

# **HARDWARE-BASED ACCELERATION OF NETWORK-ON-CHIP SIMULATION USING FPGAs**

A Thesis

Submitted in partial fulfilment of the requirements for the degree of

**DOCTOR OF PHILOSOPHY**

*by*

**PRABHU PRASAD B M**

(155113 CS15F10)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575 025

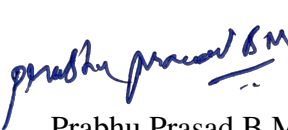
DECEMBER, 2020



## DECLARATION

*by the Ph.D. Research Scholar*

I hereby declare that the Research Thesis entitled **Hardware-based acceleration of Network-on-Chip simulation using FPGAs** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in Department of Computer Science and Engineering is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

 29/12/2020

Prabhu Prasad B M, 155113CS15F10

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: Dec 29, 2020



## CERTIFICATE

This is to certify that the Research Thesis entitled **Hardware-based acceleration of Network-on-Chip simulation using FPGAs** submitted by **Prabhu Prasad B M** (Register Number: **155113 CS15F10**) as the record of the research work carried out by him, is accepted as the Research Thesis submission in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. Basavaraj Talawar  
Research Guide

Dr. Alwyn Roshan Pais  
Chairman - DRPC



To  
**Mother Nature and My Family**





## **ACKNOWLEDGEMENTS**

Firstly, I thank my advisor, Dr. Basavaraj Talawar, for his invaluable guidance, immense support and encouragement throughout my Ph.D. at the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal.

I would like to thank my Research progress assessment committee members Dr. Ramesh Kini M, Associate Professor in the Department of Electronics and Communication Engineering, and Dr. Shashidhar G Koolagudi, Associate Professor in the Department of Computer Science and Engineering, for their encouragement and insightful comments, which helped me in improving my research work.

I whole heartedly express my gratitude to Dr. Alwyn Roshan Pais, Head of the Department, Department of Computer Science and Engineering, NITK Surathkal for his support and valuable suggestions during my doctoral studies.

I would like to thank all the faculty members and non-teaching staff of the Department of Computer Science and Engineering for helping me directly or indirectly in the completion of my research work.

My special thanks to my research lab partners and colleagues with whom I have shared many precious moments during my Ph.D.

I cannot thank my family - Appaji, Amma, Madhu, my better half Rashmi and my adorable kids Bhumika and Varshini - enough. Without you, this journey would not have been possible.

Finally, I would like to thank everyone who is directly and indirectly responsible for the successful completion of my doctoral research work.

Prabhu Prasad B M



## ABSTRACT

Replacing the conventional bus-based architectures, Network-on-Chip (NoC) has become a tangible on-chip communication framework in the many-core processors, Chip Multi-Processors (CMPs), and Multi-Processor System-on-Chips (MPSoCs). Also, NoCs have become an integral part of the heterogeneous systems with application-specific accelerators such as databases, graph processing, and deep neural networks. In these heterogeneous systems, it is the responsibility of NoCs to interconnect various components. More number of cores are being incorporated in state-of-the-art homogeneous and heterogeneous multi-core processors to achieve high performance and better power efficiency. Likewise, to achieve high performance in the target applications, various components such as processing cores, input/output peripherals, and memory components being integrated on heterogeneous systems are also increasing. When there is an increase in the number of interconnected components, the performance of the target application becomes highly dependent on the performance of NoC. Hence, there is a need to model and evaluate large NoC designs quickly and accurately as thousands of cores are targeted in the near future multi-core architectures due to the advances in CMOS technology. NoC modeling helps understand the impact of various design parameters on the overall system and the performance characteristics.

A crucial hurdle in the design and evaluation of large-scale NoC is the lack of rapid methodologies for modeling, which can deliver a high level of accuracy. Analytical models compromise accuracy to achieve results in a short period of time. Hence, to perform the design space exploration of NoCs, designers frequently employ the software simulators. The software simulators provide better accuracy than analytical modeling. When a large-scale NoC with a huge number of nodes is being simulated, the software simulators tend to become too slow. To address the issue of simulation speed, an Field Programmable Gate Arrays (FPGA) based NoC simulation acceleration framework has been proposed in this thesis. A fully parameterized FPGA based NoC simulation framework called YaNoC has been proposed. YaNoC supports the design space exploration

of various NoC topologies considering a rich set of router micro-architectural parameters. To simulate the larger topologies, the hard blocks of the FPGA, such as Block RAMs (BRAMs) and DSP blocks, have been employed to map the NoC router components such as FIFO buffers and the crossbar, respectively. Further, a lightweight NoC router architecture has been proposed to reduce the area utilization and improve network performance.

The thesis's initial work employs profiling to analyze the performance of the Booksim2.0 NoC software simulator with various design decision parameters and memory configurations. Various cache design parameters such as cache size, block size, and associativity have been considered to simulate the NoC topologies of Booksim2.0 to observe the effect of cache configurations. The hotspots of the Booksim2.0 simulator are identified, and software optimizations are employed to improve the performance of the Booksim2.0. To reduce the execution time of Booksim2.0, optimization methodologies such as vectorization and thread parallelization are employed. The OpenMP programming model is used for parallelizing and vectorizing the source code of Booksim2.0.

Due to high synchronization cost, the gain achieved in simulation speed is not significant. Higher simulation speed can be achieved by sacrificing the simulation accuracy to mitigate the complexity of synchronizations. FPGA-based simulators are becoming a promising approach for enhancing the speed of simulations. An FPGA-based NoC simulation acceleration framework called YaNoC, supporting design space exploration of standard and custom NoC topologies considering a full set of NoC router micro-architectural parameters, has been proposed. YaNoC is capable of designing custom routing algorithms, various traffic patterns. Obtained results show that the YaNoC consumes fewer hardware resources and is faster than the other FPGA based NoC simulation acceleration platforms.

Most of the state-of-the-art FPGA based simulators utilize soft logic only for modeling the NoCs, leaving out the hard blocks unutilized. The FPGA soft logic resources become a limiting factor when simulating a large NoC topology. Multiple FPGAs with off-chip memory can be employed to overcome the limitation of the FPGA resources.

The entire system becomes more complex and slow by using these approaches, leading to a reduction in the system's performance. Instead of having a multi-FPGA setup to simulate larger topologies, the hard blocks of an FPGA have been utilized efficiently to map the NoC router components. The functionality of the NoC router's buffer and crossbar switch are embedded in the BRAMs and the wide multiplexers of the DSP48E1 slices. A substantial decrease in the Configurable Logic Blocks (CLBs) utilization of NoC topologies on the FPGA is observed by embedding the functionality of the buffers and crossbar on the hard blocks of the FPGA compared to other state-of-the-art works.

Lightweight and high-performance NoC architecture is suitable for designing the heterogeneous systems to achieve area reduction and to improve the overall system performance. A low latency router with a look-ahead bypass called LBNoC has been proposed. The techniques such as single cycle router pipeline bypass, adaptive routing module, parallel virtual channel and switch allocation, combined flow control mechanism like virtual cut through, and wormhole switching are employed in designing the LBNoC router. The input buffer modules of NoC router are mapped on the FPGA BRAM hard blocks to utilize resources efficiently.

**Keywords:** Network-on-Chip, NoC, FPGA, Simulation acceleration, Performance analysis, DSP48E1, BRAM



# CONTENTS

<b>List of Figures</b>	<b>x</b>
<b>List of Tables</b>	<b>xiii</b>
<b>List of Abbreviations</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Description . . . . .	3
1.2 Thesis Contributions . . . . .	4
1.3 Thesis Organization . . . . .	5
<b>2 Background and Literature Review</b>	<b>7</b>
2.1 Field Programmable Gate Array . . . . .	7
2.2 NoCs : an overview . . . . .	9
2.3 Related work . . . . .	13
<b>3 Analysis and performance enhancement of BookSim 2.0 NoC simulator</b>	<b>19</b>
3.1 Methodology . . . . .	19
3.2 Profiling and Software optimization techniques . . . . .	21
3.3 Profiling, Performance Optimization Tools and Experimental method- ology . . . . .	23
3.4 Results and Discussion . . . . .	25
3.5 Optimization Strategies . . . . .	32
3.6 Summary . . . . .	38
<b>4 YaNoC - An FPGA based NoC simulation acceleration framework</b>	<b>39</b>
4.1 Introduction . . . . .	39
4.2 YaNoC - Design and Implementation . . . . .	40
4.3 Design of Mesh and Diagonal Mesh (DMesh) topologies . . . . .	46
4.4 Experimental Results . . . . .	54

4.5	YaNoC vs. State-Of-The-Art . . . . .	63
4.6	Summary . . . . .	64
<b>5</b>	<b>Mapping the NoC router components on the Hard-blocks of the FPGA</b>	<b>65</b>
5.1	Introduction . . . . .	65
5.2	NoC Router Architecture . . . . .	67
5.3	Block RAMs as the buffers . . . . .	68
5.4	DSP48E1 tile as the Crossbar Switch . . . . .	70
5.5	Results and Discussion . . . . .	74
5.6	Summary . . . . .	83
<b>6</b>	<b>Optimization of the NoC router for achieving low latency and area</b>	<b>85</b>
6.1	Introduction . . . . .	85
6.2	Related Work . . . . .	86
6.3	LBNOC-FPGA based Bypass NoC Framework . . . . .	88
6.4	Results and Discussion . . . . .	100
6.5	Comparison with the State-of-the-Art NoC architectures . . . . .	108
6.6	Summary . . . . .	115
<b>7</b>	<b>Conclusions and future work</b>	<b>117</b>
7.1	Conclusions . . . . .	117
7.2	Future work . . . . .	119
	<b>Bibliography</b>	<b>121</b>
	<b>Publications</b>	<b>135</b>



## LIST OF FIGURES

2.1	A generic architecture of Xilinx FPGA . . . . .	8
2.2	Architecture of the Configurable Logic Block (Xilinx Inc (2016)) . . . . .	9
2.3	An NoC router micro-architecture with M input/output ports, N-virtual channels at each input port and $M \times M$ Crossbar switch (Jerger and Peh 2009) . . . . .	11
2.4	Various NoC topologies . . . . .	12
3.1	BookSim 2.0 Execution time of k-ary n-dimensional Mesh networks(n=2,3 and 4). . . . .	20
3.2	Average I1 MPKI of BookSim 2.0 for Mesh topology.(MPKI is averaged over topology sizes mentioned in Table 3.1 and L1 cache configurations were varied as shown in Table 3.2) . . . . .	26
3.3	Average D1 MPKI of BookSim 2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 and L1 cache configurations were varied as shown in Table 3.2) . . . . .	28
3.4	Average LL MPKI of BookSim 2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 and LL cache configurations were varied as shown in Table 3.2) . . . . .	30
3.5	CPI for BookSim 2.0 running various sizes of Mesh topology . . . . .	32
3.6	Cache misses before and after optimization . . . . .	33
3.7	Speedup achieved before and after optimization . . . . .	34
3.8	Simulation execution times before and after improvements . . . . .	37
3.9	Speedups with Mesh topology of varying sizes . . . . .	37

4.1	Architecture of the proposed YaNoC FPGA based NoC simulation acceleration framework . . . . .	41
4.2	(a)Flit types and (b)Packet structure used in experiments. (Time stamp field is useful in calculating the latency of a packet) . . . . .	42
4.3	Modified router architecture supporting Congestion aware adaptive routing . . . . .	43
4.4	A High-level block diagram of YaNoC consisting of Host PC connected to an FPGA Board. . . . .	45
4.5	Simulation framework flow . . . . .	46
4.6	Mesh and Diagonal Mesh topologies (Red and Green colors indicate the routes calculated by XY and novel shortest path XY routing algorithms) . . . . .	48
4.7	Interconnection of the Router 12 with other Routers in DMesh topology . . . . .	51
4.8	Load Delay graph of 6x6 Mesh and Torus Topologies under Random Permutation Traffic patterns (a)Buffer Depth=8 flits and (b)Buffer Depth=16 flits . . . . .	59
4.9	Load delay graph of 8x8 Mesh and Torus topologies under Bit complement traffic patterns (a)buffer depth=8flits and (b)buffer depth=16flits . . . . .	60
4.10	(a) Load delay graph of Fat tree with buffer depth 8 and 16 flits under Random permutation traffic pattern(b)Load delay graph for Mesh and DMesh topologies under Uniform traffic . . . . .	61
4.11	Load Delay graph of Mesh Topology under (a)Uniform and (b)Transpose traffic patterns . . . . .	62
5.1	Functional diagram of the proposed FPGA based NoC framework. (a) an NoC topology, (b)Processing Element (PE), (c)Proposed router architecture . . . . .	67
5.2	Architecture of the Xilinx BRAM hard block (Xilinx Inc 2019) . . . . .	68
5.3	Illustration of mapping the input ports to the BRAM based buffer . . . . .	69
5.4	Bypassing an empty Buffer . . . . .	69
5.5	Two DSP48E1 slices connected by dedicated cascade links form a single DSP tile (Xilinx Inc 2018) . . . . .	71

5.6	Illustration of mapping the input ports to the DSP48E1 based crossbar . . . . .	72
5.7	(a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the $6 \times 6$ Mesh and Torus topologies under CLB and BRAM-DSP based FIFO and crossbar implementation considering NN, RP, HS, TO and BC traffic patterns . . . . .	79
5.8	(a), (b), (c), (d) - Load vs Latency comparison (e) saturation throughput of the Mesh topologies employing proposed BRAM-DSP router arhchi- tecture and CONNECT, DART NoC architectures . . . . .	82
6.1	The overall architecture of LBNoC-framework implemented on Xilinx Zynq 7000 ZC702 SoC. The PS consists of two core ARM Cortex-A9 processors and the PL has Artix-7 FPGA . . . . .	89
6.2	Two clock cycle Low latency router architecture implemented in LB- NoC framework(The router is highly parameterized with combined VC and Switch allocation stages) . . . . .	90
6.3	The architecture of Input buffer employed in designing low latency router	91
6.4	Free VC availability check and count . . . . .	92
6.5	Request filter logic . . . . .	93
6.6	Parallel virtual channel and switch allocator . . . . .	94
6.7	Pipeline stages of conventional and LBNoC router architecture . . . . .	97
6.8	Proposed adaptive look-ahead routing module . . . . .	98
6.9	Flit structure employed in LBNoC Framework . . . . .	100
6.10	Performance comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns. . . . .	105
6.11	Throughput comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns. . . . .	107
6.12	Average packet latency comparison between LBNoC, CONNECT (Pa- pamichael and Hoe 2015) and ProNoC (Monemi et al. 2017) consid- ering different types of traffic patterns . . . . .	110
6.13	Throughput comparison of LBNoC, ProNoC and CONNECT NoC ar- chitecture . . . . .	111

6.14	Area, Frequency and Power utilization of various router architectures . .	112
6.15	Average packet latency comparison between LBNoC, SOTA(Stanford Concurrent VLSI Architecture Group. 2014), Shared-buffer (Soteriou et al. 2009) and PCA (Yan et al. 2015) considering different types of traffic patterns . . . . .	114
6.16	Throughput comparison of LBNoC, SOTA, Shared-buffer and PCA NoC architectures . . . . .	115

## LIST OF TABLES

2.1	Comparison of the proposed and the other FPGA-based NoC simulators	18
3.1	Experimental setup	25
3.2	Different L1 Instruction(I1), L1 Data (D1) and Last Level (LL) Cache Configurations Used In Experiments	25
3.3	Effect on misses due to various I1 and D1 cache configurations	27
3.4	Effect on misses due to various Last Level (LL) cache configurations	30
3.5	Analysis of miss rates of Hotspot methods in BookSim 2.0	31
3.6	Unused functions in BookSim 2.0 source code	33
3.7	Replacing post-increment operator by pre-increment operator	35
3.8	Identifying the memory access pattern of BookSim 2.0 source code	36
4.1	Configurable router architectural parameters	40
4.2	Experimental setup details	54
4.3	Resource utilization of $6 \times 6$ (36 node)Mesh and Torus topologies under various configurations of Flit Width(FW) and Buffer Depth (BD)	55
4.4	Resource utilization of $8 \times 8$ (64 node)Mesh and Torus topologies under various configurations of Flit Width (FW) and Buffer Depth (BD)	55
4.5	Resource utilization of 56 node Fat tree topology under various configurations of Flit Width (FW) and Buffer Depth (BD)	56
4.6	Resource utilization of a Single Router	57
4.7	LUT Utilization of 5 and 9 Port Router Components	58
4.8	Synthesis results of YaNoC on Artix-7 FPGA device (XC7A100T, speed-3)	58

4.9	Synthesis results of 36-Node Mesh based topology on Artix-7 FPGA device (XC7A100T, speed-3) . . . . .	61
4.10	Resource utilization of CONNECT and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for $6 \times 6$ Mesh and DMesh topologies . . .	63
4.11	Resource utilization of DART and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for $3 \times 3$ Mesh topology . . . . .	64
5.1	4:1 Multiplexer operating signals based on the grant signals from the arbiter . . . . .	72
5.2	DSP48E1-I slice configuration based on the arbiter encoded signal . . . . .	72
5.3	DSP48E1-II slice configuration based on the arbiter encoded signal . . . . .	73
5.4	Experimental setup details . . . . .	75
5.5	Resource utilization of NoC Router considering CLB and BRAM-DSP mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA . . . . .	75
5.6	Resource utilization of $6 \times 6$ Mesh topology with CLB and BRAM-DSP mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA with XY routing . . . . .	77
5.7	Resource utilization of $6 \times 6$ Torus topology with CLB and BRAM-DSP48E1 mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA with XY routing . . . . .	77
5.8	FPGA synthesis results of the $6 \times 6$ Mesh topology considering the proposed BRAM-DSP implementation and CONNECT's implementation on Artix 7 (XC7A100T) board with BD=6 and FW=64 . . . . .	80
5.9	Hardware utilization results of the $3 \times 3$ Mesh topology with proposed BRAM-DSP implementation and DART's implementation on Artix 7 (XC7A100T) FPGA . . . . .	81
5.10	Features supported in the proposed BRAM-DSP NoC architecture and the other state-of-the-art NoC architectures . . . . .	83
6.1	The conventional allocator. V and P represent number of VCs per port and number of ports . . . . .	92

6.2	The proposed parallel allocator. V and P denotes number of VCs per port and number of ports . . . . .	95
6.3	Experimental setup details . . . . .	100
6.4	FPGA memory buffers using three implementation alternatives with constant flit width of 32-bit. . . . .	101
6.5	FPGA memory buffers using three implementation alternatives with constant buffer depth of 15 flits. . . . .	101
6.6	Synthesis results of various configurations of Input buffer in LBNoC router with 64-bit flit width . . . . .	102
6.7	Synthesis results of various configurations of Input buffer in LBNoC router with 128-bit of flit width . . . . .	102
6.8	Synthesis results of merged FIFO buffers at each input port and Conventional FIFO buffers . . . . .	103
6.9	Synthesis results of Queue of free VCs selection and Conventional VC allocator implementation . . . . .	103
6.10	Synthesis results of Full and Decomposed Crossbar with IN/OUT ports	104
6.11	Synthesis results of Mesh topology of size $4 \times 4$ and $5 \times 5$ with various configuration of input parameters . . . . .	104
6.12	Resource utilization and Maximum operating frequency of Different NoC configurations considering $4 \times 4$ mesh topology . . . . .	108
6.13	Resource utilization and Maximum operating frequency of Different NoC configurations considering $4 \times 4$ mesh topology . . . . .	113





## LIST OF ABBREVIATIONS

<b><u>Abbreviations</u></b>	<b><u>Expansion</u></b>
ASIC	Application Specific Integrated Circuit
CMP	Chip Multi-Processor
CPI	Cycles per Instruction
FF	Flip Flop
FPGA	Field Programmable Gate Array
IP Core	Intellectual Property Core
LFSR	Linear Feedback Shift Register
LUT	Look Up Table
MPKI	Misses per Kilo Instruction
MPSoC	Multiprocessor System-on-Chip
NoC	Networks-on-Chip
PE	Processing Element
SA	Switch Allocation
SoC	System-on-Chip
ST	Switch Traversal
TDM	Time Division Multiplexing
TG	Traffic Generator
TR	Traffic Receptor
VC	Virtual Channel



# CHAPTER 1

## INTRODUCTION

The multi-core architecture consists a number of cores that can perform application-specific tasks or more generic operations such as arithmetic logical operations. State-of-the-art applications require a more number of cores on a single chip to perform a wide variety of computations. These cores have to be interconnected efficiently to achieve better communication and computational performance. In a point-to-point bus-based communication infrastructure, the access to the bus is managed by the control logic. Also, data can only be transmitted by a single core. Although the bus-based communication provides flexibility, there lacks the provision of scalability. With the advent of VLSI technology, thousands of cores can be integrated on a single chip. With more number of cores on a chip, the wire length between the cores increases in a bus-based architecture in turn, leading to increased power consumption. The bus-based systems offer a very low throughput as only a single core can be accessed at a time.

The NoC has become the tangible on-chip communication technique (Benini and De Micheli (2002); Dally and Towles (2001)). A network of routers are used to interconnect the cores in the NoC paradigm. These routers communicate with one another, employing the packet-switching mechanism. The lightweight protocols used in the data networks can be employed in the NoC scenario. Hence, NoCs have been employed as the on-chip interconnect in many of the state-of-the-art heterogenous application specific architectures, multi-core processors, and MPSoCs (Akopyan et al. (2015); Ax et al. (2018); Balkind et al. (2016); Bohnenstiehl et al. (2017); Chen et al. (2017); Luo et al.;

Sodani et al. (2016)).

The time taken to model and evaluate large-scale NoC designs has to be less as thousands of cores are targetted in the near future. These results help in exploring the performance characteristics along with the effect on the overall system. The system architects will be able to understand the impact of various NoC design parameters before chip fabrication by reducing the total cost. NoC researchers rely upon the cycle-accurate software power and performance simulators (viz. Orion (Kahng et al. (2012)), Garnet (Agarwal et al. (2009)), SICOSYS (Puenta et al. 2002), Noxim (Catania et al. 2016), BookSim2.0 (Jiang et al. 2013)) to explore the microarchitectural design space of on-chip networks.

To offer speed, accuracy, completeness, flexibility, and usability, the computer system simulators are commonly implemented in software. A continuous advance in the computer system complexity can be observed over time rapidly at a faster pace than the growth in the performance of the computers. As a result, computer simulation performance is declining compared to the next generation of the computer system being simulated. For example, software simulators tend to become slower when the number of cores has been increased. This situation is called as the *simulation wall* (Angepat et al. (2014)). Li-Shiuan Peh and Dally (2001); Ogras et al. (2010) propose the analyticals model which are extremely fast. But, in many cases, these analytical models can be considerably inaccurate.

The fast and precise software simulators provide a platform for performing design space exploration of various NoC architectures. Employing techniques such as thread-level parallelism with these simulators to improve the simulation speed is difficult due to high synchronization cost. Higher simulation speed can be achieved by sacrificing the simulation accuracy to mitigate the complexity of synchronizations (Prasad et al. 2019). Next generation computer systems make extensive use of hardware-level parallelism to achieve high performance. A quick and advantageous solution to software only simulators is to implement FPGA based simulator accelerators which provide the hardware-level parallelism needed in the precise simulation of the computer systems. FPGAs are programmable devices composed of the humongous number of lookup ta-

bles interconnected with each other. Any arbitrary logic functions can be realized using these lookup tables. Contrary to the hard-wired ASICs, which map the silicon's logic permanently, the hardware of FPGAs can be reconfigurable based on the increments being made to the design like the software development. Thanks to its reprogramming ability, FPGAs allow the implementation and maintenance of hardware at lower cost and time than a dedicated integrated circuit would have needed. To improve the performance of the simulations, FPGA-based simulators have been proposed (Kamali and Hessabi (2016); Lotlikar et al. (2011); Papamichael (2011); Wang et al. (2014); Wolkotte et al. (2007)). Various activities of the software simulator are parallelized and executed on the FPGA fabric resources to achieve a better speed up.

## 1.1 PROBLEM DESCRIPTION

A simulation platform that is flexible, fast, and robust is needed for the realization of the NoC architectures. The NoC community uses the cycle-accurate software simulators for the design space exploration. The NoC parameters such as topology, routing algorithm, flow control, and router micro-architecture, including buffer management and allocation schemes, can be analyzed using these simulators. Since thousands of cores have a significant role in the near future many-core architectures, the large scale NoC designs have to be modeled and evaluated to analyze the performance parameters and effect on the system. As the number of simulated cores increases, software simulators tend to become slower due to the *simulation wall* phenomenon. A crucial hurdle in the design and evaluation of large-scale NoC is the lack of fast simulators for modeling, which can deliver a high level of accuracy.

### 1.1.1 Research Objectives

1. Study and performance enhancement of NoC architecture simulator.
  - Analysis and software optimizations for NoC simulator.
  - Comparative study of optimized NoC simulator with existing hardware-based NoC simulator.
2. Implementation of fully parameterized FPGA accelerated NoC architecture sim-

ulator.

- Employing various Hard blocks of the FPGAs for efficient resource utilization.
- Optimization of NoC router architecture to reduce the area utilization and improve network performance.

## 1.2 THESIS CONTRIBUTIONS

Contributions from this thesis are listed below:

1. Analysis of the performance of the BookSim 2.0 NoC simulator with various memory configurations to observe the effect on the speed of the simulation by using the profiling techniques considering various topologies and configurations of the NoC router components. Various software optimization techniques are employed to improve the performance of the BookSim 2.0 simulator.
2. A highly configurable FPGA based NoC simulation acceleration framework called YaNoC is developed for design space exploration of various NoC configurations. Various topologies considering several router micro-architectural parameters can be simulated using YaNoC.
3. The hard blocks of FPGA, such as DSP48E1 and BRAM blocks are used efficiently to map the functionality of the router micro-architectural components such as buffers and crossbar. Comparison of the proposed BRAM and DSP48E1 based router architecture with the other state-of-the-art FPGA based NoC router architectures.
4. A router architecture with an optimized area consuming less power and providing high performance for NoCs in FPGA is proposed. The proposed architecture is verified using a framework called LBNoC. LBNoC is capable of design space exploration of topologies considering various configurations.

### 1.3 THESIS ORGANIZATION

The thesis is organized as follows:

**Chapter 2: Background and literature review** introduces the FPGAs and NoC concepts briefly and summarizes the survey of NoC software simulators and methodologies employed for the FPGA based simulation acceleration.

**Chapter 3: Study and performance enhancement of NoC architecture simulator** presents the profiling techniques and software optimization techniques employed for improving the performance of BookSim 2.0 software simulations.

**Chapter 4: YaNoC - An FPGA based NoC simulation acceleration framework** presents an FPGA based NoC simulation acceleration framework called YaNoC. The framework is analysed considering various NoC configuration and topologies. A novel congestion aware routing algorithm has been proposed and a comparative analysis with the existing algorithms has been performed.

**Chapter 5: Mapping the NoC router components on the Hard-blocks of the FPGA** presents the techniques of mapping the NoC router components on the hard-blocks of the FPGA. The NoC router's crossbar has been efficiently mapped on the DSP48E1 blocks and the FIFO buffers are mapped on the BRAM blocks of the Xilinx FPGA. A comparison with soft-logic only implementation and the hard-block mapping has been carried out in this chapter.

**Chapter 6: Optimization of the NoC router for achieving low latency and area** presents an optimized NoC router architecture called LBNoC. Various techniques employed to reduce the latency and the area of the NoC router are detailed in this chapter.

In **Chapter 7: Conclusions**, the contributions of this thesis, along with some important conclusions have been summarized.





## CHAPTER 2

### BACKGROUND AND LITERATURE REVIEW

In this chapter, a brief background on FPGAs has been provided. Later, we discuss the basic concepts of NoCs and explain various factors influencing the performance of the same. Finally, various software and FPGA based NoC simulators have been reviewed.

#### 2.1 FIELD PROGRAMMABLE GATE ARRAY

The state-of-the-art FPGAs mainly contain:

- Configurable logic blocks
- Routing fabric
- Input/output blocks
- Embedded hard blocks

Fig. 2.1 shows the generic architecture of an FPGA. The Configurable logic blocks (CLBs) are placed in an island fashion. The CLBs are interconnected through the flexible routing fabric. The I/O blocks are placed at the borders of the grid to enable the off-chip communications. The DSP blocks and the Block RAMs constitute the embedded hard blocks. These hard blocks are placed in the columnar configuration and are spread across the FPGA. Further, the DSP blocks can be cascaded with one another through the dedicated interconnections.

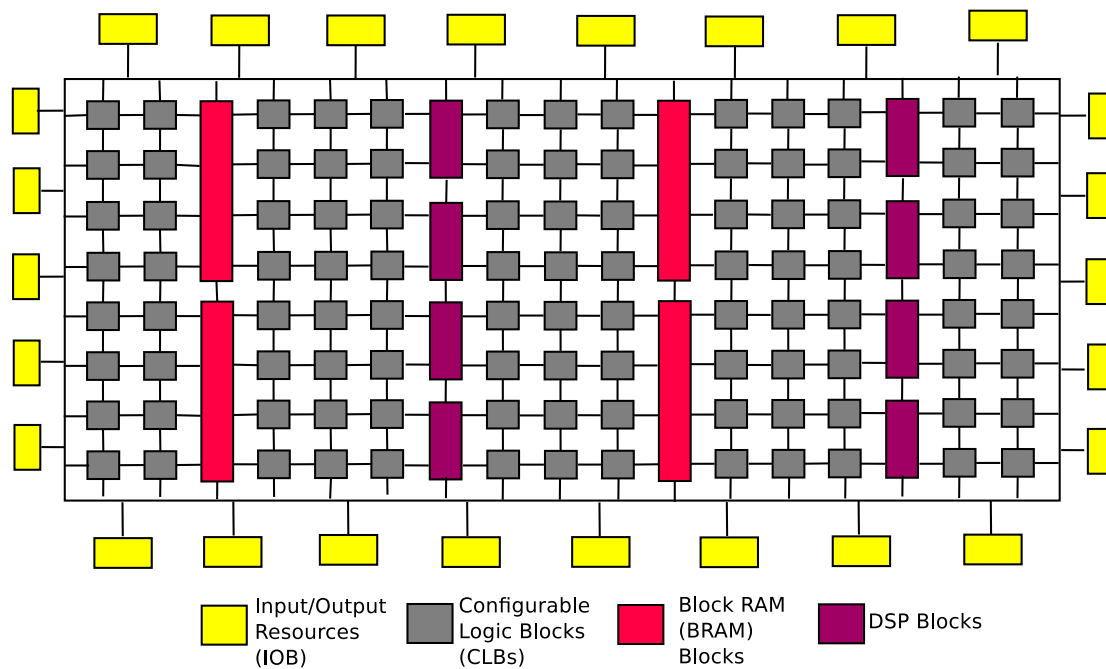


Figure 2.1: A generic architecture of Xilinx FPGA

### 2.1.1 Configurable logic blocks

All the sequential and combinational logic can be implemented employing the CLBs. In the Xilinx 7 series FPGAs, each CLB contains a pair of slices which can work independently (Xilinx Inc (2016)). Each slice is made up of four Lookup Tables (LUTs), eight Flip-flops (FFs), multiplexers and the carry logic. Various functions can be generated by combining LUTs through the multiplexers. Fig. 2.2 shows the structure of a CLB.

### 2.1.2 Routing fabric

Connections between various components of the FPGAs is facilitated through the routing fabric. The routing fabric consists of the connection boxes and the switch boxes.

### 2.1.3 Input/output resources

The I/O resources in the FPGA are placed at the periphery to connect with the external entities. A wide variety of I/O standards along with high speed serial standards are supported by the FPGAs. Further, the I/O resources are clubbed into banks.

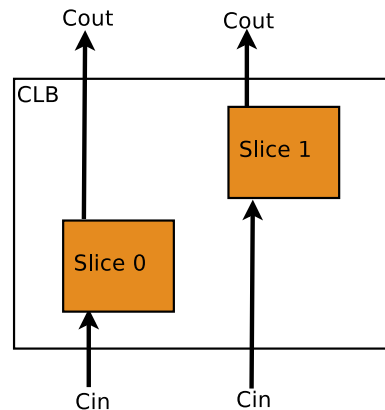


Figure 2.2: Architecture of the Configurable Logic Block (Xilinx Inc (2016))

#### 2.1.4 Embedded hard blocks

Various arbitrary arithmetic operations can be implemented by programming the CLBs. Also, the CLBs can be used as small memories. But, the CLBs consume more FPGA area and become slower when implementing many such functions. To provide the flexibility for the designers, modern FPGAs embed the optimized hard-wired ASIC like blocks such as Block RAMs (BRAMs) and DSPs to perform specific functionalities. Mapping the functions on embedded hard-blocks improves the performance and reduces the power consumption in comparison with the same functions mapped on the CLBs (Ronak and Fahmy 2016). By employing the hard-blocks in the design, a reduction in the usage of CLB resources can be observed. The BRAMs (Xilinx Inc 2019) and DSP blocks (Xilinx Inc 2018) are discussed in detail in Chapter 5.

## 2.2 NOCS : AN OVERVIEW

The NoC architecture is composed of links, routers, and network interface (Benini and De Micheli (2002); Dally and Towles (2001); Guerrier and Greiner (2000)). The communication between the routers takes place in the form of *packets*. In NoC, packets can be further divided into several flow control units (*flits*). Flit is the largest number of bits of data that can be transmitted between two routers at a given point of time. *Flow control* determines how the packets are transmitted between two nodes/cores. Specifically, flow control determines when the flits can be forwarded from one router to the next router. There exist various kinds of flow control mechanisms such as store-and-forward

packet switching (Dally and Towles (2004)), virtual cut-through (Kermani and Kleinrock (1979)), and wormhole switching (Dally et al. (1986)) to transmit the flits. The Virtual Channels (VCs) mechanism has been implemented in Dally (1992) to prevent the deadlock and to achieve better performance.

The performance of an NoC architecture depends on various factors such as topology, the configuration of the router micro-architectural components, and routing algorithms. All these entities are explained below:

### 2.2.1 Links

A communication link is composed of a set of wires connecting two routers in the network. Channel, and link mean a group of wires connecting two entities. Typically, an NoC link has two physical channels making a full-duplex connection between the routers.

### 2.2.2 Router

A router is composed of a set of input ports and output ports, switching matrix connecting the input port to output port, and a local port to access the processing element connected to this router. Fig. 2.3 depicts the micro-architecture of the router. Following are the pipeline stages of router architecture (Dally and Towles (2001, 2004); Pande et al. (2005)).

*Buffer write:* On arrival of head flit at an input port, it is first decoded and is buffered according to its input Virtual Channel (VC).

*Route computation:* Head flit contains route information. At this stage, route computation is performed to determine the output port for the packet. To this end, the head flit indicates the VC that it belongs to, the VC state is updated, and the next output port is computed based on routing algorithms. The routing algorithms can be categorized into oblivious and adaptive algorithms (Jerger and Peh (2009)).

*Virtual Channel allocation:* The head flit arbitrates for the available VC on its output port.

*Switch allocation:* The header flits are arbitrated for accessing the output port.

*Switch traversal:* On winning the arbitration, the flit moves to the switch traversal stage,

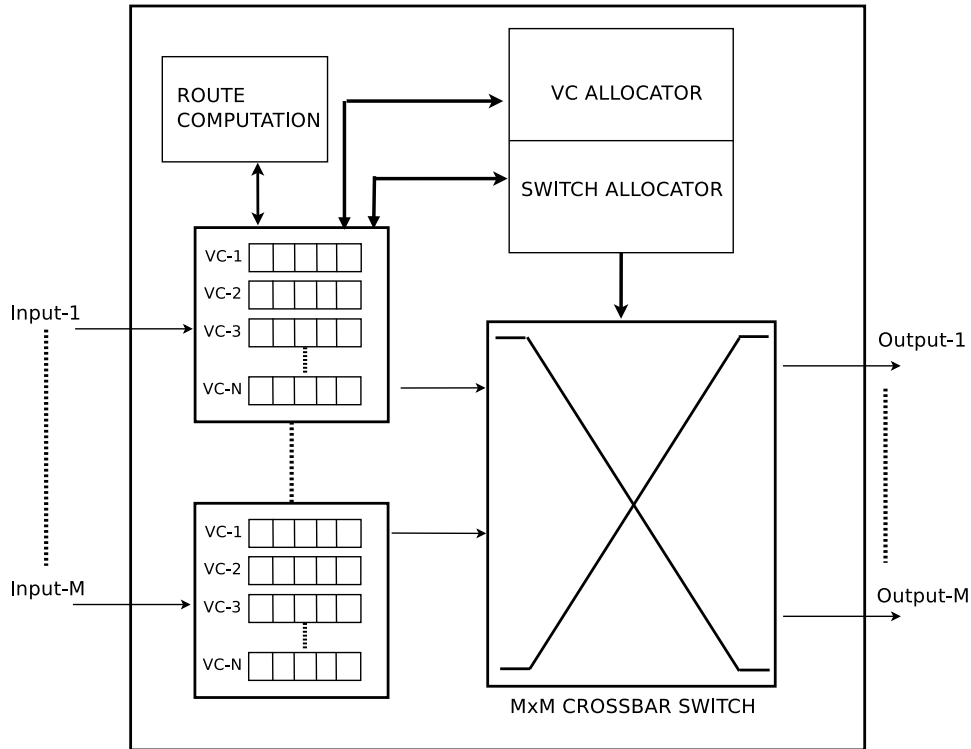


Figure 2.3: An NoC router micro-architecture with  $M$  input/output ports,  $N$ -virtual channels at each input port and  $M \times M$  Crossbar switch (Jerger and Peh 2009)

where it traverses the crossbar and is transmitted on the output port.

*Link traversal:* Flits travel to the next node.

### 2.2.3 Network adapter

A Network adapter (NA) or network interface (NI) is the third building block of NoC. The Processing Elements (PEs) and the network are connected logically, employing the NA. Each PE may have a distinct interface protocol with respect to the network. Separation of computation and communication is done with the help of NA.

### 2.2.4 Topology

The PEs are arranged in a particular pattern with the help of wires/links. This arrangement is called the *Topology*. The PEs in the NoC can be interconnected in various topologies such as Mesh-based, Tree-based, and user-specific architectures (Bjerregaard and Mahadevan 2006; Pande et al. 2005) as shown in Fig. 2.4.

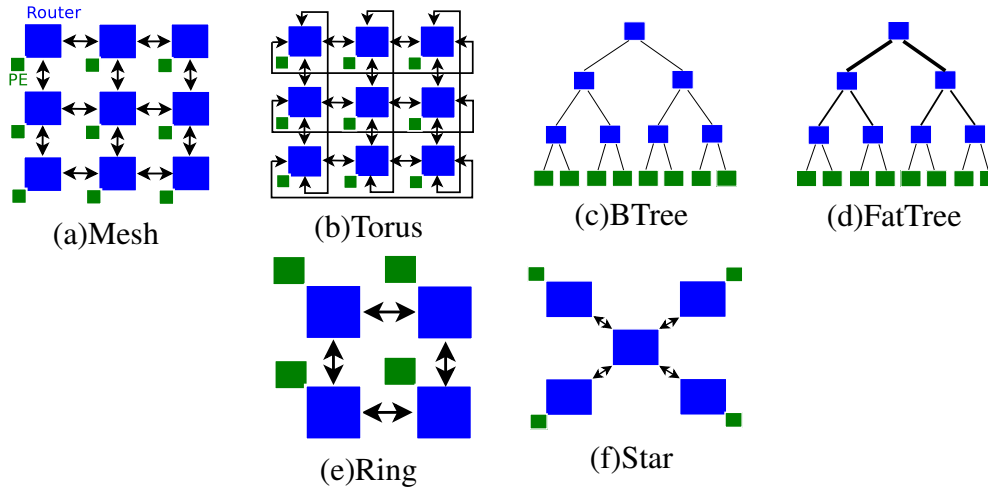


Figure 2.4: Various NoC topologies

### 2.2.5 NoC performance parameters

Different NoC architectures are compared by using a standard set of performance metrics such as throughput, latency, energy/power, and area overhead (Jerger and Peh 2009).

#### 2.2.5.1 Throughput

It is defined as the maximum traffic accepted by the network, that is, the maximum amount of information delivered per time unit. For message passing systems, message throughput can be defined as  $TP$ , (Equation 2.1):

$$TP = (Total\ Messages\ completed * Message\ length) / (Number\ of\ PE\ blocks * Total\ Time) \quad (2.1)$$

Here, *Total Messages completed* means that the whole message has arrived at the destination node; *Message length* is the total number of flits; *Number of PE blocks* is the number of functional cores involved in communication; *Total Time* is the difference of time between the first flit generated, and the last flit received.

#### 2.2.5.2 Average Packet Latency

The latency is defined as the cycle time required by the packet to travel from source processing elements to the destination processing elements. The average packet latency is

given by the equation 2.2:

$$Avg_{lat} = 1/N \sum_{i=1}^N L_i \quad (2.2)$$

where N refers to the total number of flits accepted by the all destination nodes and  $L_i$  refers to the latency of the  $i^{th}$  flit received by its destination processing element.

### 2.2.5.3 Area

In the NoC architecture design, the presence of the input buffers, Switch allocator, crossbar switch and the interfaces results in the silicon area overhead. The area of NoC architecture is given by equation 2.3 and 2.4:

$$NoC_{Area} = Routers_{Area} + Links_{Area} \quad (2.3)$$

$$Router_{Area} = IB_{Area} + RCL_{Area} + Crossbar_{Area} \quad (2.4)$$

Where IB is Input Buffer of NoC router, RCL is the Router Control Logic such as routing logic, VC and Switch allocation logic.

### 2.2.5.4 Power

The total power consumed by the NoC architecture can be broken down into router, links, input/output and clock distribution power. The router power consumption includes FIFO buffer, routing algorithm, allocator and crossbar switch power. The total power of NoC architecture is given by 2.5 and 2.6:

$$P_{NoC} = P_{router} + P_{link} + P_{Interfaces} + P_{clk} \quad (2.5)$$

$$P_{router} = P_{FIFO} + P_{routellogic} + P_{allocator} + P_{crossbar} \quad (2.6)$$

## 2.3 RELATED WORK

In this section, state-of-the-art in the area of NoC software simulators and FPGA based emulators have been introduced.

### 2.3.1 Software simulators

Area, performance and power are the important design decision parameters in designing NoCs. Hence, there is a need to estimate the NoC area, performance and power consumption in the early stages of the design. Researchers rely upon simulators to evaluate the power and performance of the NoCs. Many of the full-system simulators such as Gem5 (Binkert et al. 2011) and MARSS (Patel et al. 2011) provide the flexibility to study the NoCs with many-core systems and other components. The full system simulators are cycle-accurate. But, when there are a large number of simulated cores, the time taken is excessive. ZSim (Sanchez and Kozyrakis (2013)) is a parallelized full-system simulator that proposes a technique in which the simulation is divided into several small intervals of many thousand cycles. During the simulation of the processor cores parallelly, the resource contentions are ignored, and the zero load latency has been employed for all types of memory accesses. By doing so, ZSim achieves speed by sacrificing accuracy.

Orion simulator (Kahng et al. 2012) includes a set of the architectural area and power models for on-chip interconnection routers. A classic five-stage pipelined router with virtual channel flow control has been modeled in GARNET (Agarwal et al. 2009). GARNET has been integrated with Gem5 (Binkert et al. 2011) full system simulator. NoC micro-architectural details such as input buffers, routing logic, allocators, and the crossbar switch are modeled. The workload traffic running on Gem5 can be analyzed using GARNET.

BookSim 2.0 (Jiang et al. 2013) is a cycle-accurate simulator. It is flexible in terms of modeling network components. A large set of network parameters that are configurable such as routing algorithm, topology, flow control, and router micro-architecture, are implemented. Noxim (Catania et al. 2015, 2016), is another NoC simulator which is implemented in SystemC. Noxim is capable of simulating wireless NoCs. NOCulator (CMU-SAFARI 2018), Access Noxim (Access IC Lab 2018) and VisualNoC (Wang et al. 2016) are other popular NoC simulators. A modular, open-source NoC simulator based on OMNeT++ (Varga 1999) has been presented in HNOCS (Ben-Itzhak et al. 2012). Heterogeneous NoCs with variable link capacities and the number of



VCS per unidirectional port are supported in HNOCS. Statistical measurements such as latency in between source and destination, throughput and VC acquisition latencies are provided by HNOCS. An NoC simulator calculating the accurate cycle timings with wormhole switching has been proposed in Ting-Shuo Hsu et al. (2015). The flit propagation model calculating the flit timings at I/O ports of FIFOs and switches play a vital role in Ting-Shuo Hsu et al. (2015).

The performance of computer simulation is ever decreasing relative to the next generation of computers being simulated due to the phenomenon of *simulation wall* Angepat et al. (2014). Hence, there is a need for simulation techniques which can yield the results quickly.

### 2.3.2 FPGA based NoC simulation frameworks

Due to their prominent features supporting highly parallel operations, reconfigurability and programmability, FPGAs have become a vehicle for NoC simulation acceleration.

Employing the FPGA fine-grain parallelism, several works such as ProtoFlex (Chung et al. (2009)), RAMP Gold (Tan et al. (2010)), RAMP White (Chiou et al. (2007)), RAMP Red (Wee et al. (2007)) have shown that a remarkable improvement in the emulation performance can be achieved.

In Lotlikar et al. (2011), an NoC emulation environment on FPGA called AcENoCs has been proposed. Both of the software and hardware components of the FPGAs have been utilized by AcENoCs. The Microblaze softcore processor hosts Traffic generators, clock generation, and traffic sinks. The generation of the clock on software is flexible but potentially slow. The hardware platform is the network-on-chip to be emulated. AcENoCs supports the design of Mesh topology only.

Fast Interconnect Simulation Techniques (FIST) has been proposed in Papamichael et al. (2011). The time consuming detailed NoC models of full system simulators can be replaced by FIST as it incorporates a fast and simple packet latency estimator. The ideas from execution-driven and analytical network modeling simulation models are combined to build FIST. The latency estimation of a packet is done by determining the routers traversed by the packet. Latencies depending on the load, are then added to give

the packet latency. Due to this approach, FIST is not appropriate for a thorough analysis of networks.

An FPGA based NoC emulation supporting the direct implementation and virtualized implementation has been proposed in Papamichael (2011). The NoCs to be emulated are directly implemented on the FPGA. A Time Division Multiplexing (TDM) approach is employed to support the virtualized implementation of NoCs. The traffic tables needed for the simulation are stored in the off-chip DRAM. Thus, the overall system performance can be confined by the latency and bandwidth of the DRAM access.

An FPGA-based NoC emulator has been proposed in DART (Wang et al. 2014). Global interconnect across all the nodes is provided. Any topology can be emulated by DART, leaving out the resynthesis of design utilizing these global interconnects and employing a software tool by configuring the routing tables properly. Most of the FPGA resources are consumed by the global interconnect. DART minimizes the expense of global interconnect by clustering many nodes into a partition and employing a crossbar for the clusters instead of a full crossbar for all nodes. This leads to the complex hardware, and the size of the routing tables becomes larger on increasing the number of nodes. With a large number of nodes, the off-chip DRAM has to be used to store the routing tables, which becomes unavoidable.

An FPGA emulation platform called Ultra-Fast has been proposed in Thiem Van Chu et al. (2015). Ultra-Fast employs two methods enabling swift emulations of larger NoC architectures on a single FPGA. The time of network being simulated is decoupled from the time of traffic generation units to model the Synthetic workloads accurately. To emulate the entire network utilizing more physical nodes, the TDM approach has been employed. Authors have considered Mesh topology with the look-ahead XY routing and the credit-based flow control.

AdapNoC, a fast and flexible FPGA based NoC simulator, has been proposed in Kamali and Hessabi (2016). Various router micro-architectural parameters are configurable in AdapNoC. Transplantable Traffic Generators and Receptors running on the software side are supported. To simulate larger topologies and reducing the simulation

time drastically, Dual clock virtualization methodology has been employed. AdapNoC supports Adaptive Toggle Dimension Order Routing (ATDOR) as a known adaptive routing algorithm. Only Mesh and Torus topologies are supported by AdapNoC. DuCNoC (Kamali et al. (2018)) is another version of AdapNoC. DuCNoC employs Xilinx Zynq-7000 SoC. Two soft-core ARM processors are used to model the traffic generator and traffic receptors. Employing an approach similar to DuCNoC, an FPGA based NoC emulator has been proposed in Drewes et al. (2017). An NoC of size  $8 \times 8$  can be emulated in Drewes et al. (2017).

An NoC RTL generator called CONNECT has been proposed in Papamichael and Hoe (2015). For any topology design, the route information of packets is stored in the routing table. Packets are routed from source to destination using these tables. CONNECTs implementation of the NoC topology uses LUTs for designing input memory buffers. This causes high resource usage when using wide buffer sizes.

Bufferless customized unidirectional Torus topology with Deflective routing has been implemented in Hoplite (Kapre and Gray (2015)). By incorporating bufferless deflective routing, the hardware required by buffers can also be saved, reducing power consumption. Crossbar's cost is reduced considerably by doing so as the unidirectional Torus topology accepts packets only from two neighboring ports and a local port, thus reducing the crossbar complexity. Hoplite supports the design of unidirectional, bufferless, deflection-routed torus networks only. Hoplite DSP (Chethan and Kapre (2016)) extends the concepts of Hoplite to map the routers on the DSP48E1 blocks of the Xilinx FPGA.

Table 2.1 provides a comparison of the state-of-the-art FPGA based NoC simulators and the proposed work. As seen from Table 2.1, our work supports various standard NoC topologies and also provides the provision for designing the custom topologies. The table-based routing algorithm has been implemented to design the custom topologies and the congestion-aware adaptive routing algorithm to achieve better performance. Various hard blocks of FPGA, such as DSP48E1 and the BRAMs, are used efficiently to map the NoC router micro-architectural components.

Table 2.1: Comparison of the proposed and the other FPGA-based NoC simulators

FPGA based NoC simulation framework	Topology		Routing			Router micro-architecture	FPGA	Hw/Sw Interface	FPGA components for mapping router components				
	Mesh based	Tree based	Custom DoR based	Table based	Adaptive ports				VC	Pipeline stages	CLB	DSP	BRAM
AcENoCs Lotlikar et al. (2011)	Yes	No	No	Yes	No	5	2	1	V5	RS232	Yes	No	No
Papamichael Papamichael (2011)	Yes	No	No	Yes	No	4/8/12	2/4/8	-	V5	UART	Yes	No	No
Zhang Zhang et al. (2013)	Yes	No	No	Yes	No	5	2/4	3+	V6	-	Yes	No	No
DART Wang et al. (2014)	Yes	Yes	No	Yes	No	upto 8	upto 4	5	V6	PCIe	Yes	No	Yes
UltraNoC Thiem Van Chu et al. (2015)	Yes	No	No	Yes	No	5	1/2	4/5	V7	-	Yes	No	Yes
AdapNoC Kamali and Hessabi (2016)	Yes	No	No	Yes	Yes	5	upto 4	4/5	V7	PCIe	Yes	No	Yes
Hoplite DSP Chethan and Kapre (2016)	Unidirectional Torus	No	No	Yes	No	3	Bufferless	-	V7	-	Yes	Yes	No
SRNoC Xu et al. (2019)	Yes	Yes	No	Yes	Yes	5	upto 8	5	V7	PCIe	Yes	No	Yes
Proposed work	Yes	Yes	Yes	Yes	Yes	3/5/7	2/4/8	2/5	Artix 7 Zynq 7000	USB-UART	Yes	Yes	Yes

## CHAPTER 3

### ANALYSIS AND PERFORMANCE ENHANCEMENT OF BOOKSIM 2.0 NOC SIMULATOR

The Network-on-Chip (NoC) is now an integral component in the MPSoCs and CMPs (Dally and Towles 2001). The communication time can influence the total turnaround time of the application significantly (Pande et al. 2005). NoC researchers have relied on cycle-accurate power and performance software simulators (viz. Orion (Kahng et al. 2012) (Kahng et al. 2015)), GARNET (Agarwal et al. 2009), Noxim (Catania et al. 2015) (Catania et al. 2016)), SICOSYS (Puenta et al. 2002), BookSim 2.0 (Jiang et al. 2013)) to explore the micro-architectural design space of on-chip networks. Amongst these, BookSim 2.0 has emerged as one of the prominent NoC performance analysis tools.

In this Chapter, the performance of the BookSim 2.0 NoC simulator considering various memory configurations of the system is analysed. Also, the locations at which most of the execution time is spent during simulation are identified. Programming paradigms are applied to improve the performance of the simulator.

#### 3.1 METHODOLOGY

BookSim 2.0 offers network parameters such as topology, routing algorithm, flow control, and router micro-architecture, including buffer management and allocation schemes as input parameters for simulating NoC architectures. Simulating large NoC architectures take days together to complete. Hence, there is a need for fast design space

### 3. Analysis and performance enhancement of BookSim 2.0 NoC simulator

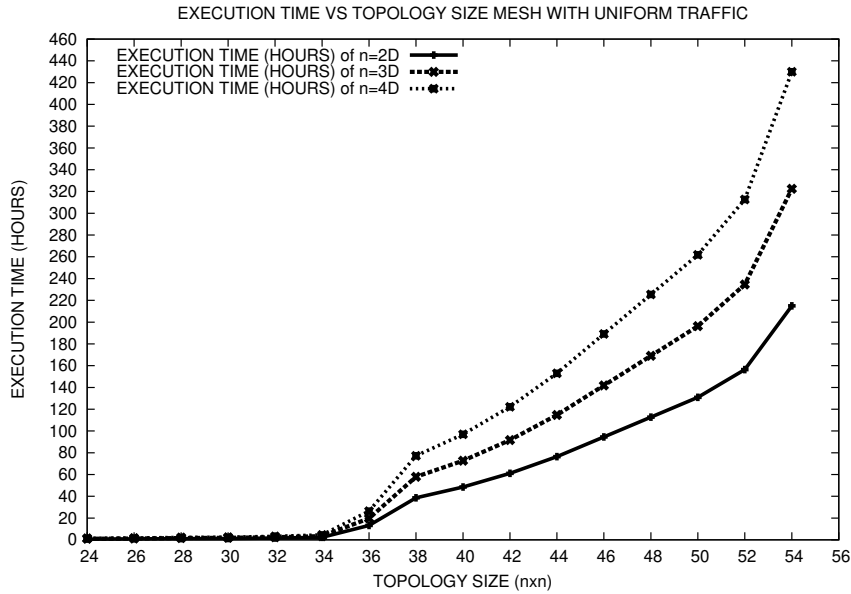


Figure 3.1: BookSim 2.0 Execution time of k-ary n-dimensional Mesh networks(n=2,3 and 4).

exploration of NoC architectures, which help designers to reduce the time and effort spent in the development of a common on-chip framework. From Fig. 3.1, it can be observed that the execution time of BookSim 2.0 simulator varies as the topology size is increased. It can be seen that the execution time varies from 6 seconds to 10 days simulating  $4 \times 4$  and  $54 \times 54$  NoC architectures of Mesh topology. By increasing the dimension from 2 to 3 and 4, the time taken can be even more. The increase in execution time could be because of cache behavior and the way memory is accessed. To analyze the cache behavior and memory access patterns, we use profiling methodology.

Profiling refers to the ability to measure an application’s performance and diagnose potential problems. Profilers can identify the Hotspots where the program spends most of its execution time and memory access patterns. Profiling BookSim 2.0 reveals the dependence of the cache and memory behavior on the input data. Valgrind (Nethercote and Seward (2007)) is used to map the cache and memory usage patterns of BookSim 2.0. Valgrind is a dynamic binary instrumentation framework for building dynamic binary analysis tools. Cachegrind tool of the Valgrind framework has been employed for profiling BookSim 2.0 considering various cache configurations.

Based on profiling, the best cache configuration in which the cache misses is mini-

num and memory access patterns of BookSim 2.0 which help in improving the performance, are identified. Further, we use software optimization techniques such as removal of unused functions, loop optimizations and pre-increment operator for non-primitive data types to minimize the cache misses.

To reduce the execution time of BookSim 2.0, optimization methodologies such as vectorization and thread parallelization are employed. The OpenMP programming model is used for parallelizing and vectorizing the source code of BookSim 2.0.

## **3.2 PROFILING AND SOFTWARE OPTIMIZATION TECHNIQUES**

The tools and principal works that have used profiling to study application behavior and works that have optimized the application for performance are discussed below. Optimizations based on program input, cache access behavior and memory reference patterns are listed.

### **3.2.1 Profiling based on program input**

Understanding the influence of input data on the overall performance of an application is a crucial aspect of software development. Frameworks have been proposed to dynamically estimate the size of the input to derive cost functions (Coppa et al. (2014b) Nistor and Ravindranath (2014)). Input-sensitive profiling (Coppa et al. (2014a)) discovers workload-dependent performance bottlenecks. The growth rate of an application is recorded as a function of input sizes to the routines of the application in the aprof tool (Coppa et al. (2014a)). Algorithmic profiler (Zaparanuks and Hauswirth (2012)) infers an empirical cost function by automatically determining the “inputs” to a program, by measuring the program’s “cost” for a given input. To achieve low overheads for deployment in data centers, instant profiling (Mahlke et al. (2013)) interleaves native execution and instrumented execution according to configurable profiling duration and frequency parameters. Causal profiling (Curtsinger and Berger (2015)) runs performance experiments to calculate the impact of any potential optimization by virtually speeding up code during program execution. Pipelined Profiling and Analysis on Multi-core Systems-PiPA (Zhao et al. (2008)) aims to reduce the cost of user-defined analysis tools in instrumentation by parallelizing dynamic program profiling in multi-core sys-

tems.

### 3.2.2 Profiling for cache performance

Prefetching and profiling can help computing systems better mitigate performance losses due to limited cache bandwidth. Cache profiling can improve a program's performance by focusing on the programmer's attention on problematic code sections and providing insight into appropriate program transformations. Several proposals exist to use profile based application-level knowledge to manage the contents of caches (Cherniack et al. (2003)). The Cachetor run-time profiling tool (Nguyen and Xu (2013)) identifies and reports operations generating invariant data values. Cachetor uses dynamic dependence profiling and value profiling to expose caching opportunities to improve program performance. Phase guided cache profiling (Sembrant et al. (2012)) has been used to model the cache miss ratio as a function of the cache allocation over time. The Pharo code profiler (Infante (2014)) addresses the problem of identifying memory savings opportunities by employing object caches in the context of the Smalltalk programming language. Pharo identifies and monitors instance creations and the mutations of these instances. Valgrind variants have been used to study the cache behavior of multimedia applications to optimize performance (Asaduzzaman and Mahgoub (2006)). The CProf cache profiling system (Lebeck and Wood (1994)) lets programmers identify hot spots by providing cache performance information at the source-line and data-structure level.

### 3.2.3 Memory usage based profiling

Memory accesses have a significant influence on the total application performance. Several profiling frameworks have been proposed to analyze memory accesses in applications and input dependent main memory growth patterns. The Toddler framework (Nistor et al. (2013)) implemented for Java applications, reports loops whose computation has repetitive and partially similar memory-access patterns across iterations. MemInsight (Jensen et al. (2015)) implements fine-tuned source-code instrumentation to provide time-varying analysis of the memory behavior of JavaScript applications. JSWhiz (Pienaar and Hundt (2013)) is a compiler extension for analyzing memory leaks in JavaScript programs. LeakChaser (Xu et al. (2011)) is a specification-based



technique that can capture unnecessary references leading to memory leaks. Reference propagation (Yan et al. (2012)) provides information specific to reference producers and their run-time contexts to reveal inefficiencies in the code. Data-centric profiling for parallel programs (Liu and Mellor-Crummey (2013)) has been used to measure memory access latency. Hardware counters are used to attribute latency metrics to variables and instructions.

#### **3.2.4 Techniques for minimizing Cache misses**

Code transformation techniques such as loop unrolling, loop fusion, loop distribution have been employed in Porterfield (1989) to minimize the cache misses of applications. In Kowarschik and Wei (2003), Data access optimizations that change the order of execution of the nested loops are used. These techniques improve the temporal locality of the cache, reducing the cache misses. In Song et al. (2003), techniques such as loop invariant code motion, loop unrolling and loop peeling have been demonstrated.

#### **3.2.5 Performance improvement of applications**

The performance of an application can be improved by using techniques such as vectorization and threading. Larsen et al. (2005) vectorizes operations in the important loops of a program to improve overall resource utilization, allowing for software pipelines with shorter initiation intervals. In Nie et al. (2010), two ways of exploiting the data parallelism in Java using vectorization are introduced. In Randall and Lewis (2002), OpenMP programming model has been employed to parallelize the Ant colony optimization algorithm.

### **3.3 PROFILING, PERFORMANCE OPTIMIZATION TOOLS AND EXPERIMENTAL METHODOLOGY**

Profiling has been employed for measuring the application performance, identifying Hotspots, and diagnosing potential problems. Valgrind (Nethercote and Seward (2007)) has been used for profiling BookSim 2.0. Cachegrind, one of the tools of Valgrind suite is used to simulate the behavior of a program with the cache hierarchy and branch predictor of the system. Cachegrind simulates a system with independent first-level (L1) instruction and data caches(I1 and D1), backed by a unified last-level cache(LL).

Callgrind - another tool of Valgrind suite has been employed to record call history of the functions in BookSim 2.0. KCachegrind - a GUI based tool is used for identifying the Hotspots of BookSim 2.0. Employing these tools, the best cache configuration in which the cache misses are minimum is identified. Also, memory access patterns of BookSim 2.0 which help in improving the performance, are identified.

Cppcheck (Daniel Marjamki (2011)) is used to detect the types of bugs that the compilers normally do not detect, such as unused functions. The techniques such as reversing loop iterations and replacing post-increment operator with pre-increment operator etc. have been adopted to reduce the cache misses. Further, vectorization and thread parallelization techniques are applied to improve the performance of BookSim 2.0. Intel Advisor suite (Intel Corporation (2017)) has been employed to identify the top time-consuming loops of BookSim 2.0. Based on these analyses, we employ the OpenMP programming model to parallelize the top time-consuming loops of BookSim 2.0.

#### **3.3.1 Experimental methodology**

The cache design and BookSim 2.0 configuration parameters considered for experiments in this work are shown in Table 3.1. Cache simulation has been performed considering the Cachegrind simulator. Various cache configurations shown in Table 3.2 are simulated using Cachegrind tool of Valgrind suite to analyze the cache behavior and memory usage of BookSim 2.0 considering various topology sizes. Based on these analyses, the best cache configuration with a minimum number of cache misses has been identified. The optimization techniques such as reversing loop iterations, removal of unused functions, and replacing post-increment operator with pre-increment operator have been adopted to reduce the cache misses.

Further, techniques such as vectorization, and thread parallelization are employed to speedup the simulation execution time of BookSim 2.0.

Table 3.1: Experimental setup

<b>System Configuration</b>	
Cache Hierarchy	(I1+D1)L1 and LL
L1 Cache size	32KB and 64KB
Last Level (LL) Cache size	512KB 4MB and 8MB
L1, LL Cache Line size	32B and 64B
Write Policy	Write Allocate
Page Replacement	LRU
L1 Associativity	2,4 and 8-way
LL Associativity	4,8 and 16-way
Tools used	Valgrind, Cachegrind, Kcachegrind, Intel Advisor and Cppcheck
<b>Network Configuration Input to BookSim 2.0</b>	
Topology Type	Mesh and Torus
Network size	$4 \times 4$ , $6 \times 6$ , ....., $30 \times 30$
Traffic Pattern	Uniform random
Number of Virtual Channels	8
Virtual Buffer Size	8
Packet Size	20 flits
Sample Period	1000 cycles
Maximum Number of Samples	10
Latency Threshold	$10^9$
Injection Rate	0.005
Routing Algorithm	Dimension Order Routing

Table 3.2: Different L1 Instruction(I1), L1 Data (D1) and Last Level (LL) Cache Configurations Used In Experiments

Sl No.	<b>L1 cache configuration</b>			<b>LL cache configuration</b>		
	I1 and D1 cache sizes	Associativity	Block Sizes	LL cache sizes	Associativity	Block Sizes
1	32KB/32KB	2,4,8-Way	32B,64B	512KB	4,8,16-Way	32B,64B
2	64KB/64KB	2,4,8-Way	32B,64B	8MB	4,8,16-Way	32B,64B

### 3.4 RESULTS AND DISCUSSION

#### 3.4.1 Identifying the best cache configuration

The performance of cache memory is studied considering various cache and topology sizes. 12 different cache configurations are employed for First level Instruction cache (I1), First Level Data cache (D1), and Last Level Cache (LL) for analyzing the effect of cache size, block size and associativity as shown in Table 3.2. BookSim 2.0 simulations are run for 2D Mesh topology of sizes ranging from  $4 \times 4$ ,  $6 \times 6$ , ... ,  $30 \times 30$ .

Cache misses are classified as Compulsory, Capacity, and Conflict misses. The

cache performance can be improved by reducing these misses. The compulsory misses can be minimized by increasing the block size. But, this may lead to an increase in conflict misses. The larger associative cache can be employed in order to minimize conflict misses. As the cache size increases, the capacity misses will be minimized as larger caches are available to store the program data. In Fig. 3.2 and 3.3, the values are obtained by computing the average MPKIs of 14 different network sizes considering all the cache configurations, as shown in Table 3.2 .

In Fig. 3.2 and 3.3, each bar represents a particular cache configuration. The values are obtained by averaging the MPKI of 14 experiments of Mesh topology from  $4 \times 4$ ,  $6 \times 6$ , ... ,  $30 \times 30$  network size. All the other values are computed in a similar way.

### 3.4.1.1 L1 instruction (I1) cache analysis

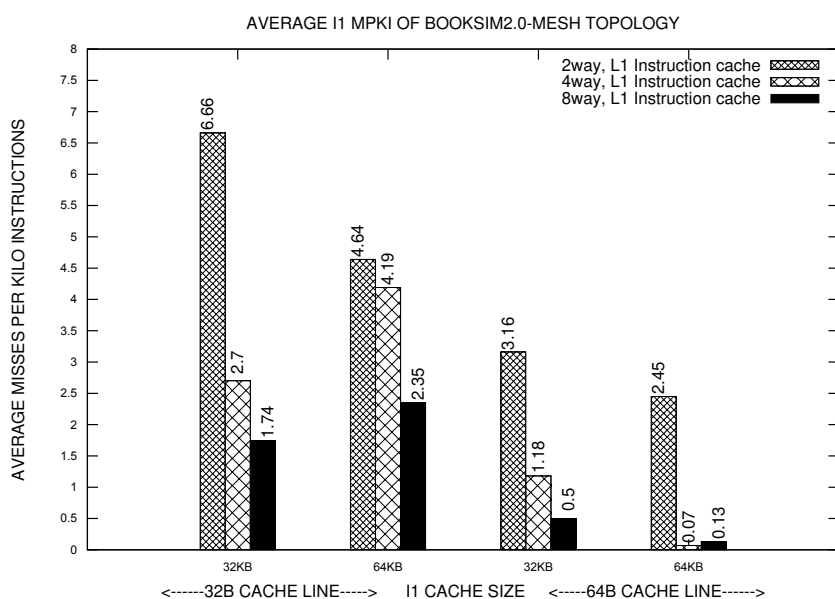


Figure 3.2: Average I1 MPKI of BookSim 2.0 for Mesh topology.(MPKI is averaged over topology sizes mentioned in Table 3.1 and L1 cache configurations were varied as shown in Table 3.2)

**Effect of cache size on I1 cache misses:** From Fig. 3.2, it can be seen that I1 cache misses are reduced by 30.3% when the cache size is increased from 32KB to 64KB for 2-way, 32B line I1 cache. Considering 64B cache line for the same configuration, the cache misses are reduced by 22.47%.

As shown in Table 3.3, 2.73% to 45.24% reduction of misses is observed for all the

other cache configurations when I1 cache size is increased from 32KB to 64KB.

**Effect of associativity on I1 cache misses:** From Fig. 3.2, it can be seen that when the cache configurations are changed from 2-way to 4-way, considering 32KB I1 cache with 32B line size, the misses are reduced by 59.45%. When the cache configurations are changed from 2-way to 8-way, the misses reduced by 73.87%.

A reduction of 56.40% to 74.00% is observed by replacing 2-way I1 cache by corresponding 8-way I1 cache as shown in Table 3.3. Conflict misses are reduced when the associativity is increased from 2 to 4-way and 2 to 8-way, as more blocks in the set can be accommodated.

**Effect of cache line size on I1 cache misses:** It can be seen from Fig. 3.2 that, for 2-way 32KB I1 cache, by increasing the cache line size from 32B to 64B, the misses are reduced by 30.33%. This is due to a good spatial locality of reference.

The reduction of misses from 32.7% to 91.8% is observed for all the other cache configurations when I1 line size is increased from 32B to 64B, as shown in Table 3.3.

Based on the above observations, maximum cache miss reduction of 98.94% can be seen when moving from 2-way, 32KB I1 cache with 32B line to 4-way, 64KB I1 cache with 64B line.

Table 3.3: Effect on misses due to various I1 and D1 cache configurations

<b>Reduction in I1 Misses</b>		
Configurations	Design Choices	Reduction in Misses
32KB vs 64KB	2,4,8-way & 32B,64B	2.73% to 45.24%
2-way vs 8-way	32KB,64KB & 32B,64B	32.7% to 91.8%
32B vs 64B	2,4,8-way & 32KB,64KB	56.40% to 74.00%
<b>Reduction in D1 Misses</b>		
Configurations	Design Choices	Reduction in Misses
32KB vs 64KB	2,4,8-way & 32B,64B	5.16% to 5.80%
2-way vs 8-way	32KB,64KB & 32B,64B	0.78% to 3.35%
32B vs 64B	2,4,8-way & 32KB,64KB	21.16% to 22.00%

### 3.4.1.2 L1 data (D1) cache Analysis

**Effect of cache size on D1 cache misses:** An increase in the size of D1 cache in an incremental manner yields to improved D1 cache performance. From Fig. 3.3, for 2-

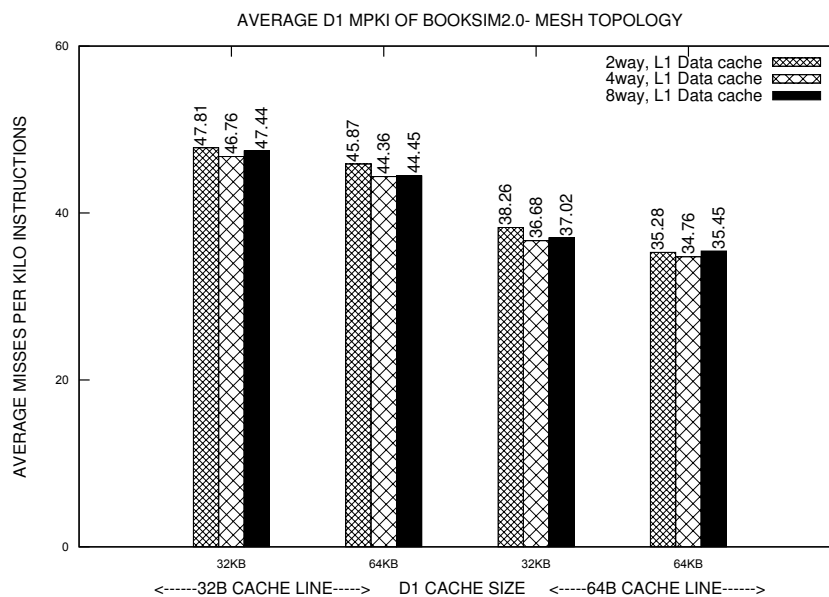


Figure 3.3: Average D1 MPKI of BookSim 2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 and L1 cache configurations were varied as shown in Table 3.2)

way D1 cache with 32B line, when the configurations are changed from 32KB to 64KB cache, the misses are reduced by 4.18%. For 64B cache line, 7.78% reduction in misses is observed when moving from 32KB to 64KB.

For all other cache configurations, 5.16% to 5.80% reduction of misses were observed on increasing D1 cache size from 32KB to 64KB as shown in Table 3.3.

**Effect of associativity on D1 cache misses:** In Fig. 3.3, increasing the associativity from 2-way to 4-way for 32KB D1 cache with 32B line, the misses are reduced by 2.12%. On changing the associativity from 2-way to 8-way, the misses are reduced by 0.77%.

From Table 3.3, 0.78% to 3.35% reduction of misses is observed for all the other cache configurations on moving from lower to higher associativity level. The conflict misses arising from blocks of main memory mapping to the same position in the cache can be reduced by increasing the associativity from 2-way to 8-way.

**Effect of cache line on D1 cache misses:** From Fig. 3.3, the misses reduced by 19.97% on increasing the cache line from 32B to 64B for 2way, 32KB D1 cache.

Reduction of misses from 21.16% to 22.00% is observed for other cache configurations on increasing cache line from 32B to 64B as shown in Table 3.3. Increasing the cache line, more data can be fetched from LL cache into D1 cache. This reduces the compulsory misses.

Based on the above observations, maximum cache miss reduction of 27.29% can be seen when moving from 2way, 32KB D1 cache with 32B line to 4way, 64KB D1 cache with 64B line.

Comparing I1 and D1 cache analysis, the reduction observed in I1 cache is much more than D1 cache as I1 caches exhibit better spatial locality of reference.

Based on the above analysis of I1 and D1 caches, medium associative, higher cache size with larger cache line performs better than all other cache configurations. Our experiments show that 4-way, (64KB+64KB) L1 cache with 64B line L1 configuration is appropriate for running the BookSim 2.0 simulations.

### 3.4.1.3 Last level (LL) cache analysis

The last level cache size of 512KB, 4MB and 8MB are used to identify the appropriate LL cache configuration. 512KB and 8MB LL cache size have been considered for last level cache analysis as the changes in cache misses can be observed more clearly.

In Fig. 3.4, the values are obtained by computing the average MPKIs of 14 different network sizes considering all the cache configurations, as shown in Table 3.2.

**Effect of cache size on LL cache misses:** From Fig. 3.4, for 4-way LL cache with 32B line, on moving from 512KB to 8MB cache, the misses are reduced by 4.13%. Similarly, for the 64B cache line, 6.67% reduction in misses is observed. As shown in Table 3.4, 2.54% to 65.90% reduction of misses were observed for all the other cache configurations when moving from 512KB to 8MB of LL cache size.

**Effect of associativity on LL cache misses:** From Fig. 3.4, increasing the associativity from 4-way to 8-way for 512KB LL cache with 32B line, the misses are reduced by 2.48%. On moving from 4-way to 16-way, the misses are reduced by 15.7%. Reduction of misses from 0.86% to 73.21% is observed for all the other cache configurations

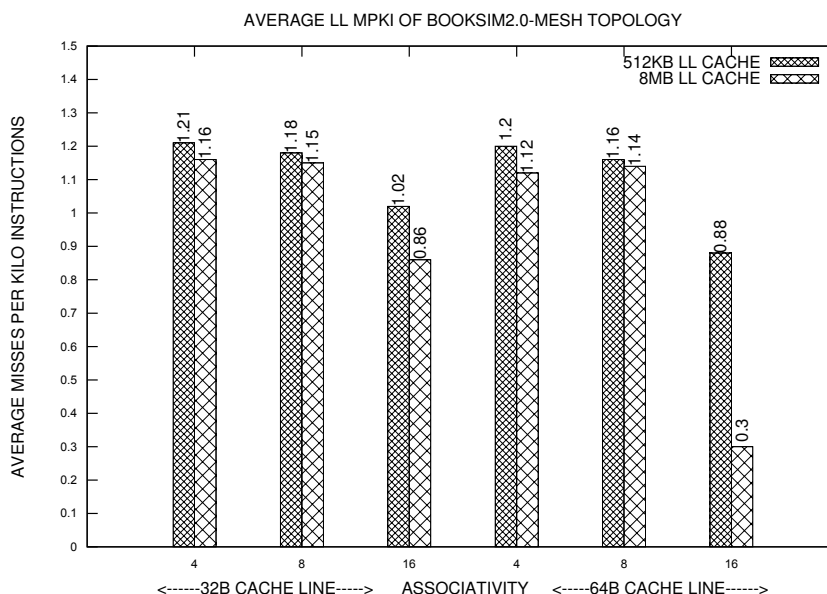


Figure 3.4: Average LL MPKI of BookSim 2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 and LL cache configurations were varied as shown in Table 3.2)

Table 3.4: Effect on misses due to various Last Level (LL) cache configurations

Reduction in LL Misses		
Configurations	Design Choices	Reduction in Misses
512KB vs 8MB	4,8,16-way & 32B, 64B	2.54% to 65.90%
4-way vs 16-way	512KB, 8MB & 32B, 64B	0.86% to 73.21%
32B vs 64B	4,8,16-way & 512KB, 8MB	0.87% to 65.11%

on moving from 4 to 8-way and 4 to 16-way respectively, as shown in Table 3.4. The conflict misses arising from blocks of main memory mapping to the same position in the cache can be reduced when moving from 4-way to 16-way.

**Effect of cache line on LL cache misses:** In Fig. 3.4, the misses are reduced by 0.83% by increasing the cache line from 32B to 64B for 4-way, 512KB LL cache. As seen from Table 3.4, the reduction of misses from 0.87% to 65.11% is observed for other cache configurations. Increasing the cache line from 32B to 64B, more data can be fetched from the main memory to LL cache and the possibility of finding the required data will be high. This reduces the compulsory misses.

Based on the above analysis of LL cache, higher associative, higher cache size with



Table 3.5: Analysis of miss rates of Hotspot methods in BookSim 2.0

<b>24 × 24 Mesh topology</b>				
Method Name	I Refs	(32KB+32KB) L1 Miss	(64KB+64KB) L1 Miss	Reduction of Misses
Simulate(BookSimConfig)	14.5	2.12	1.85	12.73%
TrafficManager::Run()	14.1	2.07	1.72	16.91%
TrafficManager::Step()	14.1	2.07	1.72	16.91%
TrafficManager::SingleSim()	14.1	2.07	1.72	16.91%
Network::Evaluate()	6.32	0.71	0.64	9.86%
Router::Evaluate()	5.81	0.53	0.48	9.44%
IQRouter::InternalStep()	5.21	0.49	0.41	16.33%
Network::WriteOutput()	3.42	0.38	0.37	2.63%
Network::ReadInputs()	3.91	0.32	0.29	9.37%
SparseAllocate::Clear()	3.12	0.26	0.21	19.23%

larger cache line performs better than all other cache configurations. From our experiments, 16-way, 8MB LL cache with 64B line LL cache configuration is found to be appropriate for running the BookSim 2.0 simulations.

By all these observations, it can be inferred that 4-way, (64KB+64KB) L1 cache with 64B cache line and 16-way 8MB LL cache with 64B line is the optimal cache configuration for running BookSim 2.0.

### 3.4.2 Hotspot and CPI analysis

The instruction references, L1(I1+D1) and LL cache misses for all the methods of BookSim 2.0 are extracted by employing Kcachegrind tool. The methods shown in Table 3.5 are identified as hotspots, as most of the execution time is spent in them.

The reduction of misses from 2.1% to 24.22% is observed on moving from 32KB+32KB L1 cache to 64KB+64KB L1 cache configuration for all the other hotspot methods of BookSim 2.0 for topology size varied from  $4 \times 4$  to  $30 \times 30$ .

CPI is one of the critical parameters to measure the performance of BookSim 2.0 with the worst and best cache configurations. From Fig. 3.5, it can be seen that for the worst cache configuration, i.e. 2-way, 32KB+32KB L1 cache, 4-way 512KB LL cache with 32B line, CPI is 5.68 for  $14 \times 14$  Mesh Topology. Employing the best cache configuration, i.e. 4-way, 64KB+64KB L1 cache, 16-way 8MB LL cache with 64B

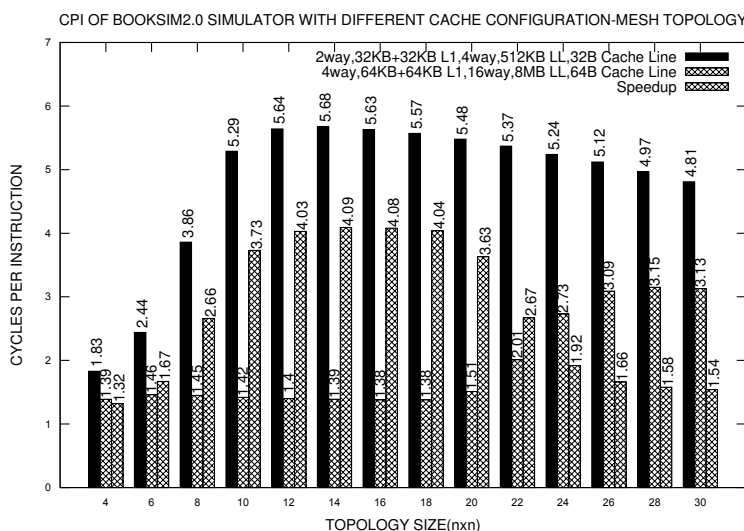


Figure 3.5: CPI for BookSim 2.0 running various sizes of Mesh topology

line size, the CPI reduces to 1.39. Speedup of  $4.1\times$  is observed when the best cache configuration is used. For the smaller Mesh topology sizes, the cache misses are lower as there is less traffic generated. Hence, the speedup for the topology sizes  $4 \times 4$  and  $6 \times 6$  is in the range of  $1.32\times$  to  $1.67\times$ . As the topology size increases, the higher cache configuration yields better performance. Hence, it can be observed in Fig. 3.5 that the speedup achieved for the larger topology sizes is in the range of  $2.66\times$  to  $4.04\times$ . From these experiments, it is evident that increasing cache configuration improves the performance of BookSim 2.0 simulations.

### 3.5 OPTIMIZATION STRATEGIES

In this section, the techniques which are used to optimize the cache misses have been explained. The best cache configuration for BookSim 2.0 has been identified using Valgrind profiling tool. Further, the cache misses have been minimized and the performance of BookSim 2.0 is improved by considering 4-way, (64KB+64KB) L1 cache and 16-way, 8MB LL cache with 64B block size+64 cache configuration.

#### 3.5.1 Minimizing the cache misses

##### 3.5.1.1 Removal of unused functions

Cppcheck (Daniel Marjamki (2011)), a static analysis tool, has been employed to reduce the cache misses of BookSim 2.0. Table 3.6 shows the list of unused functions in various

Table 3.6: Unused functions in BookSim 2.0 source code

File Name	Line Number	Unused Function Name
outputset.cpp	46	Add()
module.cpp	80	Debug()
network.cpp	260, 283	DumpChannelMap(), DumpNodeMap()
iq_router.cpp	2308	GetBufferOccupancyForClass()

source files of BookSim 2.0. These unused functions are removed from the source code. As shown in Fig. 3.6, the cache misses for  $30 \times 30$  sized mesh network is 49.23M for the execution that contains the unused function. Removing the unused functions, the misses

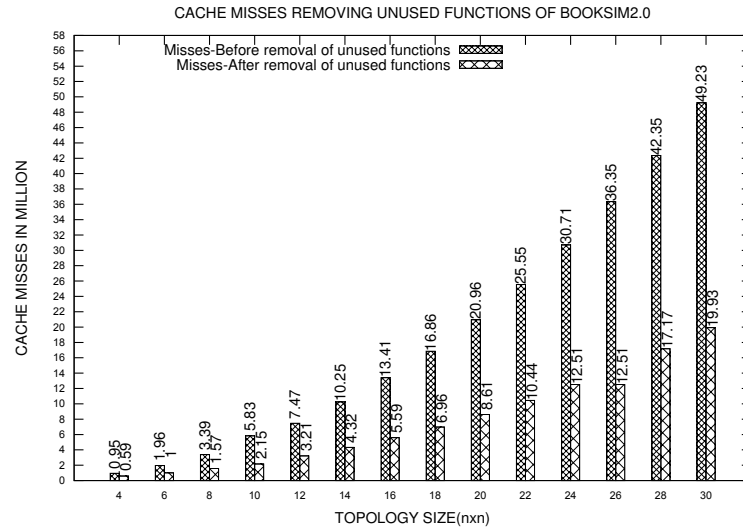


Figure 3.6: Cache misses before and after optimization

are reduced to 19.93M (48.83% reduction of misses was observed). Speedup of  $1.18 \times$  is observed as shown in Fig. 3.7. By employing this optimization technique, 18.52% average reduction of misses is observed for all the other Mesh topology. Speedup of  $1.01 \times$  to  $1.43 \times$  is observed for all the other network sizes of Mesh topology as shown in Fig. 3.7.

### 3.5.1.2 Loop optimization

Loops account for more cache misses relative to other components of a program (Porterfield (1989)). These misses can be optimized by employing techniques such as loop unrolling, loop tiling and loop rotation, etc. The technique of loop reversal has been

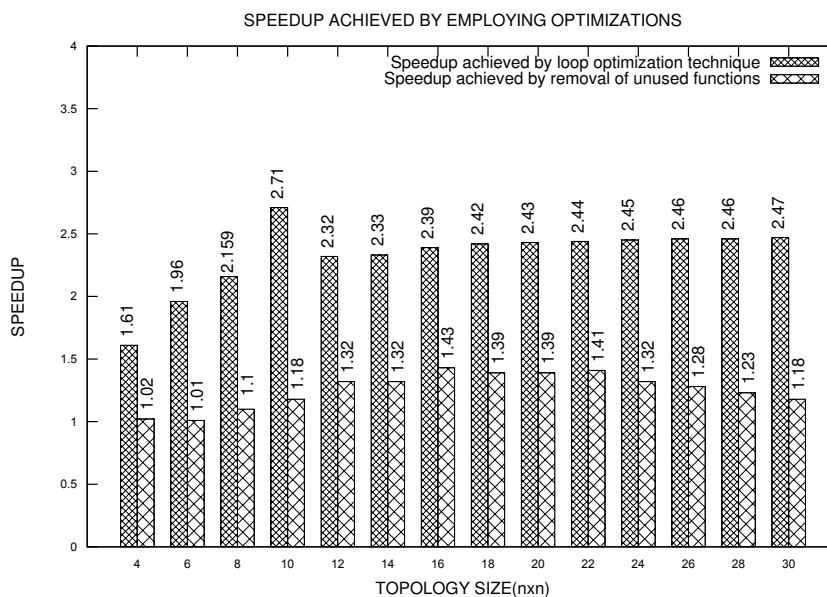


Figure 3.7: Speedup achieved before and after optimization

employed to reduce the misses. A reduction in misses of 5.34% has been observed by applying this technique for the loops. From Fig. 3.7 it can be observed that, the maximum speedup of  $2.47\times$  is observed for  $30 \times 30$  network topology. From  $1.61\times$  to  $2.71\times$  speedup has been observed for all the other topology sizes.

The below code snippet shows the technique employed:

**Before:**

```
for ( int subnet = 0; subnet < _subnets; ++subnet ) {
for ( int n = 0; n < _nodes; ++n ) {
```

**After:**

```
for ( int subnet = _subnets; subnet-- ; ) {
for ( int n = _nodes; n--; ) {
```

#### 3.5.1.3 Pre-Increment Operator

Employing the pre-increment operator instead of post-increment operator in the source code can improve the performance of an application. The pre-increment operator does a single operation such as incrementing the value but, the post-increment operator does three operations: save the current value, increment the value and return the old value.

Table 3.7: Replacing post-increment operator by pre-increment operator

File Name	Line Number	Function Name	Improvement
allocator.cpp	406, 418	SparseAllocator::PrintRequests	2.2 %
islip.cpp	78,102,134,167	iSLIP_Sparse::Allocate( )	3.2 %
selalloc.cpp	86,115,153,183	SelAlloc::Allocate( )	3.4 %
	233,244	SelAlloc::PrintRequests()	3.6 %
prio_arb.cpp	57	PriorityArbiter::AddRequest()	3.8 %
	92,119,141	PriorityArbiter::RemoveRequest()	3.8 %
	119,141	PriorityArbiter::Arbitrate( )	3.7 %
config_utils.cpp	267,278,285	Configuration::WriteFil()	3.6 %
	302,311,318	Configuration::WriteMatlabFile()	3.9 %
anynet.cpp	93,100,105,111,120	AnyNet::_ComputeSize()	3.3 %
	143,158,181,186	AnyNet::_BuildNet()	3.9 %
	272,282,297,315	AnyNet::buildRoutingTable()	3.4 %
	489	AnyNet::readFile()	3.1 %
outputset.cpp	72	OutputSet::Add	3.2 %
trafficmanager.cpp	1404,1415	TrafficManager::_DisplayRemaining	3.5 %
	1484,1590	TrafficManager::_SingleSim( )	3.9 %

The performance of an application is not affected, using pre and post-increment when the data type is primitive. For the non-primitive data types, using pre-increment operation improves the performance. Cppcheck tool has been used to identify the post-increment operators in the C++ source code of BookSim 2.0.

```
$ cppcheck --enable=performance /booksim/src/
```

The above mentioned command detects the post-increment operators in the source code of BookSim 2.0. It can be seen that the output of Cppcheck contains names of methods in the classes of source code and line number of post-increment operator.

Output:

```
/booksim2-master/src/trafficmanager.cpp:1403]: (performance)
Prefer prefix ++/-- operators for non-primitive types.
```

Table 3.7, shows the different locations of the code which are using post-increment operators. These operators are replaced by pre-increment operator in 8 source files, 42 lines of BookSim 2.0. As seen from the “Improvement” column of the table, the cache misses are reduced from 2.2% to 3.9%.

### 3.5.2 Improving performance of BookSim 2.0

The techniques such as vectorization, multi-threading and OpenMP programming models are employed to parallelize the portions of the BookSim 2.0 source code. The most

### 3. Analysis and performance enhancement of BookSim 2.0 NoC simulator

time-consuming loops are identified employing the Intel Advisor tool. Memory access patterns have an impact on improving the performance of an application. The Intel vectorization tool has been employed to identify the stride access patterns of BookSim 2.0. 17% of memory instructions are unit stride, 34% of memory instructions are fixed non-unit stride and 49% of memory instructions are variable stride after annotating Singlesim method of TrafficManager class of BookSim 2.0. Based on these observations, the performance of BookSim 2.0 has been improved by changing unaligned memory accesses to aligned memory access. The compiler directive shown below is inserted in the source files of BookSim 2.0 to change unaligned to aligned memory access based on memory access pattern analysis as shown in Table 3.8.

```
#pragma omp simd aligned()
```

Table 3.8: Identifying the memory access pattern of BookSim 2.0 source code

Memory Access Pattern	Source	Nested Function Name	Line Number
Constant stride	trafficmanager.cpp	_RetireFlit	672
			680
	stl_map.h	construct	1521
Uniform stride	credit.cpp	New	52
	network.cpp	WriteCredit	229
	new_allocator.h	construct	104
	stats.cpp	AddSample	107
	trafficmanager.cpp	_step	1267
			1268
			649
	credit.cpp	Reset	47, 48,49
	credit.cpp	New	58
	network.cpp	WriteCredit	229
new_allocator.h	construct	104	
stats.cpp	AddSample	114	
Variable stride	trafficmanager.cpp	_step	1252
			1253
		_RetireFlit	655, 660, 673
			678, 679, 681
			686, 692, 696
711, 731, 736, 752			

Further, OpenMP programming model and SIMD constructs are employed to parallelize and vectorize the most time-consuming portions of BookSim 2.0. Execution times of sequential code with parallel code are compared considering different network topology size of BookSim 2.0 with Mesh topology, as shown in Fig. 3.8. The speedup

of  $2.93\times$  as shown in Fig. 3.9 has been achieved by parallelizing the sequential code of BookSim 2.0 using OpenMP constructs considering  $30 \times 30$  network size of Mesh topology.  $1.07\times$  to  $3.0\times$  speedup was observed for all the other sizes of Mesh topology.

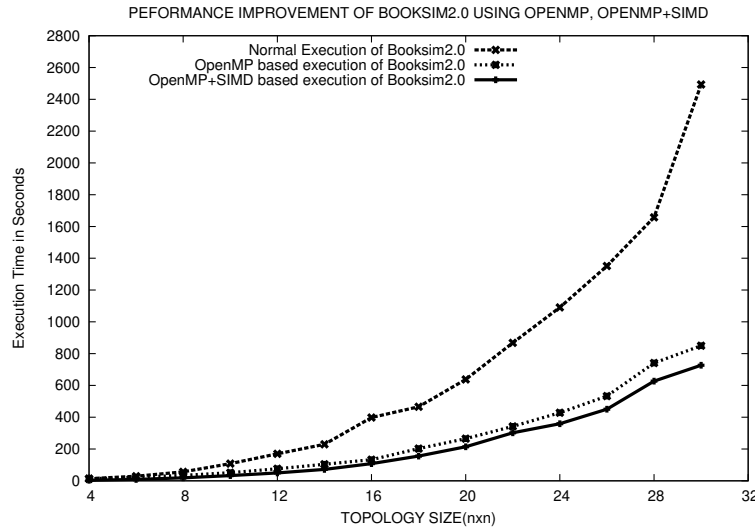


Figure 3.8: Simulation execution times before and after improvements

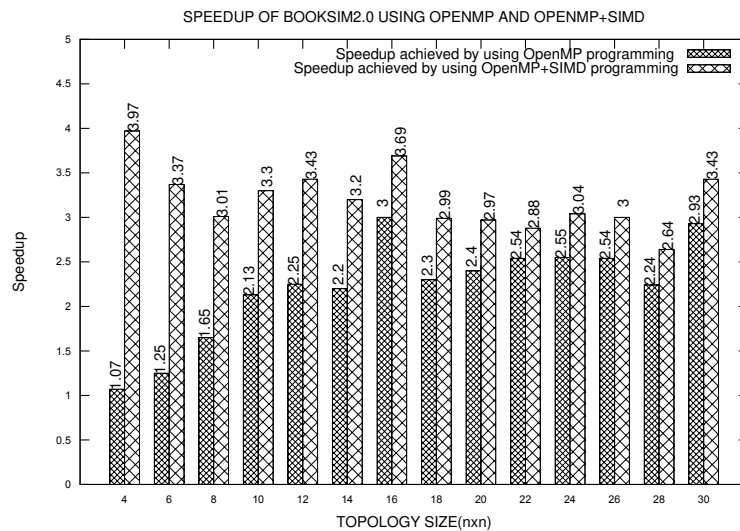


Figure 3.9: Speedups with Mesh topology of varying sizes

Also, the SIMD construct is used with the OpenMP programming model to achieve fine-grain parallelization. By using SIMD with OpenMP model, the performance improvement of  $3.97\times$  is observed for  $4 \times 4$  Mesh topology. And, the speedup from  $2.64\times$  to  $3.69\times$  is observed for all the other sizes of Mesh topology. The overall traffic statistics of BookSim 2.0 observed in both the executions matched each other. The pragma

constructs used to parallelize and vectorize the code are shown below:

```
#pragma omp parallel for  
#pragma omp parallel simd for
```

From Fig. 3.8 and 3.9, it can be inferred that parallelization and vectorization reduce the execution time of  $30 \times 30$  Mesh topology from 60 to 14 minutes and 12 minutes, respectively.

## 3.6 SUMMARY

In this chapter, the profiling and software optimization strategies that can improve the performance of BookSim 2.0 NoC simulator are discussed. Profiling BookSim 2.0 helps in understanding the effect of various input parameters.

The cache design parameters such as cache size, associativity, cache line size, replacement algorithm and cache write policy are considered for the experiments. Hotspot methods of BookSim 2.0 simulator, where most of the execution time is spent, are identified employing the KCachegrind tool. The software optimization techniques employed to reduce the cache misses. Vectorization and parallelization are employed to improve the performance of the BookSim 2.0 simulator. By using the Intel advisor tool, the top time-consuming loops in BookSim 2.0 source code have been identified. The OpenMP programming model has been used to parallelize the time-consuming loops in the simulator.

FPGA based simulation accelerations are proven to provide better speedup and accuracy compared to the software simulators (Angepat et al. (2014)). An FPGA based NoC simulation acceleration platform is proposed in the subsequent chapters. The proposed framework is capable of design space exploration of standard and custom NoC topologies considering a full set of micro-architectural parameters.



## CHAPTER 4

### YANOC - AN FPGA BASED NOC SIMULATION ACCELERATION FRAMEWORK

In this Chapter, an FPGA based NoC simulation acceleration framework supporting design space exploration of standard and custom NoC topologies considering a full set of micro-architectural parameters is presented. For conventional NoCs, the standard minimal routing algorithms are supported. For designing the custom topologies, the table-based routing has been implemented. A custom topology called Diagonal Mesh has been evaluated employing the table-based and novel shortest path routing algorithms. A congestion aware adaptive routing has been proposed to route the packets along the minimally congested path.

#### 4.1 INTRODUCTION

Highly reconfigurable LUTs act as the building blocks of FPGAs. Any arbitrary function can be realized by employing the LUTs. FPGAs allow the events to be executed in parallel. Features mentioned above helped the researchers to employ FPGAs for simulation acceleration by parallelizing various functionalities of a simulator.

To expedite the speed of simulation compared to the software simulators, an FPGA based NoC simulation framework called YaNoC has been presented in this chapter. YaNoC supports the design space exploration of standard NoC topologies such as Mesh, Torus, Ring, and Tree-based topologies along with the Custom topologies. Also, YaNoC supports the creation of standard and custom routing algorithms, generation of synthetic

Table 4.1: Configurable router architectural parameters

Router Parameter	Range of values
Topology	Mesh based, Ring based, Tree based, Custom
Flit buffer depth	Variable
Flit width	Variable
Ports	2 to 16
Routing Algorithms	Standard minimal routing, Table based Congestion aware adaptive routing, Nearest neighbor
Arbitration schemes	Round Robin and Priority based
Traffic patterns	Uniform random, Bit complement, Transpose, Random permutation

traffic patterns, and exploration of a full set of micro-architectural parameters.

## 4.2 YANOC - DESIGN AND IMPLEMENTATION

Fig. 4.1 shows the architecture of YaNoC simulation acceleration engine. YaNoC has been designed to be highly parameterizable, modular, easily adaptable to new NoC architectures as per the design requirements. The list of configuration parameters YaNoC are shown in Table 4.1. The configuration parameters and their corresponding hardware modules are detailed in this section.

### 4.2.1 YaNoC configuration parameters

To provide the maximum flexibility, YaNoC parameterizes all the components of the NoC. If a design with 32-bit flit width and buffer depth of 8 flits has to be evaluated, these parameters have to be specified in the configuration file. The Automated Verilog HDL Generator generates the Verilog code corresponding to this configuration. When there is a need to evaluate 64-bit flit width and buffer depth to be of 4 flits, the older configuration file can be modified according to the new requirement. Automated Verilog HDL Generator generates the code for considering this configuration. Similar to the flit size and flit buffer depth, if there is a need to evaluate the conventional XY and Table based routing algorithms for Mesh topology, the parameters for routing algorithms can be modified accordingly.

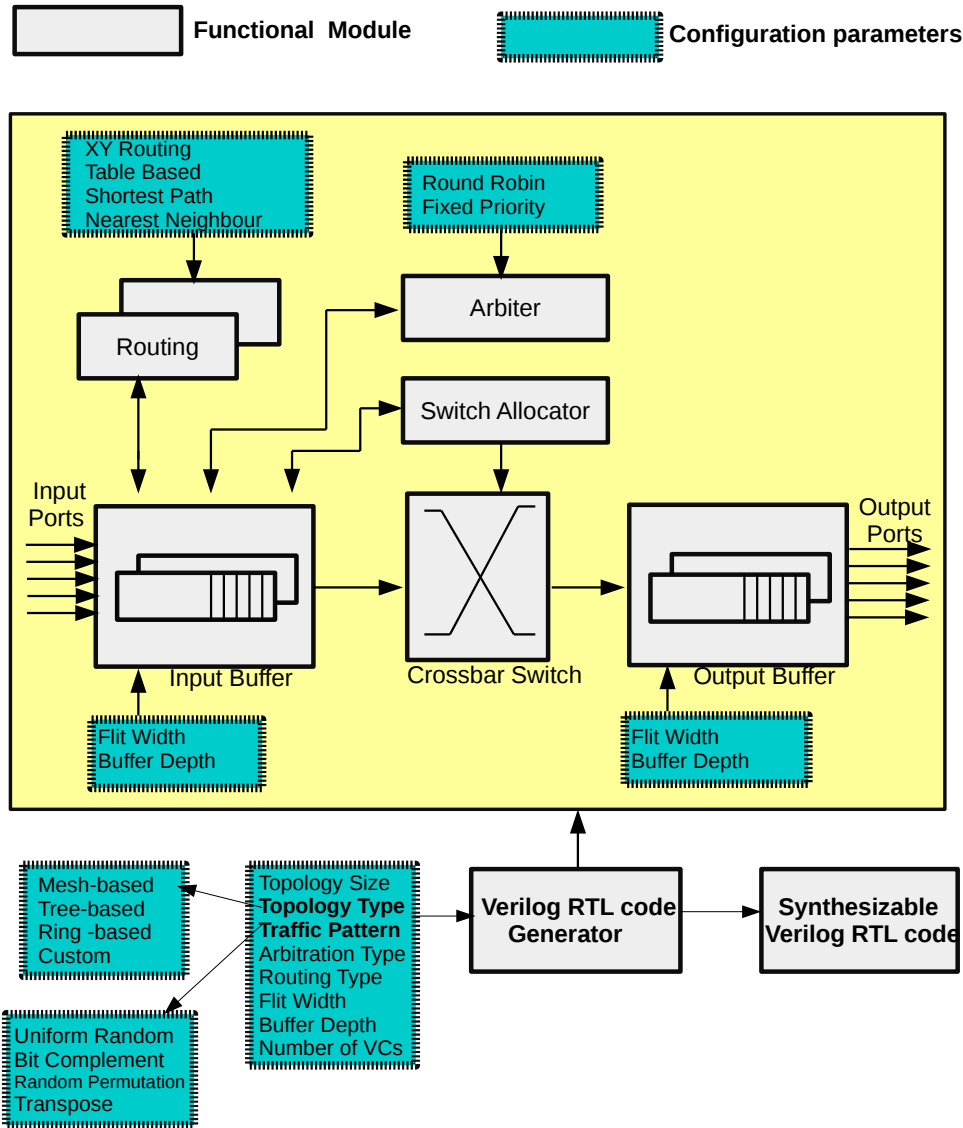


Figure 4.1: Architecture of the proposed YaNoC FPGA based NoC simulation acceleration framework

## 4.2.2 Router architecture

The router module consists of micro-architectural components such as I/O buffers, Route compute logic, Arbitration unit, Crossbar unit, and Traffic generator (Source/Sink).

### 4.2.2.1 Flit buffer

The incoming flits are stored in buffers implemented employing the FIFO mechanism. The buffer depth is parameterized to provide the flexibility to explore various kinds of

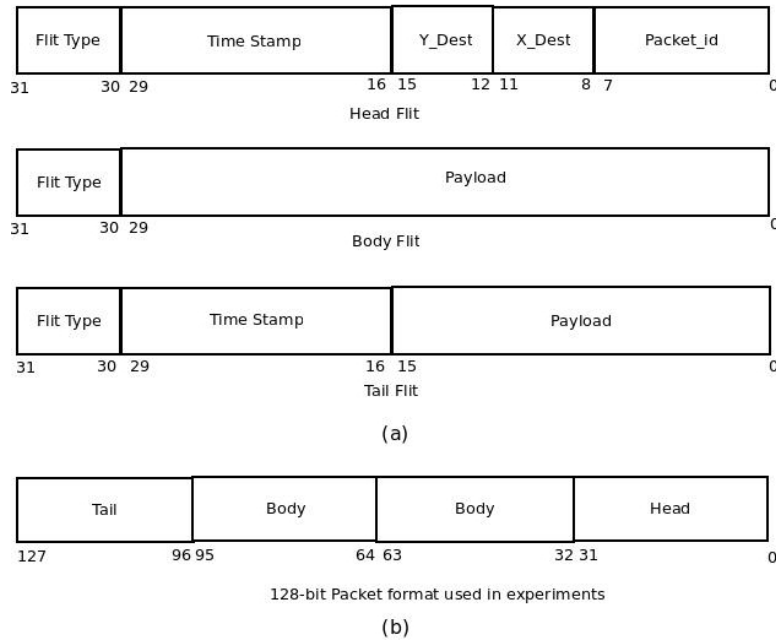


Figure 4.2: (a) Flit types and (b) Packet structure used in experiments. (Time stamp field is useful in calculating the latency of a packet)

flit width.

#### 4.2.2.2 Flit structure

YaNoC supports flits of variable widths. The structure of the head, body and tail flits are shown in Fig. 4.2(a). The size of each flit is of 32-bit. The fields for flit type, destination address, timestamp, and packet id are incorporated in the header flit. Body flit embodies the fields for flit type and payload. To calculate latency of the network, the tail flit comprises of timestamp similar to head flit. Fig. 4.2(b) shows the packet format employed in the experiments. The packet of length 128-bit length comprises of a head flit, two body flits, and a tail flit.

#### 4.2.2.3 Input/Output ports

It is advantageous to have the reconfigurable ports while building various topologies. The ring topology has 3 ports in which two of them are used to communicate between the neighboring cores, and the remaining port is used to connect to the local processing element of that core. Similarly, Mesh and DMesh topologies have 5 and 9 ports for communication. The provision for variable ports is provided in YaNoC to explore

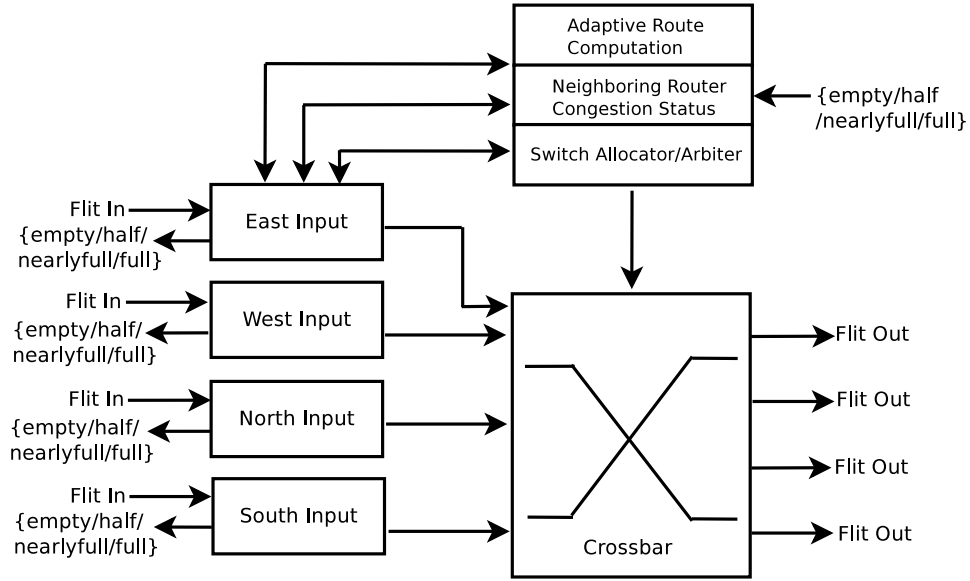


Figure 4.3: Modified router architecture supporting Congestion aware adaptive routing

various custom topologies.

#### 4.2.2.4 Routing algorithms

The standard minimal dimension-order (XY) routing algorithm for conventional NoCs is supported. Table-based routing is implemented to support the creation of custom topologies. The output ports to all the destinations in the network are stored in the LUTs. The entries in routing LUTs are large for larger networks. Distributed RAMs (DRAM) of FPGA are employed in the proposed architecture to implement the routing tables. A single DRAM is typically single-bit wide memory with 16-64 elements constrained to a specific FPGA family. As the entries in routing tables are a maximum of 3-bit wide, they are mapped very efficiently to DRAMs. A custom topology called Diagonal Mesh (DMesh) has been evaluated using table-based and a novel shortest path version of the XY routing algorithm.

The standard and table-based routing algorithms do not consider the congestion state of the network under analysis for route computation. A congestion aware adaptive routing has been proposed to consider the traffic condition in the network. The congestion aware adaptive routing algorithm has negligible FPGA area overhead compared to the conventional XY routing.

Fig. 4.3 shows the modified router architecture supporting the Congestion aware adaptive routing algorithm. The modified router architecture includes the logic for neighboring router congestion information and adaptive route computation. The working of the proposed adaptive routing algorithm has been detailed in Section 4.3.4.

##### **4.2.2.5 Arbitration schemes**

To ensure fairness in the allocation of resources, round-robin and priority based arbitration schemes have been implemented.

##### **4.2.3 Traffic generator**

The Traffic Generator (TG) module takes care of the generation of various synthetic traffic patterns. Linear Feedback Shift Register (LFSR) mechanism has been employed to introduce randomness in the traffic being generated. The TG module is incorporated into each router. The LFSR modules generate traffic according to the traffic pattern and the specified injection rate.

To calculate the latency of the network, a source generating the packet inserts 14-bit timestamp to head and tail flits. The traffic sink is responsible for ejecting the flits and calculating the latency of the network.

##### **4.2.4 Software tools supporting YaNoC**

###### **4.2.4.1 Automated Verilog HDL generator**

A Hardware Description Language (HDL) generator has been developed in python to generate the synthesizable Verilog code of a specified configuration. The automated Verilog HDL Generator generates the synthesizable Verilog HDL code based on the specified configuration parameters such as type and size of topology, link width, flit buffer depth, buffer width, routing algorithm, and arbiter type.

###### **4.2.4.2 Routing table generator**

Along with the support for the generation of synthesizable Verilog HDL, YaNoC also includes the software tools developed in Python to automatically generate routing tables for Mesh, Torus, Fat tree, and DMesh topologies. The routing table for each node

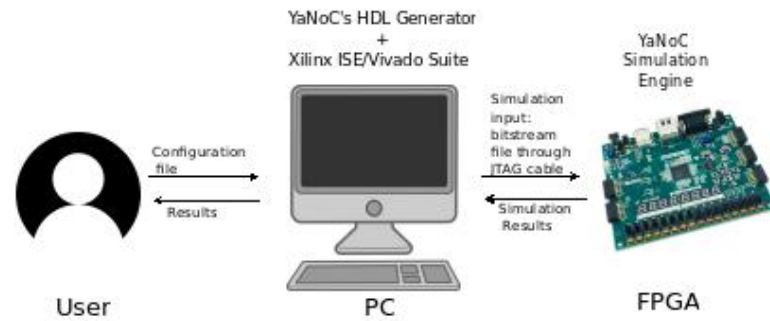


Figure 4.4: A High-level block diagram of YaNoC consisting of Host PC connected to an FPGA Board.

containing the entries of the shortest path to every other node in the topology is populated upon executing these scripts. For example, Routing tables of XY routing for Mesh-based, Shortest path routing for Ring based topologies, Nearest Ancestor First for Tree-based topologies is populated depending on the selected configuration.

#### 4.2.4.3 YaNoC portal

Also, YaNoC consists of a JTAG connection between the host PC and the FPGA board. A portal has been developed for interaction with the simulation engine located on FPGA and the host PC. Simulation results from the FPGA can be accessed by using the portal, as shown in Fig. 4.4.

#### 4.2.5 Design phase

To ensure the functional correctness of the NoC synthesized by YaNoC, we split the software cycle into the correctness and implementation phase. In the correctness phase, the design to be simulated on FPGA has been thoroughly analyzed considering clock by clock transitions. The flit traversal through each pipeline stage has been analyzed the functional correctness. In the implementation phase, the HDL for required NoC design is generated with the help of Automated Verilog HDL Generator, and it is programmed on the FPGA using Xilinx Vivado suite.

The proposed platform consists of a host PC, JTAG cable connecting the host PC and FPGA board, and Xilinx Artix 7 FPGA (XC7A100T). The NoC simulation engine is hosted on Artix7 FPGA board (Fig. 4.4). The following steps describe the flow of the YaNoC framework shown in Fig. 4.5.

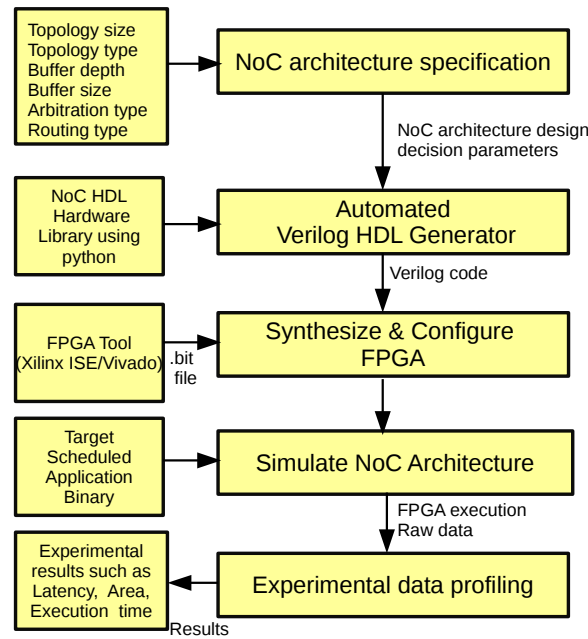


Figure 4.5: Simulation framework flow

1. The configuration file is updated by the user to reflect the correct parameters of the NoC to be simulated.
2. The YaNoCs automated HDL generator generates synthesizable Verilog HDL. The bitstream (.bit) file is obtained by Xilinx ISE/Vivado design suites.
3. The .bit file is programmed on the FPGA through the JTAG cable. UART is used for transferring data from the FPGA to the PC.
4. The NoC to be simulated is programmed on the FPGA. The latency results of simulation are extracted from the portal developed for interaction with the FPGA through UART communication.
5. The hardware resource consumption is obtained from the design summary of Xilinx ISE/Vivado.

### 4.3 DESIGN OF MESH AND DIAGONAL MESH (DMESH) TOPOLOGIES

YaNoC is capable of simulating standard and custom topologies. The design of custom topology is possible because of the table-based routing approach. Along with the sup-



port for table-based routing approach, the route computation template can be modified as per the user logic to route the packets in the network.

#### 4.3.1 Design of DMesh topology in YaNoC

To design an application-specific custom topology, interconnection in between nodes and routing tables along with the other router micro-architectural parameters have to be specified in the configuration file. This file is given as the input to the Automated Verilog HDL Generator to generate the synthesizable Verilog HDL code.

#### 4.3.2 Design of DMesh routing algorithm in YaNoC

Table based routing is used to store routing information in case of custom topologies whose route compute modules are complex to design. Along with the table-based routing approach, we demonstrate the flexibility of YaNoC in designing a user-specific routing algorithm for DMesh topology.

The novel routing algorithm for routing the packets in the shortest path has been designed. The logic for calculating the shortest path can be implemented in the route compute template. Changes made to the route compute logic can be seen in the code snippet shown below. Fig. 4.6(a) and 4.6(b) show the  $6 \times 6$  Mesh and DMesh topologies. The arrows in red color of Fig. 4.6(a) indicates the route followed by the conventional XY routing algorithm. In this case, it takes 10 hops to reach the destination “55” from the source “00”. Employing the proposed novel routing algorithm, the shortest path between a source and destination pairs has been achieved in the DMesh topology. The arrows in green in Fig. 4.6(b) represents the route followed by the flits employing the proposed routing algorithm. It can be seen that it takes only 5 hops from “00” node to “55” node through the diagonal nodes (“11”, “22” and so on).

---

Code snippet of shortest path routing algorithm for DMesh topology

---

```
module compute (Li, port_num_next);  
/* Lo, Eo, No, Wo, So, NEo, SEo, NWo and SWo are the output
```

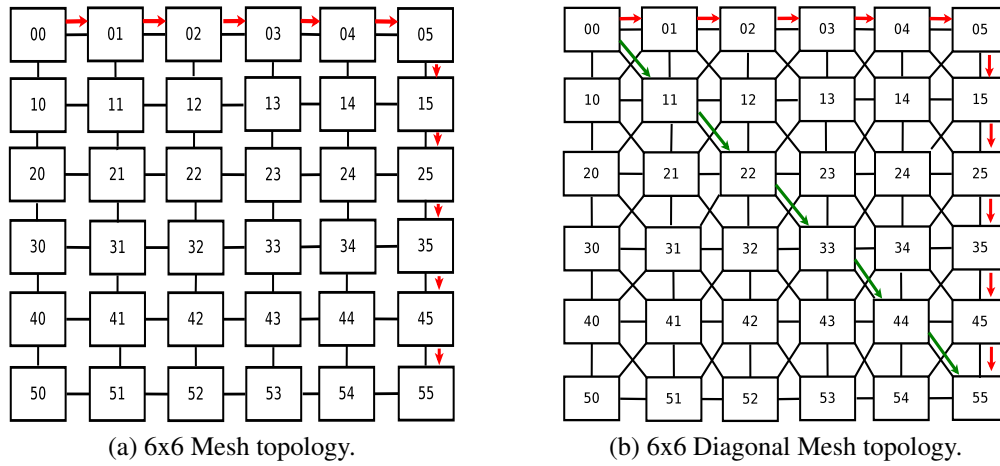


Figure 4.6: Mesh and Diagonal Mesh topologies (Red and Green colors indicate the routes calculated by XY and novel shortest path XY routing algorithms)

ports corresponding to Local, East, North, West, South, NorthEast, SouthEast, NorthWest and SouthWest directions respectively.

\*/

/\* Assign 1, 2, 3, 4, 5, 6, 7, 8 and 9 to Local, East, North, West, South, NorthEast, SouthEast, NorthWest and SouthWest ports respectively. \*/

assign Lo = 4'b0001; //LOCAL\_OUT

assign Eo = 4'b0010; //EAST\_OUT

assign No = 4'b0011; //NORTH\_OUT

assign Wo = 4'b0100; //WEST\_OUT

assign So = 4'b0101; //SOUTH\_OUT

assign NEo = 4'b0110; //NORTH\_EAST\_OUT

assign SEo = 4'b0111; //SOUTH\_EAST\_OUT

assign NWo = 4'b1000; //NORTH\_WEST\_OUT

assign SWo = 4'b1001; //SOUTH\_WEST\_OUT

/\* (xc,yc ) and (xd,yd) are the current node and destination node x and y co-ordinates respectively. The route computation is done by considering these values. \*/

assign xc = r1[2:0];

```
assign yc = r1[7:5];
assign xd = Li[2:0];
assign yd = Li[7:5];
/* Following if-else conditions are checked to find out the
shortest path computation from a current node to the destination
node. */
always@(*)begin
    // Condition for NorthEast Output port. If true, NEd will
be the output port.
    if (xc[2:0]<xd[2:0]&&yc[2:0]>yd[2:0])
        begin
            port_num_next = NEd;
        end
    //Condition for SouthEast Output port. If true, SEd will
be the output port.
    else if(xc[2:0]<xd[2:0]&&yc[2:0]<yd[2:0])
        begin
            port_num_next = SEd;
        end
    // Conditions for rest of the ports can be included as
shown above.
end
endmodule
```

#### 4.3.2.1 Routing tables

A routing table is stored in each router. The routing table contains route to all the other routers in the network. Below lines specify the routing table for a Router with ID “0” in a  $2 \times 2$  Mesh topology.

#Router_ID	Dest_Out_Port
0	0 //Local
1	1 //East

```
2           4           //South
3           1           //East
```

Above mentioned syntax is used to design the DMesh topology. The architecture of the DMesh is shown in Fig. 4.6(b). Each router in the DMesh topology consists of 9-ports for communicating with neighboring nodes. The nodes are interconnected via links from these ports. Fig. 4.7 shows the interconnection of the Router “12” with all its neighbors in the DMesh topology.

Below lines enumerate the “Network Topology” entries for the Router with ID “12”.

```
Router_Link_From      Router_Link_To
# (port_num:src_node) (port_num:neighbor_node)
  1 : R12             2 : R13
  2 : R12             1 : R11
  3 : R12             4 : R02
  4 : R12             3 : R22
  5 : R12             8 : R03
  6 : R12             7 : R01
  7 : R12             6 : R23
  8 : R12             5 : R21
```

### 4.3.3 DMesh topology configuration

YaNoC generates the synthesizable Verilog code, considering the entries in a configuration file. YaNoC supports up to 16 router ports. These ports can be used to interconnect the router modules to form a custom topology. The nodes have to be interconnected in a specific manner to form a topology. In YaNoC, the interconnection between source and destination nodes, along with the port numbers, are enumerated to specify the network topology. Below code snippet shows the interconnection of two nodes 0 and 1:

```
#Router_Link_From      Router_Link_To
```

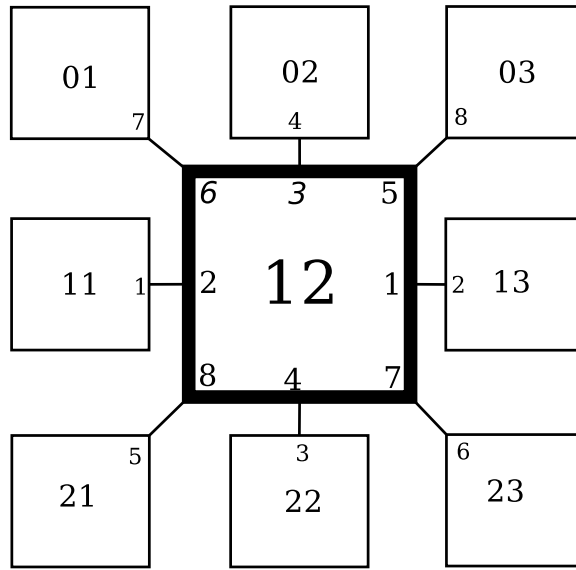


Figure 4.7: Interconnection of the Router 12 with other Routers in DMesh topology

```
# (port_num:src_node)      (port_num:neighbor_node)
    1:R0                    2:R1

#Local_Conn_Port (Port number for connecting the Processing
Element)

# (port_num:node)
    0 :   R0
    0 :   R1
```

The routing table of a node consists of output ports for all the other nodes in the network. Considering  $6 \times 6$  DMesh topology, the routing table for each Router consists of 36 entries. Each row in the routing table consists of Router ID along with its output port from the current Router. Along with the above-detailed modifications in the configuration file, the parameters shown in Table 4.2 are used.

Once all these entries are configured, the Verilog generator of YaNoC generates the synthesizable Verilog code, which can be imported in Xilinx ISE/Vivado design suite. The bitstream file generated following the Synthesis, Translation, and Place and Route processes can be programmed on the FPGA board to simulate the DMesh topology.

#### 4.3.4 Congestion aware adaptive routing algorithm for Mesh based topologies

Employing the proposed adaptive routing algorithm, packets are routed along the minimally congested communication path. In a given 2D Mesh topology, if the current and destination addresses are same (( $x_c=x_d$ ) and ( $y_c=y_d$ )), the flit has reached its destination and is forwarded via the local port to the processing element. When the current and destination nodes are different (( $x_c \neq x_d$ ) and ( $y_c \neq y_d$ )) the flit is forwarded to the neighboring nodes through E/W/N/S directions from the current node based on congestion in the network.

Congestion weights for each direction are calculated employing the weight calculation technique. The first priority is given to the West port if ( $x_{diff} < 0$ ) where  $x_{diff}$  is ( $x_d - x_c$ ). To avoid the deadlock condition, second priority is given to South port, and the third priority is shared among the East and the North ports. Similarly, when ( $x_{diff} > 0$ ), the first priority is given to the East port. For the deadlock avoidance, the second priority is given to North port and the third priority is shared among the West and the South ports. The conditions discussed above are shown in Equations 4.1 and 4.2.

$$P[E/W/N/S] = \{3/1/3/2\} \quad (4.1)$$

$$P[E/W/N/S] = \{1/3/2/3\} \quad (4.2)$$

The algorithm for calculating the priorities of all the ports of a router is shown in Algorithm 1.

Once the priorities of the ports have been calculated, weights corresponding to the buffer occupancy of the router is computed.

We employ 2-bit values for indicating the congestion in the communicating routers. The values 00, 01, 10 and 11 represent the empty(0%), half full(50%), nearly full(75%) and full (100%) occupancy of the buffers of a router. A router has to exchange these congestion status bits with its neighboring routers to make the correct routing decision. The information of all the ports of a router except the local port needs to be exchanged. Once the weight values :  $W_{empty}$ ,  $W_{half\_full}$ ,  $W_{nearly\_full}$  and  $W_{full}$  are calculated,  $W[i]$ , the total weight for each port  $i \in \{E, W, N, S\}$  is calculated by using the Equation 4.3. The values 2, 3, 5 and 10 are assigned for  $W_{empty}$ ,  $W_{half\_full}$ ,  $W_{nearly\_full}$  and  $W_{full}$  for fair

---

**Algorithm 1:** Priority calculation for E/W/N/S directions

---

**Input:** Co-ordinates of current node (xc,yc), destination node (xd,yd)  
**Output:** Priority Matrix of all the ports P[E/W/N/S]  
 $xdiff = xd - xc$   
 $ydiff = yd - yc$   
**if**  $xdiff < 0$  **then**  
    |  $P[E/W/N/S] = 3/1/3/2;$   
**else if**  $xdiff > 0 \&\&ydiff < 0$  **then**  
    |  $P[E/W/N/S] = 1/2/2/3;$   
**else if**  $xdiff > 0 \&\&ydiff > 0$  **then**  
    |  $P[E/W/N/S] = 2/3/3/1;$   
**else if**  $xdiff > 0$  **then**  
    |  $P[E/W/N/S] = 1/3/2/3;$   
**else if**  $ydiff > 0$  **then**  
    |  $P[E/W/N/S] = 2/2/3/1;$   
**else if**  $ydiff < 0$  **then**  
    |  $P[E/W/N/S] = 2/2/1/3;$

---



---

**Algorithm 2:** Congestion aware adaptive routing algorithm for Mesh based topologies

---

**Input:** Co-ordinates of current node (xc,yc), destination node (xd,yd) and the number of ports (E,W,N,S) **Output:** Selected output port  
 $W[E/W/N/S] = 0$   
**for**  $i = E$  **to**  $S$  **do**  
    | **Calculate P[i] using Algorithm 1.**  
    | **if**  $empty == 00$  **then**  
        |  $W_{empty}[i] = 2; W_{half\_full/nearly\_full/full}[i] = 0$   
    | **else if**  $half\_full == 01$  **then**  
        |  $W_{half\_full}[i] = 3; W_{empty/nearly\_full/full}[i] = 0;$   
    | **else if**  $nearly\_full == 10$  **then**  
        |  $W_{nearly\_full}[i] = 5; W_{empty/half\_full/full}[i] = 0;$   
    | **else if**  $full == 11$  **then**  
        |  $W_{full}[i] = 10; W_{empty/half\_full/nearly\_full}[i] = 0;$   
    |  $W[i] = P[i] + W_{empty}[i] + W_{half\_full}[i] + W_{nearly\_full}[i] + W_{full}[i];$   
**end**  
 $min = W[E];$   
**for**  $i = W$  **to**  $S$  **do**  
    | **if**  $w[i] < min$  **then**  
        |  $min = w[i]; output\_channel = i;$   
**end**

---

calculation of the weight matrix. The port with minimal weight value is chosen as the final output port. The adaptive routing algorithm is shown in Algorithm 2.

$$W[i] = P[i] + W_{empty} + W_{half\ full} + W_{nearly\ full} + W_{full} \quad (4.3)$$

The current status information of the  $(i+1)^{th}$  router is taken into consideration at the  $i^{th}$  router to achieve the adaptiveness in the NoC router architecture. “Neighboring Router Congestion Status” monitors the congestion status of all neighboring router ports. The “Adaptive Route Computation” unit calculates the priority for the ports in all directions as discussed above. The deadlock avoidance is taken care of implicitly in the proposed routing algorithm as there are priorities assigned to each direction, and no turns leading to a deadlock are encountered by the flit.

Table 4.2: Experimental setup details

Experimental setup	
Topology	$6 \times 6$ and $8 \times 8$ Mesh, Torus, 56 node Fat tree and $6 \times 6$ Diagonal Mesh
Buffer type	FIFO buffer
Buffer Depth	4, 8, 16, 32, 64 flits
Arbiter type	Round-robin
Routing Algorithm	XY (Dimension-order), Novel shortest path XY routing, Table based, Congestion Aware Adaptive
Router pipeline depth	5-stage
Flow control	Wormhole
Flit Width	16, 32 bits
Packet length	4 and 8 flits
Traffic pattern	Uniform random, Random Permutation, Bit complement, Transpose

#### 4.4 EXPERIMENTAL RESULTS

Synthesis results of the simulation are extracted from the Design Summary of Xilinx Vivado. Results include resource usage for Xilinx Artix-7 FPGA (XC7A100T part, CSG324 package, speed grade -3). The NoCs are tested with injection rates of 0.01 to 0.5 using Uniform random, Transpose, Bit complement, and Random permutation traffic patterns. Table 4.2 shows the experimental setup details. The proposed YaNoC framework is capable of simulating the Mesh, Torus, and Fat tree topologies. Along



Table 4.3: Resource utilization of  $6 \times 6$  (36 node) Mesh and Torus topologies under various configurations of Flit Width (FW) and Buffer Depth (BD)

	FW	16bits						32bits			
	BD	4	8	16	32	64	4	8	16	32	64
6x6 Mesh	LUT(%)	32.33	33.85	34.03	35.13	37.97	34.45	35.65	37.17	38.30	41.17
	DRAM(%)	7.58	7.58	7.58	7.58	11.37	11.37	11.37	11.37	11.37	18.95
	FF(%)	12.62	13.05	13.47	13.90	14.32	15.00	15.14	15.88	16.30	16.73
6x6 Torus	LUT	40.15	41.28	42.13	43.84	47.24	53.72	54.85	55.70	57.41	63.09
	DRAM(%)	11.37	11.37	11.37	11.37	22.74	22.74	22.74	22.74	22.74	41.68
	FF(%)	15.54	15.97	16.40	16.82	17.25	23.51	23.93	24.36	24.79	25.21

Table 4.4: Resource utilization of  $8 \times 8$  (64 node) Mesh and Torus topologies under various configurations of Flit Width (FW) and Buffer Depth (BD)

	FW	16bits					32bits		
	BD	4	8	16	32	64	4	8	16
8x8 Mesh	LUT(%)	62.15	63.67	65.05	67.37	69.58	67.84	69.33	70.93
	DRAM(%)	20.21	20.21	20.21	20.21	26.95	26.95	26.95	26.95
	FF(%)	25.57	26.33	27.07	27.84	28.04	29.26	30.02	30.77
8x8 Torus	LUT	71.32	73.34	74.85	77.88	83.94	95.68	97.83	99.40
	DRAM(%)	26.95	26.95	26.95	26.95	40.42	40.42	40.42	40.42
	FF(%)	27.57	28.33	29.09	29.84	30.60	41.66	42.42	43.18

with these topologies, the user-specific custom topologies can be designed, as explained in Section 4.3.1.

#### 4.4.1 FPGA synthesis results of Mesh based and Fat tree topologies

Tables 4.3, 4.4, and 4.5 show the synthesis results of Mesh, Torus, and Fat Tree topologies, respectively. Our Automated Verilog HDL Generator generates the Verilog HDL code for all these topologies. To optimize the FPGA resource usage, the router micro-architectural parameters are fine-tuned. The synthesis results presented in the tables include the percentage of LUTs, DRAMs, and FFs consumed for a particular NoC configuration.

In Table 4.3, Flit Width (FW) is varied from 16 to 32 bits, and Buffer Depth (BD) is varied between 4 to 64. An increase in the FPGA resources is observed when we increase FW and BD parameters. The LUT and FF usage is increased from 32.23% to 34.45% and 12.62% to 15.00%, considering for the BD of 4 and FW of 16 and 32 bits, respectively. Similar behavior can be observed for all the other configurations.

Table 4.5: Resource utilization of 56 node Fat tree topology under various configurations of Flit Width (FW) and Buffer Depth (BD)

FW		16bits					32bits				
BD		4	8	16	32	64	4	8	16	32	64
56N Fat tree	LUT(%)	41.21	42.62	43.68	45.80	50.04	56.96	58.37	59.07	60.13	67.20
	DRAM(%)	14.15	14.15	14.15	14.15	28.29	28.29	28.29	28.29	28.89	51.87
	FF(%)	16.06	16.59	17.12	17.65	18.18	26.06	26.59	27.12	27.65	28.18

The DRAM usage remains unchanged for the BD till 32. When we increase the BD beyond 32, an increase in the DRAM usage is observed. The same behavior is observed for both the  $6 \times 6$  Mesh and Torus topologies. The proposed design is optimized such that the DRAMs are capable of supporting the BD till 32 without any change in their usage. But, when we increase the BD beyond 32, more number of DRAMs are needed to support the configuration.

Comparing the  $6 \times 6$  Mesh and Torus topologies in Table 4.3, it can be observed that the Torus topology consumes more FPGA resources than the Mesh topology. Considering the BD of 4 and FW of 16, the  $6 \times 6$  Mesh topology consumes 32.33% LUTs and 12.62% of FFs. Whereas the  $6 \times 6$  Torus topology consumes 40.15% LUTs and 15.54% FFs. Similar behavior is observed for all the other configurations of BD and FW. The configuration of the boundary routers and more number of links present in the Torus topology results in the increase of FPGA resources compared to Mesh topology.

Table 4.4 shows the synthesis results of  $8 \times 8$  Mesh and Torus topologies by considering the BD of 4 to 64 and FW of 16 to 32 bits. When we increase the BD and FW parameters, an increase in the FPGA resources is observed. The behavior of DRAM resource consumption is similar to that of  $6 \times 6$  Mesh and Torus topologies. Inferences similar to that of  $6 \times 6$  Mesh and Torus topologies can be drawn with respect to  $8 \times 8$  Mesh and Torus topologies. Considering the FW of 32 bits and the BD of 32 and 64, the  $8 \times 8$  Mesh and Torus topologies exceeded the FPGA resources. Hence, the result for the same configurations has not been shown in Table 4.4.

Table 4.5 shows the synthesis results for the 56 node Fat tree topology. Increasing the BD and FW parameters yield an increase in FPGA resource utilization. The LUT and FF usage is increased from 41.21% to 56.96% and 16.06% to 26.06%, considering

Table 4.6: Resource utilization of a Single Router

Resource utilization of Router		
	5-port	9-port
LUT	775	2647
FF	550	1098
DRAM	120	216

the BD of 4 and FW of 16 and 32 bits, respectively. Comparing the 56 node Fat tree and  $6 \times 6$  Mesh and Torus topologies, the 56 node Fat tree consumes more hardware resources. This is because of the more number of nodes in the Fat tree compared to the  $6 \times 6$  Mesh and Torus topologies. The major findings from the experiments are summarized below:

- An increase in LUTs and FFs resource is observed when the FW and BD are increased considering all the topologies.
- The DRAM usage remains unchanged for BD parameters till 32. When BD parameter is increased beyond 32, an increase in DRAM usage has been observed.
- The Torus topology consumes more hardware resources than the Mesh topology as it contains more number of links and the configuration of the boundary routers.
- The Fat tree topology consumes more FPGA resources than the  $6 \times 6$  Mesh and Torus. And, fewer FPGA resources than the  $8 \times 8$  Mesh and Torus topologies.

#### 4.4.2 FPGA synthesis results of Custom (DMesh) topology

Table 4.6 shows the area utilization of 5-port and 9-port routers. 9-port router consumes  $2\times$  resources than that of the 5-port router as a complex control logic is required to implement a 9-port router.

Table 4.7 shows the resource utilization breakdown of router components on the Xilinx Artix7 XC7A100T device. Due to more number of ports in the DMesh topology, its components consume  $2\times$  the resources of Mesh topology.

Table 4.8 shows the results considering XY and the novel shortest path version of the XY routing algorithms for Mesh and DMesh topologies, respectively. It can be seen

Table 4.7: LUT Utilization of 5 and 9 Port Router Components

	5-port Router	9-port Router
Input buffer	240	522
Router logic	26	127
Arbiter	184	808
Crossbar	301	1093
Allocator	23	95

that the resource consumption of DMesh topology is more compared to the normal Mesh topology as there are more number of ports which in turn leads to more number of router micro-architectural components. Hence, the DMesh topology consumes  $2.3\times$  resources than the Mesh topology.

Table 4.8: Synthesis results of YaNoC on Artix-7 FPGA device (XC7A100T, speed-3)

Flit width=32-bits Flit buffer width=8				
	XY Novel shortest path XY		Table Based	
	Mesh	DMesh	Mesh	DMesh
%LUT	35.65	87.55	27.70	67.76
%DRAM	11.37	20.46	11.12	20.02
%FF	15.14	20.62	13.08	19.89

Our framework is also capable of supporting table-based routing algorithm for custom topologies. Table 4.8 shows the synthesis results of Mesh and DMesh topologies considering the table-based routing. The table-based routing consumes 12% and 20% fewer LUTs compared to XY and novel shortest path version of the XY routing algorithms for Mesh and DMesh topologies, respectively. This is because, the route compute logic in these algorithms has been replaced by the routing tables. The routing tables store route to all the other nodes in the topology. As the entries in routing tables are maximum of 3 bits wide, they are mapped very efficiently to the LUTs.

#### 4.4.3 Latency analysis

Average packet latency comparison of Mesh, Torus, Fat tree, and Dmesh topologies is described in this section.

Fig. 4.8 shows the average packet latency of  $6 \times 6$  Mesh and Torus topologies under

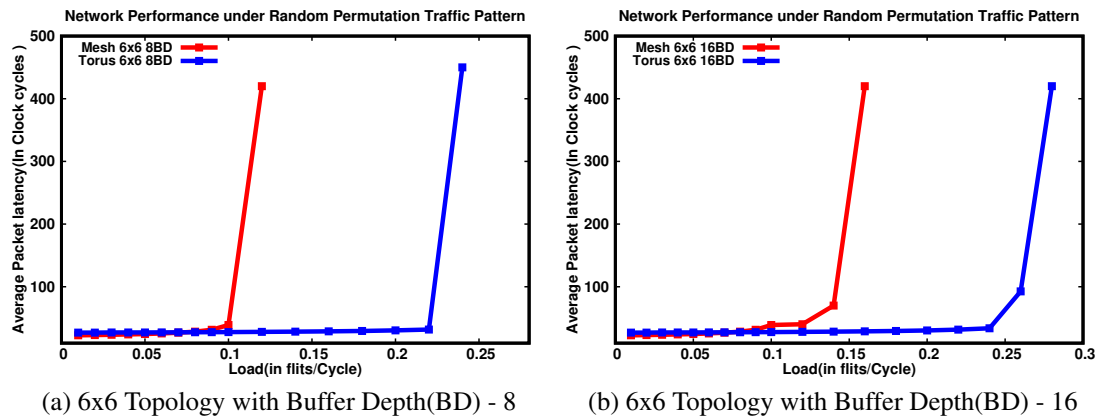


Figure 4.8: Load Delay graph of 6x6 Mesh and Torus Topologies under Random Permutation Traffic patterns (a) Buffer Depth=8 flits and (b) Buffer Depth=16 flits

random permutation traffic pattern. From Fig. 4.8(a), the Mesh topology has lower average packet latency at lower injection rates compared to the Torus topology. An increase in latency is observed with increasing the injection rate. Mesh topology is the first to saturate at about 10% of the traffic load. The Torus topology saturates at 22% of the traffic load. Under Random permutation traffic pattern, the Torus topology showed fairly good performance compared to Mesh topology. From Fig. 4.8(b), 5% (i.e., from 10% to 15%) increase in saturation throughput of Mesh topology is observed. This is due to the effect of increase in the size of the BD. Mesh and Torus topologies saturate at 15% and 24% of the traffic, respectively. The packet latency decreases significantly across all loads as we move from BD of 8 to 16 flits.

Fig. 4.9 shows the network performance of the  $8 \times 8$  Mesh and Torus topologies under Bit complement traffic. From Fig. 4.9(a) we observed that the Mesh topology saturates at lower traffic loads compared to Torus topology. The Torus topology saturates at 28% of the traffic load. As we increase the BD from 8 to 16, the saturation throughput of Mesh and Torus topologies increase by 25% and 12%, respectively as shown in Fig. 4.9(b). The average packet latency reduction of 3.5% is observed increasing the BD from 8 to 16.

Network performance of Fat tree topology under Random permutation traffic is shown in Fig. 4.10 (a). We observed that the 56-node Fat tree with BD of 16 has a higher saturation throughput compared to BD of 8. The larger BD accommodates more

packets resulting in a reduction of the packet contention in the network. The 56-node Fat tree with BD of 8 and 16 saturate at 40% and 45% of the traffic load, respectively. The average packet latency reduction of 20.5% is observed when the BD is increased from 8 to 16.

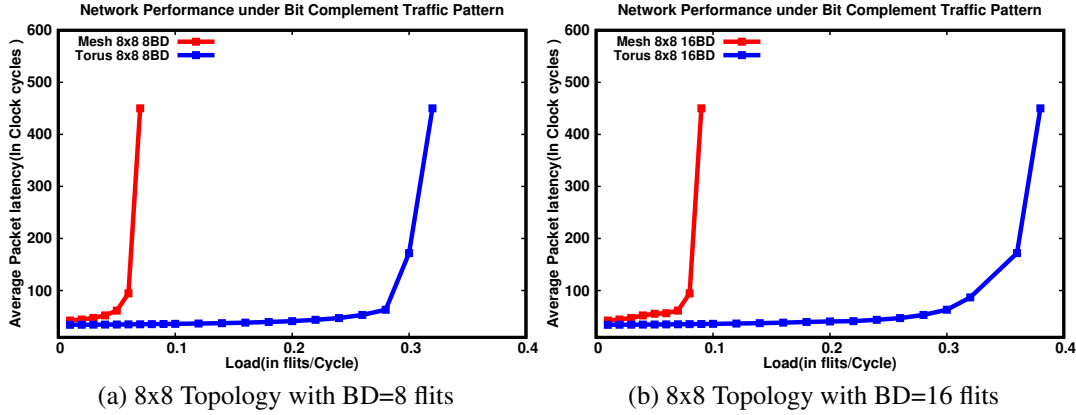


Figure 4.9: Load delay graph of 8x8 Mesh and Torus topologies under Bit complement traffic patterns (a)buffer depth=8flits and (b)buffer depth=16flits

Fig. 4.10 (b) plots the behavior of average network packet latency vs. injection rate under Uniform random traffic pattern. It can be seen that the Mesh topology saturates at the injection rate of 45%. DMesh topology sustains the traffic load till injection rate of 55%. This is because of the higher Bisection bandwidth and connectivity of DMesh topology. The maximum hops ( $H_{max}$ ) for a packet to traverse from one end to its diagonally opposite end in Mesh and DMesh topology can be calculated using the equations 4.4, 4.5, 4.6 and 4.7 where M and N are the number of nodes along X and Y axes. When M and N equal, from Equations 4.5 and 4.6 it can be seen that DMesh takes 50% of the number of hops taken in Mesh topology. Hence, the latency in DMesh is less than the latency in conventional XY routing in Mesh. As there are diagonal links between nodes in DMesh topology, our algorithm chooses the shortest path leading to the destination.

$$H_{max}(Mesh) = (M + N) - 2 \quad (4.4)$$

$$H_{max}(Mesh) = 2(N - 1) \quad \text{if } M = N \quad (4.5)$$

$$H_{max}(DMesh) = (N - 1) \quad \text{if } N > M \quad (4.6)$$

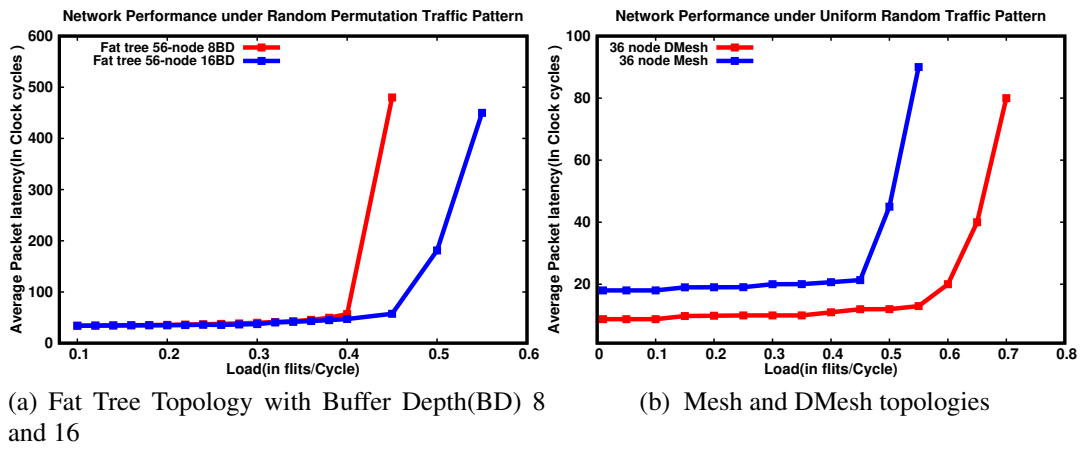


Figure 4.10: (a) Load delay graph of Fat tree with buffer depth 8 and 16 flits under Random permutation traffic pattern (b) Load delay graph for Mesh and DMesh topologies under Uniform traffic

$$H_{max}(DMesh) = (M - 1) \quad \text{if } M > N \quad (4.7)$$

On an average, DMesh topology offers 50% lesser latency than the Mesh topology.

Table 4.9: Synthesis results of 36-Node Mesh based topology on Artix-7 FPGA device (XC7A100T, speed-3)

6x6 Mesh based topology (Flit Width=32-bit, Buffer Depth=8)		
H/W	XY	Proposed
utilization in %	routing	Adaptive routing
LUTs	35.65	37.31
DRAM	11.37	11.37
FFs	15.14	15.12

#### 4.4.4 Analysis of Mesh topology with congestion aware adaptive routing algorithm

6×6 Mesh topology is evaluated considering the conventional XY and the proposed congestion aware adaptive routing algorithms. Hardware synthesis and latency analysis results are detailed below.

##### 4.4.4.1 Area resource utilization

Table 4.9 shows the detailed synthesis results of XY and proposed adaptive routing algorithms for Mesh-based topologies. It can be seen that both algorithms consume the

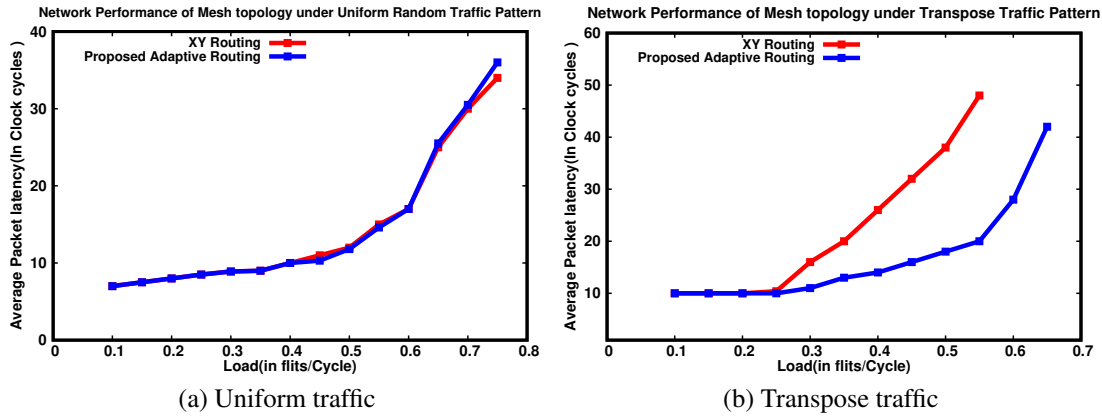


Figure 4.11: Load Delay graph of Mesh Topology under (a)Uniform and (b)Transpose traffic patterns

same amount of DRAMs and FFs. The adaptive routing algorithm consumes 1.66% more LUTs than the XY routing algorithm to store the 2-bit congestion information of the neighboring routers and the routing logic.

#### 4.4.4.2 Network latency analysis

The conventional XY and proposed adaptive routing algorithms are evaluated considering uniform and transpose traffic patterns. In the uniform traffic pattern, each node sends a fixed-size packet consisting of 8 flits to random nodes with Bernoulli distribution. From Fig. 4.11 (a) it can be seen that both the XY and proposed routing algorithms exhibit similar behavior for all the injection rates.

In transpose traffic pattern, a node  $(i,j)$  sends packets only to node  $(n-i,n-j)$  where “ $n$ ” is the network dimension. In this scenario, the XY routing saturates early compared to the proposed adaptive routing algorithm. From Fig. 4.11 (b), it can be seen that at the higher injection rates, the proposed adaptive routing algorithm for Mesh-based topologies outperforms XY routing by reducing average packet latency by 55%.

#### 4.4.5 Speedup

The simulation time of BookSim 2.0 simulator is measured on a computer with Core i7 4770 CPU and 8GB memory. The speedup is calculated as the ratio of simulation time in clock cycles of BookSim 2.0 to the simulation time of YaNoC. The simulation



for a  $6 \times 6$  network was run on both BookSim 2.0 and YaNoC. A speedup of  $2548\times$  is observed over BookSim 2.0 simulator.

## 4.5 YANOC VS. STATE-OF-THE-ART

### 4.5.1 YaNoC and CONNECT

Table 4.10: Resource utilization of CONNECT and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for  $6 \times 6$  Mesh and DMesh topologies

		XY Routing (% Utilization)		Table based (% Utilization)	
		CONNECT	YaNoC	CONNECT	YaNoC
Mesh	LUT	44.94	35.65	43.71	27.70
	DRAM	27.54	11.37	26.39	11.12
	FFs	6.71	15.14	5.91	13.08
DMesh	LUT	Exceed	87.55	Exceed	67.76
	DRAM	Exceed	20.46	Exceed	20.02
	FF	Exceed	20.62	Exceed	19.89

The Verilog HDL code of  $6 \times 6$  Mesh and custom DMesh topologies are generated from CONNECT (Papamichael and Hoe 2015) and YaNoC frameworks for comparing the hardware resource utilization.

Considering XY routing algorithm, it can be observed from Table 4.10 that YaNoC’s implementation of  $6 \times 6$  Mesh topology consumes fewer resources (35.65% LUTs) than CONNECT’s Mesh topology (44.94% LUTs). Similar behavior is observed for the Table based routing algorithm. YaNoC’s Table based routing ensures that always the shortest path is chosen between the communicating routers.

Also, for custom DMesh topology, the synthesis will not succeed as there is a resource crunch employing CONNECT’s implementation. The Input Output Blocks (IOBs) will exceed the limit of Artix7 FPGA board. Whereas, synthesis of YaNoC’s HDL code succeeds, and the corresponding resource utilization is shown in Table 4.10. Same behavior is observed for both the XY and table based routing algorithms.

Speedup of  $500-1000\times$  and  $2548\times$  has been observed in CONNECT and YaNoC, respectively with respect to BookSim 2.0. YaNoC is  $2.55\times$  faster than CONNECT NoC generator.

Table 4.11: Resource utilization of DART and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for  $3 \times 3$  Mesh topology

	DART	YaNoC
%LUTs	30	12.41
%DRAMs	21.74	5.68
%FFs	19.28	5.40

#### 4.5.2 YaNoC and DART

$3 \times 3$  network with XY routing algorithm of DART (Wang et al. 2014) and YaNoC are compared in Table 4.11. It can be observed that % LUT consumption is 12.41 and 30 for YaNoC and DART implementations, respectively. Large topologies can be analyzed by using YaNoC’s implementation on a small FPGA board like Artix7. Whereas, the DART implementation consumes more FPGA resources and hence it requires high-end FPGA boards for the analysis. DART simulation achieves over  $100\times$  speedup relative to BookSim 2.0. YaNoC is  $25\times$  times faster than DART.

#### 4.6 SUMMARY

An FPGA based simulation acceleration framework for design space exploration of Network-on-Chips called YaNoC is presented in this chapter. YaNoC supports the design space exploration of various standard and custom NoC topologies. The router micro-architectural parameters are highly configurable. The conventional routing algorithms such as XY, Nearest neighbor are supported for mesh-based and Fat tree topologies. To support the design space exploration of custom topologies, YaNoC supports the Table based routing algorithms. Also, a congestion-aware adaptive routing algorithm has been proposed to route the flits. The Flit Width and Buffer Depth parameters are varied to identify their effect on the performance of the network and the topologies. An increase in LUTs and FFs resource has been observed varying the FW and BD in all the topologies. YaNoC consumes fewer hardware resources than the state-of-the-art CONNECT and DART frameworks. And, YaNoC is  $2.55\times$  and  $25\times$  faster than CONNECT and DART frameworks, respectively.

## **CHAPTER 5**

### **MAPPING THE NOC ROUTER COMPONENTS ON THE HARD-BLOCKS OF THE FPGA**

The FPGA based NoC simulation framework - YaNoC proposed in the previous chapter and the other state-of-the-art works utilize soft logic only for modeling the NoCs on the FPGAs, leaving out the hard blocks unutilized. The functionality of the NoC router's buffer and crossbar switch have been embedded in the BRAMs and the wide multiplexers of the DSP48E1 slices in this Chapter. Employing the proposed techniques of mapping the NoC router components on the FPGA hard blocks, an FPGA based NoC simulation framework has been proposed in this chapter. A substantial decrease in the CLB utilization of NoC topologies on the FPGA has been observed by embedding the functionality of the buffers and crossbar on the hard blocks of the FPGA.

#### **5.1 INTRODUCTION**

The FPGAs are now vehicles for simulation acceleration due to their properties of parallelism. The ASIC like hard blocks are embedded in the modern FPGAs to improve the performance of common functionalities. The functionalities such as multiply-accumulate, processing and data storage can be performed with the help of embedded hard blocks such as DSP slices, Embedded processors and Block RAMs (BRAMs) respectively. In addition to their usability, these hard blocks can also be used to support the other functionalities.

The dual-port memory blocks with separate ports for writing or reading form the

BRAMs. The BRAMs are capable of storing several Kilobits of data. As the BRAMs can be configured as memories with dual or single-port supporting various port widths, they can be employed as an efficient on-chip memory. BRAMs can be configured as a memory with large capacity with the help of cascading links present in the latest FPGAs. In addition, BRAMs can be used as register files, FIFOs, etc.

DSP slices are primarily employed to perform signal processing tasks such as multiply-accumulate and multiply efficiently. The arithmetic and logic operations along with shift and pattern matching operations are supported by DSP slices of FPGAs. Multiple DSP slices can be cascaded to support the operations wider than the port widths supported. The DSP slices in the most recent Xilinx 7 Series FPGAs have a 740 MHz (Xilinx Inc 2018) operating frequency.

All these features open up the opportunities for employing BRAM and the DSP blocks for the applications other than Signal processing. The dual-port BRAM hard blocks of the FPGA are employed to support the FIFO buffer functionality. And, the five port NoC router crossbar switch has been embedded in a DSP tile consisting of two DSP48E1 slices. An FPGA based NoC simulation acceleration framework is presented in this chapter. The proposed framework is capable of utilizing both the Soft blocks (CLBs made up of LUTs and FFs) and the Hard blocks (DSP48E1 slices and BRAMs) of the Xilinx FPGAs. These characteristics allow us to make more efficient use of the FPGA resources by mapping NoC topologies on both the soft and hard blocks.

Many state-of-the-art works utilize only the CLBs components (LUTs and FFs) of the FPGA building the NoC simulators. The other components of the FPGA, such as the DSP slices and BRAM blocks, which form the hard blocks, can be utilized to map the NoC router component functionality. This results in reduced area resource utilization of the soft logic substantially. The multiplexer implementation on the FPGAs consumes more soft logic (CLBs), resulting in increased power and critical path delay. As an alternative approach of modeling the Crossbar component of NoC routers by employing the CLBs, the multiplexers present in the DSP48E1 slices can be used efficiently to configure them as the Crossbar component of an NoC router. As the DSP48E1 slices operate at a higher frequency, mapping the crossbar functionality yields the higher op-

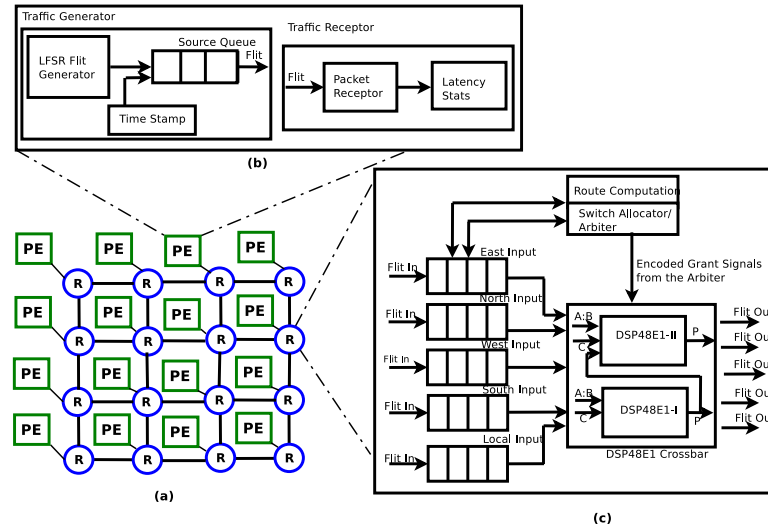


Figure 5.1: Functional diagram of the proposed FPGA based NoC framework. (a) an NoC topology, (b) Processing Element (PE), (c) Proposed router architecture

erating frequency of the whole circuit with reduced power consumption. The modern FPGAs have a large number of BRAMs and DSP slices, due to which the higher frequency of operation and an increase in execution speed and lower power consumption can be achieved.

## 5.2 NOC ROUTER ARCHITECTURE

Fig. 5.1 shows an overview of the proposed FPGA based NoC simulator. The Processing Elements (Fig. 5.1 (a)) (the traffic source and sink in our case) are interconnected with the help of routers. The traffic generation module shown in Fig. 5.1 (b) consists of the Source queue and the Traffic receptor modules. The Linear Feedback Shift Register (LFSR) technique is used to generate several types of synthetic traffic patterns such as Uniform random, Transpose, Random Permutation, Hotspot, Nearest neighbor, Tornado and Bit-complement.

The Source queue stores the flits generated by the traffic generation module before injecting them into the network. The source queues operate in the FIFO fashion. A timestamp is appended to the head flits for indicating the time of injecting the flits into the network.

The Traffic receptor module accounts for ejecting the flits once they reach their

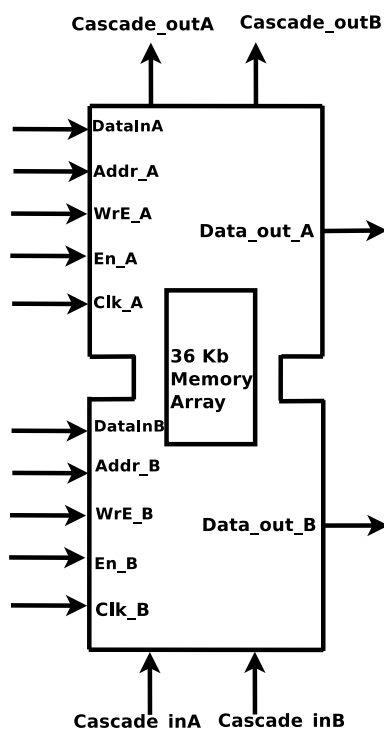


Figure 5.2: Architecture of the Xilinx BRAM hard block (Xilinx Inc 2019)

destinations. Also, the statistics of the simulation, such as the number of flits received, the number of flits transmitted, and the packet latency, are calculated based on the time stamp stored in the head flit.

In this chapter, the crossbar functionality of an NoC router is mapped on the DSP48E1 hard blocks and the BRAMs are used for implementing the buffers. The framework is capable of using both CLBs and the hard blocks for simulating NoC on the FPGA. The proposed NoC router architecture is shown in Fig. 5.1 (c).

### 5.3 BLOCK RAMS AS THE BUFFERS

Fig. 5.2 shows the architecture of the Xilinx BRAM block (Xilinx Inc (2019)). The BRAM blocks are capable of storing the data of size 36Kbits. They can also be configured either as two standalone 18 Kb RAMs or as a 36 Kb RAM. The Write and Read operations are synchronous, where these two ports are independent and symmetrical. The Write and Read ports share only the stored data. Each port can be configured independently of the other port in one of the available widths. Also, the read port's width may not be the same as the width of the write port.

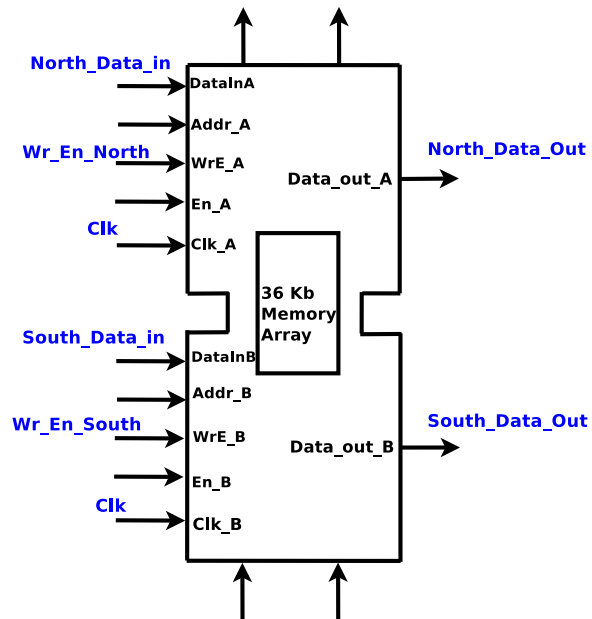


Figure 5.3: Illustration of mapping the input ports to the BRAM based buffer

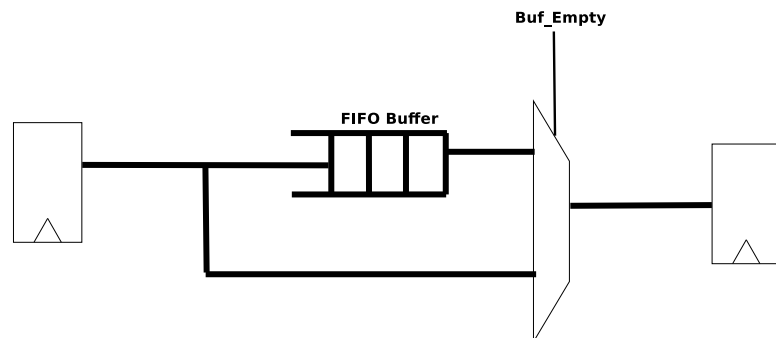


Figure 5.4: Bypassing an empty Buffer

Fig. 5.3 shows an example of mapping the input port data to the dual-port BRAM block. It can be seen that the North port's input data "North\_Data\_in" is mapped to the "DataInA" port of the BRAM block. Also, the write enable signal "Wr\_En\_North" is mapped to the "WrE\_A" port. Similarly, the "North\_Data\_in" and "Wr\_En\_North" are mapped to "DataInB" and "WrE\_B" ports along with clock "Clk" are mapped to "Clk\_A" and "Clk\_B" ports. The output data of North and South ports "North\_Data\_Out" and "South\_Data\_Out" can be seen at "Data\_out\_A" and "Data\_out\_B" ports.

To reduce the latency of the NoC under consideration, a control unit called "buf\_empty\_checker" for monitoring the status of the buffer occupancy has been employed at the input ports as shown in Fig. 5.4. When the input buffer is empty, and

the desired output port is available for the incoming flit, the “buf\_empty\_checker” control unit makes sure that there shall be no request sent to the arbiter for allocating that output port. The data from the input port shall be bypassed to the output port without going through the arbitration stage. A reduction of two clock cycles in latency can be observed by employing the “buf\_empty\_checker” unit.

## **5.4 DSP48E1 TILE AS THE CROSSBAR SWITCH**

### **5.4.1 Xilinx DSP48E1 primitives**

The architecture of the Xilinx DSP tile is shown in Fig. 5.5. The DSP48E1 slices present in each DSP tile are interconnected with the help of the cascaded links Xilinx Inc (2018). Each DSP48E1 slice contains various functional units such as a pre-adder, multiplier, and an Arithmetic and Logic Unit (ALU). The pre-adder unit is responsible for addition or the subtraction of 25-bit two inputs. The asymmetric 25-bit and 18-bit inputs along with the pre-adder unit’s output form the input to the multiplier unit. ALU’s inputs are formed from the output of the multiplier and the other 48-bit input. The DSP blocks allow the dynamic reconfiguration of the arithmetic computations and the data flow operations during runtime. Hence, by employing the technique of time-multiplexing and the dynamic reconfiguration, complex data flow expressions requiring complex computational power can be executed on the same DSP48E1 slice. The DSP slice’s dynamic operation is achieved by modifying the OPMODE control signal of the multiplexers.

### **5.4.2 Crossbar functionality on the DSP48E1 multiplexers**

The ALU unit, multiplexers (namely X, Y, and Z) present in the DSP slice and their control signals such as INMODE, ALUMODE, and OPMODE play an important role in configuring the Crossbar switch functionality of the NoC router on the DSP tile. The multiplexers are controlled dynamically based on the configuration of the OPMODE signal. The mode of operation of the ALU unit such as the logical or the arithmetic mode is controlled by the ALUMODE signal. In this work, the ALUMODE signal is configured to make the ALU unit perform  $(X+Y+Z)$  operation.

A DSP tile which contains two DSP48E1 slices is used to map the crossbar func-



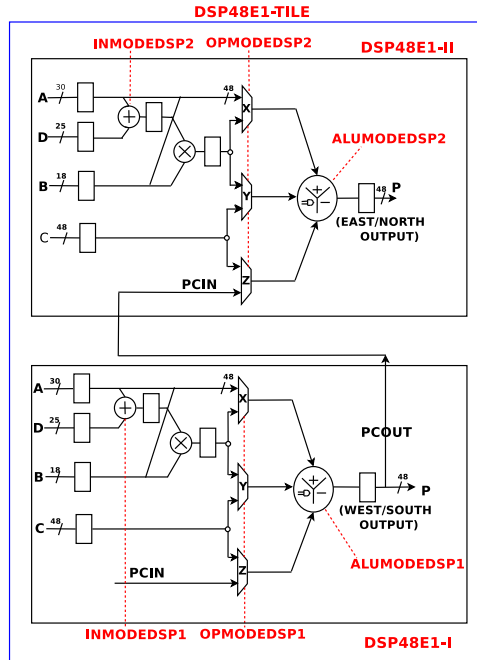


Figure 5.5: Two DSP48E1 slices connected by dedicated cascade links form a single DSP tile (Xilinx Inc 2018)

tionality of a NoC router. These two DSP48E1 slices are interconnected with the help of the PCIN, and PCOUT cascaded links. All the input ports except the local input have been mapped to a 4:1 multiplexer to achieve better performance. The 4:1 multiplexer bypasses the packets to the local output port depending on the arbiter signals for local output. Using this circuitry has the advantage of bypassing the packets instead of heading towards the crossbar to map the inputs to the local output.

An illustration of the implementation of the NoC Crossbar switch using the DSP48E1 slices is shown in Fig. 5.6. The A:B and C input ports of the DSP48E1-I and DSP48E1-II slice are used for mapping the crossbar switch's inputs. As the DSP48E1 slices support the dynamic reconfiguration, the crossbar's inputs are mapped on the A:B and C inputs of the DSP48E1 slices efficiently. To reduce the latency for the packets destined for the sink at the LOCAL output port, a 4:1 multiplexer has been introduced in the circuitry. By having the 4:1 multiplexer, the packets destined to the LOCAL output port can now be bypassed directly to the sink instead of going through the crossbar traversal.

In the proposed work, the configuration of the OPMODE signals for X, Y, and Z multiplexers of the DSP48E1 slices is highly dependent on the one-hot encoded signals

## 5. Mapping the NoC router components on the Hard-blocks of the FPGA

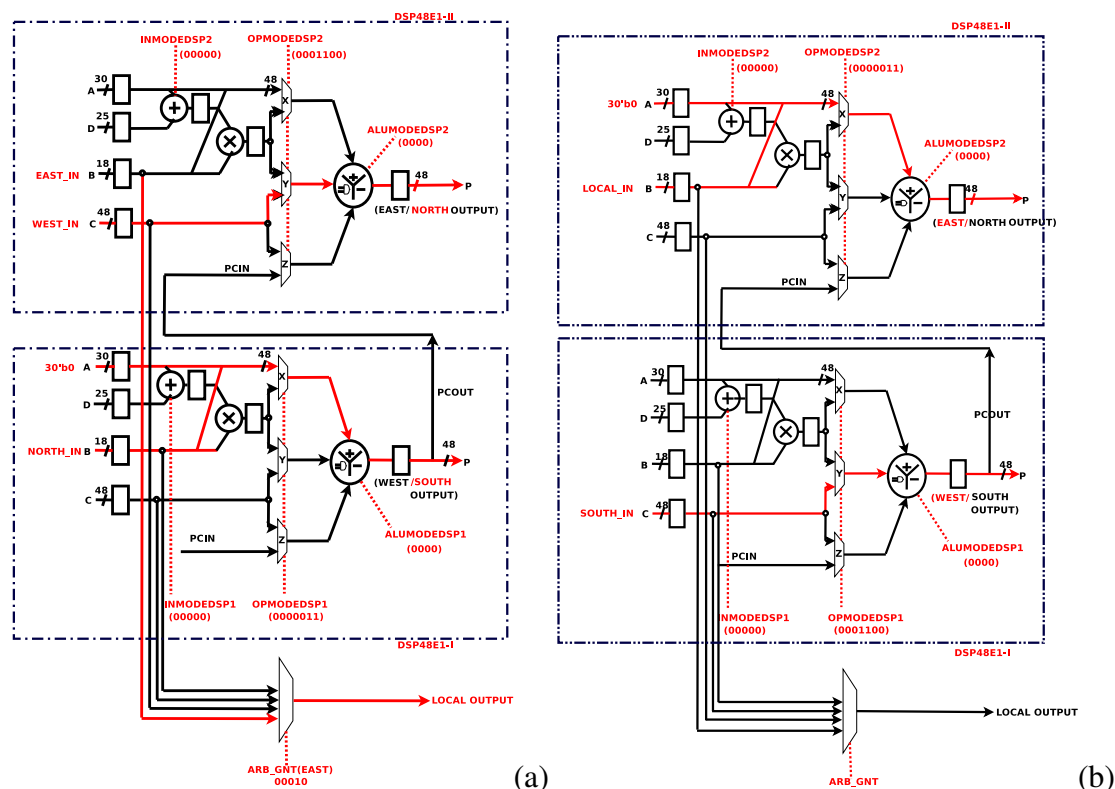


Figure 5.6: Illustration of mapping the input ports to the DSP48E1 based crossbar

Table 5.1: 4:1 Multiplexer operating signals based on the grant signals from the arbiter

Output port	Input port	Arbiter encoded signal
Local	East	00010
	North	00100
	West	01000
	South	10000

Table 5.2: DSP48E1-I slice configuration based on the arbiter encoded signal

Output port	Input port	Arbiter encoded signal	DSP48E1-I input	OPMODE signal
West	Local	00001	C	0001100
	East	00010	A:B	0000011
	North	00100	C	0001100
	South	10000	A:B	0000011
South	Local	00001	A:B	0000011
	East	00010	C	0001100
	North	00100	A:B	0000011
	West	01000	C	0001100

Table 5.3: DSP48E1-II slice configuration based on the arbiter encoded signal

Output port	Input port	Arbiter encoded signal	DSP48E1-II input	OPMODE signal
East	Local	00001	A:B	0000011
	North	00100	C	0001100
	West	01000	A:B	0000011
	South	10000	C	0001100
North	Local	00001	C	0001100
	East	00010	A:B	0000011
	West	01000	C	0001100
	South	10000	A:B	0000011

provided by the Arbiter. The one-hot encoded signals from the Arbiter for particular input ports are as follows: 00001 - Local, 00010 - East, 00100 - North, 01000 - West, 10000 - South. The enable signals for the 4:1 Multiplexer depending on arbiter grant signals are shown in Table 5.1. Various configurations of OPMODE signals for other input ports and the mapping of the input ports to the corresponding DSP48E1 slice input ports are shown in Tables 5.2 and 5.3. Various arbitration schemes, such as fixed priority, weighted round-robin, and round-robin, are supported by the proposed framework. The round-robin arbitration scheme is employed in this work to generate the grant signals. The 5-bit one-hot encoded signal is allocated to each input port winning the arbitration. The crossbar maps the input to the respective output port, which wins the arbitration stage based on these signals. An approach similar to this is used in our work: depending on these one-hot encoded signals from the arbiter, router input ports are mapped to DSP48E1 slice inputs. The one-hot encoded signals from the arbiter are used to configure the OPMODE control signals.

To overcome the deadlock state arising due to turns taken by the packets, the turn models of Glass and Ni (1992) have been employed in this work. We also make an assumption that an input port  $i$  only sends the data to the output port  $j$  where  $i \neq j$ . The deadlock situations and the critical path delay can be avoided by the assumption being made.

The configuration of the inputs at the right DSP48E1 slice has been carried out employing the Time-multiplexing technique. An illustration of mapping the NoC router's

crossbar switch on the DSP tile is shown in Fig. 5.6. Suppose the flits of the router's WEST input are allocated with the NORTH output port and NORTH inputs are allocated with the SOUTH output port in the first clock cycle as shown in Fig. 5.6(a), the arbiter grant signals are monitored to configure the router inputs to the DSP crossbar. Corresponding to the arbiter grant signals i.e., 01000 and 00100, the WEST input is configured on the C input of DSP48E1-II and the NORTH input is configured on the A:B input of DSP48E1-I by setting OPMODEDSP2 to 0001100 and OPMODEDSP1 to 0000011. This configuration maps the WEST and NORTH input ports to the NORTH and SOUTH output ports effectively. Further, when the EAST input is granted with the LOCAL output port, the 4:1 multiplexer maps the EAST input port's flits to the sink at LOCAL output port, avoiding the DSP tile.

In the second clock cycle, when the LOCAL and SOUTH input ports are granted with the EAST and WEST output ports, the arbiter grants 00001 and 10000 signals to achieve this configuration. With these grant signals, the LOCAL input is mapped to A:B input of DSP48E1-II slice, and SOUTH input is mapped to the DSP48E1-I slice. And the OPMODEDSP1 signal is configured to 0001100 and OPMODEDSP2 signal is configured to 0000011, as shown in Fig. 5.6 (b).

## 5.5 RESULTS AND DISCUSSION

The impact of varying the NoC parameters such as flit width, buffer width, buffer type, crossbar type, and traffic patterns is studied in the experiments. The area and latency performances are analyzed. The Xilinx Artix 7 FPGA board with XC7A100 T chip has been employed in the experiments. The micro-architectural components of the NoC architectures are designed by using Verilog HDL. The synthesis results are obtained from Xilinx Vivado. The configurations used in the experiments is shown in Table 5.4.

### 5.5.1 FPGA utilization results

#### 5.5.1.1 Router implementation

Table 5.5 shows the resource utilization of the Router architecture employing the CLB and BRAM-DSP slices is shown. The resource utilization results for the CLB based implementation of the topologies are obtained through YaNoC (Prabhu Prasad B M

Table 5.4: Experimental setup details

Experimental setup	
Topology	$3 \times 3$ and $6 \times 6$ Mesh-based topologies
Arbiter type	Fixed priority and Round-robin
Flow control	Wormhole switching
Crossbar mapping	DSP48E1 and CLB based crossbar
Routing Algorithm	Dimension-order (XY)
Router pipeline depth	5-stage
FIFO Buffer type	CLB and BRAM based buffer
Buffer depth	6,8,10
Traffic pattern	Random Permutation, Hotspot, Nearest Neighbor, Tornado, Bit-complement, Uniform Random, Transpose
Packet length	4-flits
Flit size	64, 128
FPGA Board	Xilinx Artix 7 (XC7A100T)

Table 5.5: Resource utilization of NoC Router considering CLB and BRAM-DSP mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA

Router Type	Components	LUTs		FFs		BRAMs		DSPs	
		#	%	#	%	#	%	#	%
CLB Router	Router	735	1.16	545	0.43	-	-	-	-
	FIFO	45	0.07	62	0.05	-	-	-	-
	Crossbar	243	0.38	258	0.2	-	-	-	-
BRAM DSP Router	Router	544	0.86	373	0.29	5	3.7	2	0.83
	FIFO	18	0.03	14	0.01	1	0.74	-	-
	Crossbar	163	0.25	148	0.11	-	-	2	0.83

et al. (2018)). The router with only CLB implementation consumes 1.16%, 0.43% LUTs and FFs, respectively, without any BRAM or the DSP slice utilization. The router implementation employing the FIFO buffer based BRAMs and DSP based crossbar consumes 0.86%, 0.29% LUTs and FFs along with 3.7% and 0.83% BRAMs and DSPs, respectively.

The CLB based FIFO implementation consumes 0.07%, 0.05% LUTs and FFs respectively. And, the BRAM based FIFO implementation consumes 0.03%, 0.01% LUTs and FFs along with 0.74% BRAMs. The CLB based Crossbar implementation consumes 0.38%, 0.2% LUTs and FFs respectively. And, the DSP based Crossbar implementation consumes 0.25%, 0.11% LUTs and FFs along with 0.83% DSPs.

The height of a DSP48E1 tile is equal to that of five CLBs Xilinx Inc (2018). Every CLB is composed of 16 FF and 8 LUTs. Thus, each DSP48E1 is equivalent to 80FFs and 40 LUTs . For the applications that leave the hard blocks resources unused, crossbar configuration on the DSP tiles does not reflect any loss in the FPGA area utilization.

### 5.5.1.2 Topology implementation

Tables 5.6, 5.7 show the FPGA synthesis results of 36-node Mesh and Torus topologies considering the proposed BRAM-DSP implementation and the CLB based implementation. XY routing algorithm is employed in the experiments. By supporting the parameterized values of Flit Width (FW) and BD (Buffer Depth), the proposed framework is capable of modelling various NoC architectures. Various configurations of BD and FW are shown in ‘Configuration’ column. The ‘CLB Implementation’ column shows the FPGA resource consumption in ‘%’ considering the CLB implementation of the topologies, and the ‘BRAM-DSP implementation’ column shows the FPGA resource consumption in ‘%’ considering the BRAM based FIFO and DSP based crossbar implementation.

From both tables it can be shown that the CLB based implementation of the Topologies consume more LUTs and FFs than the BRAM-DSP based implementation. This is because of the efficient mapping of the NoC Router’s FIFO and Crossbar components on the BRAM and DSP48E1 blocks.

In Table 5.6, considering the  $6 \times 6$  Mesh topology, for the BD of 6, when the FW is increased from 64 to 128, an increase in the LUT utilization from 37.42% to 41.08%, FF utilization from 15.32% to 15.69% and BRAM utilization from 49.55% to 54.13% are observed. When we increase the FW and BD configurations, the consumption of LUT, FF, and BRAM resources also increased with respect to all the topologies. Similar behavior can be observed in the Torus topology. As the Torus topology consists of wrap-around links in the boundary routers, the Torus topology has a higher hardware resource consumption compared to the Mesh topology.

On average, the topologies implemented considering the proposed BRAM-DSP components of the FPGA consumes 43.47%, 41.66% fewer LUTs, FFs, respectively,

compared to the topologies implemented considering the CLB only implementation. It can be from all the experimental observations that the topologies implemented with the proposed BRAM-DSP mapping of NoC router components occupy fewer FPGA soft logic resources compared to the the topology implementation based on the CLBs.

Table 5.6: Resource utilization of  $6 \times 6$  Mesh topology with CLB and BRAM-DSP mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA with XY routing

Configuration		CLB			BRAM-DSP			
Flit width	Buffer depth	LUT%	FF%	LUTRAM%	LUT%	FF%	BRAM%	DSP%
64	6	61.89	23.04	41.68	37.42	15.32	49.55	30
	8	62.74	23.48	41.68	40.83	16.19	61.18	30
	10	63.60	23.89	41.68	42.81	17.02	66.67	30
128	6	88.86	32.12	83.87	41.08	15.69	54.13	30
	8	89.72	32.55	83.87	42.65	16.65	64.81	30
	10	90.57	32.98	83.87	43.2	17.4	67.14	30

Table 5.7: Resource utilization of  $6 \times 6$  Torus topology with CLB and BRAM-DSP48E1 mapping of FIFO and Crossbar on Artix 7(XC7A100T) FPGA with XY routing

Configuration		CLB			BRAM-DSP			
Flit width	Buffer depth	LUT%	FF%	LUTRAM%	LUT%	FF%	BRAM%	DSP%
64	6	62.74	23.48	41.68	41.83	15.78	51.86	30
	8	62.91	23.61	41.68	42.57	16.22	62.96	30
	10	64.98	24.28	41.68	43.73	16.73	66.67	30
128	6	88.98	33.21	83.87	45.98	15.96	59.26	30
	8	90.31	33.84	83.87	47.55	16.94	66.73	30
	10	90.97	34.13	83.87	48.47	17.63	68.14	30

## 5.5.2 Latency and saturation throughput analysis

The network latency is the amount of clock cycles taken by a packet from a given source to reach a destination. Equation 5.1 gives the average packet latency:

$$L_{\text{avg}} = 1/N \sum_{i=1}^k L_i \quad (5.1)$$

where  $L_{\text{avg}}$  is the average latency, the total amount of flits reached the destination is denoted by  $N$ . And, the latency experienced by  $i^{\text{th}}$  flit at the destination is denoted by  $L_i$ . The amount of flits transported from the source to the destination per a given unit of time is called the throughput of the network.

The workload for NoC is modeled by the injection rate, length of the packets, and the pattern of the traffic. Increased injection rate results in an increased number of network packets. When the number of packets is more, a considerable amount of clock cycles are taken by the packets to reach the destination from a given source. This contributes to higher packet latency. When the injection rate is further increased, the network nears a threshold, which leads to congestion. Packets are subject to exponential latency growth due to congestion.

Various traffic patterns such as Bit-complement(BC), Random Permutation(RP), Hotspot(HS), Nearest Neighbor(NN), and Tornado(TO) are used in the experiments to analyze the latency-throughput performance of the  $6 \times 6$  Mesh and Torus topologies considering the CLB and BRAM-DSP architecture of the NoC router.

The latency behavior of the 36-node Mesh and Torus topology implementations based on the CLB and BRAM-DSP with respect to various traffic patterns referred above is shown in Fig. 5.7. The average packet latency is denoted in clock cycles, and the injection load is denoted flits per clock cycle per node.

From Fig. 5.7, it can be seen that the topologies employing the proposed router architecture perform better compared to the CLB based router architecture under all the traffic patterns. This is because of the “buf\_empty\_checker” control unit present in the router. When the input buffer is empty, and the desired output port is available for the incoming flit, the buf\_empty\_checker control unit makes sure that there shall be no packet sent to the arbiter for allocating that output port. The data from the input port shall be bypassed to the output port without going through the arbitration stage. A reduction of two clock cycles in latency can be observed by employing the “buf\_empty\_checker” unit.

In Fig. 5.7, considering the 36-node Mesh topology implemented with the proposed BRAM-DSP based router architecture, a reduction in average latency by 3.78%, 2.39%, 3.14%, 2.15%, 1.61% under the RP, BC, NN, TO and HS traffic patterns has been observed compared to the CLB based topology implementation. Similarly, the 36-node Torus topology with the proposed router architecture achieves the average latency re-



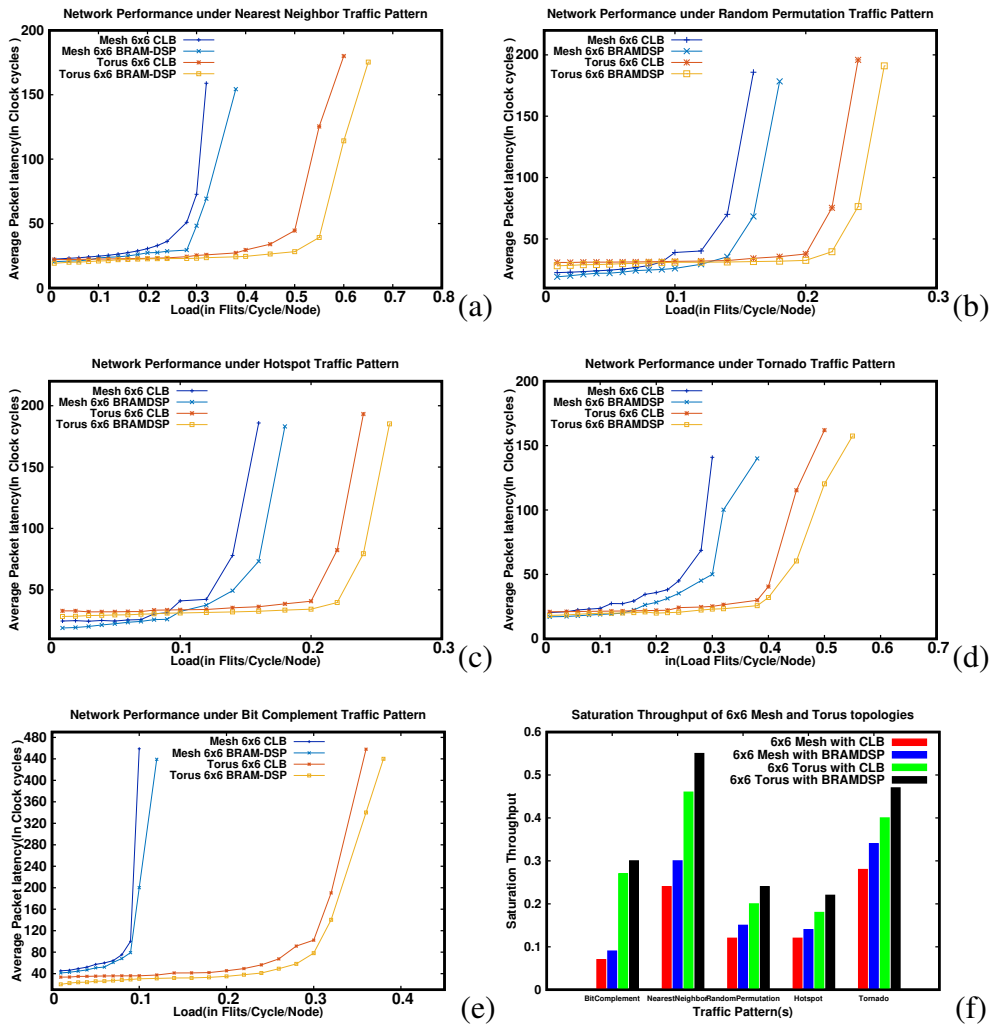


Figure 5.7: (a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the  $6 \times 6$  Mesh and Torus topologies under CLB and BRAM-DSP based FIFO and crossbar implementation considering NN, RP, HS, TO and BC traffic patterns

reductions of 2.05%, 3.93%, 2.78%, 3.09%, 4.15% considering the RP, BC, NN, TO and HS traffic patterns compared to the CLB based topology implementation.

The saturation throughput of 36-node Mesh and Torus topologies with respect to the traffic patterns mentioned above considering the CLB and BRAM-DSP topology implementation is shown in Fig. 5.7(f). The topologies perform better under the proposed mapping of FIFO buffer on BRAM and Crossbar on the DSP implementation along with the “buf\_empty\_checker” control unit to bypass the flits the output unit. Due to its higher bisection bandwidth, the Torus topology can sustain traffic loads at higher

Table 5.8: FPGA synthesis results of the  $6 \times 6$  Mesh topology considering the proposed BRAM-DSP implementation and CONNECT's implementation on Artix 7 (XC7A100T) board with BD=6 and FW=64

	CONNECT	PROPOSED WORK
LUT%	41.99	37.42
FF%	3.61	15.32
LUTRAM%	31.92	-
BRAM%	-	49.55
DSP%	-	30
Frequency(MHz)	146	306

injection rates compared to the Mesh topology.

### 5.5.3 Comparison with the CONNECT (Papamichael and Hoe (2015)) and DART (Wang et al. (2014))

#### 5.5.3.1 Topology implementation

The work proposed is compared with the CONNECT framework (Papamichael and Hoe 2015). The FPGA utilization of the 36-node Mesh topology implemented with the proposed BRAM-DSP NoC architecture and the CONNECT NoC architecture considering the buffer configuration of 6 BD and 64 FW are shown in Table 5.8. A single-stage pipeline is used in CONNECT NoC architecture. Hence, CONNECT NoC consumes lesser FF resources than the proposed BRAM-DSP NoC architecture, which employs the five-stage pipeline. The topology implementation employing the proposed router architecture consumes 10.88% fewer LUT resources than the CONNECT implementation. Also, as the hard blocks are employed in designing the router components, an increase in the frequency of operation is observed. The architecture proposed is  $2.09 \times$  faster than CONNECT.

Table 5.9: Hardware utilization results of the  $3 \times 3$  Mesh topology with proposed BRAM-DSP implementation and DART’s implementation on Artix 7 (XC7A100T) FPGA

	DART	PROPOSED WORK
LUT%	36.37	9.85
FF%	11.12	3.72
BRAM%	-	16.67
DSP%	-	7.5
Frequency(MHz)	171	328

Table 5.9 provides the comparison of 9-node Mesh topology implemented considering the proposed BRAM-DSP NoC architecture and the DART (Wang et al. (2014)) NoC architecture. The 9-node Mesh topology implemented with the proposed BRAM-DSP router architecture consumes 73.38% and 66.55% fewer LUTs and FFs, respectively, than the DART NoC architecture. The proposed architecture is  $1.91 \times$  faster than the DART framework.

### 5.5.3.2 Latency evaluation

Fig. 5.8 shows the comparison of the latency performance of the Mesh topology based on the proposed BRAM-DSP architecture and the CONNECT and DART architectures with various traffic patterns such as Transpose, Bit-complement, Uniform random and Nearest neighbor and . Million flits per second and Nanosecond (ns) are used to represent the load and delay, respectively. Mesh topology implementation based on the proposed BRAM-DSP NoC router architecture has lower latency compared to the Mesh topology based on the DART and CONNECT architectures. Mesh topology based on the proposed BRAM-DSP NoC router architecture can sustain twice the load which can be sustained by the Mesh topology based on DART and CONNECT architectures. The topology based on CONNECT and DART NoC architectures saturate at lower injection rates compared to the topology based on the proposed BRAM-DSP NoC architecture. The proposed NoC architecture’s average packet latency is 24.8% and 19.1% lesser than the CONNECT and DART architectures under Uniform random traffic (Fig. 5.8(a)). Similarly, a reduction of 21.6% and 17.6% in average latency is observed under the Transpose traffic pattern (Fig. 5.8 (b)). Under the NN traffic pattern, latency reduction

## 5. Mapping the NoC router components on the Hard-blocks of the FPGA

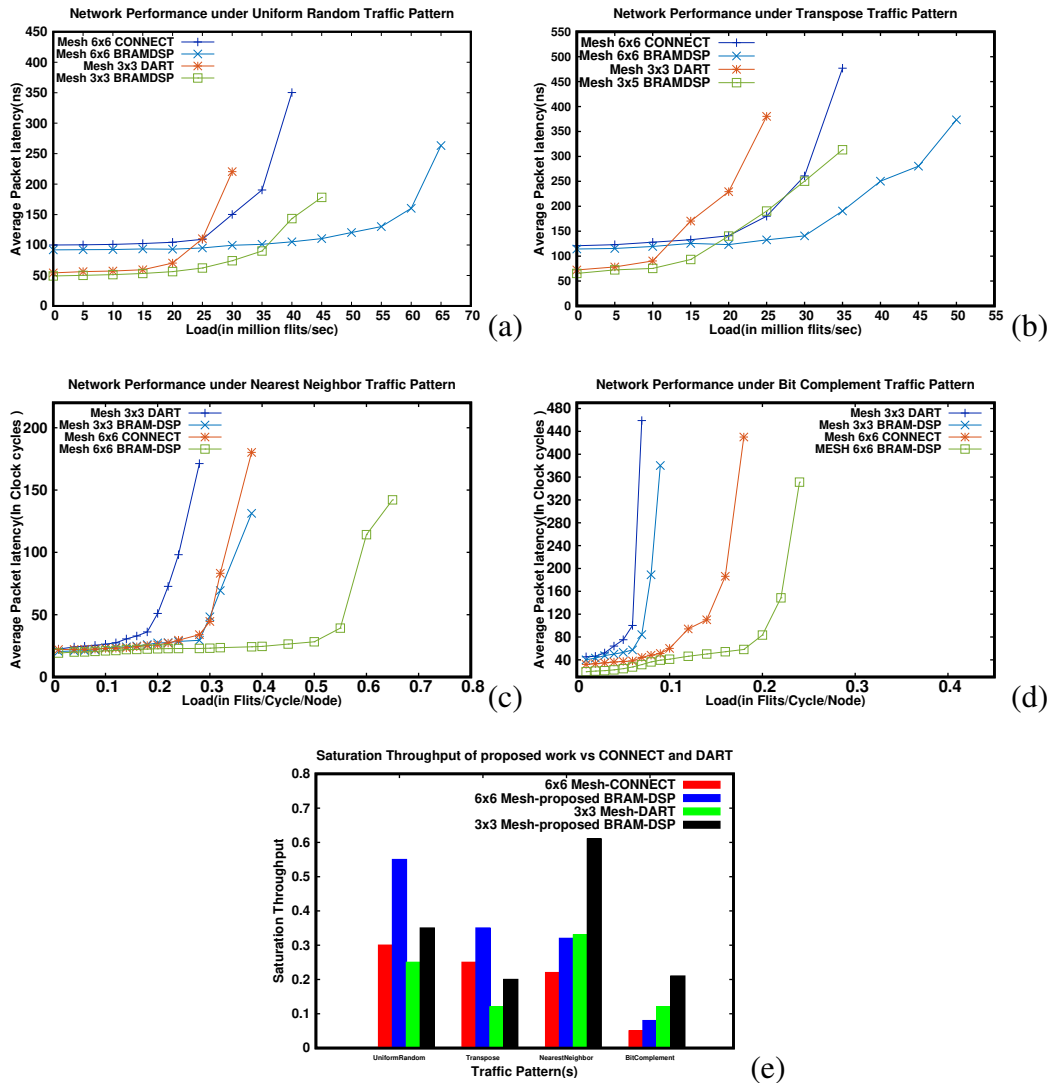


Figure 5.8: (a), (b), (c), (d) - Load vs Latency comparison (e) saturation throughput of the Mesh topologies employing proposed BRAM-DSP router architecture and CONNECT, DART NoC architectures

Table 5.10: Features supported in the proposed BRAM-DSP NoC architecture and the other state-of-the-art NoC architectures

	FPGA components			Router components	
	CLBs	BRAM	DSP	Buffered Router	Bidirectional Ports
Hoplite-DSP	Yes	No	Yes	No	No
CONNECT	Yes	No	No	Yes	Yes
DART	Yes	Yes	No	Yes	Yes
Proposed work	Yes	Yes	Yes	Yes	Yes

of 21.11% and 23.39% is observed with respect to CONNECT and DART (Fig. 5.8 (c)). Considering the BC traffic pattern and the topologies with proposed router architecture, a reduction of 18.37% and 17.21% is observed with respect to CONNECT and DART (Fig. 5.8(d)). Fig. 5.8(e) shows the saturation throughput of the 9-node and 36-node Mesh topology with the proposed NoC architecture and the DART and CONNECT NoC architectures.

The supported features by Hoplite-DSP (Chethan and Kapre (2016)), CONNECT, and DART NoC architectures frameworks and the proposed BRAM-DSP NoC architecture are shown in Table 5.10. The proposed work efficiently utilizes the BRAMs and DSP48E1 blocks of the FPGAs to configure the functionality of the components of an NoC router.

## 5.6 SUMMARY

The unused hard blocks of the FPGA, such as BRAM and DSP48E1 blocks are employed to map the functionality of NoC router’s FIFO and Crossbar components. A reduction of soft logic has been observed employing the proposed technique. A control unit called “buf\_empty\_checker” has been included in the circuit to reduce the latency of the network. The proposed framework performs favorably compared to the other state-of-the-art FPGA based NoC simulation platforms. Further, optimizations to the router architecture can be made to reduce the area and power consumed and providing a better throughput. A router architecture with the optimizations such as single cycle router bypass, parallel VC and SA, combined virtual cut-through and wormhole switching has been proposed in the next Chapter.



## CHAPTER 6

### OPTIMIZATION OF THE NOC ROUTER FOR ACHIEVING LOW LATENCY AND AREA

An FPGA based NoC using a low latency router with a look-ahead bypass (LBNoC) is designed in this Chapter. The proposed design targets the optimized area with improved network performance. The techniques such as single cycle router bypass, adaptive routing module, parallel virtual channel and switch allocation, combined virtual cut-through and wormhole switching are employed in the design of the LBNoC router. The LBNoC router is parameterizable with the network topology, traffic patterns, routing algorithms, buffer depth, buffer width, number of VCs, I/O ports being configurable. A table-based routing algorithm is employed to support the design of custom topologies. The input buffer modules of NoC router are mapped on the FPGA BRAM hard blocks to utilize resources efficiently.

#### 6.1 INTRODUCTION

The NoC router stores the incoming flits in the buffers, and the route for the destination is computed based on the routing algorithms. The wormhole switching mechanism is employed to divide the large packets into smaller chunks of data called flits for efficient buffer utilization. However, in the case of a single physical buffer per port, there can be a chance of head-of-line(HoL) blocking, which degrades the performance of the NoC. To overcome this issue, Virtual Channel (VC) buffers are introduced (Becker 2012) (Dally 1992). The express virtual channels are employed in the applications demanding high

throughput, which leads to complex router microarchitecture (Kumar et al. 2007).

The bufferless router is designed by removing the buffers at the input port. Removing the buffers saves the router area (Moscibroda and Mutlu 2009) (Hayenga et al. 2009). At high traffic loads, the performance of bufferless router degrades, the incoming packets are dropped or deflected because of no buffer in the router design. This, in turn increases the network contention and leads to higher power consumption than a buffered router (Michelogiannakis et al. 2010).

The number of pipeline stages in an NoC router, and the number of hops along the route affect the overall network latency significantly. In this work, we reduce the pipeline stages in the NoC router by employing parallel VC and switch allocation schemes and router bypass techniques. The adaptive routing module is designed to avoid network congestion by dynamically conforming with the adversarial traffic conditions. It achieves high performance under high traffic load.

An FPGA based framework has been developed to demonstrate a prototype of the parameterized low-latency router architecture employing the lookahead bypass technique called LBNoC is proposed in this chapter. Various NoC design space exploration parameters such as buffer depth, number of VCs, flit width, traffic patterns and routing algorithm can be tuned in the framework for regular and user-defined custom topologies.

## 6.2 RELATED WORK

In this section, we introduce the state-of-the-art techniques proposed for low latency router architecture.

An efficient NoC router microarchitecture is proposed in Becker (2012). The non-atomic VC reallocation method and full crossbar architecture are employed in the router micro-architecture. This results in the increased area overhead of the router. In the lookahead routing technique (Galles 1997), the route computation is done in advance in a neighboring router, and this routing information is appended to the header flit. The next router need not compute the route for the head flit and can send the flit for allocation unit depending on the precomputed output port. Switch allocation with speculation (Peh and



Dally 2001) is presented to eliminate the VC and switch allocation dependency. The speculative allocation performs well under the low traffic condition. As we increase the traffic load, there is an increase in unsuccessful speculation, which leads to the inefficient use of the speculation technique. The technique of precomputing arbitration is proposed by Mullins et al. (2004). Employing this technique, the critical path delay is reduced for the separable input-first VC allocator. In the pipeline stages, the switch allocation stage is removed from the critical path. When the traffic is high, removing the switch allocation stage is not efficient as there can be an unused crossbar time slots for the newly arrived flits. The design of FPGA based low latency router micro-architecture is presented in Lu et al. (2011). Two clock cycle router architecture is designed by combining the VC and switch allocation. The atomic VC allocation is employed in this work. This results in higher average packet latency and lowers the saturation throughput in the early stages of traffic injection. Becker (2012) proposes a combined VC and switch allocation technique in which the queues of free VCs are employed to replace the VC allocation for each destination port. As similar to the speculative approach, this technique demands to have a higher priority for the non-header flit requests. There can be a possibility where it may not be able to assign the Output VC (OVC) for a header flit granted by the switch allocation. A prototyping platform called ProNoC for many-core SoCs employing the low latency NoC is proposed in Monemi et al. (2017). ProNoC supports the emulation of Torus and Mesh topologies. ProNoC employs the full crossbar architecture to implement routers. This leads to a higher area overhead and increased latency.

The express VCs (Kumar et al. 2007), dynamic allocation of VC called ViChar (Nicopoulos et al. 2006), and flit reservation flow control (Peh and Dally 2000) are proposed for achieving high throughput. These designs are more complex leading to more area utilization and increased dynamic power. ViChar (Nicopoulos et al. 2006) improves buffer utilization by designing a complex control circuit. The main problems with ViChar are the complexity, setup limitation, and longer pipeline for flit arrival/departure. The router architecture with distributed shared-buffer is proposed in Ramanujam et al. (2010) Ramanujam et al. (2011) and Soteriou et al. (2009). An output-buffered

router(OBR) has been emulated in these works. The proposed router architecture in Ramanujam et al. (2010), Ramanujam et al. (2011) and Soteriou et al. (2009) has a higher zero load latency than a virtual channel router(VCR). This is due to the fact that a packet must travel through input buffer, two crossbars, and shared queues at each router even at lower traffic load. The design of complex router with two crossbars and the timestamp-based flow control consumes 35% and 58% more area and power than a conventional router architecture, respectively. The dynamic buffer management and flow control is proposed in Becker et al. (2012). This implementation leads to an increase in hardware cost, delay, and power. The predefined priority cooperation and centralized priority management based round-robin arbiter are proposed in Yan et al. (2015) and Yan and Sridhar (2018). These designs increase the allocation matching quality. Hence, the requested input port gets the grant signal for packet transmission much more accurately. These designs have area overhead and are difficult to maintain synchronization among each arbiter.

### **6.3 LBNOC-FPGA BASED BYPASS NOC FRAMEWORK**

LBNoC framework contains components on both the software and the hardware partitions of the FPGA. The operations which are processed on the hardware side are controlled by the processing unit of the software side. Also, the software side is responsible for generating traffic and performing the statistics calculation. The hardware side includes the NoC router micro-architecture, programmable logic, memory and interfaces, flow control, and the packet traversal. Fig. 6.1 shows the architecture of the LBNoC framework in which the Xilinx Zynq 7000 ZC702 SoC is used for the implementation. The software side of the framework is implemented on the Zynq 7000's Processing System (PS) containing dual-core ARM Cortex 9 soft processors. The hardware side is implemented on the Artix 7 FPGA chip of Programmable Logic(PL). As seen in Fig. 6.1, a USB-UART driver is employed to establish communication between the host PC and the FPGA. This guarantees the performance of dynamic traffic transmission. The FIFOs are implemented between (i) the USB-UART interface and the DDR3 for transferring the trace files from the host to DDR3 of PS, (ii) the DDR3 and Programmable Logic (PL) bridge for transferring the traces to the emulated NoC routers on the FPGA

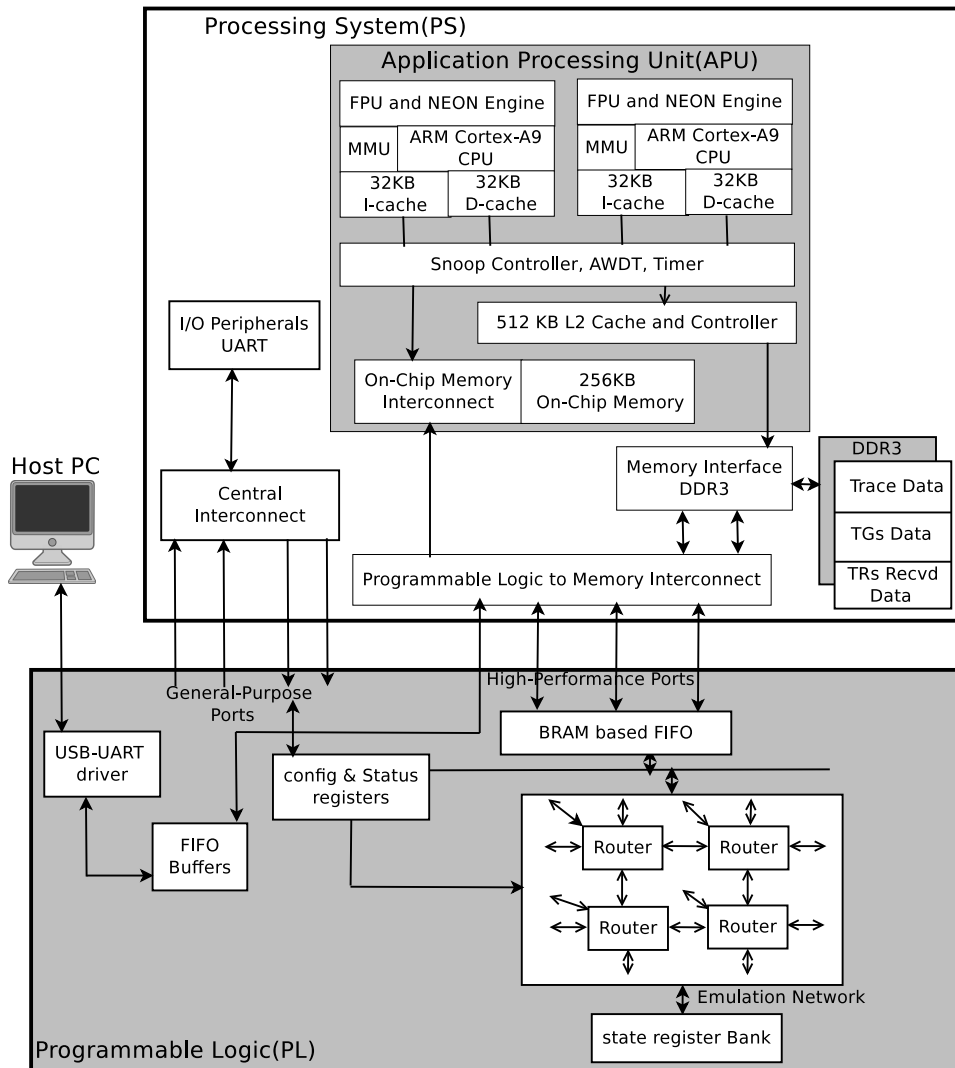


Figure 6.1: The overall architecture of LBNoC-framework implemented on Xilinx Zynq 7000 ZC702 SoC. The PS consists of two core ARM Cortex-A9 processors and the PL has Artix-7 FPGA

and (iii) in between Traffic receptors (TRs) and Traffic generators (TGs) which are modeled on one of the two available ARM Cortex 9 processors. The Memory interface is connected to Processor1 for writing or reading the generated or the received packets.

### 6.3.1 Hardware components

The NoC architecture is implemented on the PL side of the Zynq 7000 SoC. The two-stage pipeline router architecture is designed. The router micro-architectural parameters such as flit width, buffer depth, virtual channel, ports, routing algorithm, link width, and topology are configurable in the LBNoC framework.

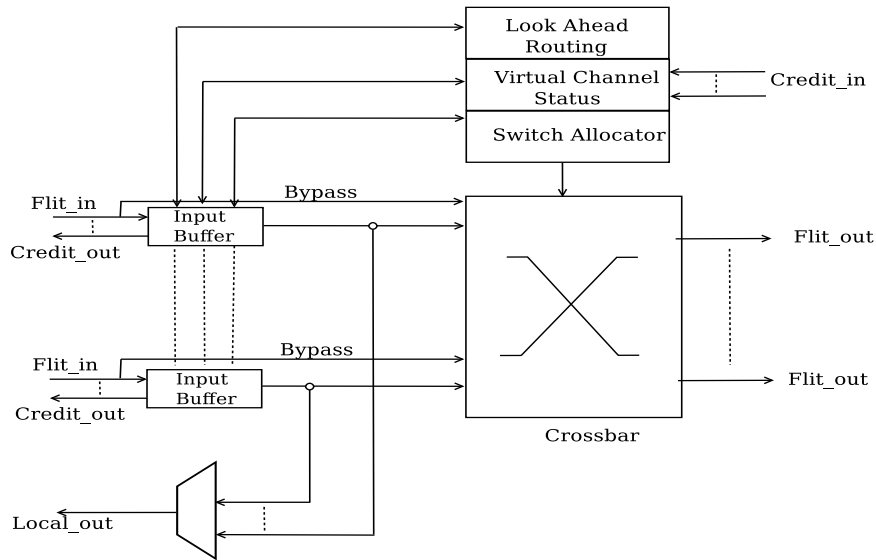


Figure 6.2: Two clock cycle Low latency router architecture implemented in LBNoC framework(The router is highly parameterized with combined VC and Switch allocation stages)

### 6.3.1.1 NoC router

Fig. 6.2 shows the LBNoC router architecture. The two-stage pipelined router architecture employing the lookahead bypass technique comprises an input buffer, route computation unit, combined virtual and switch allocation unit, output module, and the crossbar. The Traffic generator is connected to the injection port of the router, and ejection port is connected to the Traffic receptor in the emulation platform.

### 6.3.1.2 Buffer implementation

In the conventional VC based router architecture, a separate buffer is designated for each VC. Multiplexers and demultiplexers are used to write and read the data from these dedicated buffers for each VCs. The multiplexer/demultiplexer implementation consumes more number of FPGA resources (Monemi (2015)). When the number of VCs are increased in the router input ports, there is a need for large width multiplexers (Monemi et al. (2017)) that leads to significant area utilization. In LBNoC design, large width multiplexers are replaced with a single, dual-port BRAM memory and two multiplexers with narrow a width, as shown in Fig. 6.3. The multiplexers with narrow width select the read and write pointers from the active virtual channel by combining all the input VC buffers at each input port. The VC ID from the incoming header flit

and Write pointer from the controller are combined to form Addr1 for writing incoming flit into the input buffer. For reading the flit from the input buffer, the Addr2 comprises an associated grant signal and read pointer from the controller. The ports of dual-port memory are used for writing the incoming flits and reading the outgoing flits for sending them to the desired crossbar output port of the router.

Fig. 6.3 shows the implementation of input buffers based on the dual-port RAM. The write and read pointers of the dual-port RAM are used to select the address for reading and writing from the memory. This approach removes the multiplexer and demultiplexer in the input buffers of the router and efficiently maps all input VC buffers of an input to a single BRAM. This scheme efficiently utilizes FPGA BRAMs. We have implemented configurable parameters for efficient mapping of the input buffer to FPGA memory based on the flit width and buffer depth. FPGA supports two kinds of memories, viz., soft logic that is LUT based memory (Distributed RAM(DRAM)), and the Hard logic memory(BRAM). When the size of the input buffer is small, LBNoC utilizes the DRAMs to map input buffers. In case of large input buffers, BRAMs are used instead of DRAMs thus improving the performance of LBNoC.

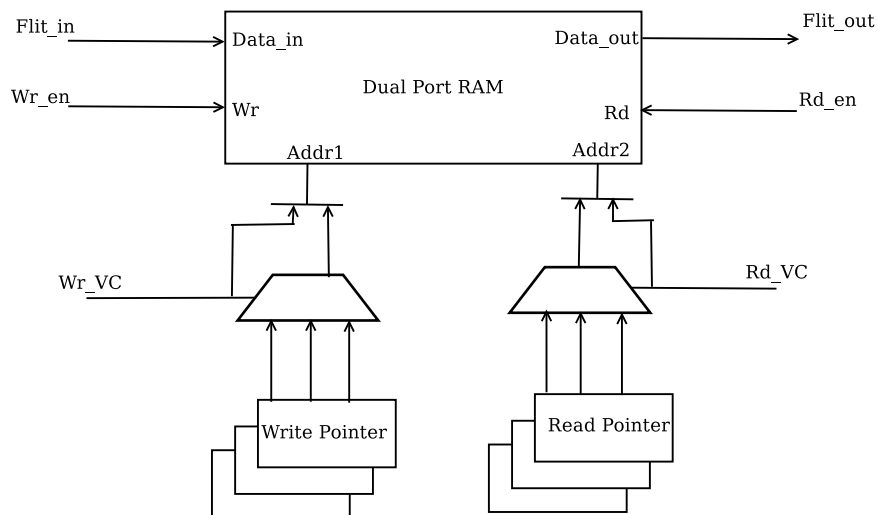


Figure 6.3: The architecture of Input buffer employed in designing low latency router

### 6.3.1.3 Routing algorithms

The incoming flits are sent to the route computation module to determine the output port. The framework supports deterministic, table-based, and minimal adaptive routing

Table 6.1: The conventional allocator. V and P represent number of VCs per port and number of ports

Allocation type	First stage allocator		Second stage allocator	
	Number	size of arbiter	Number	size of arbiter
Virtual channel allocator	(PV)	(V:1)	(PV)	((P-1)V:1)
Switch allocator	(P)	(V:1)	(P)	((P-1):1)

for regular and custom topologies. Table-based routing is implemented for designing the custom topologies. The LUTs holding the output ports to all the destinations from a node under considerations are maintained. The output port entries for the large networks have large number entries in LUTs. DRAMs are used in LBNoC to store the routing tables. A single DRAM is typically of a single-bit wide memory with 16-64 elements constrained to a specific FPGA family. As the entries in the routing tables are maximum of 3 bits wide, they are mapped very efficiently to DRAMs.

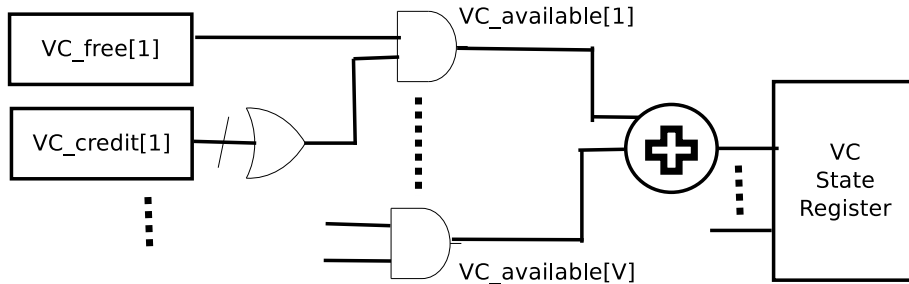


Figure 6.4: Free VC availability check and count

### 6.3.1.4 Parallel VC allocation and Switch allocation

The large circuit complexity and higher critical path delay of traditional allocator results in area overhead and performance reduction (Monemi et al. (2017)). The most challenging task in the NoC router design is the implementation of the allocator module as it lies in the critical path of the NoC router, and also influences the performance and area overhead significantly. From Table 6.1, it can be seen that the VC allocation consumes a large number of resources compared to the Switch allocation. Due to this large consumption, the VC allocation is replaced with queues of free VCs selection for each destination port, as shown in Fig. 6.4. As the queues of free VCs selection require a single arbitration stage, it results in faster execution and less area overhead. The con-

ventional VC allocator stage requires the second arbitration stage to remove conflicts of assigning one Output VC to several input VCs. Whereas, in the queues of free VCs selection technique, the switch allocator removes conflicts of assigning one Output VC to several input VCs.

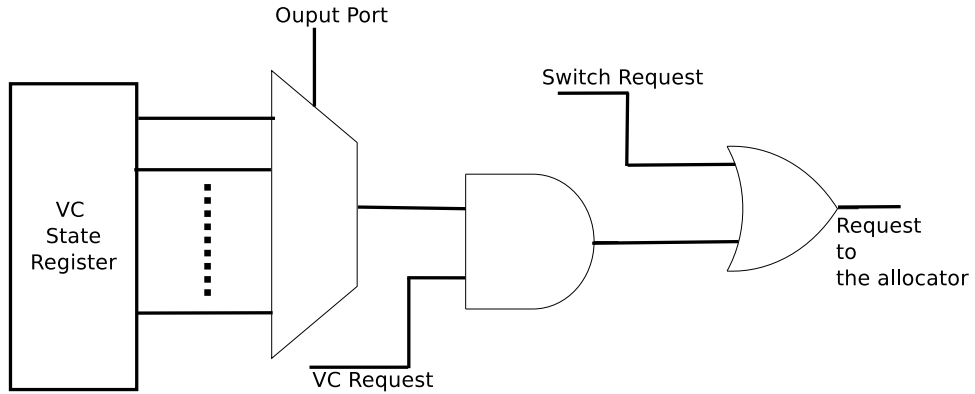


Figure 6.5: Request filter logic

To implement the two-stage router pipeline architecture, the VC and switch allocation stages of the conventional router are combined to form a single VSA stage. In the VSA stage, the VC allocation and switch allocation are performed in a single clock cycle (Lu et al. (2011)). The VC allocation fails when all the virtual channels of the given output channel are busy for head flit, and there is a lack of free space in VCs for body and tail flits. A novel filtering method is implemented to monitor all switch allocation requests that are unable to transfer flits through the output channel. The newly proposed request filter logic is shown in Fig. 6.5. The request for output channel is filtered, when there are free VCs and free space in the VCs. These requests are sent to the switch allocation logic.

The non-speculative parallel virtual channel and switch allocation approach is implemented as shown in Fig. 6.6. For each input port, only one request is granted from all the other requests that are sent to the first stage arbiter of size  $V:1$ . After encoding the winner, the arbiter selects the output port among the rest of the requests (Virtual channel or switch). The header flit requests both VC and SW allocator. The non-header flit requests only switch allocator, as the header flit has already done the VC allocation. The VC allocation done for the header flit is reserved for the entire duration of the packet.

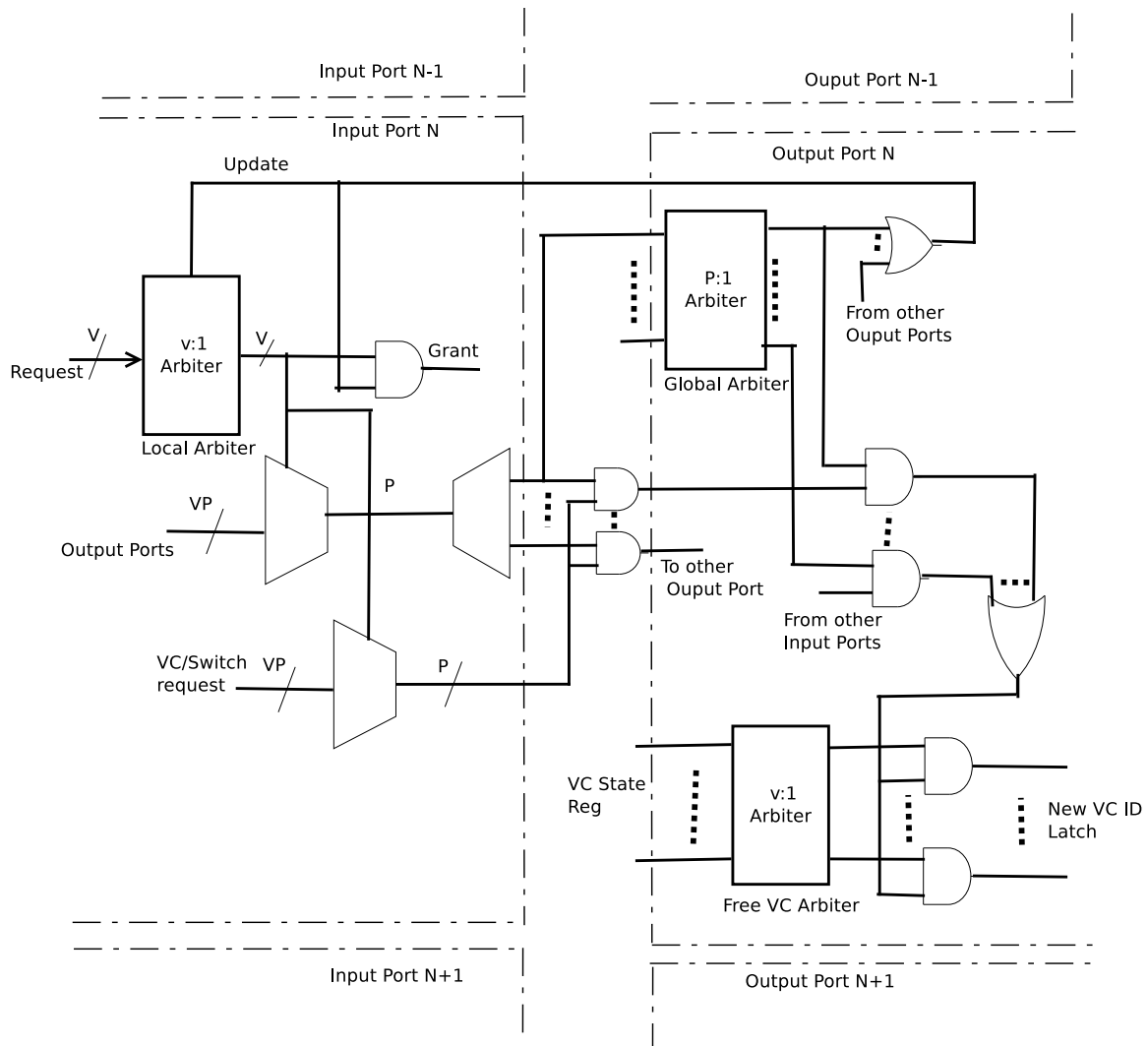


Figure 6.6: Parallel virtual channel and switch allocator

However, the switch requests are allocated flit by flit basis. Each time the flit has to request for the switch allocator. To differentiate header and non-header flit request signals, the VC and switch requests are designated with “1” and “0” bits, respectively. The selected output port is sent to the second stage of the arbiter. The second stage arbiter performs arbitration among requests of different input ports that request the same output port. The second stage arbitration results are sent back to the respective inputs. The request signal after winning both the first and second stage arbitration is only granted the access. If the request is of switch type designated with “0”, then the allocation process is completed. If the request is of VC type designated with “1”, then it continues for the VC allocation, that is free VC selection. The arbitration of free VCs is done using



Table 6.2: The proposed parallel allocator. V and P denotes number of VCs per port and number of ports

Parallel allocator		
Type of allocator	Virtual channel allocator+Switch allocator	
Size of arbiter	(V:1)	((P-1):1)
Number	(2P)	(P)

the V:1 arbiter at the output channel. An encoding of the winner of V:1 arbiter can be done and latched as a new Virtual Channel Identifier (VCID). The input request for output channel, which does not have free space in VC is eliminated by request monitoring logic. The resultant is at most one grant signal, which can be made available at the output channel. This always results in successful VC allocation. The free VC selection is not performed on the critical path; therefore, it is performed parallelly with switch allocation. Arbitration is needed in the scenario of two levels of allocators. The design of parallel VC allocation and switch allocation results in the reduced circuit complexity and critical path delay in NoC router architecture, as shown in Table 6.2. LBNoC supports round-robin, weighted round-robin, and fixed priority arbitration schemes.

### 6.3.1.5 Crossbar

The decomposed crossbar architecture is implemented in the LBNoC router architecture. Based on the route computed employing lookahead routing, the flits destined to the local output are sent through the multiplexer instead of going through the switch allocation and switch traversal. This results in the reduction of 2 clock cycles at the destination nodes. Employing decomposed crossbar results in reduced area utilization. As there is less contention as the connections are less, the probability of contention at the output port is reduced.

### 6.3.1.6 Design of hybrid flow control

A hybrid flow allocation mechanism allows the improvement of network throughput. To keep the flits of the same packet together in the network, a communication flow is established employing the hybrid flow control. As the packet stays across a lesser number of routers, less number of network resources such as VCs are required to hold

the packet at any given time. The hybrid flow control mechanism is implemented by combining the virtual-cut through and wormhole flow switching based on priority rule. The output port is allocated on flit by flit basis of different packets, but the priority being given to the same packet flits requesting contiguous in switch allocation mechanism. That is, both the first(local) and second(global) stage arbitration in switch allocation favor flits of the same packet. The degradation of throughput can be avoided in the network employing the hybrid flow only if the same packet flits contiguous requests exist. The starvation does not happen in the hybrid flow due to the breakage of the hybrid flow of the same packet flits. Once the switch allocation request to the tail flit of the packet is granted, the resources are free to be allocated to flits of other packets. Thus, across the network, the flows are created by keeping the same packet flits together, and less number of resources can be occupied at any given time.

#### **6.3.1.7 The Pipeline bypass**

The output ports of the router which have little contention are often free at low network loads. In such circumstances, the pipeline stages can be bypassed, and flit traversal delay can be reduced to a single clock cycle. The switch traversal path of the crossbar is set up earlier by configuring control signals based on an advance setup signal which arrived one cycle ahead than the actual flit. The following three conditions are to be fulfilled for pipeline bypassing: First, the buffer at the input port is empty, when the advance setup signal arrives. Second, there is no conflict for the output port with existing flits. The advance setup signal requests for a conflict-free output port. Third, multiple advance setup signal does not have output port conflicts. The flit follows normal pipeline stages of Fig. 6.7 (b) when the above conditions are not satisfied. If the above conditions are satisfied, the pipeline delay of the flit is just a single clock cycle.

#### **6.3.1.8 Pipeline architecture**

Fig.6.7 shows the conventional 5-stage pipeline, 2-stage pipeline, and 2-stage with bypass pipeline of the NoC router architecture. The conventional 5-stage pipeline router comprises of buffer write(BW), route computation(RC), virtual allocation(VA), switch allocation(SA), and switch traversal(ST) stages. In the 2-stage router pipeline architec-

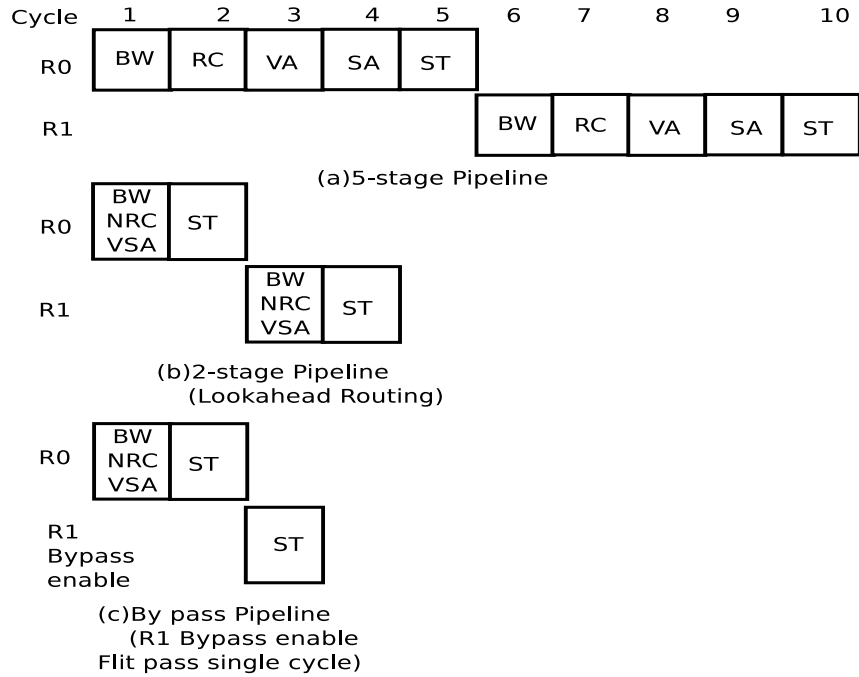


Figure 6.7: Pipeline stages of conventional and LBNoC router architecture

ture, the pipeline structure of the router is optimized by employing lookahead routing and combining the non-speculative Virtual channel and switch allocation stages. Also, the buffer write, Lookahead Route Compute (LRC) and combined VSA allocation are done in one clock cycle, and switch traversal(ST) is performed in the next clock cycle. At low traffic loads, the flit passes through a single cycle in a 2-stage router with bypass architecture. In this pipeline architecture, flits perform only Switch Traversal(ST) stage by eliminating all other stages of router pipeline architecture by employing the bypass technique.

### 6.3.1.9 Adaptive routing algorithm

Fig. 6.8 shows the block diagram of the proposed adaptive route computation module. The adaptive route computation module consists next router address predictor, two-hop neighbor status information, and adaptive routing algorithm for route computation. A small area overhead can be observed due to the additional status information in transmission links and register for storing the 2-bit status information. This area overhead is negligible in FPGA due to the abundant availability of wires and FFs (Papamichael and Hoe (2015)). The adaptive routing employs 2-bit values for indicating the congestion

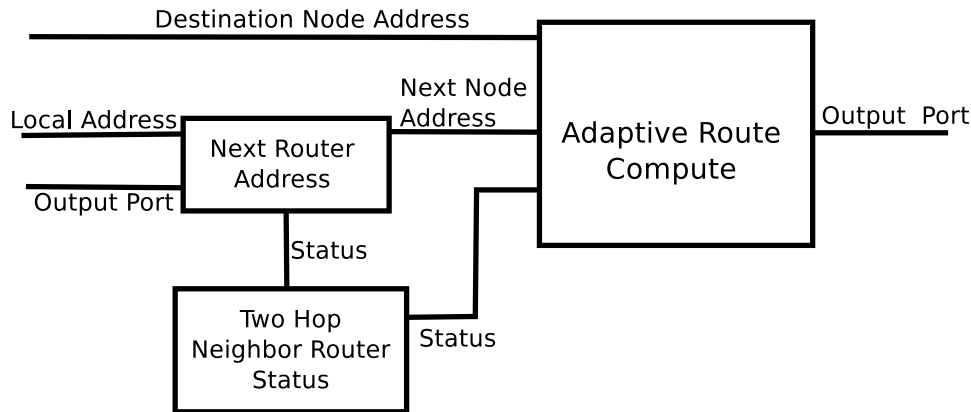


Figure 6.8: Proposed adaptive look-ahead routing module

in the communication routers. 00, 01, 10, and 11 are used to indicate the empty(0%), half full(50%), nearly full(75%), and full (100%) occupancy of the buffers of a router respectively. Each router has to exchange these congestion status bits with its two-hop neighboring routers to make the correct routing decisions. The adaptive routing algorithm from Parane et al. (2018) with negligible area overhead of 1.66% compared to XY routing algorithm is used in the proposed LBNoC architecture.

### 6.3.2 Software components in LBNoC

The Traffic generation, Source queue, and Traffic receptors are implemented on the software side. These components are implemented on the embedded ARM Cortex 9 soft processors.

#### 6.3.2.1 Traffic generation

The Traffic Generation (TG) modules are used to produce various synthetic traffic patterns. Each router of the NoC topology is associated with TG module. The TG modules are responsible for generating the packets and storing them in the source queue. Also, the flit generation logic is incorporated in the TG module. The TG module is responsible for generating traffic. The framework supports various types of synthetic traffic patterns such as uniform random, bit complement, transpose, bit reverse, hotspot, and tornado.

### 6.3.2.2 Source queue

The source queues are implemented to store the flits generated by the TG module prior to injecting the packet in the network. The source queues operate in FIFO fashion. The source queue takes care of the injection of the flits into the NoC topology based on the emulation cycles.

### 6.3.2.3 Traffic receptor

The Traffic Receptor (TR) associated with each NoC router is responsible for validating the destination. TR also decodes the information in the flit. The TR module calculates the packet latency based on the time stamp stored in the head flit. Also, it monitors the number of total packets received, the number of packets transmitted, and average packet latency.

### 6.3.2.4 Global clock generation

The Global Clock Generation (GCG) module is responsible for maintaining the synchronization between the software (ARM Cortex) and the FPGA side. GCG generates a clock on the software side. The FPGA side is driven by the clock generated in the software side.

## 6.3.3 Flit structure

Fig. 6.9 shows the 32-bit flit structure. The flit size can be configured to any size varying from 32, 64, and 128 bits. A 2-bit 'Type' field is used to represent the type of flit that is being traversed in the network such as Head, Body and Tail. The 'VC\_Id' field depends on the number of configured VCs. The 'LHR' field specifies the output port of the next-hop router computed using Look ahead routing. The 'Class' field specifies the message class. Source and Destination fields are specified in 'Source' and 'Dest' fields. The 'Timestamp' field is used to specify flit creation time at the Source. The Body flit contains the 'Payload' field to accommodate the actual data to be transferred. The 'Wr\_TimeStamp' field of Tail flit marks the arrival time of a flit at its destination. The latency is calculated by considering the 'TimeStamp' and 'Wr\_TimeStamp' fields.

## 6. Optimization of the NoC router for achieving low latency and area

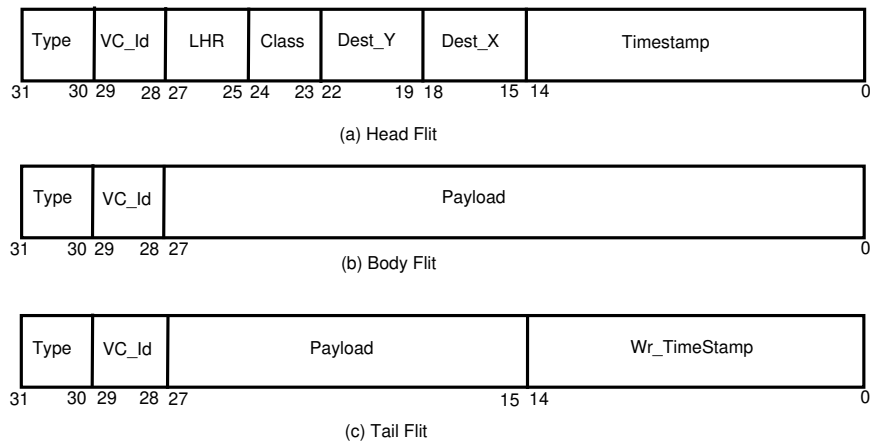


Figure 6.9: Flit structure employed in LBNoC Framework

### 6.4 RESULTS AND DISCUSSION

The microarchitectural components of the NoC architecture are implemented in Verilog. Synthesis results are extracted from Xilinx Vivado suite. Results include resource usage on the Xilinx Zynq 7000 SoC (ZC 702 board). The hardware side is implemented on the Artix 7 FPGA, and the software side is implemented on the ARM Cortex 9 soft processors. The software side accounts for traffic generators and traffic receptors. The UART interface is employed for transceiving the traffic from the software to the hardware side of the framework. Another USB-UART interface is employed for communication between the host PC and the PL side of the Zynq SoC.

Table 6.3: Experimental setup details

Experimental setup details	
Topology	$4 \times 4$ & $5 \times 5$ Mesh topology
Buffer type	FIFO buffer
Packet length	4 flits
Flit width	32, 64 and 128
Buffer depth	4, 8 and 16
Routing algorithm	DoR Lookahead
Router Pipeline depth	2-stage and 2-stage with Bypass
Flow control	Credit based
Arbiter type	Round robin, Priority
Traffic pattern	Uniform random, Bit complement, Transpose, Bit reverse, Tornado, Hotspot

The experimental setup employed in evaluating LBNoC is shown in Table 6.3. Various configurations of buffer depth, number of VCs, flit width, the number of I/O ports, traffic patterns, and different topologies have been employed to evaluate the LBNoC

framework.

#### 6.4.1 FPGA resource utilization

In the FPGA platform, memory can be mapped on the soft and hard memory blocks. The soft memory block such as Register RAM, Distributed RAM (also called LUT based RAM) and the hard memory block such as BRAM. Table 6.4 shows the buffer implementation mapping on three alternative memory modules of the FPGA platform with fixed 32-bit of Flit width and varying buffer depth from 5 to 55 flits. From Table 6.4, it can be observed that mapping memory logic on the Register RAM consumes more FPGA resources compared to the LUTRAM and BRAM resources. This increases the circuit complexity and decreases the operating frequency. Comparing the three alternative memory modules, BRAM based implementation is best suitable to map the entire memory logic into the single BRAM block. This will reduce the circuit complexity and increases the operating frequency. In Table 6.5, we fixed the buffer depth to 15 flits, and flit width is varied from 4 to 256-bit. It can be observed that the FPGA memory resource consumption increases with the increase in the number of bits. The results show that BRAM is best suitable for explicit memory mapping, which consumes hardly four BRAM blocks with a flit size of 256 bit compared to Register RAM and LUTRAM.

Table 6.4: FPGA memory buffers using three implementation alternatives with constant flit width of 32-bit.

Flit Width		32-bit										
Buffer depth		5	10	15	20	25	30	35	40	45	50	55
FIFO buffer	RegisterRAM	160	320	512	640	800	960	1120	1280	1440	1600	1760
	LUTRAM	24	24	24	24	24	24	44	44	44	44	44
	BRAM	1	1	1	1	1	1	1	1	1	1	1

Table 6.5: FPGA memory buffers using three implementation alternatives with constant buffer depth of 15 flits.

Buffer depth		15										
Flit width		4	8	16	32	64	128	150	180	200	210	256
FIFO buffer	RegisterRAM	64	128	256	512	1024	2048	2400	2880	3200	3360	4095
	LUTRAM	8	8	16	24	48	88	100	120	136	140	176
	BRAM	1	1	1	1	1	2	3	3	3	3	4

Tables 6.6 and 6.7 show the FPGA BRAM resource utilization for various config-

## 6. Optimization of the NoC router for achieving low latency and area

urations of NoC router input buffers targeting Zynq 7000 SoC. The results in Tables 6.6 and 6.7 are specific to the input buffer in a single LBNoC router. It can be seen that the LBNoC framework is capable of design space exploration of the NoC architecture by allowing the parametrized values for Input buffer configurations. Tables 6.6 and 6.7 infer that increasing the number of VCs, flit width, and buffer depth yields in higher utilization of FPGA resources. Changes made to the buffer depth and flit width affect the buffering requirement. This plays a major role in the performance of NoC architecture.

Table 6.6: Synthesis results of various configurations of Input buffer in LBNoC router with 64-bit flit width

Flit width		64 bits								
Number of VCs		4			8			16		
Buffer depth		4	8	16	4	8	16	4	8	16
Input buffer	LUT	46	58	76	94	124	158	190	254	323
	FFs	28	41	52	56	80	104	112	160	208
	BRAM36	1	1	1	1	1	1	1	1	1

Table 6.7: Synthesis results of various configurations of Input buffer in LBNoC router with 128-bit of flit width

Flit width		128 bits								
Number of VCs		4			8			16		
Buffer depth		4	8	16	4	8	16	4	8	16
Input buffer	LUT	46	58	76	94	124	158	192	254	323
	FFs	28	41	52	56	80	104	112	160	208
	BRAM36	2	2	2	2	2	2	2	2	2

In designing the low latency router architecture in LBNoC, we employ the merging of all input VCs buffer at input ports. Table 6.8 shows the resource utilization of merged buffer implementation and CONNECT’s (Papamichael and Hoe (2015)) conventional buffer implementation. It can be seen from Table 6.8 that the merged implementation consumes 28 % and 44 % fewer LUTs for 32 and 64-bit flit width. This implementation will be mapped efficiently to a single Block RAM of FPGA. An increase in the number of LUTs increases the critical path, thus reducing the operating frequency.

The queues of free selectors are considered for implementing the VC allocator in low latency router architecture. Table 6.9 shows the resource utilization of queues of



free VCs selection module and the two-level VC allocator module. It can be seen that the queues of free VCs selection consume 58.05 % fewer hardware compared to the conventional VC allocator (Becker (2012)).

Table 6.8: Synthesis results of merged FIFO buffers at each input port and Conventional FIFO buffers

Flit Width		32 bits	64 bits
Num. of VCs		4	4
Buf. Depth		8	8
Merged Input buffer	LUT	56	58
	FF	40	40
	DRAM	-	-
	RAMB18	1	-
	RAMB36	-	1
CONNECT buffer	LUT	78	104
	FF	42	42
	DRAM	24	48
	RAMB18	-	-
	RAMB36	-	-

The decomposed crossbar architecture is implemented to directly route the packet, which is destined for the local input port based on the look-ahead routing. Table 6.10 shows the resource utilization of the Full crossbar (Monemi et al. (2017)) and the Decomposed crossbar. The Decomposed implementation consumes 10.64 % fewer LUTs than the Full crossbar.

Table 6.9: Synthesis results of Queue of free VCs selection and Conventional VC allocator implementation

VC		4
In/Out Port		5
Queue of free VCs	LUT	370
	FF	155
VC allocator (Becker 2012)	LUT	882
	FF	201

#### 6.4.1.1 Topology implementation

The  $4 \times 4$  and  $5 \times 5$  prototypes have been implemented employing LBNoC to demonstrate resource utilization. Table 6.11 shows the synthesis results for the designed

Table 6.10: Synthesis results of Full and Decomposed Crossbar with IN/OUT ports

	Full Crossbar (Monemi et al. 2017) 6-IN and 6-OUT ports	Decomposed Crossbar 6-IN and 5-OUT ports
LUT	141	126

Table 6.11: Synthesis results of Mesh topology of size  $4 \times 4$  and  $5 \times 5$  with various configuration of input parameters

Topology	4x4				5x5				4x4				5x5			
VC	2				2				4				4			
Buffer depth	2	4	2	4	2	4	2	4	2	4	2	4	2	4	2	4
Flit Width	32	64	32	64	32	64	32	64	32	64	32	64	32	64	32	64
LUT(%)	21	31	26	35	36	52	43	57	41	49	50	59	67	81	78	90
FF(%)	5	7	7	7	6	12	11	14	9	11	12	14	15	19	19	23
BRAM(%)	22	45	22	45	37	75	37	75	22	45	22	45	37	75	37	75
Power(mW)	374	529	422	607	550	821	634	942	633	837	704	917	964	1314	1078	1481

topologies. Increasing the flit width increases the FPGA resource utilization. Considering the  $4 \times 4$  mesh topology, when the flit width is increased from 32 to 64 bits, the LUT utilization increases from 26% to 35% for buffer depth 4 considering 2 VCs. Similar behavior is observed for the  $5 \times 5$  mesh topology. Flit width affects the buffer requirement in the NoC router architecture. It can be seen that increasing the flit width from 32 to 64-bit leads to higher utilization of BRAMs. When the 32-bit flits are utilized, the 18Kb BRAMs of Xilinx FPGA are used. When the 64-bit flits are used, 36Kb BRAMs are utilized. This is because the Xilinx 18Kb BRAM can be configured to accommodate 512 flits each of 32-bit width. As we increase the flit width to 64-bit, the 18Kb BRAM are not be sufficient to map the required width. Hence, the 64-bit flits occupy the 36Kb BRAMs by configuring 512 of flits each of 64-bits. As the topology size increases, the FPGA resources also increase. The  $5 \times 5$  mesh topology with 4 VCs and 64 bit flit width consumes 65.30 % and 40 % more LUTs and 36Kb BRAMs, respectively than the  $4 \times 4$  mesh topology.

#### 6.4.2 Latency analysis

The packet latency analysis results under various synthetic traffic patterns are presented in this section. We use six traffic patterns: Uniform Random, Hot-spot, Transpose, Bit complement, Bit Reverse, and Tornado. Table 6.3 shows the basic experimental setup

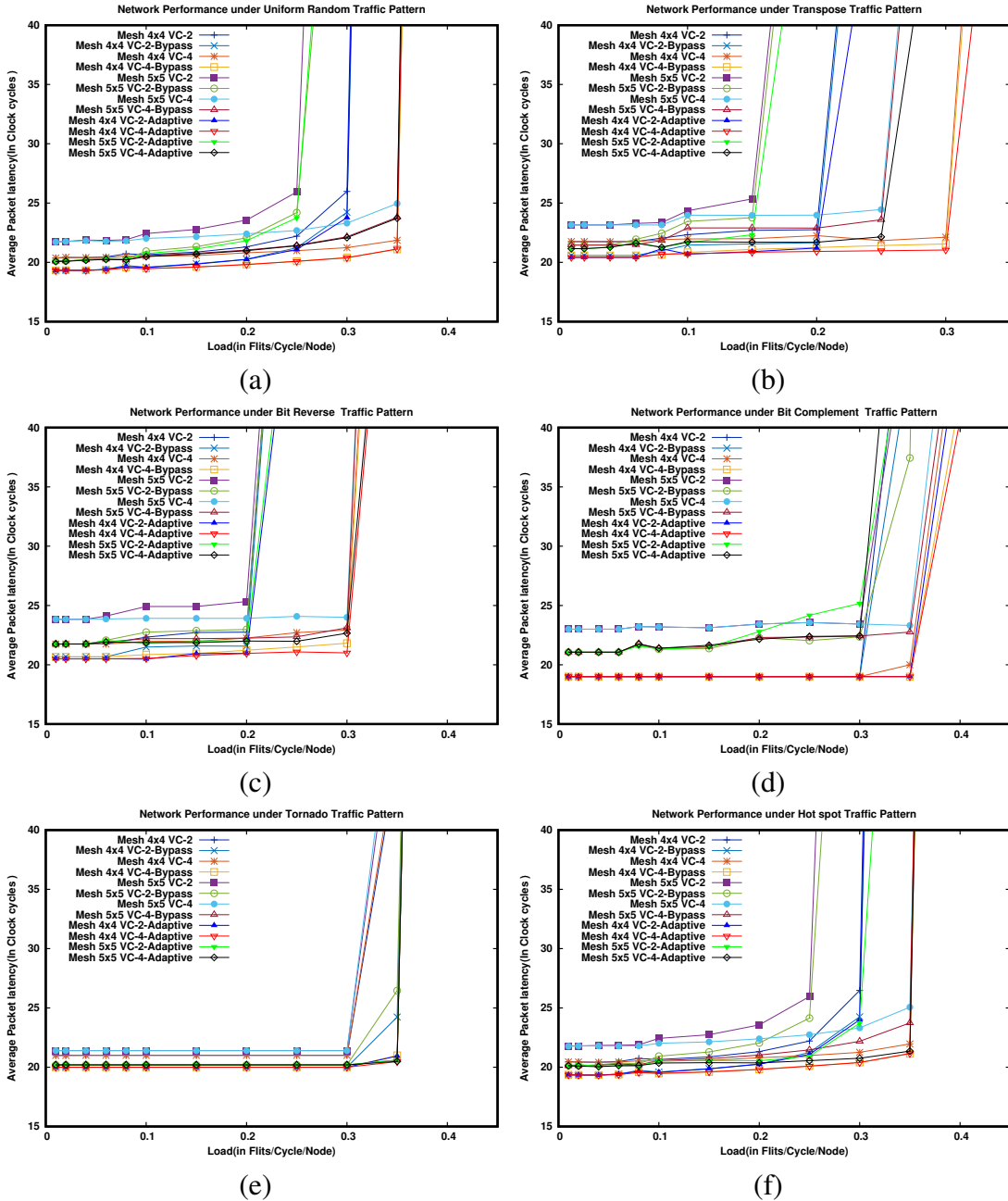


Figure 6.10: Performance comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns.

of latency analysis. From Fig. 6.10, it can be seen that the single cycle router with bypass path and adaptive routing performs better compared to the baseline Two clock cycle router architecture for all the traffic patterns. From Fig. 6.10 (a), the  $4 \times 4$  mesh topology has a lower average packet latency compared to  $5 \times 5$  mesh topology. This is due to the lower diameter of the  $4 \times 4$  mesh. The single-cycle Bypass path reduces the average communication latency by 5.4%, 5.6%, 5.2%, 9.2%, 4.99% and 5.6% for Uniform random, Transpose, Bit reverse, Bit complement, Tornado and Hot-spot traffic patterns respectively for  $4 \times 4$  mesh topology. The  $5 \times 5$  mesh topology with a single cycle Bypass path reduces the average packet latency by 6.78%, 7.93%, 9.5%, 10.3%, 6.45% and 7.93% for Uniform random, Transpose, Bit reverse, Bit complement, Tornado and Hot-spot traffic patterns respectively. Employing both the single-cycle Bypass path and the adaptive routing strategy, reduction in the average communication latency by 6.31%, 6.53%, 6.12%, 10.42%, 6.01% and 6.7% for Uniform random, Transpose, Bit reverse, Bit complement, Tornado and Hot-spot traffic patterns respectively have been observed for  $4 \times 4$  mesh topology. For the  $5 \times 5$  mesh topology with single-cycle Bypass path and the adaptive routing strategy, reduction in the average packet latency by 7.84%, 8.85%, 10.7%, 11.63%, 7.5% and 8.79% for Uniform random, Transpose, Bit reverse, Bit complement, Tornado, and Hot-spot respectively have been observed.  $4 \times 4$  mesh topology employing the bypass path and adaptive routing strategy with 4 VCs performs better compared to all the other configurations under Uniform Random, Transpose, Bit Reverse, and Hot-spot traffic patterns. From Fig. 6.10(d), it can be observed that  $4 \times 4$  with 2 VCs considering the bypass and baseline two clock cycle architecture saturates early compared to all other configurations. Fig. 6.10(e) shows that  $5 \times 5$  mesh with 2 VCs, Bypass path, and the adaptive routing strategy has lower average packet latency compared to the baseline two clock cycle, but it saturates early. The  $4 \times 4$  mesh topology with 4 VCs, and the Bypass path yields better performance compared to all other configurations, as shown in Fig. 6.10(e).

### 6.4.3 Throughput analysis

Fig. 6.11 shows an increase in the saturation throughput as virtual channel increases from 2 to 4. This is due to the large size buffers to hold more number of flits in the

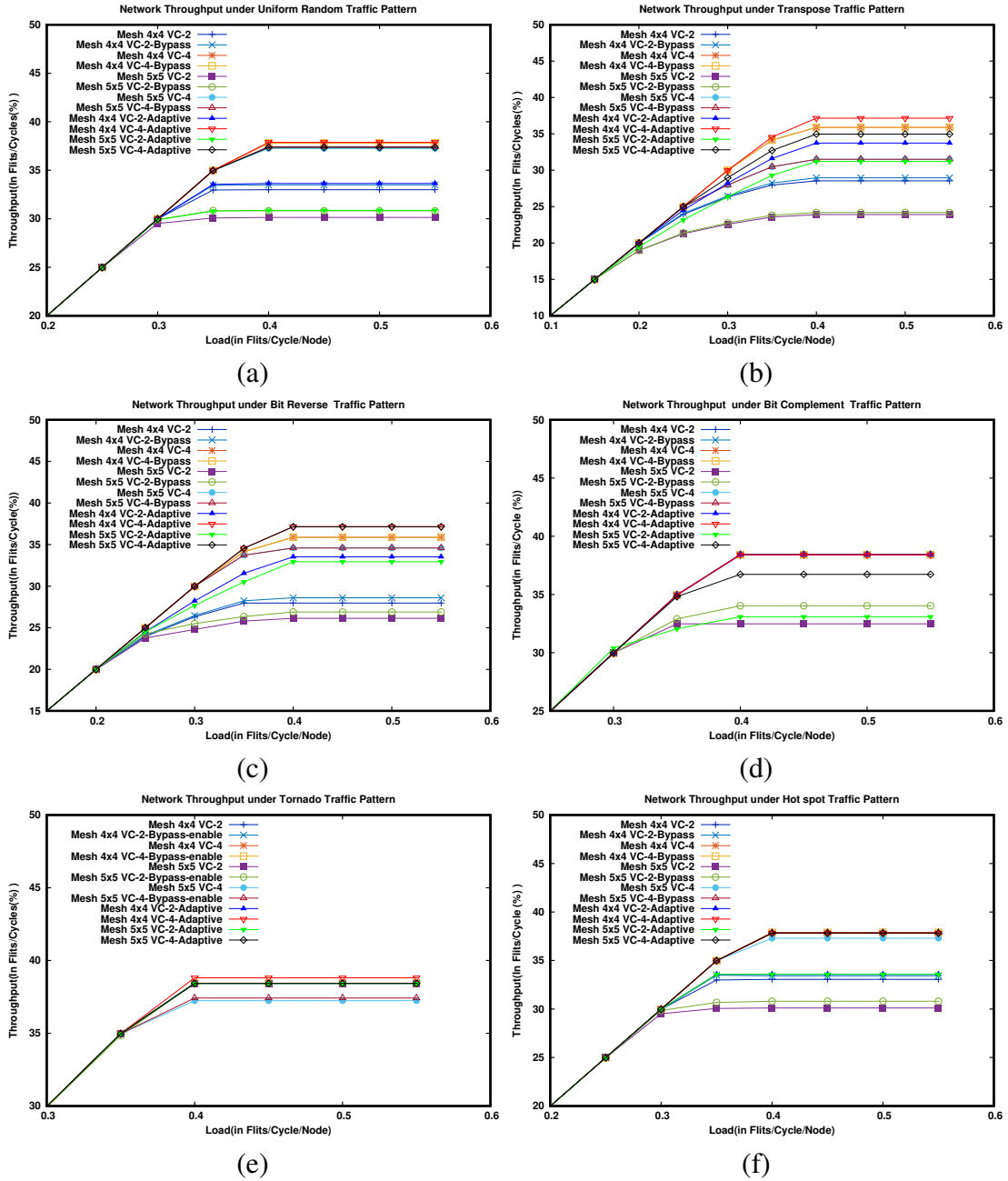


Figure 6.11: Throughput comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns.

network. The  $4 \times 4$  mesh topology with 4 VC employing single cycle Bypass path and adaptive has a higher saturation throughput compared to all the other configurations in Fig. 6.11(a), (b),(c) and (f). From Fig. 6.11(d) it can be seen that  $4 \times 4$  mesh topology with 2 VC baseline architecture has lower saturation throughput compared to all other configurations. The  $5 \times 5$  mesh topology with 2 VC single cycle Bypass path has lower saturation throughput compared to all other configurations, as shown in Fig. 6.11(e).

#### 6.4.4 Power analysis

Table 6.11 shows power consumption for various configurations of  $4 \times 4$  and  $5 \times 5$  mesh topologies. The dynamic power is estimated using Xilinx Xpower by supplying switching activity rates. We extract these switching activity rates from simulation data. It can be observed that power consumption increases as and when there is an increase in the size of the virtual channel, flit width, buffer depth, and size of the topology. For example, from 374mW for a low configuration ( $4 \times 4$  mesh topology with VCs of 2, buffer depth of 2 and Flit width of 32) to 1481mW for the highest configuration ( $5 \times 5$  mesh topology with VCs of 4, buffer depth of 4 and Flit width of 64) as shown in Table 6.11. This is due to the large resource requirements and utilization required for larger topology size. The larger topology is capable of processing more flits than a smaller topology within the same time frame.

Table 6.12: Resource utilization and Maximum operating frequency of Different NoC configurations considering  $4 \times 4$  mesh topology

	Resource Utilization (%)	Max. Operating Frequency(MHz)	Power (mW)
CONNECT	69	98	810
ProNoC	53	170	734
LBNOC	50	205	704

## 6.5 COMPARISON WITH THE STATE-OF-THE-ART NOC ARCHITECTURES

### 6.5.1 Comparison with FPGA state-of-the-art

#### 6.5.1.1 Area, frequency and power:

Table 6.12 shows the resource utilization, and maximum operating frequency of different NoC architectures.  $4 \times 4$  mesh topology implemented employing the LBNOC NoC

architecture consumes 4.5% and 27.1% fewer hardware resources than the ProNoC and CONNECT architectures with identical NoC configuration parameters. This is because of the design optimizations such as merged input buffers, decomposed crossbar architecture, and employing queues of free VCs. This, in turn, results in a lower critical path delay. Hence, LBNoC NoC architecture operates at a higher frequency of 205MHz compared to CONNECT(100MHz) and ProNoC(172MHz) architectures. We observe a 4.1% and 13.1% reduction in power consumption than ProNoC and CONNECT NoC architecture, respectively. The lower power due to its fewer FPGA resource utilization.

### 6.5.1.2 Latency and throughput analysis:

To compare the performance of different NoC architectures, the experiments considering the same configuration parameters such as 32-bit flit width, 4 VCs per port, buffer depth of 4 flits are conducted. Fig. 6.12 shows the load vs. delay graph for various synthetic traffic patterns in  $5 \times 5$  mesh topology. The average latency increases with an increase in the packet injection rate.

Fig. 6.12(a) shows the load vs delay graph of  $5 \times 5$  mesh topology under Uniform traffic pattern. The average packet latency of LBNoC architecture is 25% and 13% less than the CONNECT and ProNoC architectures. From Fig. 6.12(b), it can be seen that LBNoC architecture has 30% and 15% lesser average packet latency than CONNECT and ProNoC architectures under the Transpose traffic pattern. Fig. 6.12(c),(d),(e) and (f) shows the load vs. delay graph of  $5 \times 5$  mesh topology under Tornado, Hot-spot, Bit complement and Bit Reverse traffic patterns. LBNoC has less average packet latency compared to CONNECT and ProNoC. It can be seen that the LBNoC architecture has 36%, 16.7%, 35.3%, 30% lesser average packet latency than the CONNECT architecture under Tornado, Hot spot, Bit complement, and Bit-reversal traffic patterns respectively. Compared to ProNoC, LBNoC has 11.2%, 10.4%, 4.56%, and 6.3% lesser average packet latency under Tornado, Hot spot, Bit complement and Bit reversal traffic patterns respectively.

Fig. 6.13 shows the saturation throughput of the CONNECT, ProNoC, and LBNoC architectures under Uniform, Transpose, Tornado, Hot-spot, Bit complement, and Bit

6. Optimization of the NoC router for achieving low latency and area

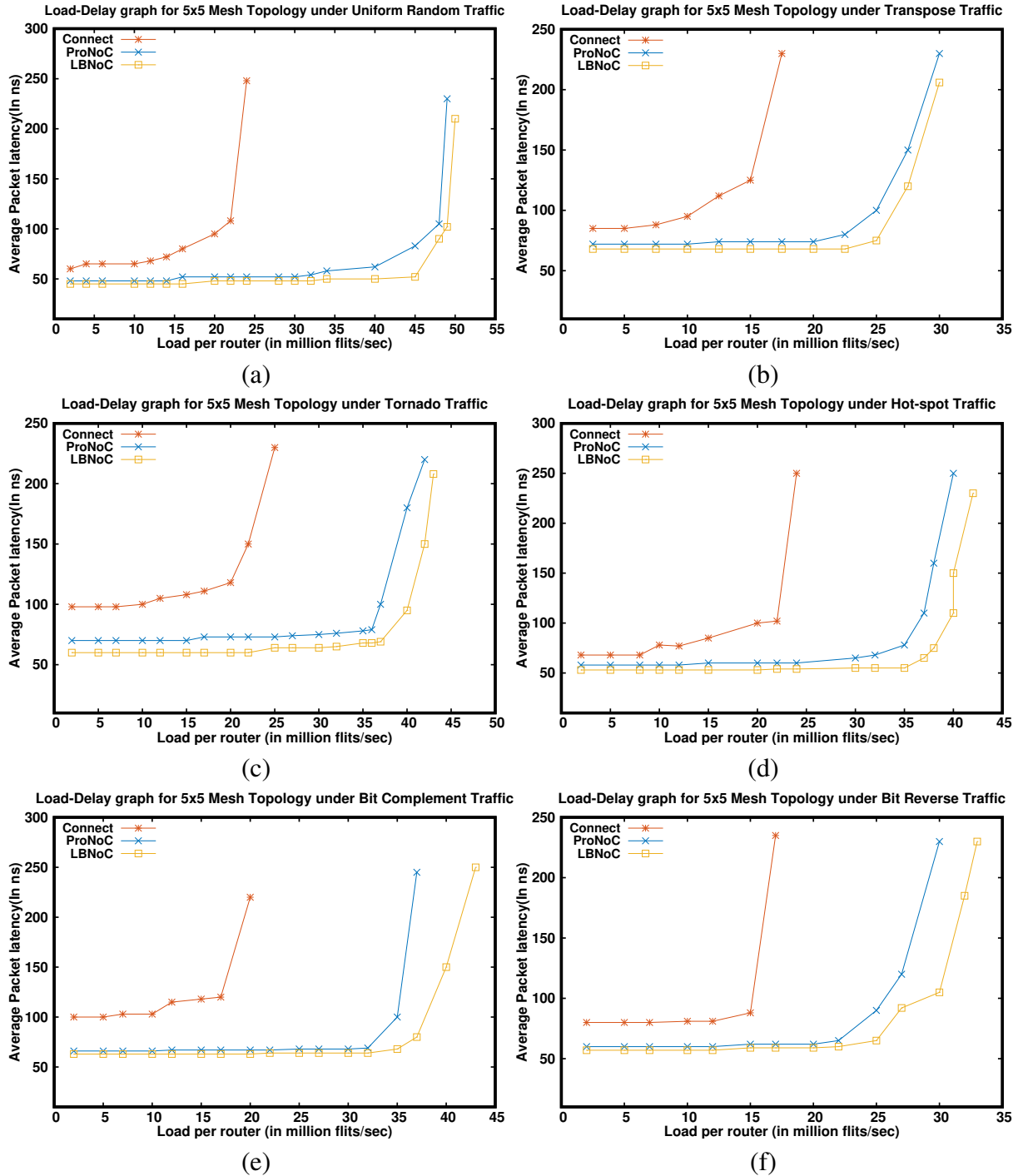


Figure 6.12: Average packet latency comparison between LBNoC, CONNECT (Pamichael and Hoe 2015) and ProNoC (Monemi et al. 2017) considering different types of traffic patterns



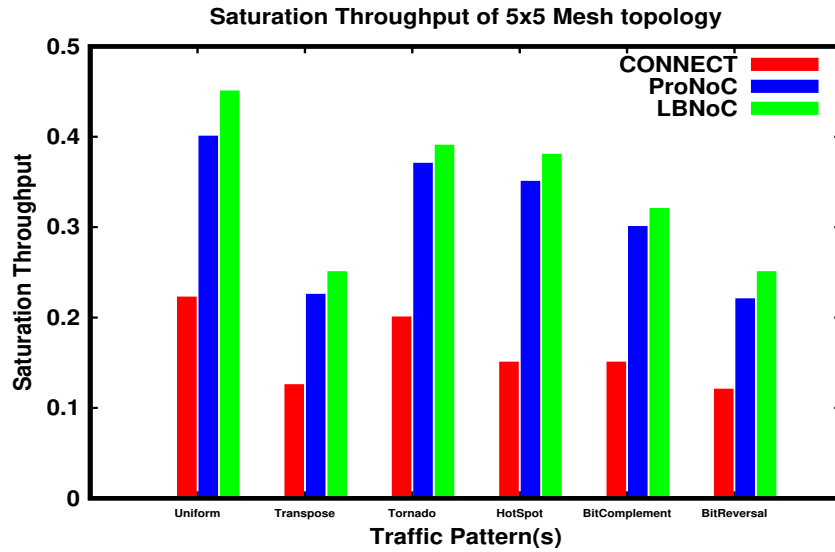


Figure 6.13: Throughput comparison of LBNoC, ProNoC and CONNECT NoC architecture

reverse traffic patterns. It can be observed that the LBNoC architecture saturates at a higher injection load compared to CONNECT and ProNoC architectures. Under the Uniform traffic pattern, LBNoC achieves an improvement of  $2.16\times$  and  $1.06\times$  with respect to CONNECT and ProNoC architectures. Similarly, under the Transpose traffic pattern, LBNoC achieves an improvement of  $2.6\times$  and  $1.16\times$  with respect to CONNECT and ProNoC architectures. LBNoC has higher saturation throughput of  $0.95\times$ ,  $1.5\times$ ,  $1.13\times$  and  $1.08\times$  under Tornado, Hot spot, Bit complement and Bit reversal traffic patterns, respectively compared to CONNECT. LBNoC has  $0.05\times$ ,  $0.08\times$ ,  $0.07\times$ , and  $0.13\times$  higher saturation throughput under Tornado, Hot spot, Bit complement and Bit reversal traffic patterns, respectively compared to ProNoC architecture.

## 6.5.2 Comparison with ASIC targetted state-of-the-art NoC architectures

### 6.5.2.1 Area, frequency and power:

Fig. 6.14 shows the FPGA resource utilization, frequency, and power results of the proposed and state-of-the-art ASIC NoC router designs.

Despite the design flow of FPGA and the design flow of ASIC have much in common in their RTL-based design and synthesis environments, they actually vary in making design decisions which affect performance and cost optimizations. When synthe-

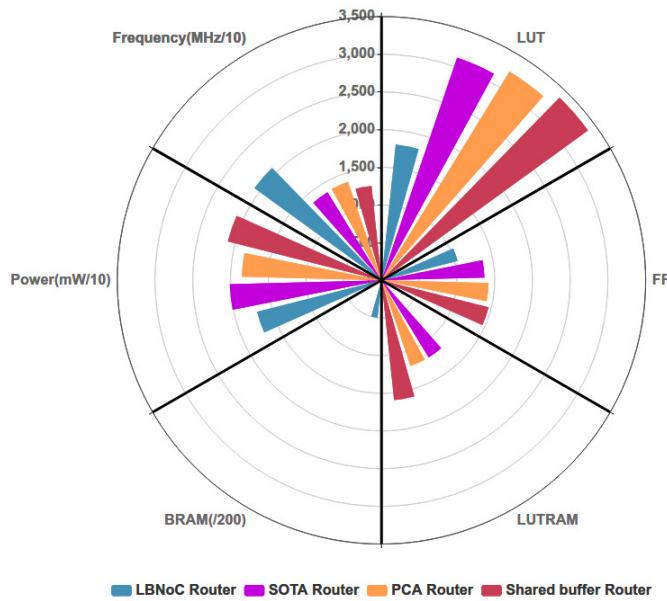


Figure 6.14: Area, Frequency and Power utilization of various router architectures

sized on an FPGA, a compactly optimized router on an ASIC may occupy more FPGA resources because of the various relative cost trading between wires, memory, and logic of the FPGA. It can be observed that the proposed NoC router architecture occupies 41.98%, 44.30%, and 46.49% fewer FPGA resource than state-of-art ASIC NoC router architectures such as publicly available RTL of router based on VCs (Stanford Concurrent VLSI Architecture Group. (2014)), which we will refer to as SOTA, Priority cooperation based round robin-arbiter (Yan et al. (2015)) which we refer to as PCA and shared buffer Becker et al. (2012) respectively. The reduction of power consumption up to 10.5%, 5.3%, and 15.84% can be observed in the proposed router with respect to SOTA, PCA, and shared buffer NoC router architectures. The proposed NoC architecture operates at a higher frequency than the state-of-the-art NoC router architectures.

Table 6.13 shows the resource utilization, maximum operating frequency and power consumption of different NoC architectures.  $4 \times 4$  mesh topology implemented employing the LBNoC NoC architecture consumes 15.25%, 19.35%, and 27.53% fewer hardware resources than the SOTA, Priority and shared buffer architectures with identical NoC configuration parameters. This is because of the design optimizations such as merged input buffers, decomposed crossbar architecture, and employing the queues of

free VCs in the proposed NoC router architecture. This in turn, results in a lower critical path delay. Hence, LBNoC NoC architecture operates at a higher frequency of 205MHz compared to SOTA(101.72MHz), PCA(106.5MHz), and Shared buffer(98.5MHz) architectures. Reduction in power consumption by 6.1%, 3.82% and 20.45% have been observed with respect to SOTA, PCA, and Shared-buffer NoC architectures. The lower power is observed due to the less FPGA resource utilization of LBNoC NoC architecture.

Table 6.13: Resource utilization and Maximum operating frequency of Different NoC configurations considering  $4 \times 4$  mesh topology

	Resource Utilization (%)	Max. Operating Frequency(MHz)	Power (mW)
SOTA	59	101.72	750
PCA	62	106.5	732
Shared buffer	69	98.5	885
LBNoC	50	205	704

### 6.5.2.2 Latency and throughput analysis:

The network performance of LBNoC is compared with publicly available state-of-the-art RTL of VC-based router Stanford Concurrent VLSI Architecture Group. (2014) and Yan et al. (2015) NoC architectures. The results for Shared buffer architecture (Soteriou et al. 2009) are obtained by modifying SOTA (Stanford Concurrent VLSI Architecture Group. 2014) RTL code. And, the results for PCA are obtained from Yan et al. (2015). Fig. 6.15 shows the average latency comparison between LBNoC, SOTA, and PCA under various traffic patterns.  $5 \times 5$  mesh topology has been considered for the experiments. An increase in the average latency can be observed with the increase in the packet injection rate. Fig. 6.15 (a) shows the load vs. latency behavior of the  $5 \times 5$  mesh topology considering various traffic patterns. It can be observed that the LBNoC architecture outperforms all the other NoC architectures consistently offering lower packet latency considering all the traffic patterns.

In Fig. 6.15, considering the  $5 \times 5$  mesh topology with LBNoC router architecture, a reduction in average latency by 10.5%, 8.1%, 1.67%, 13.68%, 9.5%, and 4.5% under the Uniform Random, Transpose, Tornado, Neighbor, Bit-complement, and Bit-reverse

## 6. Optimization of the NoC router for achieving low latency and area

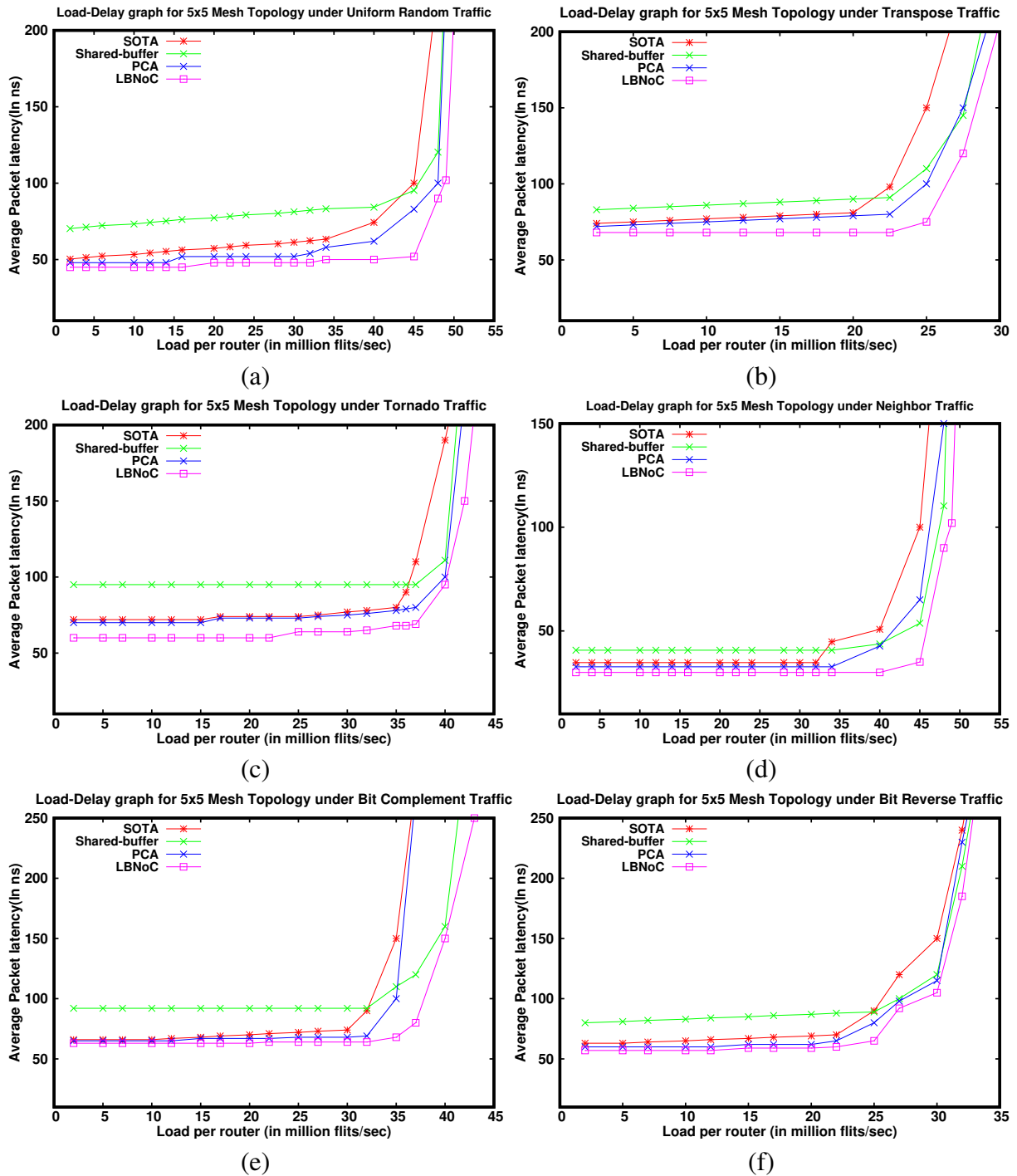


Figure 6.15: Average packet latency comparison between LBNoC, SOTA(Stanford Concurrent VLSI Architecture Group. 2014), Shared-buffer (Soteriou et al. 2009) and PCA (Yan et al. 2015) considering different types of traffic patterns

traffic patterns compared to SOTA NoC architecture. Similarly, considering the  $5 \times 5$  mesh topology LBNoC router architecture, a reduction in average latency by 30.02%, 18.01%, 32.8%, 26.3%, 28.75%, and 31.5% are observed with respect to Shared buffer

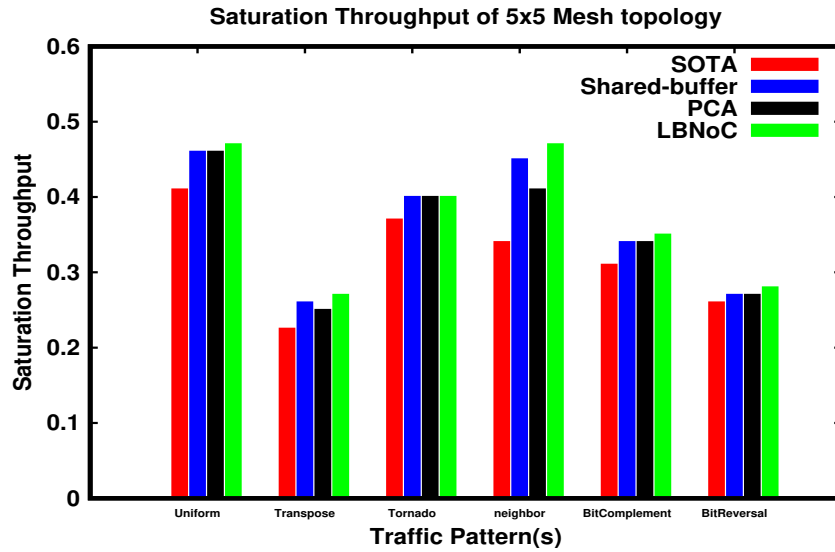


Figure 6.16: Throughput comparison of LBNOC, SOTA, Shared-buffer and PCA NoC architectures

NoC architecture under the afore-mentioned traffic patterns. And, a reduction of 6.25%, 5.5%, 1.4%, 8.2%, 5%, and 3.07% have been observed with respect to PCA.

Fig. 6.16 shows the saturation throughputs of  $5 \times 5$  mesh topology under the traffic patterns mentioned before considering the LBNOC, Shared-buffer, and PCA NoC architectures. The  $5 \times 5$  mesh topology under LBNOC router architecture is capable of sustaining more load compared to the other NoC architectures under Uniform, Transpose, Neighbor, Bit-complement, and Bit-reversal traffic patterns. Under Tornado traffic pattern, Shared buffer, and PCA architectures achieve similar saturation throughput as that of LBNOC architecture.

## 6.6 SUMMARY

In this chapter, LBNOC: an FPGA based NoC architecture is designed to reduce the area cost, latency, and improve the performance. The NoC router architecture is designed by considering the single-cycle bypass router and employing the techniques of combined flow control, parallel VC, and switch allocation. The single-cycle bypass router architecture accelerates the packets traversing long distances. The combined flow control improves the network performance by keeping the flits of the same packet together along the path. To reduce the area overhead, and to improve the network performance,

parallel VC and switch allocator are designed along with a merged input buffer and a decomposed crossbar. An FPGA based fully parameterized framework is developed to evaluate the proposed NoC architecture. The LBNoC architecture consumes fewer hardware resources and achieves lesser average packet latency than CONNECT and ProNoC architectures. Speedup of  $1.15\times$  and  $1.18\times$  are observed for LBNoC architecture with respect to ProNoC and CONNECT NoC architectures. A comparison of LBNoC architecture with the ASIC implementations of the NoC architectures such as SOTA, Shared-buffer NoC router, and PCA router is made. It is found that the LBNoC architecture achieves low latency, higher throughput, and consumes lesser power compared to the ASIC counterparts.

## CHAPTER 7

### CONCLUSIONS AND FUTURE WORK

#### 7.1 CONCLUSIONS

The contributions of this thesis are: Profiling of BookSim 2.0 simulator to analyze and improve the performance. An FPGA-based NoC simulation acceleration framework for design space exploration, the efficient techniques of mapping the NoC router components on the FPGA's hard blocks. And, the design of lightweight router architecture for NoCs using FPGAs to achieve high performance.

The initial part of the thesis deals with analyzing the performance of the BookSim 2.0 software simulator by employing profiling. Various software mechanisms have been employed to improve the performance of BookSim 2.0 NoC simulator. The speedup of  $2.93\times$  has been achieved by parallelizing the sequential code of BookSim 2.0 using OpenMP constructs considering  $30 \times 30$  Mesh topology. The parallelization and vectorization techniques reduced the execution time of  $30 \times 30$  Mesh topology from 60 minutes (normal BookSim 2.0 execution time) to 14 minutes and 12 minutes.

An FPGA based NoC simulation acceleration framework called YaNoC has been proposed. YaNoC supports the design space exploration of various standard and custom NoC topologies. The router micro-architectural parameters are highly configurable. A custom topology called Diagonal Mesh (DMesh) has been designed and evaluated considering a novel shortest path, and the Table based routing algorithms. YaNoC consumes 9.29% fewer resources and is  $2.5\times$  faster than the CONNECT framework. Also, YaNoC consumes 17.59% fewer resources and  $25\times$  faster than the DART simulator.

The speedup of  $2548\times$  compared to the BookSim 2.0 software simulator was observed using YaNoC.

YaNoC utilizes only the soft blocks (CLBs) for mapping the NoC on the FPGA. The hard-blocks such as DSP48E1 and BRAM slices can be used to map the functionality of NoC router components. The unused hard blocks of the FPGA, such as BRAM and DSP48E1 blocks, have been employed to map the functionality of NoC routers buffer and crossbar components. A reduction of soft logic has been observed employing the proposed technique. The topologies implemented considering the proposed router architecture consume 43.47%, 41.66% fewer LUTs, and FFs, respectively, compared to the topologies implemented with CLBs. A control unit called `buf_empty_checker` has been included in the circuit to reduce the latency of the network. The  $6 \times 6$  Mesh topology with proposed router architecture consumes 10.88% fewer LUT resources than the CONNECT implementation. The  $3 \times 3$  Mesh topology with the proposed router architecture consumes 73.38% and 66.55% fewer LUTs and FFs, respectively than the DARTs implementation. The average latency of the proposed work is 24.8% and 19.1% lesser than the CONNECT and DART frameworks.

An optimized FPGA-based NoC router architecture called LBNoC has been proposed to improve network performance and reduce resource utilization. The NoC router architecture has been designed by considering the single-cycle bypass router and employing the techniques of combined flow control, parallel VC, and Switch Allocation. The single-cycle bypass router architecture accelerates the packets traversing long distances. The combined flow control improves the network performance by keeping the flits of the same packet together along the path. To reduce the area overhead and improve the network performance, parallel VC and Switch Allocator have been designed along with a merged input buffer and a decomposed crossbar. The LBNoC architecture is compared with state-of-the-art CONNECT and ProNoC NoC architectures. The  $4 \times 4$  Mesh topology implemented employing the LBNoC architecture consumes 4.5% and 27.1% fewer hardware resources than the ProNoC and CONNECT architectures. The average packet latency of the LBNoC NoC architecture is 30% and 15% lesser than the CONNECT and ProNoC architectures. Speedup of  $1.15\times$  and  $1.18\times$  have been ob-



served for LBNoC architecture concerning ProNoC and CONNECT NoC architectures.

## **7.2 FUTURE WORK**

The proposed FPGA based NoC simulation platform concentrates on the design space exploration of 2-dimensional (2D) NoC architectures. As technology advances, the 3-dimensional (3D) NoC architectures play an important role in achieving better throughput and reducing the power, area, and latency compared to 2D NoCs. There is a need for designing, optimizing and evaluating novel routing algorithms, arbitration schemes and crossbar architectures for the next generation 3D NoC architectures. Future efforts can be directed towards the development of an FPGA based NoC simulator capable of design space exploration of 3D NoC architectures.



## BIBLIOGRAPHY

- Access IC Lab (2018). “Access Noxim”. <http://access.ee.ntu.edu.tw/noxim/index.html>.
- Agarwal, N., Krishna, T., Peh, L.-S. and Jha, N. (2009). “GARNET: A detailed on-chip network model inside a full-system simulator.” In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, 33–42.
- Akopyan, F., Sawada, J., Cassidy, A., Alvarez-Icaza, R., Arthur, J., Merolla, P., Imam, N., Nakamura, Y., Datta, P., Nam, G., Taba, B., Beakes, M., Brezzo, B., Kuang, J. B., Manohar, R., Risk, W. P., Jackson, B. and Modha, D. S. (2015). “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10), 1537–1557.
- Angepat, H., Chiou, D., Chung, E. S. and Hoe, J. C. (2014). “FPGA-Accelerated Simulation of Computer Systems.” *Synthesis Lectures on Computer Architecture*, 9(2), 1–80.
- Asaduzzaman, A. and Mahgoub, I. (2006). “Cache modeling and optimization for portable devices running mpeg-4 video decoder.” *Multimedia Tools and Applications*, 28(2), 239–256.
- Ax, J., Sievers, G., Daberkow, J., Flasskamp, M., Vohrmann, M., Jungeblut, T., Kelly, W., Pormann, M. and Rckert, U. (2018). “CoreVA-MPSoC: A Many-Core Architecture with Tightly Coupled Shared and Local Data Memories.” *IEEE Transactions on Parallel and Distributed Systems*, 29(5), 1030–1043.

- Balkind, J., McKeown, M., Fu, Y., Nguyen, T., Zhou, Y., Lavrov, A., Shahrads, M., Fuchs, A., Payne, S., Liang, X., Matl, M. and Wentzlaff, D. (2016). “OpenPiton: An Open Source Manycore Research Framework.” *SIGPLAN Not.*, 51(4), 217232.
- Becker, D. U. (2012). *Efficient microarchitecture for Network-on-Chip routers*. PhD dissertation, Stanford University.
- Becker, D. U., Jiang, N., Michelogiannakis, G. and Dally, W. J. (2012). “Adaptive Backpressure: Efficient buffer management for on-chip networks.” In *30th International IEEE Conference on Computer Design, ICCD 2012, Montreal, QC, Canada, September 30 - Oct. 3, 2012*, 419–426.
- Ben-Itzhak, Y., Zahavi, E., Cidon, I. and Kolodny, A. (2012). “HNOCS: Modular open-source simulator for Heterogeneous NoCs.” In *2012 International Conference on Embedded Computer Systems (SAMOS)*, 51–57.
- Benini, L. and De Micheli, G. (2002). “Networks on chips: a new SoC paradigm.” *Computer*, 35(1), 70–78.
- Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A. (2011). “The Gem5 Simulator.” *SIGARCH Comput. Archit. News*, 39(2), 1–7.
- Bjerregaard, T. and Mahadevan, S. (2006). “A Survey of Research and Practices of Network-on-Chip.” *ACM Comput. Surv.*, 38(1), 1es.
- Bohnenstiehl, B., Stillmaker, A., Pimentel, J. J., Andreas, T., Liu, B., Tran, A. T., Adeagbo, E. and Baas, B. M. (2017). “KiloCore: A 32-nm 1000-Processor Computational Array.” *IEEE Journal of Solid-State Circuits*, 52(4), 891–902.
- Catania, V., Mineo, A., Monteleone, S., Palesi, M. and Patti, D. (2015). “Noxim: An open, extensible and cycle-accurate network on chip simulator.” In *26th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2015, Toronto, ON, Canada, July 27-29, 2015*, 162–163.

- Catania, V., Mineo, A., Monteleone, S., Palesi, M. and Patti, D. (2016). “Cycle-Accurate Network on Chip Simulation with Noxim.” *ACM Trans. Model. Comput. Simul.*, 27(1), 4:1–4:25.
- Chen, Y., Krishna, T., Emer, J. S. and Sze, V. (2017). “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks.” *IEEE Journal of Solid-State Circuits*, 52(1), 127–138.
- Cherniack, M., Galvez, E. F., Franklin, M. J. and Zdonik, S. (2003). “Profile-driven cache management.” In *Proceedings 19th International Conference on Data Engineering (Cat. No.03CH37405)*, 645–656.
- Chethan, K. H. B. and Kapre, N. (2016). “Hoplite-DSP: Harnessing the Xilinx DSP48 multiplexers to efficiently support NoCs on FPGAs.” In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 1–10.
- Chiou, D., Sunwoo, D., Kim, J., Patil, N. A., Reinhart, W., Johnson, D. E., Keefe, J. and Angepat, H. (2007). “FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators.” In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 40*, IEEE Computer Society, USA, 249261.
- Chung, E. S., Papamichael, M. K., Nurvitadhi, E., Hoe, J. C., Mai, K. and Falsafi, B. (2009). “ProtoFlex: Towards Scalable, Full-System Multiprocessor Simulations Using FPGAs.” *ACM Trans. Reconfigurable Technol. Syst.*, 2(2).
- CMU-SAFARI (2018). “*NOculator*”. <http://access.ee.ntu.edu.tw/noxim/index.html>.
- Coppa, E., Demetrescu, C. and Finocchi, I. (2014a). “Input-Sensitive Profiling.” *IEEE Trans. Software Eng.*, 40(12), 1185–1205.
- Coppa, E., Demetrescu, C., Finocchi, I. and Marotta, R. (2014b). “Estimating the Empirical Cost Function of Routines with Dynamic Workloads.” In *Proceedings of Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '14*, ACM, New York, NY, USA, 230:230–230:239.

- Curtsinger, C. and Berger, E. D. (2015). “Coz: Finding Code That Counts with Causal Profiling.” In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, ACM, New York, NY, USA, 184–197.
- Dally, W. J. (1992). “Virtual-Channel Flow Control.” *IEEE Trans. Parallel Distrib. Syst.*, 3(2), 194–205.
- Dally, W. J., Intel, B. F., Chips, A. N. and Plesiochronous, M. (1986). “The Torus Routing Chip.” *Distributed Computing*, 187–196.
- Dally, W. J. and Towles, B. (2001). “Route packets, not wires: on-chip interconnection networks.” In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, 684–689.
- Dally, W. J. and Towles, B. P. (2004). *Principles and Practices of Interconnection Networks*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Daniel Marjamki (2011). “Cppcheck, A tool for static C/C++ code analysis”. <http://cppcheck.sourceforge.net/>.
- Drewes, T., Joseph, J. M. and Pionteck, T. (2017). “An FPGA-based prototyping framework for Networks-on-Chip.” In *2017 International Conference on ReConfigurable Computing and FPGAs (ReConFig)*, 1–7.
- Galles, M. (1997). “Spider: a high-speed network interconnect.” *IEEE Micro*, 17(1), 34–39.
- Glass, C. J. and Ni, L. M. (1992). “The Turn Model for Adaptive Routing.” In *[1992] Proceedings the 19th Annual International Symposium on Computer Architecture*, 278–287.
- Guerrier, P. and Greiner, A. (2000). “A generic architecture for on-chip packet-switched interconnections.” In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, 250–256.

- Hayenga, M., Jerger, N. D. E. and Lipasti, M. H. (2009). “SCARAB: a single cycle adaptive routing and bufferless network.” In *42st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-42 2009), December 12-16, 2009, New York, New York, USA*, 244–254.
- Infante, A. (2014). “Identifying Caching Opportunities, Effortlessly.” In *Companion Proceedings of the 36th International Conference on Software Engineering, ICSE Companion 2014, ACM, New York, NY, USA*, 730–732.
- Intel Corporation (2017). “*Intel Advisor XE*”. <https://software.intel.com/en-us/intel-advisor-xe>.
- Jensen, S. H., Sridharan, M., Sen, K. and Chandra, S. (2015). “MemInsight: Platform-independent Memory Debugging for JavaScript.” In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, ACM, New York, NY, USA*, 345–356.
- Jerger, N. D. E. and Peh, L.-S. (2009). “On-chip networks.” In *On-Chip Networks*.
- Jiang, N., Becker, D., Michelogiannakis, G., Balfour, J., Towles, B., Shaw, D., Kim, J. and Dally, W. (2013). “A detailed and flexible cycle-accurate Network-on-Chip simulator.” In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, 86–96.
- Kahng, A. B., Li, B., Peh, L.-S. and Samadi, K. (2012). “ORION 2.0: A Power-Area Simulator for Interconnection Networks.” *IEEE Trans. Very Large Scale Integr. Syst.*, 20(1), 191196.
- Kahng, A. B., Lin, B. and Nath, S. (2015). “ORION3.0: A Comprehensive NoC Router Estimation Tool.” *IEEE Embedded Systems Letters*, 7(2), 41–45.
- Kamali, H. M., Azar, K. Z. and Hessabi, S. (2018). “DuCNoC: A High-Throughput FPGA-Based NoC Simulator Using Dual-Clock Lightweight Router Micro-Architecture.” *IEEE Transactions on Computers*, 67(2), 208–221.

- Kamali, H. M. and Hessabi, S. (2016). “AdapNoC: A fast and flexible FPGA-based NoC simulator.” In *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 1–8.
- Kapre, N. and Gray, J. (2015). “Hoplite: Building austere overlay NoCs for FPGAs.” In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 1–8.
- Kermani, P. and Kleinrock, L. (1979). “Virtual cut-through: A new computer communication switching technique.” *Computer Networks (1976)*, 3(4), 267 – 286.
- Kowarschik, M. and Wei, C. (2003). “An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms.” 213–232.
- Kumar, A., Peh, L.-S., Kundu, P. and Jha, N. K. (2007). “Express Virtual Channels: Towards the Ideal Interconnection Fabric.” *SIGARCH Comput. Archit. News*, 35(2), 150–161.
- Larsen, S., Rabbah, R. and Amarasinghe, S. (2005). “Exploiting vector parallelism in software pipelined loops.” In *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’05)*, 11 pp.–129.
- Lebeck, A. R. and Wood, D. A. (1994). “Cache profiling and the SPEC benchmarks: a case study.” *Computer*, 27(10), 15–26.
- Li-Shiuan Peh and Dally, W. J. (2001). “A delay model for router microarchitectures.” *IEEE Micro*, 21(1), 26–34.
- Liu, X. and Mellor-Crummey, J. (2013). “A Data-centric Profiler for Parallel Programs.” In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC ’13*, ACM, New York, NY, USA, 28:1–28:12.
- Lotlikar, S., Pai, V. and Gratz, P. V. (2011). “AcENoCs: A Configurable HW/SW Platform for FPGA Accelerated NoC Emulation.” In *2011 24th International Conference on VLSI Design*, 147–152.



- Lu, Y., McCanny, J. and Sezer, S. (2011). “Exploring Virtual-Channel architecture in FPGA based Networks-on-Chip.” In *2011 IEEE International SOC Conference*, 302–307.
- Luo, T., Liu, S., Li, L., Wang, Y., Zhang, S., Chen, T., Xu, Z., Temam, O. and Chen, Y. “dadiannao: A neural network supercomputer.” *IEEE Transactions on Computers*.
- Mahlke, S., Moseley, T., Hank, R., Bruening, D. and Cho, H. K. (2013). “Instant Profiling: Instrumentation Sampling for Profiling Datacenter Applications.” In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, CGO '13, IEEE Computer Society, Washington, DC, USA, 1–10.
- Michelogiannakis, G., Sánchez, D., Dally, W. J. and Kozyrakis, C. (2010). “Evaluating Bufferless Flow Control for On-chip Networks.” In *NOCS 2010, Fourth ACM/IEEE International Symposium on Networks-on-Chip, Grenoble, France, May 3-6, 2010*, 9–16.
- Monemi, A. (2015). “Low Latency Network-on-Chip Router Microarchitecture Using Request Masking Technique.” *Int. J. Reconfig. Comput.*, 2015(2), 1–7.
- Monemi, A., Tang, J. W., Palesi, M. and Marsono, M. N. (2017). “ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform.” *Microprocessors and Microsystems*, 54, 60 – 74.
- Moscibroda, T. and Mutlu, O. (2009). “A case for bufferless routing in on-chip networks.” In *36th International Symposium on Computer Architecture (ISCA 2009)*, June 20-24, 2009, Austin, TX, USA, 196–207.
- Mullins, R., West, A. and Moore, S. (2004). “Low-latency virtual-channel routers for on-chip networks.” *SIGARCH Comput. Archit. News*, 32(2), 188–.
- Nethercote, N. and Seward, J. (2007). “Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation.” In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '07*, ACM, New York, NY, USA, 89–100.

- Nguyen, K. and Xu, G. (2013). “Cachetor: Detecting Cacheable Data to Remove Bloat.” In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, ACM, New York, NY, USA, 268–278.
- Nicopoulos, C., Park, D., Kim, J., Vijaykrishnan, N., Yousif, M. S. and Das, C. R. (2006). “ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers.” In *39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39 2006)*, 9-13 December 2006, Orlando, Florida, USA, 333–346.
- Nie, J., Cheng, B., Li, S., Wang, L. and Li, X. F. (2010). “Vectorization for Java.” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6289 LNCS, 3–17.
- Nistor, A. and Ravindranath, L. (2014). “SunCat: Helping Developers Understand and Predict Performance Problems in Smartphone Applications.” In *Proceedings of the 2014 International Symposium on Software Testing and Analysis, ISSTA 2014*, ACM, New York, NY, USA, 282–292.
- Nistor, A., Song, L., Marinov, D. and Lu, S. (2013). “Toddler: Detecting Performance Problems via Similar Memory-access Patterns.” In *Proceedings of the 2013 International Conference on Software Engineering, ICSE ’13*, IEEE Press, Piscataway, NJ, USA, 562–571.
- Ogras, U. Y., Bogdan, P. and Marculescu, R. (2010). “An Analytical Approach for Network-on-Chip Performance Analysis.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12), 2001–2013.
- Pande, P. P., Grecu, C., Jones, M., Ivanov, A. and Saleh, R. (2005). “Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures.” *IEEE Transactions on Computers*, 54(8), 1025–1040.
- Papamichael, M. K. (2011). “Fast scalable FPGA-based Network-on-Chip simulation models.” In *Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMPCODE2011)*, 77–82.

- Papamichael, M. K. and Hoe, J. C. (2015). “The CONNECT Network-on-Chip Generator.” *Computer*, 48(12), 72–79.
- Papamichael, M. K., Hoe, J. C. and Mutlu, O. (2011). “FIST: A fast, lightweight, FPGA-friendly packet latency estimator for NoC modeling in full-system simulations.” In *Proceedings of the Fifth ACM/IEEE International Symposium*, 137–144.
- Parane, K., Prasad, P. B. M. and Talawar, B. (2018). “FPGA based NoC Simulation Acceleration Framework Supporting Adaptive Routing.” In *2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*.
- Patel, A., Afram, F., Chen, S. and Ghose, K. (2011). “MARSS: A full system simulator for multicore x86 CPUs.” In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 1050–1055.
- Peh, L. and Dally, W. J. (2000). “Flit-reservation flow control.” In *Proceedings of the Sixth International Symposium on High-Performance Computer Architecture, Toulouse, France, January 8-12, 2000*, 73–84.
- Peh, L. S. and Dally, W. J. (2001). “A delay model and speculative architecture for pipelined routers.” In *HPCA 2011*, 255–266.
- Pienaar, J. A. and Hundt, R. (2013). “JSWhiz: Static Analysis for JavaScript Memory Leaks.” In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, CGO ’13, IEEE Computer Society, Washington, DC, USA, 1–11.
- Porterfield, A. K. (1989). *Software Methods for Improvement of Cache Performance on Supercomputer Applications*. PhD thesis, Rice University.
- Prabhu Prasad B M, Parane, K. and Talawar, B. (2018). “YaNoC: Yet Another Network-on-Chip Simulation Acceleration Engine Using FPGAs.” In *2018 31st International Conference on VLSI Design and 17th International Conference on Embedded Systems (VLSID)*, 67–72.

- Prasad, P. B. M., Parane, K. and Talawar, B. (2019). “Analysis of cache behaviour and software optimizations for faster on-chip network simulations.” *Int. J. Syst. Assur. Eng. Manag.*, 10(4), 696–712.
- Puente, V., Gregorio, J. and Beivide, R. (2002). “SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems.” In *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euro-micro Workshop on*, 15–22.
- Ramanujam, R. S., Soteriou, V., Lin, B. and Peh, L. (2010). “Design of a High-Throughput Distributed Shared-Buffer NoC Router.” In *NOCS 2010, Fourth ACM/IEEE International Symposium on Networks-on-Chip, Grenoble, France, May 3-6, 2010*, 69–78.
- Ramanujam, R. S., Soteriou, V., Lin, B. and Peh, L. (2011). “Extending the Effective Throughput of NoCs With Distributed Shared-Buffer Routers.” *IEEE Trans. on CAD of Integrated Circuits and Systems*, 30(4), 548–561.
- Randall, M. and Lewis, A. (2002). “A Parallel Implementation of Ant Colony Optimization.” *Journal of Parallel and Distributed Computing*, 62(9), 1421–1432.
- Ronak, B. and Fahmy, S. A. (2016). “Mapping for Maximum Performance on FPGA DSP Blocks.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(4), 573–585.
- Sanchez, D. and Kozyrakis, C. (2013). “ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-core Systems.” *SIGARCH Comput. Archit. News*, 41(3), 475–486.
- Sembrant, A., Black-Schaffer, D. and Hagersten, E. (2012). “Phase Guided Profiling for Fast Cache Modeling.” In *Proceedings of the Tenth International Symposium on Code Generation and Optimization, CGO '12*, ACM, New York, NY, USA, 175–185.
- Sodani, A., Gramunt, R., Corbal, J., Kim, H., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R. and Liu, Y. (2016). “Knights Landing: Second-Generation Intel Xeon Phi Product.” *IEEE Micro*, 36(2), 34–46.

- Song, L., Kavi, K. and Cytron, R. (2003). *Software and Compilers for Embedded Systems: 7th International Workshop, SCOPES 2003, Vienna, Austria, September 24-26, 2003. Proceedings*, chapter An Unfolding-Based Loop Optimization Technique, 117–132. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Soteriou, V., Ramanujam, R. S., Lin, B. and Peh, L. (2009). “A High-Throughput Distributed Shared-Buffer NoC Router.” *Computer Architecture Letters*, 8(1), 21–24.
- Stanford Concurrent VLSI Architecture Group. (2014). “Open Source Network-on-Chip Router RTL”. <https://github.com/anan-cn/Open-Source-Network-on-Chip-Router-RTL>.
- Tan, Z., Waterman, A., Avizienis, R., Lee, Y., Cook, H., Patterson, D. and Asanovic, K. (2010). “RAMP gold: An FPGA-based architecture simulator for multiprocessors.” In *Design Automation Conference*, 463–468.
- Thiem Van Chu, Sato, S. and Kise, K. (2015). “Ultra-fast NoC emulation on a single FPGA.” In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 1–8.
- Ting-Shuo Hsu, Jun-Lin Chiu, Chao-Kai Yu and Jing-Jia Liou (2015). “A fast and accurate network-on-chip timing simulator with a flit propagation model.” In *The 20th Asia and South Pacific Design Automation Conference*, 797–802.
- Varga, A. (1999). “Using the OMNeT++ Discrete Event Simulation System in Education.” *IEEE Trans. on Educ.*, 42(4), 11 pp.–.
- Wang, D., Lo, C., Vasiljevic, J., Enright Jerger, N. and Gregory Steffan, J. (2014). “DART: A Programmable Architecture for NoC Simulation on FPGAs.” *IEEE Transactions on Computers*, 63(3), 664–678.
- Wang, J., Huang, Y., Ebrahimi, M., Huang, L., Li, Q., Jantsch, A. and Li, G. (2016). “VisualNoC: A Visualization and Evaluation Environment for Simulation and Mapping.” In *Proceedings of the Third ACM International Workshop on Many-Core Embedded Systems, MES 16*, Association for Computing Machinery, New York, NY, USA, 1825.

- Wee, S., Casper, J., Njoroge, N., Tesylar, Y., Ge, D., Kozyrakis, C. and Olukotun, K. (2007). “A Practical FPGA-Based Framework for Novel CMP Research.” In *Proceedings of the 2007 ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays*, FPGA 07, Association for Computing Machinery, New York, NY, USA, 116125.
- Wolkotte, P. T., Holzspies, P. K. F. and Smit, G. J. M. (2007). “Fast, Accurate and Detailed NoC Simulations.” In *First International Symposium on Networks-on-Chip (NOCS’07)*, 323–332.
- Xilinx Inc (2016). “7 Series FPGAs Configurable Logic Block”. [https://www.xilinx.com/support/documentation/user\\_guides/ug474\\_7Series\\_CLB.pdf](https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf).
- Xilinx Inc (2018). “7 Series DSP48E1 Slice User Guide”. [https://www.xilinx.com/support/documentation/user\\_guides/ug479\\_7Series\\_DSP48E1.pdf](https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf).
- Xilinx Inc (2019). “7 Series FPGAs Memory Resources”. [https://www.xilinx.com/support/documentation/user\\_guides/ug473\\_7Series\\_Memory\\_Resources.pdf](https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf).
- Xu, C., Liu, Y. and Yang, Y. (2019). “SRNoC: An Ultra-fast Configurable FPGA-based NoC Simulator Using Switch-Router Architecture.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1–1.
- Xu, G., Bond, M. D., Qin, F. and Rountev, A. (2011). “LeakChaser: Helping Programmers Narrow Down Causes of Memory Leaks.” *SIGPLAN Not.*, 46(6), 270–282.
- Yan, D., Xu, G. and Rountev, A. (2012). “Uncovering Performance Problems in Java Applications with Reference Propagation Profiling.” In *Proceedings of the 34th International Conference on Software Engineering, ICSE ’12*, IEEE Press, Piscataway, NJ, USA, 134–144.

- Yan, P., Jiang, S. and Sridhar, R. (2015). “A high throughput router with a novel switch allocator for network on chip.” In *28th IEEE International System-on-Chip Conference, SOCC 2015, Beijing, China, September 8-11, 2015*, 160–163.
- Yan, P. and Sridhar, R. (2018). “Centralized Priority Management Allocation for Network-on-Chip Router.” In *31st IEEE International System-on-Chip Conference, SOCC 2018, Arlington, VA, USA, September 4-7, 2018*, 290–295.
- Zaparanuks, D. and Hauswirth, M. (2012). “Algorithmic Profiling.” In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12*, ACM, New York, NY, USA, 67–76.
- Zhang, Y., Qu, P., Qian, Z., Wang, H. and Zheng, W. (2013). “Software/hardware hybrid network-on-chip simulation on fpga.” In Hsu, C.-H., Li, X., Shi, X. and Zheng, R., editors, *Network and Parallel Computing*, Springer Berlin Heidelberg, Berlin, Heidelberg, 167–178.
- Zhao, Q., Cutcutache, I. and Wong, W.-F. (2008). “Pipa: Pipelined Profiling and Analysis on Multi-core Systems.” In *Proceedings of the 6th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO '08*, ACM, New York, NY, USA, 185–194.





## PUBLICATIONS BASED ON THE RESEARCH WORK

### Journal Publications

1. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, FPGA friendly NoC simulation acceleration framework employing the Hard Blocks , *Computing*, Springer, (Accepted).
2. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, An efficient FPGA based Network-on-Chip simulation framework utilizing the Hard blocks, *Circuits, Systems, and Signal Processing* , Springer, 2020, <https://doi.org/10.1007/s00034-020-01411-z>
3. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, P-NoC: Performance Evaluation and Design Space Exploration of NoCs for chip multiprocessor architecture using FPGA, *Wireless Personal Communications*, Springer, 2020, <https://doi.org/10.1007/s11277-020-07529-2>
4. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, LBNoC-Design of low-latency router architecture with Lookahead Bypass for Network-on-Chip using FPGA, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, ACM, 2020, 25(1), <https://doi.org/10.1145/3365994>
5. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, Analysis of cache behaviour and software optimizations for faster on-chip network simulations, *International Journal of System Assurance Engineering and Management*, Springer, 2019, 10(4), 696712, <https://doi:10.1007/s13198-019-00799-5>

6. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, YaNoC: Yet Another Network-on-Chip Simulation Acceleration Engine Supporting Congestion-Aware Adaptive Routing Using FPGAs, *Journal of Circuits Systems and Computers*, World Scientific, 2019, 28:12, [https://doi:0.1142/S0218126619502025](https://doi.org/10.1142/S0218126619502025)

### Conference Publications

1. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, High-Performance NoC Simulation Acceleration Framework Employing the Xilinx DSP48E1 Blocks, in *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. Hsinchu, Taiwan:IEEE, April 2019.
2. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, Design of an Adaptive and Reliable Network on Chip Router Architecture Using FPGA, in *2019 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. Hsinchu, Taiwan:IEEE, April 2019.
3. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, High-Performance NoCs Employing the DSP48E1 Blocks of the Xilinx FPGAs, in *20th International Symposium on Quality Electronic Design, ISQED*. Santa Clara, CA, USA, USA:IEEE, March 2019.
4. G S Sangeetha, Vignesh Radhakrishnan, **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, Trace-Driven Simulation and Design Space Exploration of Network-on-Chip Topologies on FPGA, in *2018 8th International Symposium on Embedded Computing and System Design (ISED)*. Cochin, India:IEEE, Dec 2018.
5. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, FPGA based NoC Simulation Acceleration Framework Supporting Adaptive Routing, in *2018 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*. Bangalore, India:IEEE, Oct 2018.
6. **Prabhu Prasad B M**, Khyamling Parane, and Basavaraj Talawar, YaNoC: Yet Another Network-on-Chip Simulation Acceleration Engine Using FPGAs, in *VLSI*

*Design and 2018 17th International Conference on Embedded Systems (VLSID), 2018 31st International Conference on.* Pune, India:IEEE, Jan 2018.

7. Khyamling Parane, **Prabhu Prasad B M**, and Basavaraj Talawar, Cache analysis and software optimizations for faster on-chip network simulations, in *2016 11th International Conference on Industrial and Information Systems (ICIIS)*. IIT Roorkee, India:IEEE, Dec 2016.



## BIO-DATA

Name: Prabhu Prasad B M

Date of Birth: 15/05/1990

email-id: prabhuprasad1990@gmail.com

Contact No: +91 90088 62681

Present Address: Prabhu Prasad B M, Research Scholar, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, Mangalore - 575 025

Permanent Address: D.No: 613, Basava Prabhu, 1st main, 2nd cross, DCM township, Davanagere - 577 003

Educational Qualifications: B.E in Computer Science and Engineering - U.B.D.T College of Engineering, Davanagere  
M.Tech in Computer Science and Engineering - Siddaganga Institute of Technology, Tumkur

Work Experience: MTS - Quality Engineer Developer, Sep 2013 to April 2015, VMware Software India Pvt. Ltd., Bangalore  
MTS - Intern, Aug 2012 to Sep 2013, VMware Software India Pvt. Ltd., Bangalore