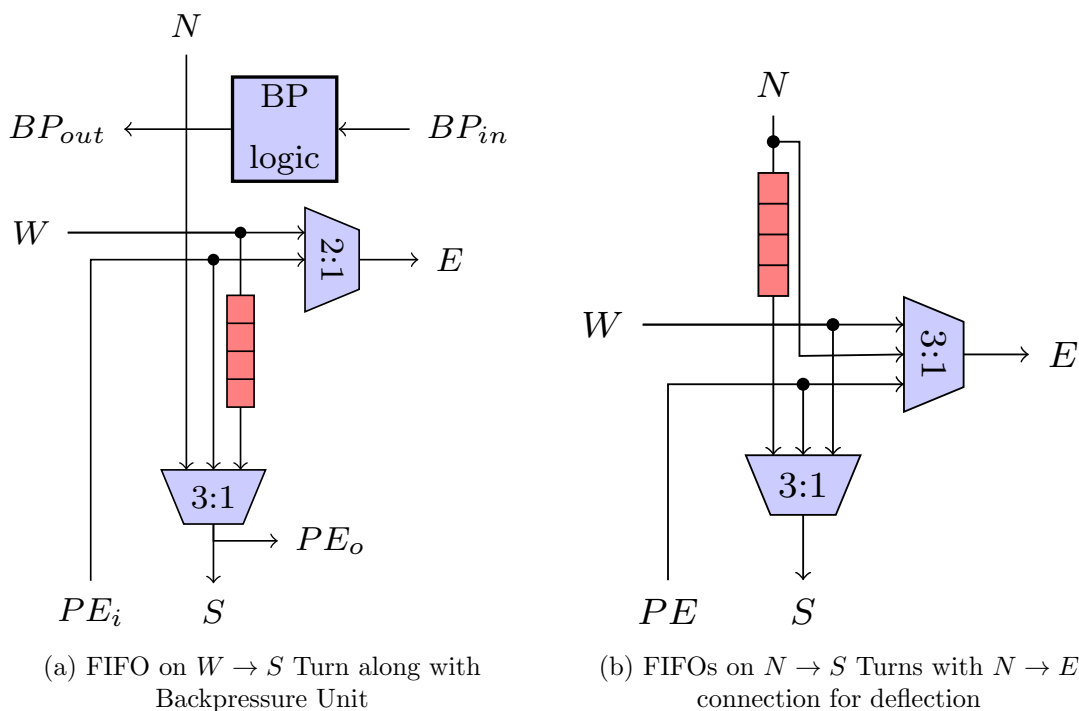


Figure 3.5: Buffered NoC designs with backpressuring and deflections

3.7 $N \rightarrow S$ Buffer Design with Deflections

The idea of this design is somewhat similar in concept to Pater Noster NoC [42] where in both designs the packets are deflected/misrouted in case the FIFO becomes full. However, the design and arbitration is entirely different from Pater Noster NoC. Unlike Pater Noster, which deflects W packets to E , we deflect N packets to E same as HopliteRT [57] in the event when $N \rightarrow S$ buffer becomes full. Now, one might argue that we could have added the buffer on the $W \rightarrow S$ turn and have deflected packets from $W \rightarrow E$ in case FIFO becomes full, however, this would have created the [livelock](#) problem as we saw with Hoplite [31] if the N port dominates the router with continuous packets.

The microarchitecture of this NoC is similar to HopliteRT with an added buffer on $N \rightarrow S$ link as shown in Figure 3.5b. The arbitration scheme is also the same as HopliteRT which prioritizes W packets over N and PE , and PE port has the least priority. The routing policy forces N packets to go E in case FIFO becomes full and W is not accessing E port. In the event of W going E and FIFO is full, N is simply moved into the FIFO as W is not accessing the S port and hence FIFO can be read at the same cycle as it is being loaded with a new N packet.

By deflecting $N \rightarrow E$ in the event of FIFO going full, it is noted that the deflected packet comes back at the high-priority W port and is guaranteed to have progress towards its destination. Hence, we can say that this design reduces to HopliteRT when FIFO becomes full. We will look at the performance comparison of this NoC design with other designs in the evaluation section.

Chapter 4

Network Calculus and Analysis

We now turn our attention to static analysis of the NoC traffic to bound buffer sizes and worst-case injection (source queueing) and in-flight traversal latencies. This is important to establish whether we can realize these buffers in distributed FPGA RAMs (SRLs and MLABs). We first introduce our regulation and traffic model. We then develop a network calculus approach to FIFO size and worst-case latency analysis for HopliteBuf. The presence of cycles in the torus topology make this analysis susceptible to instability, but we are able to provide an analytic solution that employs a topology linearization alternative (Figure 1.1b) to eliminate cycles and get accurate buffer bounds and latencies.

4.1 Client Traffic Regulation

Injection regulation is a known technique from network calculus to establish well-defined behavior of network traffic at runtime for off-chip internet-scale systems. We adapt token bucket regulation, explained in the background section 2.6.2 for use in an on-chip context at the NoC clients to enforce traffic discipline on the NoC. This is done transparently and the datapath design just needs to obey the standard NoC valid-ready interface (AXI-stream). We can implement this regulation on the FPGA using two simple counters per NoC client and require **no** buffers at the client-NoC interface. The regulator is programmed with a rate ρ and burst b that reflects the communication requirements of the application. At run-time, the regulator maintains a token counter. A packet can only be injected if the NoC is ready (no other packet is blocking the client) and there is at least one token in the counter; the token is consumed upon sending the packet. New tokens are generated

and added to the counter at a rate ρ , provided that the counter has not saturated to its maximum value of b tokens.

4.2 Traffic Model

Definition 1 Traffic curve: We revisit the concept of Traffic curves to analyze the traffic characteristics. We represent a traffic curve $\lambda_{b,\rho}(t)$ to denote the maximum number of packets sent on a NoC link in any interval of t cycles. By definition, a token bucket regulator with parameters b, ρ provides a traffic curve:

$$\lambda_{b,\rho}(t) = \min(t, b + \lfloor \rho \cdot (t - 1) \rfloor) \quad (4.1)$$

An extended version of Figure 2.11 is shown in Figure 4.1 that shows traffic curves for two regulators with $b = 2, \rho = 1/4$ and $b = 3, \rho = 1/4$ (the arrival curve γ will be introduced in Section 4.4).

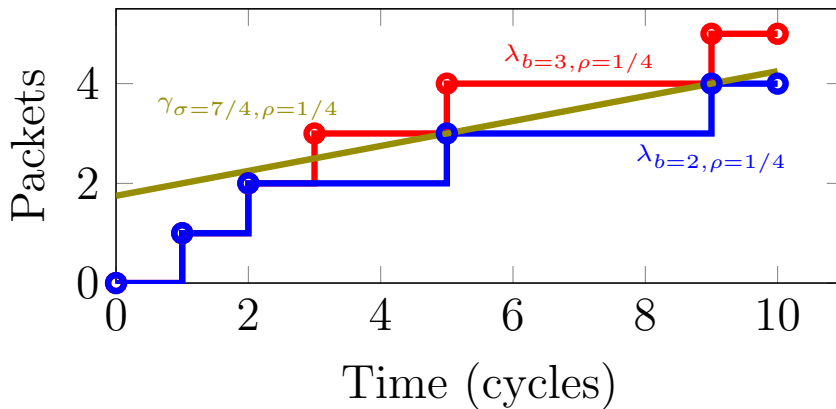


Figure 4.1: Example traffic curves for $\lambda_{b=3, \rho=1/4}$ and $\lambda_{b=2, \rho=1/4}$ along with an arrival curve for $\gamma_{b=7/4, \rho=1/4}$.

In this analysis, we consider an $(m \times m)$ matrix of clients (x, y) . Each client sends packets as part of one or more flows; all packets within the same flow have the same destination and use the same token bucket regulator. Hence, we use $F = \{f_1, \dots, f_i, \dots\}$ to denote the set of flows in the system, where for each flow f : $(f.xs, f.ys)$ represents the source client of the flow; $(f.xd, f.yd)$ represents the destination client; and $f.b, f.\rho$ represent the regulator parameters. Note that two different flows f_i and f_j might share the same source, or the same destination.

Flows entering from different ports into a switch might affect the traffic injection rate by the client at that switch. To determine the affect, we need to analyze the combined traffic rate of all the flows entering the switch. To calculate the combined traffic load we consider two traffic flows, λ_{b_1, ρ_1} and λ_{b_2, ρ_2} entering a node from two different ports and directed to the same output. Lemma 1 adapted from [57] defines an operator \oplus that combines the two traffic curves to compute a tight bound on the resulting aggregated traffic.

Lemma 1 *Let λ_{b_1, ρ_1} and λ_{b_2, ρ_2} bound the traffic on input ports (West, North or PE) directed to the same output port (East or South). Then the traffic on the output port is bounded by the following curve:*

$$(\lambda_{b_1, \rho_1} \oplus \lambda_{b_2, \rho_2})(t) = \min(t, b_1 + b_2 + \lfloor \rho_1 \cdot (t - 1) \rfloor + \lfloor \rho_2 \cdot (t - 1) \rfloor). \quad (4.2)$$

in general terms,

$$\oplus \mathcal{A}(t) = \min \left(t, b(\mathcal{A}) + \sum_{\forall \lambda_{b, \rho} \in \mathcal{A}} \lfloor \rho \cdot (t - 1) \rfloor \right) \quad (4.3)$$

Deriving Conflicting Flows Γ^C : Following Lemma 1, on how to combine traffic curves for multiple flows, the next step is to define the set of **conflicting flows**, that is, those flows that block the injection of packets at the analyzed client. It comprises:

- all other flows injected by the same source client, since a client can inject only one packet per cycle. For the example shown in Figure 4.2, Table 4.1 shows that there is no conflicting flows for flow f_1 since no other flows are originating from source (0,1). However, f_3 becomes a conflicting flow for flow f_2 since f_3 is originating from the same source at f_2 .
- Second, we need to add to Γ^C all the flows generated by other clients that traverse the same mux used by f at its source router ($f.xs, f.xs$). If f injects packets to the East port, then it suffers conflicts from any flow $W \rightarrow S$. If f injects packets to the South port, then it suffers conflicts from flow $W' \rightarrow S$ or $N \rightarrow S$. For the example shown in Figure 4.2, Table 4.1 shows that f_2 is injecting packets to East port and the East traversing f_1 is conflicting with it.

Example: We present a running example of a NoC with five flows $f_{1...5}$ using the $W \rightarrow S$ buffer design in Figure 4.2. Note that we use f'_i to denote a flow after it leaves a buffer, as buffering can increase the burstiness of the flow (packets queued up in a buffer can be flushed directly back-to-back). Relevant flow parameters are tabulated in Table 4.1. We discuss how to apply the analysis to the $W \rightarrow S + N$ design in Section 4.6 as the flows

have been linearized and have no loops. For the $W \rightarrow S$ design, the analysis is harder due to the loopback of the vertical ring. The instability created by loopbacks is a notoriously challenging problem in network calculus [38] and results in lower provable bounds on link utilization. We analyze the unique problem formulation presented by the HopliteBuf torus network and propose a technique for deriving these bounds and improving link utilization through linearization of the vertical ring.

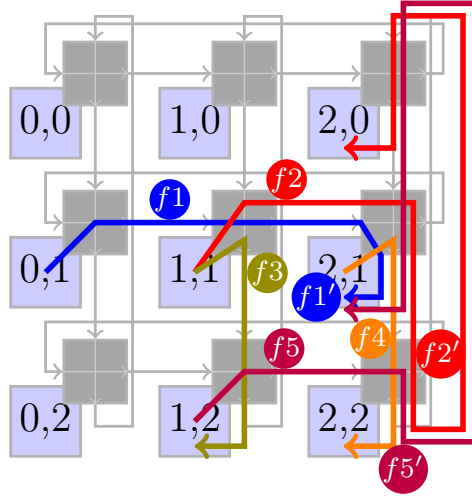


Figure 4.2: Example $W \rightarrow S$ NoC design with five flows $f_{1..5}$.

Table 4.1: Flow parameters for the example NoC. Γ^C are the conflicting flows used in Section 4.3; $f_{W \rightarrow S}$ and $f_{N \rightarrow S}$ are the $W \rightarrow S$ and $N \rightarrow S$ interfering flows used in Section 4.4. ‘-’ denotes not applicable.

| flow | source | dest | Γ^C | $f_{W \rightarrow S}$ | $f_{N \rightarrow S}$ |
|-------|--------|-------|--------------------|-----------------------|-----------------------|
| f_1 | (0,1) | (2,1) | none | f_2 | f'_5 |
| f_2 | (1,1) | (2,0) | f_3, f_1 | f_1 | f'_5 |
| f_3 | (1,1) | (1,2) | f_2 | - | - |
| f_4 | (2,1) | (2,2) | f'_1, f'_2, f'_5 | - | - |
| f_5 | (1,2) | (2,1) | none | none | $f'_2 + f_4$ |

Our analysis derives three sets of parameters:

- Injection latency $Injection(f)$ for each flow $f \in F$; this is the maximum time that the source client $(f.xs, f.ys)$ can be stalled waiting to send a packet of f .

- Maximum queuing delay $Delay(f)$ for each turning flow f .
- Backlog for each router; this is the maximum number of packets that are queued waiting to be transmitted (excluding the packet that might be transmitted in the current clock cycle).

4.3 Injection Latency

Once the set of conflicting flows Γ^C has been derived, we can now compute the maximum delay suffered by a client (x, y) to inject a sequence of k packets of flow f , where $k \leq f.b$. Assume that each flow in Γ^C is bounded by a traffic curve $\lambda_{b,\rho}(t)$; we define $b(\Gamma^C)$ as the sum of burstiness parameters b of traffic curves for all flows in Γ^C , and $\rho(\Gamma^C)$ as the sum of their rate parameters.

In any window of time of length t , by definition there must be at least $t - \oplus\Gamma^C(t)$ free clock cycles, that is, when client is free to inject upto clock cycles $t - \oplus\Gamma^C(t)$ packets and is not obstructed by conflicting flows.

To compute the upper bound on packet injection latency, we first see Lemma 2 and 3 from HopliteRT [57] which proves that a client is guaranteed to receive free cycles for injection, if $\rho(\Gamma^C) < 1$ and also that the number of free cycles is given by equation 4.4 shown below:

$$t - \oplus\Gamma^C(t) \geq \max(0, \lfloor (t - (T^s + 1)) \cdot (1 - \rho(\Gamma^C)) \rfloor + 1) \quad (4.4)$$

where,

$$T^s = \left\lceil \frac{b(\Gamma^C)}{1 - \rho(\Gamma^C)} \right\rceil. \quad (4.5)$$

This implies that the flow might receive no free cycles for T^s clock cycles, but is then guaranteed to receive slots at a rate of $1 - \rho(\Gamma^C)$. Using Lemma 2 and 3 from [57], Theorem 1 adapted from the same work [57], proves that the first packet in the sequence waits for at most $\lceil 1/f.\rho \rceil - 1 + T^s$ cycles; successive packets are sent either every $1/f.\rho$ or every $1/(1 - \rho(\Gamma^C))$ cycles, whichever is higher.

Theorem 1 *Assume $\rho(\Gamma^C) < 1$ and the client wishes to inject a sequence of $k \leq f.b$ packets for flow f . Then the delay to inject all packets in the sequence is upper bounded by:*

$$\lceil 1/f.\rho \rceil - 1 + T^s + \left\lceil (k - 1) \cdot \max\left(\frac{1}{f.\rho}, \frac{1}{1 - \rho(\Gamma^C)}\right) \right\rceil. \quad (4.6)$$

For simplicity, we assume $k = 1$ with $f.b = 1$. Hence, from Equation 4.6, the maximum time that a source client can be stalled waiting to send a packet is represented by:

$$\text{Injection}(f) = \lceil 1/f.\rho \rceil - 1 + \left\lceil \frac{b(\Gamma^C)}{1 - \rho(\Gamma^C)} \right\rceil \quad (4.7)$$

It remains to determine the traffic curve $\lambda_{b,\rho}(t)$ for each interfering flow. For a flow f_i that has not yet traversed a buffer, the curve is simply $\lambda_{f.b,f.\rho}(t)$. We show how to derive the traffic curve for a flow f'_i that leaves a $W \rightarrow S$ buffer in the next section.

4.4 Vertical Ring Analysis $W \rightarrow S$ Design

We now analyze the behaviour of flows turning on a vertical ring through a $W \rightarrow S$ buffer. We employ the theory of network calculus [38] for FIFO-arbitrated flows to derive deterministic bounds on queuing delay and backlog. In particular, we show that the delay and backlog depend on the burstiness and rate of flows entering the FIFO buffer, as well as the burstiness and rate of flows routed $N \rightarrow S$. To apply the theory, we need to introduce a new type of curve.

Definition 2 *Leaky bucket arrival curve:* A flow f is said to be bounded by a leaky bucket arrival curve $\gamma_{\sigma,\rho}(t)$ if the number of packets transmitted by the flow in any time interval t is bounded by:

$$\gamma_{\sigma,\rho}(t) = \sigma + \rho \cdot t.$$

In this case, $f.\sigma$ and $f.\rho$ are arrival curve parameters for the flow.

Luckily, we can convert between traffic curves of the form $\lambda_{b,\rho}(t)$ and arrival curves $\gamma_{\sigma,\rho}(t)$ according to the following lemma (a formal proof is provided in Lemma 2):

- to convert $\lambda_{b,\rho}(t)$ into $\gamma_{\sigma,\rho}(t)$, we set $\sigma = b - \rho$;
- to convert $\gamma_{\sigma,\rho}(t)$ into $\lambda_{b,\rho}(t)$, we set $b = \lceil \sigma + \rho + 1 \rceil$.

Lemma 2 (1) A flow bounded by traffic curve $\lambda_{b,\rho}(t)$ is also bounded by arrival curve $\gamma_{b-\rho,\rho}(t)$.
(2) Similarly, a flow bounded by arrival curve $\gamma_{\sigma,\rho}(t)$ on any NoC link is also bounded by traffic curve $\lambda_{\lceil \sigma + \rho + 1 \rceil, \rho}(t)$.

Proof: Part (1). Based on the curve definitions, we have:

$$\begin{aligned}\lambda_{b,\rho}(t) &= \min(t, b + \lfloor \rho \cdot (t - 1) \rfloor) \\ &\leq b + \rho \cdot (t - 1) = b - \rho + \rho \cdot t = \gamma_{b-\rho,\rho}(t).\end{aligned}$$

Part (2). Again by definition:

$$\begin{aligned}\gamma_{\sigma,\rho}(t) &= \sigma + \rho \cdot t = \sigma + \rho + \rho \cdot (t - 1) \leq \sigma + \rho + \lfloor \rho \cdot (t - 1) \rfloor + 1 \\ &\leq \lceil \sigma + \rho + 1 \rceil + \lfloor \rho \cdot (t - 1) \rfloor.\end{aligned}$$

Since furthermore a NoC link cannot transmit more than one packet every clock cycle, the flow is bounded by:

$$\min(t, \lceil \sigma + \rho + 1 \rceil + \lfloor \rho \cdot (t - 1) \rfloor) = \lambda_{\lceil \sigma + \rho + 1 \rceil, \rho}(t).$$

Example: Refer again to Figure 4.1. The traffic curve $\lambda_{b=2,\rho=1/4}(t)$ is upper bounded by $\gamma_{\sigma=7/4,\rho=1/4}(t)$. Similarly, arrival curve $\gamma_{\sigma=7/4,\rho=1/4}(t)$ is upper bounded by $\lambda_{b=\lceil 7/4+1/4+1 \rceil,\rho=1/4}(t) = \lambda_{b=3,\rho=1/4}(t)$; $\gamma_{\sigma=7/4,\rho=1/4}(t) > \lambda_{b=3,\rho=1/4}(t)$ for $t = 1, 2$, but since the NoC link cannot send more than one packet per cycle, $\lambda_{b=3,\rho=1/4}(t)$ is still a valid traffic bound. In essence, Lemma 2 allows us to “convert” a flow with a traffic curve $\lambda_{b,\rho}(t)$ into an arrival curve $\gamma_{\sigma,\rho}(t)$ and vice-versa, albeit at some loss of precision.

Finally, there are situations where we need to aggregate (combine) flows transmitted on the same link; for example, flows f'_2 and f_4 entering router (2, 2) from N . Note that for two arrival curves $\gamma_{\sigma',\rho'}(t)$ and $\gamma_{\sigma'',\rho''}(t)$, it immediately holds that $\gamma_{\sigma',\rho'}(t) + \gamma_{\sigma'',\rho''}(t) = \gamma_{\sigma'+\sigma'',\rho'+\rho''}(t)$; hence, the arrival curve for the aggregate of flows traversing the same link can be expressed by summing the σ and ρ parameters of the arrival curves for the individual flows.

Let us take a look at the equations for **Backlog and Delay** calculation at a switch:

Figure 4.3 illustrates the flows required for analysis at one NoC router. Here, f and f' represent a flow under analysis before and after leaving the $W \rightarrow S$ buffer; $f_{W \rightarrow S}$ represents the aggregate of all other interfering flows traversing the buffer; $f_{N \rightarrow S}$ represents the aggregate of all interfering flows traversing the router in the $N \rightarrow S$ direction; and $f_{PE \rightarrow S}$ represents the aggregate of all flows injected by the client at that router directly S . As discussed in Section 3.5, the S mux arbitration gives lowest priority to the client; hence, we do not have to consider flow $f_{PE \rightarrow S}$ when analyzing flow f , but it will interfere in the $N \rightarrow S$ direction on the next router. Regarding the other flows, $f_{N \rightarrow S}$ has higher priority than f , while $f_{W \rightarrow S}$ and f are FIFO scheduled as they traverse the same buffer.

Assuming that each flow is described by an arrival curve, we now introduce some propositions and corollaries from [38] that will help us derive the equations for Backlog and Delay:

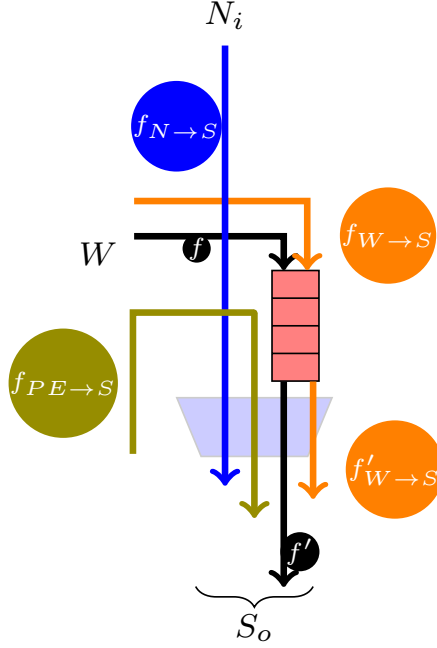


Figure 4.3: Flows through a $W \rightarrow S$ router.

Proposition 1 : (Proposition 1.10 in [38]) Consider a constant bit rate server, with rate C , serving two flows, H and L , where H is of high-priority than L . If flow H is $\gamma_{\sigma,\rho}$ -smooth, with $\rho < C$, then the low priority flow, L , is guaranteed a rate-latency service curve with rate $C - \rho$ and latency $\frac{\sigma}{C-\rho}$.

Corollary 1 : (Corollary 6.3 [38]) Consider a node serving two flows, 1 and 2 in FIFO order. Assume that flow i is constrained by one leaky bucket with rate ρ_i and burstiness σ_i . Assume that the node guarantees to the aggregate of the two flows a rate latency service curve $\beta_{R,T}$. If $\rho_1 + \rho_2 < R$, then flow 1 has a service curve equal to the rate latency function with rate $R - \rho_2$ and latency $T + \frac{\sigma_2}{R}$ and at the output, flow 1 is constrained by one leaky bucket with rate ρ_1 and burstiness σ'_1 with

$$\sigma'_1 = \sigma_1 + \rho_1 \left(T + \frac{\sigma_2}{R} \right) \quad (4.8)$$

Since we know that flow $f_{PE \rightarrow S}$ does not affect the backlog and delay at a particular switch, we are going to ignore $f_{PE \rightarrow S}$ from figure 4.3 in our calculations.

Following Proposition 1, we can now generate a service curve for the combined flows (f and $f_{W \rightarrow S}$) on W port as shown in Figure 4.4. α is the arrival curve for the flows on W and $\beta_{R,T}$ is the service curve that is delayed due to the obstruction caused by high-priority $N \rightarrow S$ packets.

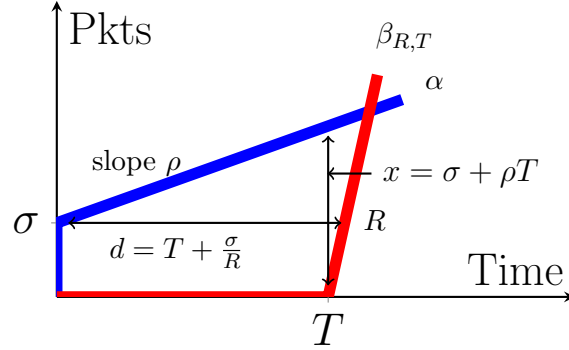


Figure 4.4: Arrival (α) and Service ($\beta_{R,T}$) curve that shows the delay (d) and rate (R) for the aggregate flow on W port when obstructed by the high-priority $N \rightarrow S$ flow. x denotes the backlog (amount of buffering) on $W \rightarrow S$.

x denotes the **backlog**, σ and ρ denotes the burstiness and rate of the aggregate ($f + f_{W \rightarrow S}$) flows on the W port, respectively. Backlog and Delay calculation follow Theorems 1.4 and 1.5 in [38]. We can now calculate Backlog as:

From Proposition 1,

$$T = \frac{f_{N \rightarrow S} \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho}, \quad (4.9)$$

$$R = 1 - f_{N \rightarrow S} \cdot \rho, \quad (4.10)$$

and backlog is given as $x = \sigma + \rho T$, here, σ and ρ is the total burstiness and rate of flow f and $f_{W \rightarrow S}$.

hence,

$$\text{Backlog} = f \cdot \sigma + f_{W \rightarrow S} \cdot \sigma + (f \cdot \rho + f_{W \rightarrow S} \cdot \rho) \cdot \frac{f_{N \rightarrow S} \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho} \quad (4.11)$$

Since buffering does not increase the **rate** of flows (Corollary 1), Equation 4.12 shows that rate of flow before and after buffering is the same. And, we can compute the burstiness of the flow from the buffer output, $f' \cdot \sigma$, using Equation 4.8 from Corollary 1:

$$f' \cdot \rho = f \cdot \rho \quad (4.12)$$

$$f' \cdot \sigma = f \cdot \sigma + f \cdot \rho \cdot \frac{f_{N \rightarrow S} \cdot \sigma + f_{W \rightarrow S} \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho} \quad (4.13)$$

under the condition that $f \cdot \rho + f_{N \rightarrow S} \cdot \rho + f_{W \rightarrow S} \cdot \rho < 1$ (that is, the link is not saturated).

To compute the delay for flow f , $Delay(f)$ we need to separate flow f from $W \rightarrow S$ flow and hence we can make another service curve $\beta_{R',T'}$ using Proposition 1.

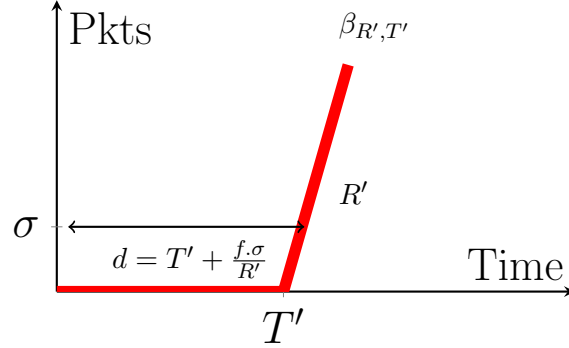


Figure 4.5: Service curve ($\beta_{R',T'}$) that extracts flow f from the aggregate flow ($f + f_{W \rightarrow S}$) $\beta_{R,T}$ shown in Figure 4.4. d is the queueing delay for turning flow, f .

Again from Proposition 1,

$$T' = \frac{f_{N \rightarrow S} \cdot \sigma + f_{W \rightarrow S} \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho} \quad (4.14)$$

$$R' = 1 - f_{N \rightarrow S} \rho - f_{W \rightarrow S} \rho \quad (4.15)$$

Substituting values of T' and R' in the delay expression, d , we get:

$$Delay(f) = \frac{f \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho - f_{W \rightarrow S} \cdot \rho} + \frac{f_{N \rightarrow S} \cdot \sigma + f_{W \rightarrow S} \cdot \sigma}{1 - f_{N \rightarrow S} \cdot \rho} \quad (4.16)$$

under the condition that $f \cdot \rho + f_{N \rightarrow S} \cdot \rho + f_{W \rightarrow S} \cdot \rho < 1$ (that is, the link is not saturated).

Based on Equation 4.12, buffering does not increase the rate of flows. Furthermore, based on Lemma 2, for any flow f_i that has not been buffered, including flow f , we have $f_i \cdot \sigma = f_i \cdot b - f_i \cdot \rho$. Hence, the only unknowns in Equation 4.13 are the values $f'_i \cdot \sigma$ for flows that have crossed a buffer. To analyze the system, we thus apply the so-called Time Stopping Method in network calculus [38]: we treat the values $f'_i \cdot \sigma$ as variables, and write a system of linear equations by applying Equation 4.13 to each flow that enters a given vertical ring. If the values of $f'_i \cdot \sigma$ obtained by solving the system of equations are valid (that is, bounded and positive), then $\gamma_{f'_i \cdot \sigma, f'_i \cdot \rho}(t)$ upper bounds flow f'_i . Otherwise, the network cannot be analyzed.

Example: Let us consider flows at switch (2,2) from Figure 4.2. $f_1.\rho + f_2.\rho + f_5.\rho < 1$. For flow f_1 , $f_{W \rightarrow S}$ comprises flow f_2 , while $f_{N \rightarrow S}$ comprises flow f'_5 . Since for any flow $f_i.\sigma = f'_i.\sigma$ and $f_i.\sigma = f_i.b - f_i.\rho$, we obtain:

$$f'_1.\sigma = f_1.b - f_1.\rho + f_1.\rho \cdot (f'_5.\sigma + f_2.b - f_2.\rho)/(1 - f_5.\rho).$$

Similarly, applying Equation 4.13 to flows f_2, f_5 under the added assumption $f_2.\rho + f_4.\rho + f_5.\rho < 1$ yields:

$$\begin{aligned} f'_2.\sigma &= f_2.b - f_2.\rho + f_2.\rho \cdot (f'_5.\sigma + f_1.b - f_1.\rho)/(1 - f_5.\rho), \\ f'_5.\sigma &= f_5.b - f_5.\rho + f_5.\rho \cdot (f'_2.\sigma + f_4.b - f_4.\rho)/(1 - f_2.\rho - f_4.\rho). \end{aligned}$$

Hence, we solve a linear system of three equations to determine the value of variables $f'_1.\sigma, f'_2.\sigma, f'_5.\sigma$, which can then be used to determine the backlog at each router and delay for each flow according to Equations 4.11, 4.16. Furthermore, by applying Lemma 2, we derive equivalent traffic curves $\lambda_{\lceil f'_i.\sigma + f'_i.\rho + 1 \rceil, f'_i.\rho}(t)$ for f'_1, f'_2 and f'_5 , which we use to bound the injection latency of f_4 . As an example, if we set $b = 1, \rho = 1/4$ for all regulators, we obtain $f'_1.\sigma = f'_2.\sigma = 33/20$, and $f'_5.\sigma = 39/20$, which result in backlogs of $14/5$ at (2, 1) and $39/20$ at (2, 2). Hence, we need a minimum $W \rightarrow S$ buffer size of $\lceil 14/5 \rceil + 1 = 3$ at (2, 1) and $\lceil 39/20 \rceil + 1 = 2$ at (2, 2); note we add 1 to the buffer size to account for a packet being read from the buffer and transmitted in the current clock cycle.

Despite this, it is known [38, 8] that the circular dependencies introduced by a ring design can reduce the sustainable (provable) per-link utilization of the network by up to 50%. We explain cyclic dependencies in $W \rightarrow S$ buffer design of HopliteBuf in the next section.

4.5 Cyclic Dependencies in HopliteBuf $W \rightarrow S$ design

Figure 4.6 shows an example flow in the rightmost column of torus shown in Figure 4.2 where each switch wants to communicate with the switch above. We have three flows: f_1 from (2,0) \rightarrow (2,2), f_2 from (2,1) \rightarrow (2,0), and f_3 from (2,2) \rightarrow (2,1). Flow f'_i represents the output flow from the buffer at each switch.

We can recreate the set of linear equations for these flows to get the bounds on buffer sizes in the same way as we did in the example above. Using Equation 4.13, we can compute the output burstiness for each of these flows (under the same assumption that the aggregate rate of flows at a switch doesn't exceed 100%) as follows:

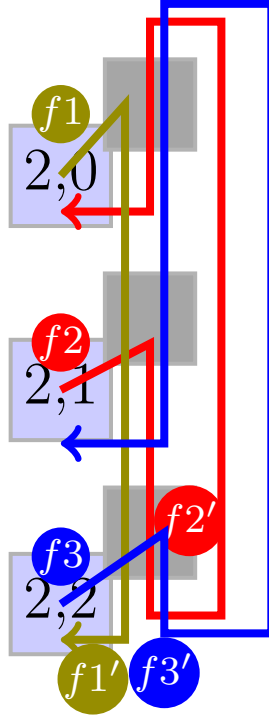


Figure 4.6: Cyclic dependency flow example in $W \rightarrow S$ HopliteBuf: rightmost column.

$$f'_1 \cdot \sigma = f_1 \cdot \sigma + f_1 \cdot \rho \cdot \frac{f'_2 \cdot \sigma + f'_3 \cdot \sigma}{1 - f_2 \cdot \rho - f_3 \cdot \rho} \quad (4.17)$$

$$f'_2 \cdot \sigma = f_2 \cdot \sigma + f_2 \cdot \rho \cdot \frac{f'_1 \cdot \sigma + f'_3 \cdot \sigma}{1 - f_1 \cdot \rho - f_3 \cdot \rho} \quad (4.18)$$

$$f'_3 \cdot \sigma = f_3 \cdot \sigma + f_3 \cdot \rho \cdot \frac{f'_1 \cdot \sigma + f'_2 \cdot \sigma}{1 - f_1 \cdot \rho - f_2 \cdot \rho} \quad (4.19)$$

$$(4.20)$$

Let us assume, $f_1 \cdot \sigma = f_2 \cdot \sigma = f_3 \cdot \sigma = \sigma$ and $f_1 \cdot \rho = f_2 \cdot \rho = f_3 \cdot \rho = \rho$. We can re-write the above equations as,

$$f'_1 \cdot \sigma = \sigma + \rho \cdot \frac{f'_2 \cdot \sigma + f'_3 \cdot \sigma}{1 - 2\rho} \quad (4.21)$$

$$f'_2 \cdot \sigma = \sigma + \rho \cdot \frac{f'_1 \cdot \sigma + f'_3 \cdot \sigma}{1 - 2\rho} \quad (4.22)$$

$$f'_3 \cdot \sigma = \sigma + \rho \cdot \frac{f'_1 \cdot \sigma + f'_2 \cdot \sigma}{1 - 2\rho} \quad (4.23)$$

because, $\rho \leq \frac{1}{3}$ (aggregare sum of rate should be < 1) we can use Corollary 6.2 [38], and the above equations can be represented as,

$$S \leq A \cdot S + K \quad (4.24)$$

where, A is a non-negative matrix and K is non-negative vector represented as,

$$S = \begin{bmatrix} f'_1 \cdot \sigma \\ f'_2 \cdot \sigma \\ f'_3 \cdot \sigma \end{bmatrix}, A = \begin{bmatrix} 0 & \frac{\rho}{1-2\rho} & \frac{\rho}{1-2\rho} \\ \frac{\rho}{1-2\rho} & 0 & \frac{\rho}{1-2\rho} \\ \frac{\rho}{1-2\rho} & \frac{\rho}{1-2\rho} & 0 \end{bmatrix}, K = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \end{bmatrix}$$

Assuming that the spectral radius of A is less than 1. In that case the power series $I + A + A^2 + A^3 + \dots$ converges and is equal to $(I - A)^{-1}$, where I is a identity matrix of same size as A . Since A is a non negative matrix, $(I - A)^{-1}$ is also non-negative; thus, we can multiply equation 4.24 to the left by $(I - A)^{-1}$ and obtain:

$$S \leq (I - A)^{-1} \cdot K \quad (4.25)$$

Defining $\eta = \frac{\rho}{1-2\rho}$ and using some linear algebra we get

$$(I - A)^{-1} = \begin{bmatrix} \frac{1-\eta^2}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} \\ \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{1-\eta^2}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} \\ \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{\eta^2+\eta}{1-3\eta^2-2\eta^3} & \frac{1-\eta^2}{1-3\eta^2-2\eta^3} \end{bmatrix}$$

If $0 \leq \eta < \frac{1}{2}$ then $(I - A)^{-1}$ is positive. Hence,

$$\begin{aligned} 0 &\leq \frac{\rho}{1-2\rho} < \frac{1}{2} \\ \text{or, } 0 &\leq \rho < 0.25 \end{aligned} \quad (4.26)$$

It is evident from Equation 4.26 that the maximum per flow injection rate has to be less than 25%, making a combined injection rate supported to be less than 75% while we should be able to support 100%.

Example: We analysed the same example flow design with our analysis tool to see what is the maximum rate supported before our analysis becomes unstable:

The file to generate the flows looks like this:

```
//filename: flows_ws.dat
//command: python analyze.py -N 3 -f flows_ws.dat

sX, sY, dX, dY, B, R
1, 0, 2, 2, 1, 0.24000
1, 1, 2, 0, 1, 0.24000
1, 2, 2, 1, 1, 0.24000
```

where, sX , sY , dX , dY are the X and Y coordinates for source and destination. B , R are the burst and rate of the individual flows. What we observed with our analysis tool is that the design becomes unstable after 24% injection rate and burstiness of 1. This clearly states that due to cyclic dependencies, we cannot support 33% per flow injection from each of these nodes which is clearly possible. This is a weakness of analysis and not the architecture itself.

Now we will look at the linearized HopliteBuf $W \rightarrow S+N$ Design which breaks these dependencies and helps us getting concrete bounds on the buffer sizes.

4.6 Linearized Analysis: $W \rightarrow S + N$ Design

The analysis for the $W \rightarrow S + N$ design proceeds in a similar manner, but is much simpler as no vertical loopback exists. The same injection latency computation is performed, albeit the set Γ^C can be different compared to the $W \rightarrow S$ design since a flow that was conflicting on the S mux could now turn N instead. Similarly, the same conditions in Equations 4.11, 4.12, 4.13, 4.16 can be applied after decoupling each router in two parts: a south component containing the $W \rightarrow S$ buffer and S mux, and a north component containing the $W \rightarrow N$ buffer and N mux. Since packets are transmitted in different directions for the two components, when writing the equation for the north components we use flows $f_{S \rightarrow N}$ and $f_{W \rightarrow N}$ in place of $f_{N \rightarrow S}$ and $f_{W \rightarrow S}$.

Example: Figure 4.7 shows the resulting decomposition for the rightmost column of the flow set depicted in Figure 4.2. Note that the topmost router (2,0) only implements the south component, as no flow can be injected north at (2,0). The sets of conflicting flows Γ_f^C and interfering flows $f_{N \rightarrow S}, f_{W \rightarrow S}, f_{S \rightarrow N}, f_{W \rightarrow N}$ are provided in Table 4.2. Compared to the $W \rightarrow S$ design, the number of conflicting and interfering flows is reduced.

Table 4.2: Conflicting and interfering flows for the $W \rightarrow S + N$ design. ‘-’ denotes not applicable, as the flow is not buffered in that direction.

| flow | Γ^C | $f_{W \rightarrow S}$ | $f_{N \rightarrow S}$ | $f_{W \rightarrow N}$ | $f_{S \rightarrow N}$ |
|-------|--------------|-----------------------|-----------------------|-----------------------|-----------------------|
| f_1 | none | none | f'_5 | - | - |
| f_2 | f_1, f_3 | - | - | none | f'_5 |
| f_4 | f'_1, f'_5 | - | - | - | - |
| f_5 | none | - | - | none | none |

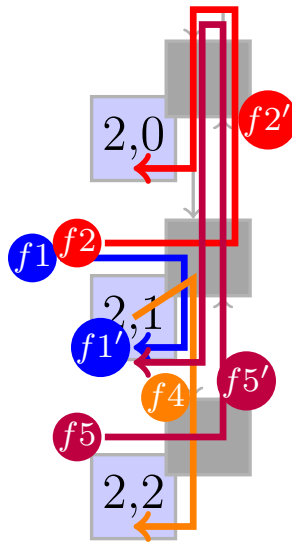


Figure 4.7: $W \rightarrow S + N$ design example: rightmost column.

When compared to the $W \rightarrow S$ design, we do not need to solve a system of equations to compute the $f'_i \cdot \sigma$ values: since the $W \rightarrow S + N$ design disconnects vertical rings, we can apply Equation 4.13 to flows with destinations on a column x by ordering the flows based on the router at which they turn, in the order of packet propagation: from $(x, m - 1)$ to $(x, 1)$ for flows turning N , and then from $(x, 0)$ back to $(x, m - 1)$ for flows turning S . As long as no link is saturated, it is guaranteed that the analysis will compute bounded delay and backlog. Let us see this with the help of the same example as $W \rightarrow S$ design reproduced with $W \rightarrow S + N$ design (shown in Figure 4.8)

Using Equation 4.13, we can compute the output burstiness for each of these flows as follows:

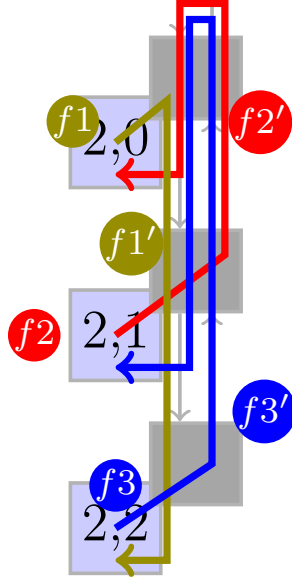


Figure 4.8: Solving cyclic dependency with linearization in $W \rightarrow S + N$ design example: rightmost column.

$$f'_1 \cdot \sigma = f_1 \cdot \sigma + f_1 \cdot \rho \cdot \frac{f'_2 \cdot \sigma + f'_3 \cdot \sigma}{1 - f_2 \cdot \rho - f_3 \cdot \rho} \quad (4.27)$$

$$f'_2 \cdot \sigma = f_2 \cdot \sigma + f_2 \cdot \rho \cdot \frac{f'_3 \cdot \sigma}{1 - f_3 \cdot \rho} \quad (4.28)$$

$$f'_3 \cdot \sigma = f_3 \cdot \sigma \quad (4.29)$$

By assuming, $f_1 \cdot \sigma = f_2 \cdot \sigma = f_3 \cdot \sigma = \sigma$ and $f_1 \cdot \rho = f_2 \cdot \rho = f_3 \cdot \rho = \rho$ and using Corollary 6.2 [38], the above set of equations can be represented as,

$$S \leq A \cdot S + K \quad (4.30)$$

where, A is non-negative matrix and K is non-negative vector represented as,

$$S = \begin{bmatrix} f'_1 \cdot \sigma \\ f'_2 \cdot \sigma \\ f'_3 \cdot \sigma \end{bmatrix}, A = \begin{bmatrix} 0 & \frac{\rho}{1-2\rho} & \frac{\rho}{1-2\rho} \\ 0 & 0 & \frac{\rho}{1-\rho} \\ 0 & 0 & 0 \end{bmatrix}, K = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \end{bmatrix}$$

Again, since A is a non negative matrix, $(I - A)^{-1}$ is also non-negative; thus, we can multiply equation 4.30 to the left by $(I - A)^{-1}$ and obtain:

$$S \leq (I - A)^{-1} \cdot K \quad (4.31)$$

Defining $\eta = \frac{\rho}{1-2\rho}$ and using some linear algebra we get

$$(I - A)^{-1} = \begin{bmatrix} 1 & \eta & \eta^2 + \eta \\ 0 & 1 & \eta \\ 0 & 0 & 1 \end{bmatrix}$$

If $\eta \geq 0$ then $(I - A)^{-1}$ is positive. Hence,

$$\begin{aligned} \frac{\rho}{1-2\rho} &> 0 \\ \text{or, } 0 &\leq \rho < 0.5 \end{aligned} \quad (4.32)$$

It is evident from equation 4.32 that as long as no link is saturated, this guarantees that the analysis computes bounded delay and backlog for $W \rightarrow S + N$ design. For this case we get a bound of FIFO sizes if the per flow rate of injection does not exceed 33%.

Example: We analysed the same example again with our analysis tool to see what is the maximum rate supported with the dual buffer design before our analysis becomes unstable:

The file to generate the flows looks the same as before:

```
//filename: flows_wsn.dat
//command: python analyze_dualbuffer.py -N 3 -f flows_wsn.dat

sX, sY, dX, dY, B, R
1, 0, 2, 2, 1, 0.33000
1, 1, 2, 0, 1, 0.33000
1, 2, 2, 1, 1, 0.33000
```

where, sX , sY , dX , dY are the X and Y coordinates for source and destination. B , R are the burst and rate of the individual flows. This time we observed that we support a maximum injection rate of 33% before our analysis becomes unstable. Hence, not only is the dual buffer design eliminates cyclic dependencies, it also supports higher injection rate and enhances the static analysis as well.

The analysis tool is open-source: <https://git.uwaterloo.ca/watcag-public/hoplitebuf-bounds>

Chapter 5

Evaluation

We present the performance measurement results for our FPGA optimized NoC and associated results from static analysis. We are interested in understanding the worst case NoC routing latency properties, its breakdown, buffer depth bounds, as well as routing coverage. We also want to confirm the properties of static analysis bounds and understand their impact of distributed FPGA RAM mapping costs. We show results for 5×5 NoCs to retain narrative consistency, but can generate other RTL networks and bounds for other sizes as well. We study four synthetic workloads which are commonly used in the real-time systems community:

- We use **ALL-TO-ONE** pattern that gets all NoC clients to target a same NoC address (destination PE (0,0)): a shared resource like an external DRAM, PCIe, or Network port.
- We use **ALL-TO-ROW** pattern that gets all NoC clients to target a same row in the torus (destination row 0).
- We use **ALL-TO-COLUMN** pattern that gets all NoC clients to target a same column in the torus (destination column 0).
- We also use synthetic uniform **RANDOM** traffic pattern that is expressed a set of flows, *i.e.* flowsets. We evaluate the NoCs using 100 separately-generated synthetic flowsets. Each flowset is a collection of m^2 distinct streaming flows. Each flow captures data communication between a source-destination pair of clients. All flows have the same burstiness and rate which is increased until the links saturate.

5.1 RTL Simulation Results

We first examine the results (feasibility, latency, FIFO sizing) of cycle-accurate RTL simulations of the different NoCs.

5.1.1 Flowset Feasibility

For our designs, we cap the maximum FIFO occupancy at 128 to enable low-cost realizations. As a result some combination of flowset communication pattern and injection rate ρ will likely be infeasible. If any FIFO ever goes full, we classify that configuration as not feasible. We want to know what fraction of our 100 randomly-generated flowsets were able to route without any of the NoC FIFOs every going full at a given rate.

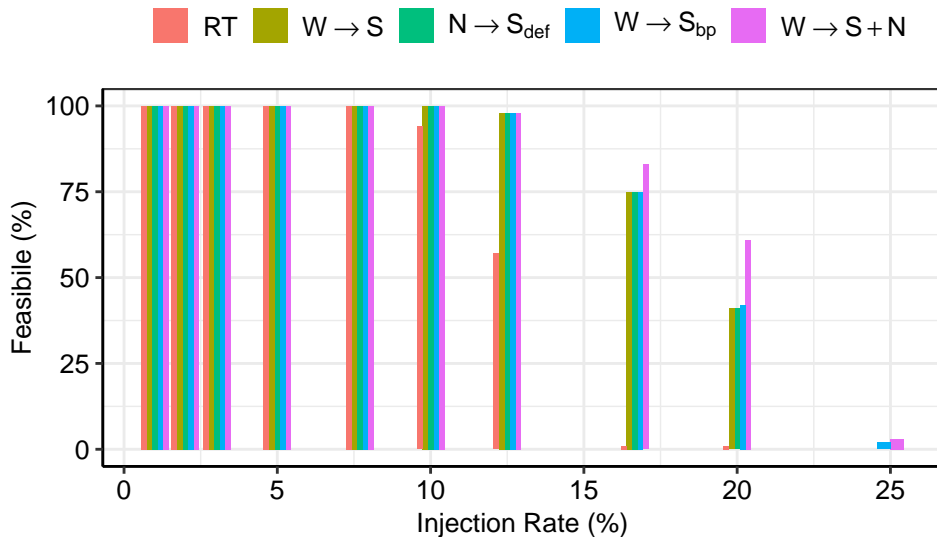


Figure 5.1: Feasible flowsets for RANDOM traffic with $b=1$ at 5×5 system size with 128-deep FIFOs in the NoC routers.

In Figure 5.1, we plot the number of feasible flowsets for RANDOM traffic pattern on the different NoCs. For HopliteRT, there are no FIFOs, but we know that flowsets are not feasible when the interfering flows on any link exceed the link bandwidth, *i.e.* you cannot use more than 100% of any link capacity. For $W \rightarrow S$ buffer design with Backpressuring ($W \rightarrow S_{bp}$) and $N \rightarrow S$ buffer design with Deflections ($N \rightarrow S_{def}$), FIFO occupancy can never exceed the programmed depth (128 in this case) and hence a flow in these designs become infeasible when the link bandwidth exceeds 100%.

The deflection pattern for HopliteRT forces traffic to travel through longer paths through the NoC thereby interfering with a lot of other traffic flows. Hence, the feasibility trends for HopliteRT fall drastically above 10% injection rates. The HopliteBuf NoCs ($W \rightarrow S$, $W \rightarrow S+N$) are more resilient and support a larger fraction of the flowsets for larger injection rates. At the peak supported injection rate of 20%, HopliteBuf supports up to 50–60% of the flowsets, while HopliteRT only routes 1–2% of the flowsets. As predicted from the linearization analysis in Section 4.6, the $W \rightarrow S + N$ topology allows the system to support more traffic and a slightly greater fraction of the synthetic combinations are feasible at even 20% injection rates. The $N \rightarrow S_{def}$ design is identical in performance to HopliteBuf $W \rightarrow S$ and it performs much better than HopliteRT even though it reduces to HopliteRT once the buffer is full. The backpressure design, $W \rightarrow S_{bp}$, performs identical to HopliteBuf $W \rightarrow S$ design, however, at 20% we see a slightly better performance with this design. But the dual buffer design $W \rightarrow S + N$ still outperforms all the other designs. At 25%, we observe that no other design is feasible except for $W \rightarrow S_{bp}$ and $W \rightarrow S + N$, where $W \rightarrow S + N$ is still outperforming the backpressure design. Higher feasibility translates into more FPGA developer freedom in being able to support their communication requirements.

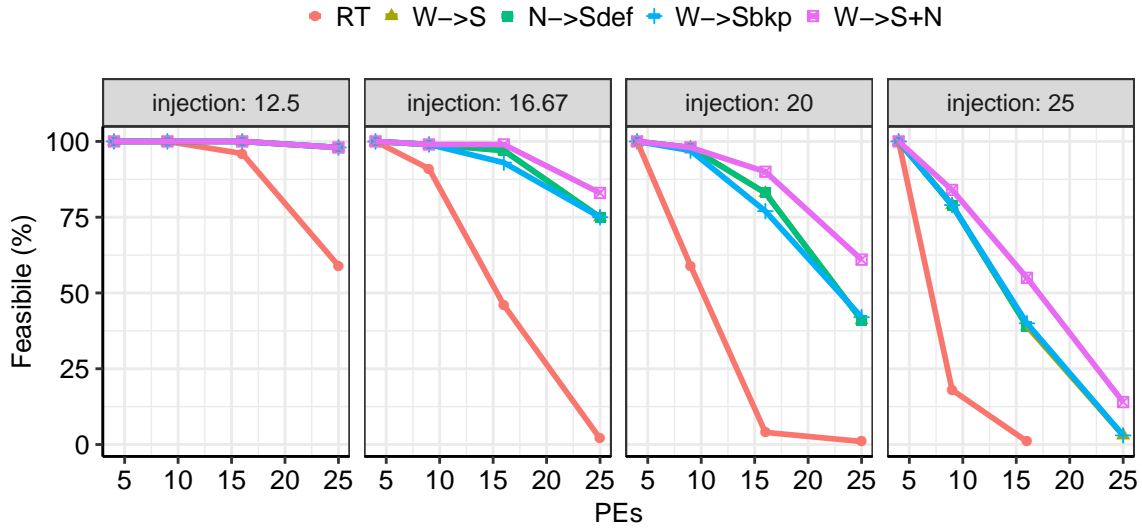


Figure 5.2: Feasible flowsets for **RANDOM** traffic with $b=1$, FIFO depth=128 with different system sizes and injection rates.

In Figure 5.2, we plot the number of feasible flowsets for the **RANDOM** traffic pattern on the different system-sizes. With increasing injection rates, HopliteRT becomes less feasible for all system sizes. For smaller systems, HopliteRT performs identical to buffered NoC designs, however, for large systems HopliteRT’s feasibility reduces drastically at higher injection rates.

5.1.2 Worst-Case Latency Trends

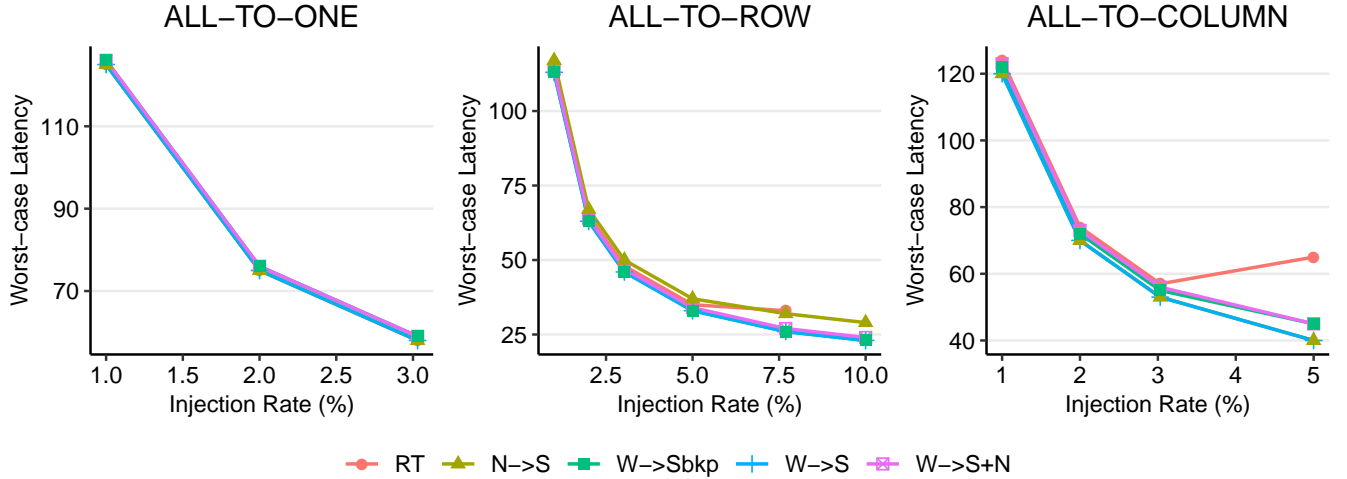


Figure 5.3: Worst-case latency trends for ALL-TO-ONE, ALL-TO-ROW, and ALL-TO-COLUMN traffic patterns on NoCs with 5×5 system sizes and $b=1$. HopliteBuf designs offers no improvements for these traffic patterns.

We expect the use of buffering will help reduce worst-case routing latencies as we eliminate deflections. However, the improvements will be balanced by the penalty of waiting in the FIFOs. In Figure 5.3 we show this effect for three traffic patterns with burst $b=1$ and in Figure 5.4 we show the same effect for 100 RANDOM flowsets. The common odd trend here is the *decrease* in injection latency as a function of injection rate. This is not an illusion, and is a result of the fact that the client is regulated and may miss the token cycle which scales with the injection rate ρ of the regulator. At large enough injection rates we eventually start to see an increase due to network congestion but this is marginal. For the ALL-TO-ONE traffic pattern, the waiting time in the FIFOs lines up with the penalty of deflections resulting in no observable difference between the different designs. For ALL-TO-ROW traffic pattern, there is no visible difference among the buffered NoC designs except for $N \rightarrow S_{def}$ which has slightly higher worst-case latency than other cases. Another thing to note is that HopliteRT saturates at 7.5% injection rate while buffered NoCs are saturating at a higher injection rate of 10%. For ALL-TO-COLUMN, all the NoC designs are saturating at around 5% and they all perform almost identically except for HopliteRT which shows substantially higher worst-case latency than other designs. The saturation rate for these designs are in agreement with the system size where all the other nodes send packets randomly to each other except to itself. For RANDOM traffic, we show a distribution of measured cycle counts across the 100 flowsets. There is a clear benefit to using buffers to avoid deflections as bufferless HopliteRT shows a wider spread of achieved worst-case latencies. The buffer waiting

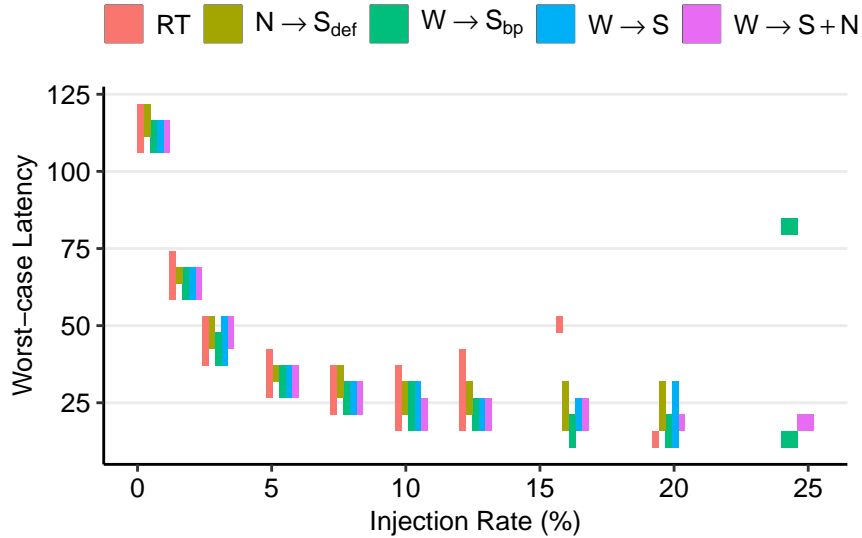


Figure 5.4: Worst-case latency trends for **RANDOM** traffic pattern on NoCs with 5×5 system sizes and $b=1$. HopliteBuf performs better than other designs for this workload.

time is lower than the penalty of deflections resulting in tighter latency spreads for HopliteBuf NoCs. Furthermore, $W \rightarrow S$ designs suffer a buffer wait only at a single turn, it exhibits slightly better or identical performance to the two-FIFO $W \rightarrow S + N$ design. $N \rightarrow S_{def}$ design suffer a buffer wait at each switch when travelling in vertical direction and hence it performs worse than HopliteBuf but much better than HopliteRT. The backpressure design $W \rightarrow S_{bp}$, however, performs similar to HopliteBuf $W \rightarrow S$ variant and in some case even better than the dual-buffer $W \rightarrow S + N$ design. Overall HopliteBuf is 1.2–2 \times better than HopliteRT in terms of worse-case routing latencies. We also see that HopliteRT is poorly unable to support the highest injection rate of 20% that is well-supported by the HopliteBuf NoCs. Thus, the presence of buffers not only improves (reduces) worst-case latencies, but also supports higher data rates. This is expected as HopliteRT, $N \rightarrow S_{def}$, and $W \rightarrow S_{bp}$ steals unnecessary bandwidth in the X-ring due to deflection or backpressuring.

5.1.3 Worst-Case Latency Breakdown

In Figure 5.5 we show a breakdown of worst-case latency into its **source-queueing latency** (waiting time at PEs) and **in-flight latency** (actual routing time in the NoC). The improvements due to elimination of deflections does show up in better in-flight routing latencies for HopliteBuf designs, but larger wins are visible during source queueing. This is because the NoC is blocking the PE

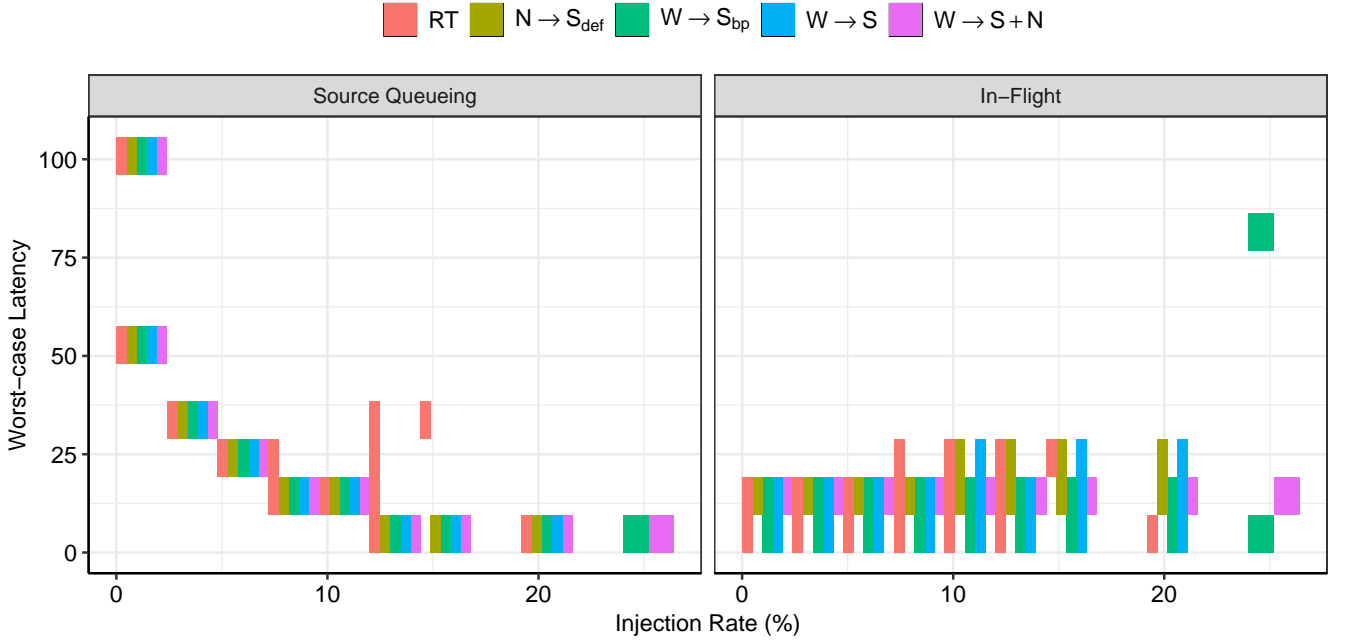


Figure 5.5: Breakdown of Source-Queueing and In-Flight NoC latencies for **RANDOM** workload with $b=1$, FIFO depth=128 at 5×5 system size. Both metrics improved due to buffering.

injection ports less often by keeping packets in the buffers instead of wasting injection slots due to deflection. For HopliteRT routing scheme, the $N \rightarrow E$ deflection potentially sends packets along the *scenic route* around each X-ring (at most once) generating traffic conflicts where none would exist for conventional DOR routing. $N \rightarrow S_{def}$ design has better source-queueing latency than HopliteRT from the fact that it absorbs most of the conflicting traffic in the buffer and resort to deflection when the buffer is full.

We expect $N \rightarrow S_{def}$ at most as worse as HopliteRT when the FIFO size is 0. As the FIFO size goes down, $N \rightarrow S_{def}$ design shows higher *inflight* latencies as number of deflections increase with small FIFO size. We observe this behaviour in *inflight latency vs injection rate plot* in Figure 5.6b for burst=1 and FIFO depth of 32.

$W \rightarrow S_{bp}$ design has comparable in-flight latencies but we expect the source-queueing latency for this design to go high in case of small buffer sizes as PE will be backpressured and packets will wait longer in the client if the buffer is full. We observe this behaviour in *source queueing vs injection rate plot* in Figure 5.6a for burst=1 and FIFO depth of 32. HopliteBuf chooses FIFO waiting on conflicts thereby reducing contention in other X-rings and a drop in

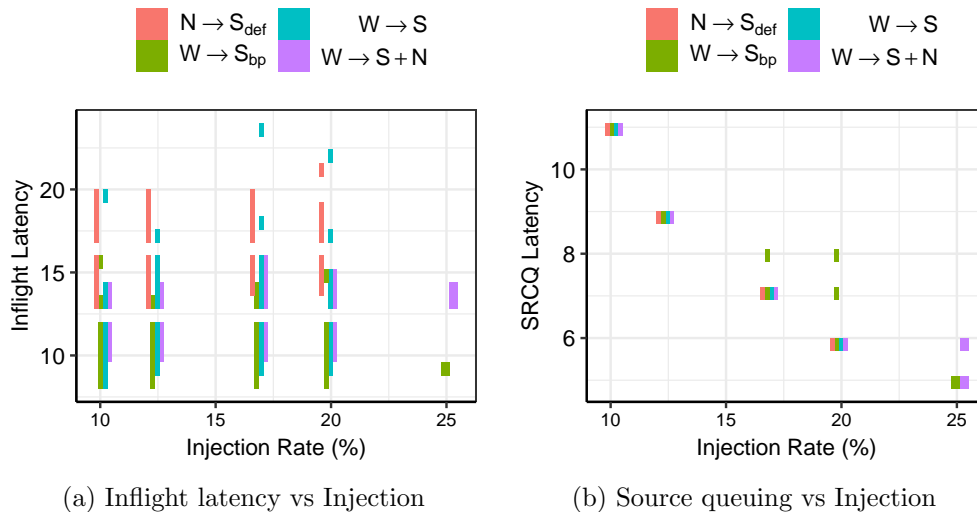


Figure 5.6: Source-Queueing and In-Flight NoC latencies for **RANDOM** workload with $b=1$, FIFO depth=32 at 5×5 system size.

source queueing delays. As we see, the NoC traversal time is mostly unaffected even in presence of FIFOs.

5.1.4 Latency Distribution

In Figure 5.7, we show the histogram of worst-case packet latencies for the different NoCs for **RANDOM** traffic with burst $b=1$, and injection rate $\rho=7.5\%$ at 5×5 system size. We note that the HopliteRT NoC has a much wider spread than the FIFO designs. This is because deflections create unpredictable trips through the NoC X-rings. In contrast, a victimized packet just sits in a buffer and the waiting time in the buffer is much lower than round-trips around the ring. $N \rightarrow S_{def}$ design is much better in terms of worst-case latency than HopliteRT given the size of FIFO that absorbs most of the conflicting packets and avoid deflections. We see that $W \rightarrow S_{bp}$ design performs better than Hoplite $W \rightarrow S$ design as more cases are feasible with backpressuring than it was with just buffers. As expected, all the other buffered design has a marginally wider distribution than the $W \rightarrow S+N$ design as the packets have an extra choice during the turn.

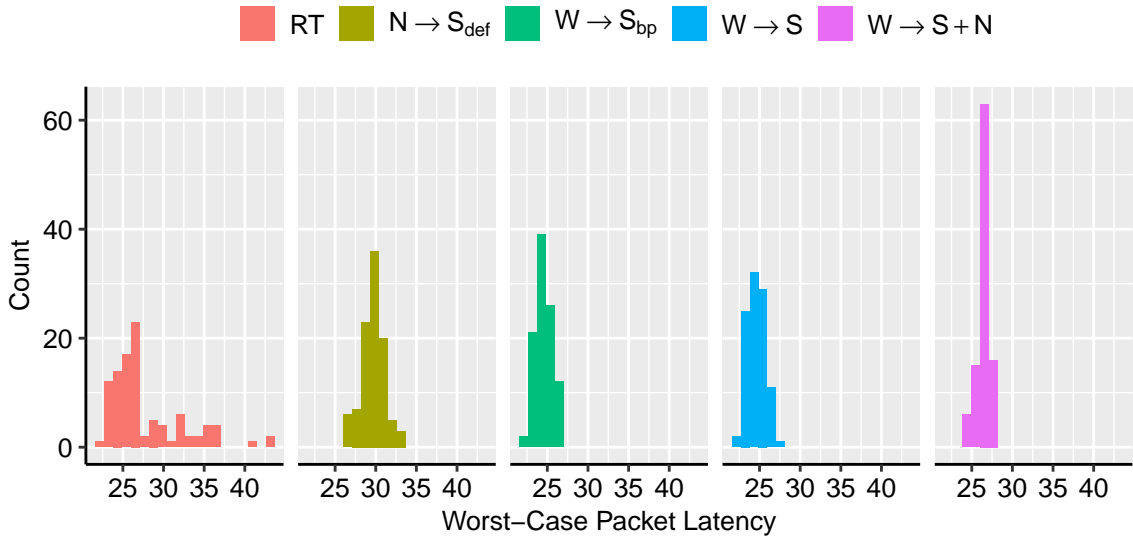


Figure 5.7: Distribution of worst-case packet latencies for RANDOM workload with $b=1$, $\rho=7.5\%$ at 5×5 system size. HopliteRT has a wider spread due to the unpredictable nature of the deflections. HopliteBuf has narrower spreads.

5.1.5 FIFO Sizing

Ultimately, the NoC with improved worst-case latencies is useful to us only if the buffer sizes are reasonable to realize on modern FPGAs. For a single LUT we can get 16–32 storage bits for our FIFOs making it possible to build LUTs using these low-cost components. We cap our experiments at 128-deep FIFO sizes to keep NoC LUT cost at 4 LUTs/bit.

For ALL-TO-ONE traffic pattern shown in Figure 5.8, we will observe high FIFO usage in the column containing the destination client. As burst length increases, the FIFO usage also scales linearly with very low utilization with a burst length of 1–2. A similar FIFO usage is observed with ALL-TO-ROW (shown in Figure 5.9) and ALL-TO-COLUMN (shown in Figure 5.10) traffic patterns as they target a specific row and column in the torus as destination.

RANDOM traffic shown in Figure 5.11 exhibits slightly lower FIFO usage and demonstrates a spread of occupancies depending on connectivity pattern. While no experiment occupies more than 50 entries in the FIFO, on average, we only need ≈ 20 –25 entries. We note an odd reduction in FIFO occupancy above 10% injection rate. This is because an increasing subset of flowsets are not feasible with 128-deep FIFO limit *i.e.* FIFOs start going full.

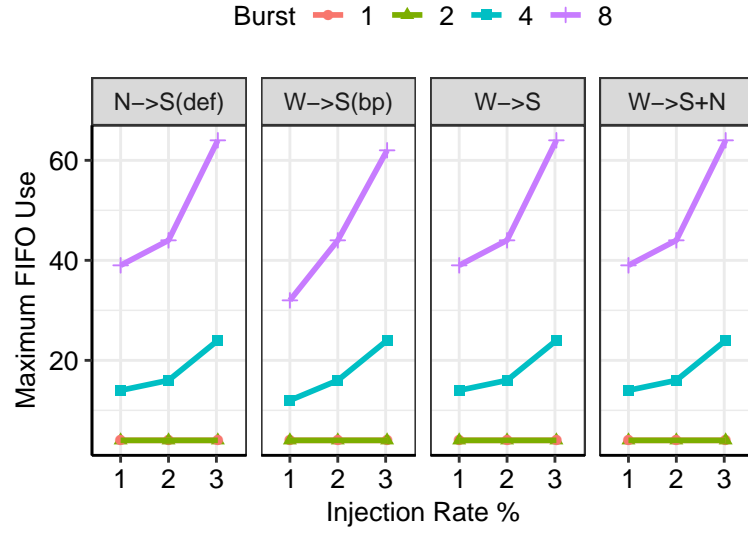


Figure 5.8: Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-ONE pattern

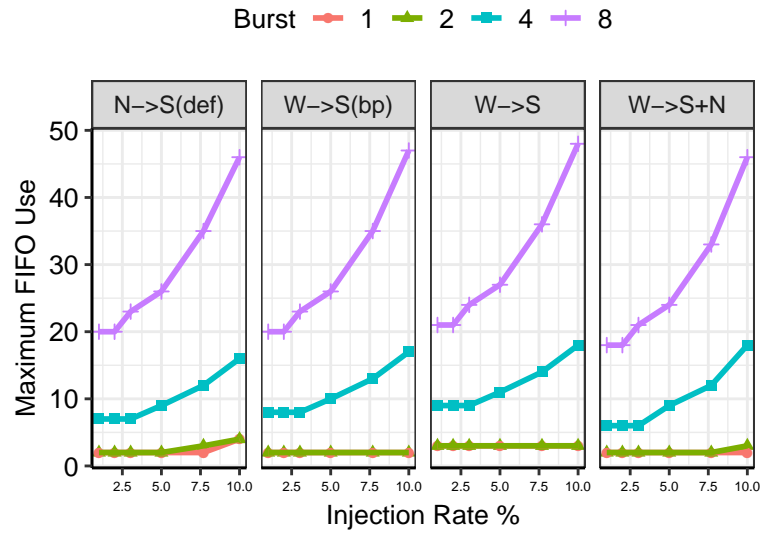


Figure 5.9: Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-ROW pattern

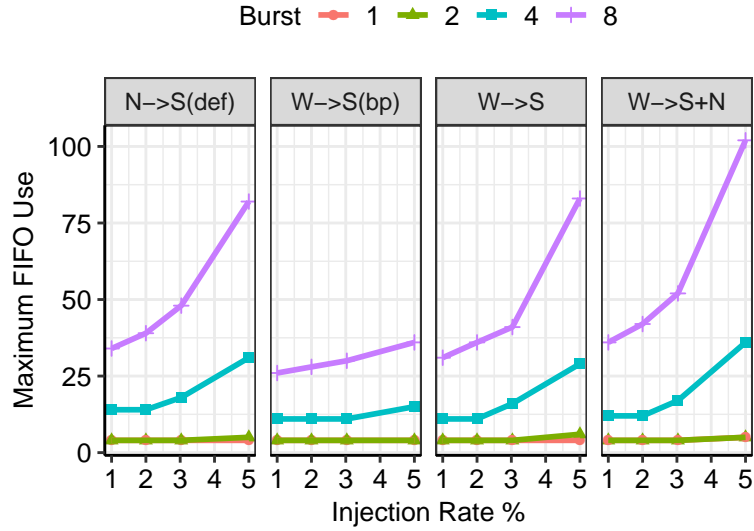


Figure 5.10: Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for ALL-TO-COLUMN pattern

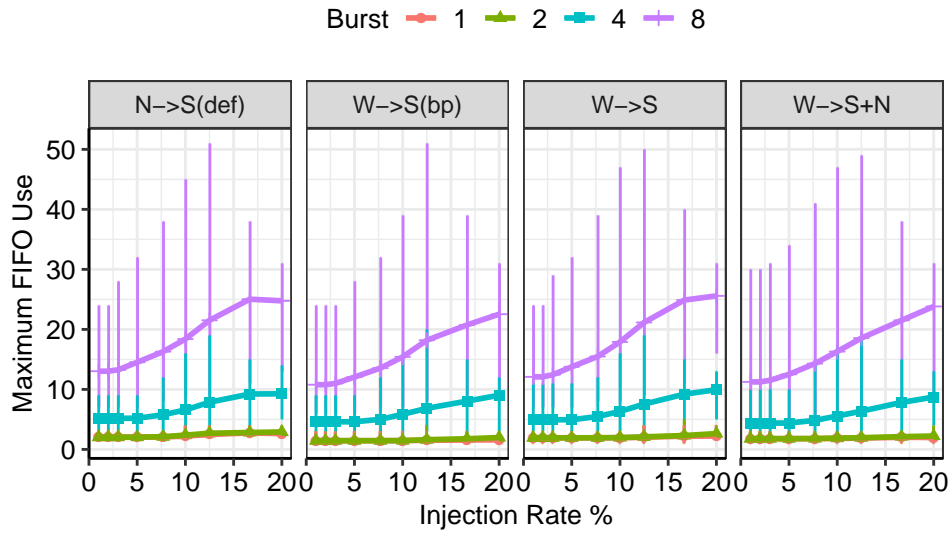


Figure 5.11: Maximum FIFO usage trends from RTL simulations of NoCs with 5×5 system sizes for RANDOM pattern

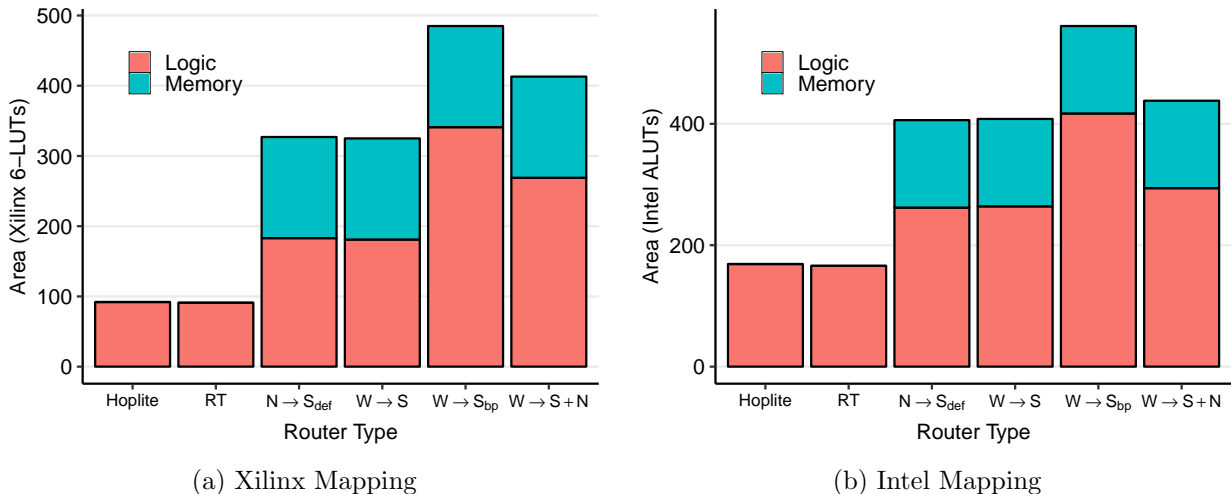


Figure 5.12: LUT utilization for logic and memory across various Hoplite routers on Xilinx and Intel FPGAs with Payload=64b and FIFO=64 deep.

5.2 FPGA Implementation Results

Now we compare the FPGA implementation costs of HopliteBuf to other FPGA-overlay NoCs presented in this thesis. We quantify the LUT utilization of the various Hoplite routers in Figure 5.12. We present resource costs on Xilinx and Intel FPGAs.

For HopliteBuf $W \rightarrow S + N$ design, we are essentially dividing the traffic into two buffers going up and down and hence, the total distributed RAM capacity stays same as it just split into two SRL32 or MLAB instantiations instead of a longer single distributed RAM block. Here, with limited opportunity for input sharing, the resulting design is larger in LUT cost, but as we have seen already, this allows efficient use of the NoC links.

As shown in the figure, the design size scales linearly with the product of Datawidth of the NoC \times Depth of the FIFO on both vendor parts. With 64-deep FIFOs mapped to distributed RAMs, the storage fraction \blacksquare increases design size by $\approx 2 \times$.

The logic cost \blacksquare varies with different router types. The $N \rightarrow S_{def}$ and HopliteBuf $W \rightarrow S$ are nearly identical in logic costs as they use same size multiplexers while $N \rightarrow S_{def}$ uses a slightly more logic for implementing deflections. For the dual-FIFO HopliteBuf $W \rightarrow S + N$ design, the extra multiplexing needed for upward route also increases cost of the switching logic. We observe the worst logic usage for $W \rightarrow S_{bp}$ backpressure design, where the classic flow-control logic implementation uses almost $2 \times$ more resources than other single-buffer designs. Both HopliteBuf achieves better performance than $W \rightarrow S_{bp}$ while maintaining low implementation

costs.

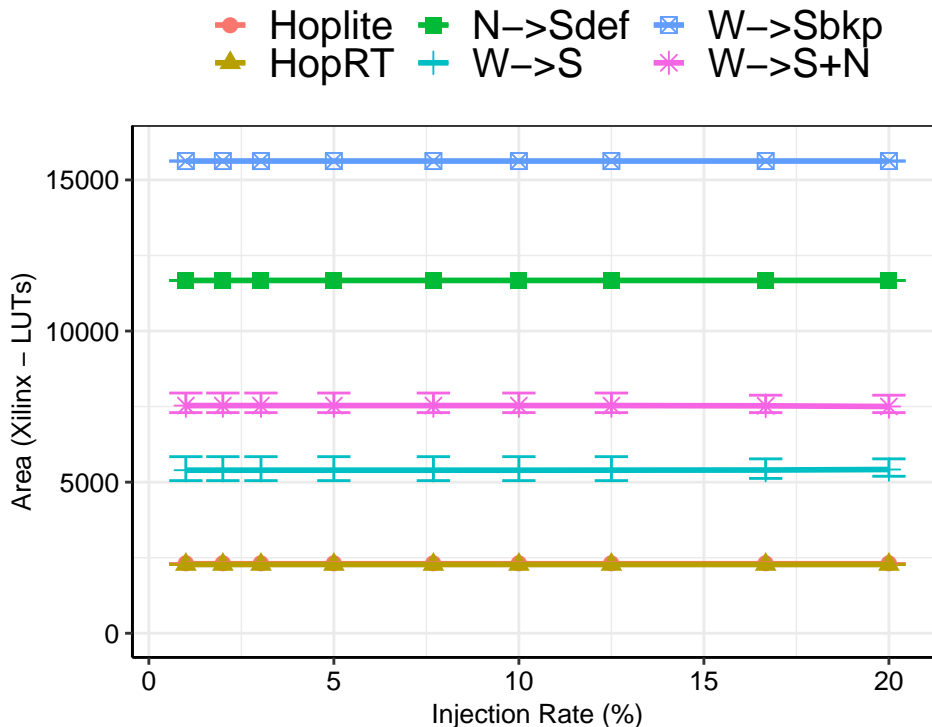


Figure 5.13: Maximum Area usage at different injection rates for **RANDOM** pattern with 5×5 system size. Each node in the system uses carefully picked FIFO sizes through static analysis in case of $W \rightarrow S$ and $W \rightarrow S + N$ HopliteBuf designs.

In Figure 5.13, we show the area usage to implement a complete 5×5 system with different NoC designs. The experiment is done for 100 synthetic **RANDOM** flows where the prototype designs, $W \rightarrow S_{bp}$ and $N \rightarrow S_{def}$, uses a fixed FIFO size of 128 whereas HopliteBuf ($W \rightarrow S$, $W \rightarrow S + N$) implements carefully analyzed FIFO sizes by our analysis tool. As shown in figure, HopliteBuf designs are $\approx 2 - 3 \times$ worse than Hoplite(RT) for all injection rates whereas $N \rightarrow S_{def}$ design is around $5 \times$ worse and $W \rightarrow S_{bp}$ design is ≈ 7 times worse than Hoplite(RT). This clearly shows the benefit of carefully analyzing the FIFO sizes for a particular application. Hence, we say that not only are the HopliteBuf designs outperforms other NoCs in terms of latency and throughput the overall hardware requirement is much better than the backpressure and deflection (fixed FIFO size) designs as well.

5.3 Analysis Results

We now examine the quality of our static analysis predictions and compare them to simulated data. The Static analysis was only done for HopliteRT and the two variants of HopliteBuf: $W \rightarrow S$ and $W \rightarrow S + N$. And hence, we only compare the simulation and analysis results for these three router designs.

5.3.1 Feasible Flowsets

Our analysis tools take the communication pattern of a flowset, its injection rate ρ and burst b to determine if it can route successfully without making a FIFO ever go full. Analysis is more conservative, and you will note that Figure 5.14 is different from the simulation data in Figure 5.1. Our simulation results are for 1024 packets per client, and there may be longer simulation conditions that ultimately go infeasible. Hence, we trust our analysis data as it is backed by the formal proofs explained in Section 4. Here, we see HopliteRT dropping dramatically above 8% while HopliteBuf clones closely track simulation results. At 11% rates, we see analysis predict feasibility of only 2–3% of HopliteRT and $\approx 90\%$ for HopliteBuf. Back in Figure 5.1, simulation results showed 50% of HopliteRT were feasible and $\approx 90\%$ for HopliteBuf. This suggests tighter analysis bounds for HopliteBuf resulting in better provable utilization of resources.

5.3.2 Worst-Case Latency: Analysis vs. Simulation

In Figure 5.15, we show the predicted worst-case latency count as a result of our static analysis vs. actual observed latencies through simulation. As expected, the predicted bounds are worse with analysis due to pessimistic assumptions regarding interference of traffic flows. HopliteRT predictions are as much as $1.5\times$ worse than the $W \rightarrow S + N$ predictions due to pessimism inherent in the HopliteRT routing algorithm. It is interesting to see that a few flowsets mapped to HopliteRT at 16–20% injection rates actually simulate fine, but are discarded by analysis as infeasible, yet again due to analytic pessimism. Furthermore, we see that the $W \rightarrow S + N$ predictions are significantly tighter than the $W \rightarrow S$ predictions. This is primarily due to the challenges associated with analyzing loopy flows in the vertical ring. When comparing the wider $W \rightarrow S$ spread to HopliteRT, it is important to note that a significant chunk of flowsets were infeasible when mapped to HopliteRT (See Figure 5.14). Thus, the larger latencies are due to $W \rightarrow S$ being able to feasibly route flowsets and doing so with high latencies than not being able to do so at all. The $W \rightarrow S + N$ analysis is significantly better than $W \rightarrow S$ and has a larger feasibility to compound the matter.

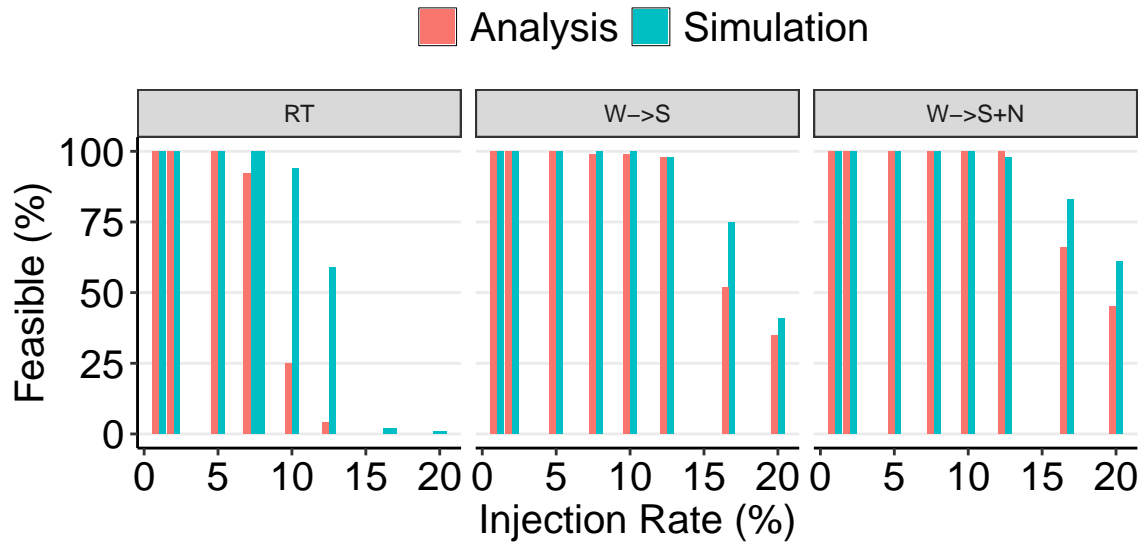


Figure 5.14: Feasible flowsets predicted by static analysis. Analysis is more conservative than simulation for HopliteRT, but much tighter for HopliteBuf.

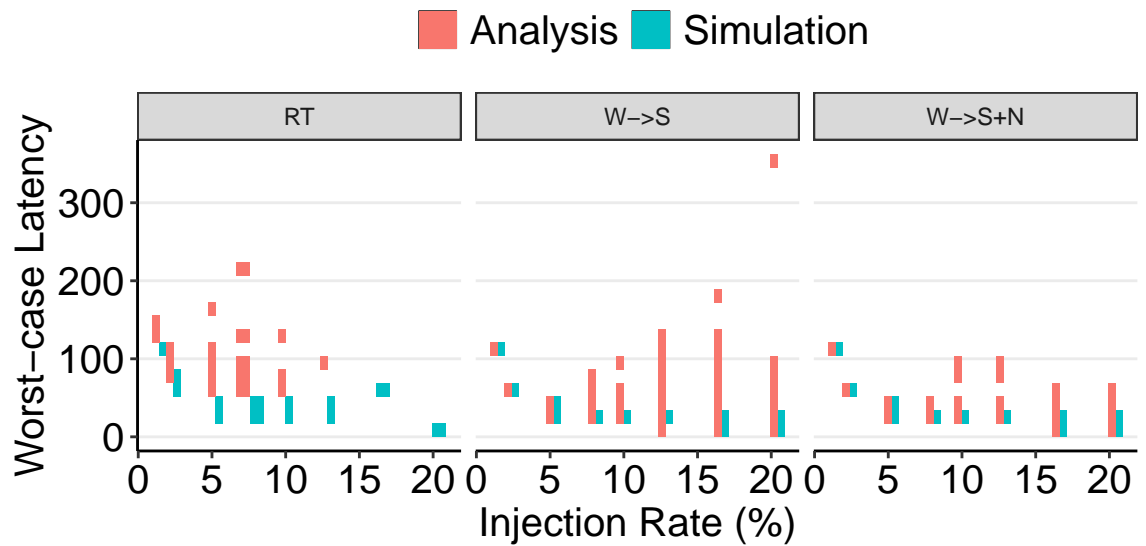


Figure 5.15: Worst-Case Latency Prediction-vs-Simulation, RANDOM traffic, $b=1$, 5×5 system size, 128-deep FIFOs.

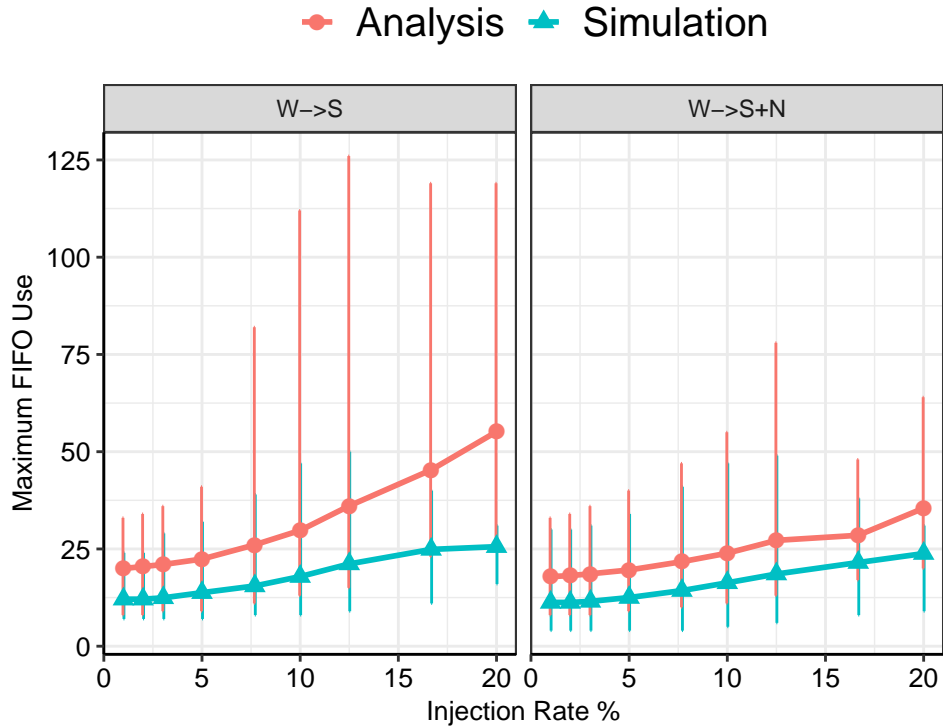


Figure 5.16: Worst-Case FIFO size Prediction-vs-Simulation for RANDOM traffic at 5×5 system size with 128-deep FIFOs and burstiness of 8.

5.3.3 FIFO Sizing: Analysis vs. Simulation

In Figure 5.16, we compare the result of static analysis with simulated data for FIFO usage for a 5×5 NoC with RANDOM traffic and a worst-case burstiness of 8. For the $W \rightarrow S$ topology, we cap the maximum FIFO size to 128 to stay within reasonable 4 LUTs/bit FIFO cost. In this case, the FIFOs go full for a few flowsets only above a healthy 15% injection rate. For the $W \rightarrow S + N$ topology, the FIFO sizes are capped at 64 (for a sum of 128) and only go full at a higher 20% injection rate. For both cases, we observe that simulated data shows lower occupancies than the prediction by as much as $2.5 \times$ (on average $1.5 \times$). This is expected due to the pessimism in the analysis but the LUT cost impact is limited due to SRL32 packing quantization. For FPGA implementation, we can choose to size all FIFOs in the NoC to the largest size, or customize each FIFO independently as per the static analysis. We observe that the largest size of 128 is rarely observed, and roughly 50% of our occupancies are below the 32 threshold. Thus, we can customize the right SRL depth to further save resources by as much as $2 \times$.

Chapter 6

Conclusion and Future Research

Wider adoption of FPGAs to implement compute intensive applications has driven enhancements in FPGA architecture. Embedded hard blocks like DSPs, high-speed memory interfaces like HBM, and various IOs require an efficient way to move data among these high-speed blocks over FPGA fabric. Network-on-Chips provide a scalable solution to satisfy the intra-chip communication requirements over these dense systems. In this work, we present an efficient FPGA-overlay NoC design with lightweight buffers. Existing state-of-the-art deflection routing NoCs like Hoplite and HopliteRT provide a low-cost and high-speed design. However, they pay higher penalties due to deflections like higher-latency, out-of-order delivery and low-throughput. The proposed NoC design exploits the architectural benefits of deflection routed NoCs and use small stall-free FIFOs to eliminate deflections and hence gain in-order delivery, lower latency and higher sustained throughput. The NoC design also uses a static analysis tool to compute these buffer bounds and also provide latency and bandwidth information of the workloads.

We summarize the main contributions of this work:

- Design of an FPGA NoC torus topology to enhance static analysis for computing buffer bounds and NoC router microarchitecture redesign with stall-free FIFOs to eliminate deflections and provide in-order packet delivery. Optimization and customization of the NoC router RTL to match Xilinx and Intel FPGAs.
- Development of a buffer sizing algorithm to compute the worstcase bounds on FIFO occupancies. Use of vertical NoC link linearization to improve provable link utilization. Static analysis tools compute upper bounds on size of FIFOs required for stall-free operation, source queueing delay, in-flight routing latency under various conditions.
- Engineering of a robust simulation infrastructure to compute cycle counts of packet traversals in the NoC. Resource and performance analysis of the NoC under various synthetic workloads.

We saw two variants of HopliteBuf: single-buffer $W \rightarrow S$ and dual-buffer $W \rightarrow S + N$. While the $W \rightarrow S$ design overcomes the problems with deflection routed NoCs and provide an efficient solution, it complicates the static analysis due to its architecture and Cyclic dependencies. Cyclic dependencies reduce the provable network utilization and hence we get pessimistic bounds on the buffer sizes needed to support a workload. The dual buffer $W \rightarrow S + N$ design linearizes the analysis by breaking the vertical rings that create cyclic dependencies. We show that not only does the dual-buffer design enhances the static analysis, it also shows significant performance gains over Hoplite, HopliteRT, and the single buffer $W \rightarrow S$ design. Our static analysis tools that can compute worst-case buffer occupancy bounds, along with latency bounds for communication patterns with rate, and burst information known up-front. In our experiments with 100 randomly-generated flowsets, we show that HopliteBuf is able to deliver 40-50% feasibility at 20% injection rates while the competing state-of-the-art HopliteRT NoC only supports 25% feasibility at 10% injection rates at $2 \times$ worse latency bounds.

6.1 Future Research

In this section, we list the promising avenues for research – building on the work proposed in this thesis.

Analysing Hard-NoCs

We anticipate our static analysis tools to help compute latency bounds on the newly-announced Xilinx Versal NoC with hardened FIFOs of known sizes. Unlike LUT-based HopliteBuf NoC used in this study, the Versal hard-NoC FIFOs do not need to be designed with LUT-FIFO capacity constraints. If the workload information is known statically, the analysis tool can be used to determine maximum rate and latency supported on a fixed hardware size.

References

- [1] Bluespec system verilog: Efficient, correct rtl from high level specifications. In *Proceedings of the Second ACM/IEEE International Conference on Formal Methods and Models for Co-Design, MEMOCODE '04*, pages 69–70, Washington, DC, USA, 2004. IEEE Computer Society.
- [2] P Abad, P Prieto, L G Menezes, A Colaso, V Puente, and J A Gregorio. TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers. *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 99–106, 2012.
- [3] M S Abdelfattah and V Betz. Design tradeoffs for hard and soft FPGA-based Networks-on-Chip. In *Field-Programmable Technology*, pages 95–103, 2012.
- [4] M. S. Abdelfattah, A. Bitar, and V. Betz. Design and applications for embedded networks-on-chip on fpgas. *IEEE Transactions on Computers*, 66(6):1008–1021, June 2017.
- [5] David H. Albonesi, Margaret Martonosi, David I. August, and José F. Martínez, editors. *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*. ACM, 2009.
- [6] Altera. Applying the benefits of network on a chip architecture to fpga system design. Altera White Paper, apr 2011.
- [7] Altera Corp. Arria 10 Core Fabric and General Purpose I/Os Handbook, May 2015.
- [8] Ahmed Amari and Ahlem Mifdaoui. Worst-case timing analysis of ring networks with cyclic dependencies using network calculus. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2017.
- [9] Krste Asanović and David Patterson. Instruction sets should be free: the case for risc-v. Technical Report No. UCB/EECS-2014-146, August 2014.

- [10] M. Becker, B. Nikolic, D. Dasari, B. Akesson, V. Nelis, M. Behnam, and T. Nolte. Partitioning and Analysis of the Network-on-Chip on a COTS Many-Core Platform. In *proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017.
- [11] Vaughn Betz and Jonathan Rose. FPGA routing architecture: Segmentation and buffering to optimize speed and density. In *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99, pages 59–68, New York, NY, USA, 1999. ACM.
- [12] Christian Bienia and Kai Li. Parsec 2.0: A new benchmark suite for chip-multiprocessors. In *Proceedings of the 5th Annual Workshop on Modeling, Benchmarking and Simulation*, 2009.
- [13] Buchholz. Comments on csma. *IEEE*, 802(11):802–11, 1992.
- [14] Y. Cai, K. Mai, and O. Mutlu. Comparative evaluation of fpga and asic implementations of bufferless and buffered routing algorithms for on-chip networks. In *Sixteenth International Symposium on Quality Electronic Design*, pages 475–484, March 2015.
- [15] T. E. Carlson, W. Heirmant, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, Nov 2011.
- [16] S. Corbetta, V. Rana, M. D. Santambrogio, and D. Sciuto. A light-weight network-on-chip architecture for dynamically reconfigurable systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 49–56, July 2008.
- [17] W J Dally and B Towles. Route packets, not wires: on-chip interconnection networks. In *Design Automation Conference, 2001. Proceedings*, pages 684–689, 2001.
- [18] Benoît Dupont de Dinechin and Amaury Graillat. Network-on-chip service guarantees on the kalray mppa-256 bostan processor. In *Proceedings of the 2Nd International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, AISTECS '17, pages 35–40, New York, NY, USA, 2017. ACM.
- [19] C. Fallin, C. Craik, and O. Mutlu. Chipper: A low-complexity bufferless deflection router. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 144–155, Feb 2011.
- [20] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu. Minbd: Minimally-buffered deflection routing for energy-efficient interconnect. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 1–10, May 2012.

- [21] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu. Minbd: Minimally-buffered deflection routing for energy-efficient interconnect. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 1–10, May 2012.
- [22] Chris Fallin, Chris Craik, and Onur Mutlu. Chipper: A low-complexity bufferless deflection router. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, HPCA '11*, pages 144–155, Washington, DC, USA, 2011. IEEE Computer Society.
- [23] Mohammad Fattah, Antti Airola, Rachata Ausavarungnirun, Nima Mirzaei, Pasi Liljeberg, Juha Plosila, Siamak Mohammadi, Tapio Pahikkala, Onur Mutlu, and Hannu Tenhunen. A low-overhead, fully-distributed, guaranteed-delivery routing algorithm for faulty network-on-chips. In *Proceedings of the 9th International Symposium on Networks-on-Chip, NOCS '15*, pages 18:1–18:8, New York, NY, USA, 2015. ACM.
- [24] Kees Goossens and Andreas Hansson. The ethereal network on chip after ten years: Goals, evolution, lessons, and future. In *47th DAC*, pages 306–311. IEEE, 2010.
- [25] J. Gray. Keynote 3 2014; the past and future of fpga soft processors. In *ReConFigurable Computing and FPGAs (ReConFig)*, pages 1–1, Dec 2014.
- [26] Jan Gray. GRVI-Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In *Proc. 24th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 17–20. IEEE, 2016.
- [27] Yutian Huan and A DeHon. FPGA optimized packet-switched NoC using split and merge primitives. In *Field-Programmable Technology*, pages 47–52, December 2012.
- [28] Mike Hutton. Understanding how the new hyperflex architecture enables next-generation high-performance systems. Altera White Paper, Apr. 2015.
- [29] S. Jeon, J. Cho, Y. Jung, S. Park, and T. Han. Automotive hardware development according to iso 26262. In *13th International Conference on Advanced Communication Technology (ICACT2011)*, pages 588–592, Feb 2011.
- [30] N. Kapre. Marathon: Statically-scheduled conflict-free routing on fpga overlay nocs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 156–163, May 2016.
- [31] N. Kapre and J. Gray. Hoplite: Building austere overlay nocs for fpgas. In *Field Programmable Logic and Applications*, pages 1–8, Sept 2015.
- [32] Nachiket Kapre, Nikil Mehta, Michael deLorimier, Raphael Rubin, Henry Barnor, Michael J Wilson, Michael Wrighton, and Andre DeHon. Packet switched vs. time multiplexed FPGA

- overlay networks. In *Proc. 14th IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 205–216. IEEE, 2006.
- [33] H. Kashif and H. Patel. Bounding buffer space requirements for real-time priority-aware networks. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 113–118, Jan 2014.
- [34] Hany Kashif and Hiren Patel. Buffer Space Allocation for Real-Time Priority-Aware Networks. In *proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, April 2016.
- [35] John Kim. Low-cost router microarchitecture for on-chip networks. In Albonese et al. [5], pages 255–266.
- [36] John Kim. Low-cost router microarchitecture for on-chip networks. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 255–266. ACM, 2009.
- [37] B S Landman and Roy L Russo. On a Pin Versus Block Relationship For Partitions of Logic Graphs. *Computers, IEEE Transactions on*, (12):1469–1479, 1971.
- [38] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, 2001.
- [39] Lu Ting, Kenny Ryan, and Atsatt Sean. Secure Device Manager for Intel Stratix 10 Devices Provides FPGA and SoC Security, March 2018.
- [40] E Matthews, L Shannon, and A Fedorova. Polyblaze: From one to many bringing the microblaze into the multicore era with Linux SMP support. In *22nd Field Programmable Logic and Applications*, pages 224–230, 2012.
- [41] G Michelogiannakis, D Sanchez, W J Dally, and C Kozyrakis. Evaluating Bufferless Flow Control for On-chip Networks. *Networks-on-Chip (NOCS), 2010 Fourth ACM/IEEE International Symposium on*, pages 9–16, 2010.
- [42] Jörg Mische and Theo Ungerer. Low power flitwise routing in an unidirectional torus with minimal buffering. In *Proceedings of the Fifth International Workshop on Network on Chip Architectures, NoCArc '12*, pages 63–68, New York, NY, USA, 2012. ACM.
- [43] Jörg Mische and Theo Ungerer. Guaranteed Service Independent of the Task Placement in NoCs with Torus Topology. In *Proc. 22Nd RTNS.*, RTNS '14, page 151:160. ACM, 2014.
- [44] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09*, pages 196–207, New York, NY, USA, 2009. ACM.

- [45] Thomas Moscibroda, Onur Mutlu, Thomas Moscibroda, and Onur Mutlu. *A case for bufferless routing in on-chip networks*, volume 37. ACM, New York, New York, USA, June 2009.
- [46] T. D. A. Nguyen and A. Kumar. PR-HMPSoC: A versatile partially reconfigurable heterogeneous multiprocessor system-on-chip for dynamic FPGA-based embedded systems. In *Field Programmable Logic and Applications*, pages 1–6, Sept 2014.
- [47] Andreas Olofsson, Tomas Nordström, and Zain-ul-Abdin. Kickstarting high-performance energy-efficient manycore architectures with epiphany. *CoRR*, abs/1412.5538, 2014.
- [48] M. Panic, C. Hernandez, E. Quinones, J. Abella, and F. J. Cazorla. Modeling High-Performance Wormhole NoCs for Critical Real-Time Embedded Systems. In *proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.
- [49] Michael K Papamichael and James C Hoe. Connect: re-examining conventional wisdom for designing nocs in the context of fpgas. In *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, pages 37–46. ACM, 2012.
- [50] Michael K Papamichael and James C Hoe. CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs. In *the ACM/SIGDA international symposium*, page 37, New York, New York, USA, 2012. ACM Press.
- [51] T. Pionteck, R. Koch, and C. Albrecht. Applying partial reconfiguration to networks-on-chips. In *Field Programmable Logic and Applications*, pages 1–6, Aug 2006.
- [52] M Ramirez, M Daneshtalab, J Plosila, and P Liljeberg. NoC-AXI interface for FPGA-based MPSoC platforms. In *Field Programmable Logic and Applications*, pages 479–480, 2012.
- [53] Zheng Shi and Alan Burns. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Second ACM/IEEE NOCS (nocs 2008)*, NOCS '08, pages 161–170. IEEE, apr 2008.
- [54] J. Specht and S Samii. Urgency?based scheduler for time?sensitive switched ethernet networks. In *Proceedings of the Euromicro Conference on Real?Time Systems (ECRTS)*, pages 75–86, 2016.
- [55] Ian Swarbrick, Dinesh Gaitonde, Sagheer Ahmad, Brian Gaide, and Ygal Arbel. Network-on-chip programmable platform in versaltm acap architecture. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '19, pages 212–221, New York, NY, USA, 2019. ACM.
- [56] Kizheppatt Vipin and Suhaib A Fahmy. Dyract: A partial reconfiguration enabled accelerator and test platform. In *Field Programmable Logic and Applications*, pages 1–7. IEEE, 2014.

- [57] Saud Wasly, Rodolfo Pellizzoni, and Nachiket Kapre. HopliteRT: An efficient FPGA NoC for real-time applications. In *F. Program. Technol. (ICFPT), 2017 Int. Conf.*, pages 64–71. IEEE, 2017.
- [58] Xilinx Inc. 7 Series FPGAs Configurable Logic Block User Guide, February 2015.
- [59] Xilinx Inc. AXI Interconnect v2.1, LogiCORE IP Product Guide, December 2017.