FPGA BASED SIMULATION ACCELERATION OF ON-CHIP NETWORKS

Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

Khyamling

(155034 CS15FV05)

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 **FPGA based simulation acceleration of on-chip networks** 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.

Khyamling, 155034CS15FV05
Department of Computer Science and Engineering

Place: NITK, Surathkal.
Date: December 01, 2020

CERTIFICATE

This is to certify that the Research Thesis entitled **FPGA based simulation acceleration of On-Chip Networks** submitted by **Khyamling** (Register Number: 155034 CS15FV05) 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)

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor Dr. Basavaraj Talawar, for his guidance, support and encouragement throughout my research work at the Department of Computer Science and Engineering NITK, Surathkal. His guidance and insightful comments helped me in all the time of research and writing of this thesis.

I express heartfelt thanks to my research progress assessment committee members Dr. Manu Basavaraj, Associate Professor in Department of Computer Science & Engineering, and Dr. A. V. Narasimhadhan, Assistant Professor in Electronics & Communication Engineering, for giving their valuable suggestions, inspiration and moral support while evaluating our work time to time. Also, I would like to thank Dr.Alwyn Roshan Pais Head of the Department of Computer Science & Engineering, NITK, Surathkal. I am also grateful to all the faculty members and non-teaching staff of CSE Department for their generous support throughout this work.

I would like to say thanks to my friends and colleagues Dr. Shashidhara, Dr.Girish G.N, Mr. Madhu Biradar, Mr.Subodh Ingaleshwar, Mr.Shivananda Poojar Mr.Manoj Patil, Dr.Praveen Tumkur.R, Dr.Anil Kadam, Mr. Nikhil, Dr.D.V.N. Sivakumar, Dr.Amit Praseed, Dr.Manjunath Mulimani, Mr.Manjunatha, Mr.Ramteke Pravin Bhaskar, Dr.Sachin Patil, Mr.Nutan Prasad and Mr.Vishal Rathod for their company and support during my research course.

I would like to show my sincere thanks to all lab mates for their support and valuable technical discussions. It would be my pleasure to extend my gratitude to Mr.Prabhu Prasad B. M, Mr.Pramod Yelmewad, Mr.Anil Kumar, Dr. Bheemappa Halavar, Mr. Kallinatha and Mrs.Sadhana Shetty for their massive supports during my research.

My family's support has been enormous throughout my journey. Biggest thanks to my father, mother, uncle, aunty, sisters, and brothers, and all my relatives for their

remarkable supports. Thank you all once again for constant support and blessings throughout my journey. Thanks to my wife Mrs.Revati Patil for her patience, understanding, prayers and continuing support to complete this research work. Thank you dear for always being with me in tough times. Now, it's time to start new phase of my life. Thank you all!!!

Khyamling

ABSTRACT

As the number of processing cores in the Systems-on-Chip(SoC) increases, the traditional bus based interconnect will be the major bottleneck to achieving the performance required by modern applications. Further, bus based communication may not provide the required bandwidth and latency to the systems with intensive parallel communication. An efficient interconnection architecture is required to achieve high performance and scalability in many-cores SoC. The Network-on-Chip(NoC) architecture has emerged as the most promising interconnection architecture for the modern Chip Multiprocessor(CMP) and Multi/Many-Processor System-on-Chip(MPSoC) systems. The components in these systems, the cores, accelerators, memory blocks, and peripherals are interconnected using one or more NoCs composed of links and routers. The choice of router parameters and NoC topologies can have a significant impact on the overall performance of heterogeneous many-core systems.

The evaluation methodologies of NoCs for future computing systems with a large number of interconnected components rely heavily on analytical models and simulations. The fast modeling of large scale NoCs have been done through analytical models with significant inaccuracy. Fast and flexible NoC simulator frameworks are needed for modeling the large scale NoC based heterogeneous many-core systems, which can deliver a high level of accuracy.

Detailed software simulators used for design space exploration of NoCs, provide better accuracy than analytical modelings. However, software simulators are slow when simulating large scale NoCs for interconnection of various components.

This thesis presents the optimization of software based NoC simulator and a Field programmable gate arrays(FPGA) based NoC simulation acceleration framework to address the issue of simulation speed, accuracy, and flexibility. Initial work in the thesis involves profiling of the Booksim2.0 software simulator, as it is used extensively for the design and evaluation of NoC architectures. The Booksim2.0 is profiled with the various NoC design parameters and memory configurations to analyze its performance.

The performance analysis of Booksim2.0 is based on cache misses, memory usage, and hotspots. Profiling helped in applying focussed software optimization techniques on the simulator. Further, Booksim2.0 was parallelized using OpenMP and SIMD constructs to improve its overall performance.

Going beyond software optimization, an FPGA based NoC simulation acceleration framework called YaNoC is proposed to explore the impact of microarchitectural parameters on the performance of the NoC. YaNoC supports for design space exploration of custom topologies with custom routing algorithm along with standard minimal routing algorithm for conventional NoCs. The YaNoC is used to study NoC architectures of a CMP using various traffic patterns, the results show that the YaNoC utilize fewer FPGA resources and is faster than the other state-of-art FPGA based NoC simulation acceleration platforms.

The next challenge was to optimize the resources consumed by YaNoC. The FPGA fabric provides hard resources such as Block RAM(BRAM) and DSP48E1 units along with specialized interconnect. Most of the state-of-art FPGA based simulators utilize soft logic only for modeling the NoCs, leaving out the hard blocks to be unutilized. The Input buffer and crossbar functionality of NoC routers embed onto the hard block of Xilinx BRAM and DSP48E1 units thereby reducing the dependence on soft logic. A pure configurable logic block implementation and a hard block based implementation of the NoC router exhibit identical latency and performance behaviour. The utilization of hard units for the design of NoCs results in high performance with low cost design compared to state-of-art frameworks.

Next, the design of an FPGA based parameterized framework called P-NoC with configurable Topology, Router and Traffic modules for performance evaluation and design space exploration has been presented. The P-NoC enables the designer to choose from a variety of architectural parameters like Input buffers, Virtual Channels, routing algorithms, traffic patterns, topology for exploration of NoC design. The P-NoC also supports a flexible communication model and traffic generation.

In the last piece of work, an FPGA based NoC using a low latency router with a look ahead bypass(LBNoC) has been proposed. The LBNoC design targets the optimized

area with improved network performance. The techniques such as a single-cycle router bypass, adaptive routing module, parallel Virtual Channel (VC), and Switch allocation, combined virtual cut through and wormhole switching, have been employed in the designing optimized LBNoC router. The LBNoC architecture consumes fewer hardware resources, reduction in average packet latency and gain in speedup than the state-of-art NoC architectures.

Keywords: Network-on-chip(NoC), Field Programmable Gate Arrays(FPGAs), Simulation framework, Simulation Acceleration, Performance Analysis, DSP48E1, Block RAM, Adaptive Routing.

CONTENTS

List of Figures **xi**

List of Tables **xvi**

List of Abbreviations **xvii**

1 Introduction **1**

1.1 Problem Description 4

1.2 Research Objectives 4

1.3 Thesis Contributions 6

1.4 Thesis Organization 7

2 Background and Review of Related work **9**

2.1 Network-on-Chip: An Overview 9

2.2 NoC Performance Parameters 13

2.3 Field Programmable Gate Array 14

2.4 Related Work 16

3 Analysis of cache behaviour and software optimizations for faster on-chip network simulations **23**

3.1 Methodology 24

3.2 Profiling and Software optimization techniques 26

3.3 Profiling, Performance Optimization Tools and Experimental methodology 29

3.4 Results and Discussion 31

3.5 Experimental results based on Optimization Strategies 38

3.6 Summary 44

4 YaNoC - FPGA based simulation acceleration Framework **47**

4.1 Introduction 47

4.2 YaNoC - Design and Implementation 48
4.3 Design of Mesh and Diagonal Mesh (DMesh) topologies 55
4.4 The proposed Reliable Network on Chip router 63
4.5 Experimental Results 64
4.6 YaNoC vs. State-Of-The-Art 76
4.7 Summary 78

5 Mapping the NoC Router Components on the DSP48E1 Hard blocks of the FPGA 79

5.1 Introduction 79
5.2 NoC Architecture 81
5.3 DSP48E1 tile as the Crossbar Switch 82
5.4 Results and Discussion 87
5.5 Summary 98

6 P-NoC: Performance Evaluation of NoCs architecture using FPGA 99

6.1 P-NoC: FPGA-based parameterized framework 99
6.2 Design Cost and Performance Analysis 105
6.3 Results and Discussion 108
6.4 summary 119

7 Design of Low latency and Area efficient Router Architecture for NoC using FPGA 121

7.1 Introduction 121
7.2 Related Work 122
7.3 LBNoC-FPGA based Bypass NoC Framework 124
7.4 Results and Discussion 135
7.5 Comparison with the State-of-the-Art NoC architectures 143
7.6 summary 151

8 Conclusions and Future Works 153

Appendix 157

A.1 Shortest Path Routing Algorithm for DMesh topology 157

Bibliography 159

Publications 173

LIST OF FIGURES

1.1 Booksim2.0 Execution Time Of k-ary n-dimensional Mesh Networks(n=2,3 and 4) Parane et al. (2016). 3

2.1 An NoC router microarchitecture with M input/output ports, N-virtual channels at each input port and M × M Crossbar switch (Enright and Peh (2009)). 11

2.2 4×4 NoC Mesh topology, Each PEs connects to a local port of router through NI, other ports of router connects to North, East, South and West neighbours using links (Partha Pratim Pande et al. (2005)). 12

2.3 A generic architecture of Xilinx FPGA (Xilinx Inc (2019b)) 15

2.4 Architecture of the Configurable Logic Block(Xilinx Inc (2016)) 16

3.1 Booksim2.0 Execution Time Of k-ary n-dimensional Mesh Networks(n=2,3 and 4). 25

3.2 Average I1 MPKI of Booksim2.0 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) 32

3.3 Average D1 MPKI of Booksim2.0 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) 34

3.4 Average LL MPKI of Booksim2.0 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) 35

3.5 CPI For Booksim2.0 Running Various Sizes Of Mesh Topology 38

3.6 Cache misses before and after optimization 39

3.7 Speedup achieved before and after optimization 40
3.8 Simulation execution times before and after improvements 43
3.9 Speedups with Mesh topology of varying sizes 43
3.10 Average packet latency for Booksim2.0 and Optimized Booksim2.0 for a 3×3 Mesh topology 44

4.1 Architecture of the proposed YaNoC FPGA based NoC simulation acceleration framework 49
4.2 (a)Flit types and (b)Packet structure used in experiments. (Time stamp field is useful in calculating the latency of a packet) 50
4.3 Modified Router architecture supporting Congestion-aware Adaptive routing 51
4.4 5 Stage Router pipeline 53
4.5 A High-level block diagram of YaNoC consisting of Host PC connected to an FPGA Board. 54
4.6 Simulation framework flow 55
4.7 Mesh and Diagonal Mesh topologies (Red and Green colors indicate the routes calculated by XY and novel shortest path XY routing algorithms) 57
4.8 Interconnection of the Router 12 with other Routers in DMesh topology 58
4.9 A deadlock avoidance in the YaNoC using VCs 63
4.10 (a)Input buffer fault tolerance strategy, (b)Crossbar fault tolerance strategy and (c) proposed Adaptive and Reliable router architecture 63
4.11 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 69
4.12 Load Delay graph of 8x8 Mesh and Torus Topologies under Bit Complement Traffic patterns (a)Buffer Depth=8flits and (b)Buffer Depth=16flits 70
4.13 (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 71

4.14 Load Delay graph of Mesh Topology under (a)Uniform and (b)Transpose traffic patterns 72
4.15 proposed Adaptive and Reliable router architecture 74

5.1 NoC router with the proposed DSP48E1 based crossbar 81
5.2 Two DSP48E1 slices connected by dedicated cascade links form a single DSP tile (Xilinx Inc (2018)) 83
5.3 Illustration of mapping the input ports to the DSP48E1 based crossbar . 85
5.4 (a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the 6×6 Mesh and Torus topologies under CLB and DSP48E1 based crossbar implementation under Nearest Neighbor, Random Permutation, Hotspot, Tornado and Bit complement traffic patterns employing the XY routing and LA routing 93
5.5 (a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the 8×8 Mesh and Torus topologies with CLB and DSP48E1 based crossbar implementation under Nearest Neighbor, Random Permutation, Hotspot, Tornado and Bit complement traffic patterns employing the XY routing and LA routing 95
5.6 (a), (b) - Load vs Latency comparison of the Mesh topologies employing proposed DSP48E1 crossbar architecture and CONNECT, DART topologies 97

6.1 Layout of P-NoC: An FPGA based Parameterized framework for design space exploration and performance analysis of NoCs for Chip Multiprocessor architecture 100
6.2 (a)16-Node Mesh Topology(16-nodes and 16-routers), (b) parameterized router architecture and (c) 16-Node ML-Mesh topology(16-nodes and 4-routers) 101
6.3 Architecture of the Parameterized VC based Input buffer employed in designing the router(Virtual Channel, Input Buffer Depth, Width can be configurable) 102
6.4 Packet generator and receptor 104

6.5 The configurable Flit Structure. 32 and 64-bit flit have been used in this work. We show the example of only 32-bit Flit Structure, each field in Flit Structure can been configurable according the size of Topology, Packets and Number of virtual channels 105
6.6 Clock frequency performance for Mesh and ML-Mesh topologies . . . 111
6.7 Critical path delay for Mesh and ML-Mesh topologies 112
6.8 Performance comparison of 16-Node ML-Mesh and Mesh NoC topologies under (a)Uniform Random, (b)Transpose, (c)Random Permutation, (d)Bit Complement, (e)Nearest neighbor and (f)Bit Shuffle Traffic Patterns. 113
6.9 Saturation Throughput comparison between 16-node Mesh, ML-Mesh topologies with network configurations of 2 VCs and BD of 8 115
6.10 Saturation Throughput comparison between 16-node Mesh, ML-Mesh topologies with network configurations of 4 VCs and BD of 8 115
6.11 Performance comparison of P-NoC and CONNECT topologies under (a)Uniform Random, (b)Transpose. 118

7.1 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 125
7.2 Two clock cycle Low latency router architecture implemented in LB-NoC framework(The router is highly parameterized with combined VC and Switch allocation stages) 126
7.3 The architecture of Input buffer employed in designing low latency router 127
7.4 Free VC availability check and count 128
7.5 Request filter logic 129
7.6 Parallel Virtual Channel and Switch allocator 130
7.7 Pipeline stages of conventional and LBNoC router architecture 132
7.8 Proposed adaptive look-ahead routing module 134
7.9 Performance comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns. 141

7.10 Throughput comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns. 142
7.11 Average packet latency comparison between LBNoC, CONNECT (Papamichael and Hoe (2015)) and (ProNoCMonemi et al. (2017)) considering different types of traffic patterns 145
7.12 Throughput comparison of LBNoC, Pronoc and Connect NoC architecture 146
7.13 Area, Frequency and Power utilization of various router architectures . . 148
7.14 Average packet latency comparison between LBNoC, SOTA(Group. (2012)), Shared-buffer (Soteriou et al. (2009)) and PCA (Yan et al. (2015)) considering different types of traffic patterns 149
7.15 Throughput comparison of LBNoC, SOTA, Shared-buffer and PCA NoC architectures 150

LIST OF TABLES

2.1 Comparison of the proposed and the other FPGA based NoC simulators 21

3.1 Experimental Setup Details 30

3.2 Different L1 Instruction(I1), L1 Data (D1) and Last Level (LL) Cache Configurations Used In Experiments 30

3.3 Effect on misses due to various I1 And D1 cache configurations 33

3.4 Effect on misses due to various Last Level (LL) cache configurations . . 36

3.5 Analysis Of Miss Rates Of Hotspot Methods In Booksim2.0 37

3.6 Unused functions in Booksim2.0 source code. 39

3.7 Replacing Post-Increment Operator By Pre-Increment Operator 41

3.8 Identifying the memory access pattern of Booksim2.0 source code . . . 42

4.1 Configurable Router Architectural Parameters 48

4.2 Experimental Setup Details 65

4.3 Resource utilization of 6×6 (36 node)Mesh and Torus topologies under various configurations of Flit Width(FW) and Buffer Depth (BD) 65

4.4 Resource utilization of 8×8 (64 node)Mesh and Torus topologies under various configurations of Flit Width (FW) and Buffer Depth (BD) . . . 66

4.5 Resource utilization of 56 node Fat tree topology under various configurations of Flit Width (FW) and Buffer Depth (BD) 67

4.6 Resource utilization of a Single Router 68

4.7 LUT Utilization of 5 and 9 Port Router Components 68

4.8 Synthesis results of YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) 69

4.9 Synthesis results of 36-Node Mesh based Topology on Artix-7 FPGA device (XC7A100T, speed-3) 72

4.10 Area utilization of 4×4 Mesh Topology on Artix-7 FPGA device (XC7A100T, speed-3) 74

4.11 SPF comparison of the proposed router with other faults tolerant router designs. 76

4.12 Resource utilization of CONNECT and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for 6×6 Mesh and DMesh topologies . . . 77

4.13 Resource utilization of DART and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for 3×3 Mesh topology 77

5.1 4:1 Multiplexer operating signals based on the grant signals from the Arbiter . 84

5.2 DSP48E1-I slice configuration based on the Arbiter Encoded Signal 84

5.3 DSP48E1-II slice configuration based on the Arbiter Encoded Signal 84

5.4 Experimental Setup Details 86

5.5 Resource utilization of the DSP48E1 and CLB based Crossbar implementation ONLY on the Artix 7 (XC7A100T) board 87

5.6 Resource utilization of 6×6 and 8×8 Mesh topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with XY routing 88

5.7 Resource utilization of 6×6 and 8×8 Mesh topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with LA routing 89

5.8 Resource utilization of 6×6 and 8×8 Torus topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with XY routing 90

5.9 Resource utilization of 6×6 and 8×8 Torus topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with LA routing 92

5.10 FPGA synthesis results of the 6×6 Mesh topology considering the proposed DSP48E1 crossbar implementation and CONNECT's implementation on Artix 7 (XC7A100T) board 96
5.11 Hardware utilization results of the 3×3 Mesh topology with proposed DSP48E1 based crossbar and DART's implementation on Artix 7 (XC7A100T) FPGA 97

6.1 Flit structure employed in the experiments. All the fields are configurable. 32-bit flit structure has been employed in this work for reference. 106
6.2 Experimental Setup Details 109
6.3 Synthesis results of 16-node Mesh and ML-Mesh topologies with FW 32 bits, Virtual channels 2,4VCs and BD of 2 to 32 on Artix-7 FPGA board 110
6.4 Synthesis results of 16-node Mesh and ML-Mesh topologies with FW 64 bits, Virtual Channels 2,4VCs and BD of 2 to 32 on Artix-7 FPGA board 110
6.5 Average Hop count of Mesh and ML-Mesh topologies 116
6.6 Power analysis of Mesh and ML-Mesh topologies configurations with(FW 32 bits, VCs 2 to 4 and BD of 2 to 32) on Artix-7 FPGA board 117
6.7 Synthesis results of 16-node Mesh and ML-Mesh topologies on Artix-7 FPGA board 117

7.1 The conventional allocator. V and P represent number of VCs per port and number of ports 128
7.2 The proposed parallel allocator. V and P denotes number of VCs per port and number of ports 131
7.3 Experimental Setup Details 136
7.4 FPGA memory buffers using three implementation alternatives with constant flit width of 32-bit. 137
7.5 FPGA memory buffers using three implementation alternatives with constant buffer depth of 15 flits. 137

7.6 Synthesis results of various configurations of Input buffer in LBNoC router with 64-bit flit width 137
7.7 Synthesis results of various configurations of Input buffer in LBNoC router with 128-bit of flit width 138
7.8 Synthesis results of merged FIFO buffers at each input port and Conventional FIFO buffers 138
7.9 Synthesis results of Queue of free VCs selection and Conventional VC allocator implementation 139
7.10 Synthesis results of Full and Decomposed Crossbar with IN/OUT ports 139
7.11 Synthesis results of Mesh topology of size 4×4 and 5×5 with various configuration of input parameters 139
7.12 Resource utilization and Maximum operating frequency of Different NoC configurations considering 4×4 Mesh topology 144
7.13 Resource utilization and Maximum operating frequency of Different NoC configurations considering 4×4 Mesh topology 148

LIST OF ABBREVIATIONS

Abbreviations	Expansion
ASIC	Application Specific Integrated Circuit
AMBA	Advanced Microcontroller Bus Architecture
AXI	Advanced eXtensible Interface
CMesh	Concentrated Mesh
CMP	Chip Multi-Processor
CMOS	Complementary Metal Oxide Semiconductor
CPI	Cycles per Instruction
DOR	Dimension Order Routing
FF	Flip Flop
FLIT	FLow control digIT
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
NI	Network Interface
LT	Link Traversal
NoC	Network-on-Chip
PE	Processing Element
RC	Route Computation
RR	Round-Robin
RTL	Register-transfer level

Abbreviations	Expansion
SA	Switch Allocation
SoC	System-on-Chip
ST	Switch Traversal
TDM	Time Division Multiplexing
TG	Traffic Generator
TR	Traffic Receptor
VA	Virtual-Channel Allocation
VC	Virtual Channels

CHAPTER 1

INTRODUCTION

A Systems-on-Chip(SoC) is a tightly coupled system containing general purpose processing units, graphics processing units, memory blocks, peripheral controllers, and accelerators fabricated on a single die for better power and performance characteristics. SoC components have to be interconnected efficiently to achieve the best possible power and performance tradeoff. SoCs use dedicated point-to-point and traditional buses as interconnection systems for interconnecting the various cores. However, a dedicated point-to-point and traditional buses suffer from scalability, unpredictable delays, noise issues, and efficient power consumption (Ho et al. (2001)).

The Network-on-Chip has become the de facto on-chip interconnection technique (Dally and Towles (2001); Benini and De Micheli (2002)) for the modern many-cores SoCs. A NoC is composed of routers and links to interconnect components on the chip. The NoC yields to a modular design, which is easier to verify and fabricate. NoCs provide high performance while servicing high bandwidth with better on-chip resource utilization. Hence, NoCs are the favoured on-chip communication framework in many of the state-of-art MPSoCs, CMP, heterogeneous systems and Domain specific hardware accelerators (Akopyan et al. (2015); Sodani et al. (2016); Balkind et al. (2016); Bohnenstiehl et al. (2017); Chen et al. (2017); Luo et al. (2017); Ax et al. (2018); Joardar et al. (2019); Jang et al. (2019)).

With the increase in the number of interconnected components, the performance of the target application becomes highly dependent on the performance of NoC. There is a

need to model and evaluate large size NoC designs quickly and accurately. NoC simulation frameworks explore the performance characteristics along with the effect on the overall system. System architects will be able to understand the impact of various design parameters before chip fabrication thereby reducing total cost. Academic and Industrial NoC designers have traditionally used cycle-accurate software based simulators((viz. Orion Kahng et al. (2012); Garnet Agarwal et al. (2009); SICOSYS Puente et al. (2002); Noxim Catania et al. (2015); Booksim Jiang et al. (2013)) to explore the microarchitectural design space of on-chip networks. Orion (Kahng et al. (2012)) provides a set of architectural power models for on-chip interconnection routers. On-chip network area power consumption and can be estimated accurately in the early phases. The details such as input buffers, routing logic, crossbar switch and allocators are implemented in Garnet (Agarwal et al. (2009)). Noxim Catania et al. (2015), is another NoC simulator which is implemented in SystemC. SICOSYS (Puente et al. (2002)) is a general-purpose interconnection network simulator that allows to model a wide variety of routers in a precise way. The parameters such as traffic pattern, applied load, message length etc., can be provided as input for simulation. Booksim2.0 (Jiang et al. (2013)) is a cycle-accurate simulator. It is flexible in terms of modeling network components.

Computer system simulators implemented in software provide an accurate and easy prediction of power and performance characteristics and are relatively easy to modify. However, systems have been continuously increasing in complexity and it is reflected in the rapid 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. From Fig. 1.1, it can be observed that execution time of BookSim2.0 (Jiang et al. (2013)) software based simulator varies as topology size increases from 6 seconds to 10 days simulating 4×4 to 54×54 NoC architectures of mesh topology on a Core i7-4770 CPU. By increasing the dimension from two to three and four, the time taken will be even more. This situation is called as the *simulation wall* (Angepat et al. (2014)). The fast modeling of large scale NoCs have been done through analytical models (Li-Shiuan Peh and Dally (2001); Suboh et al. (2010);

Ogras et al. (2010)) in many cases but significant inaccuracy in the results. The fast

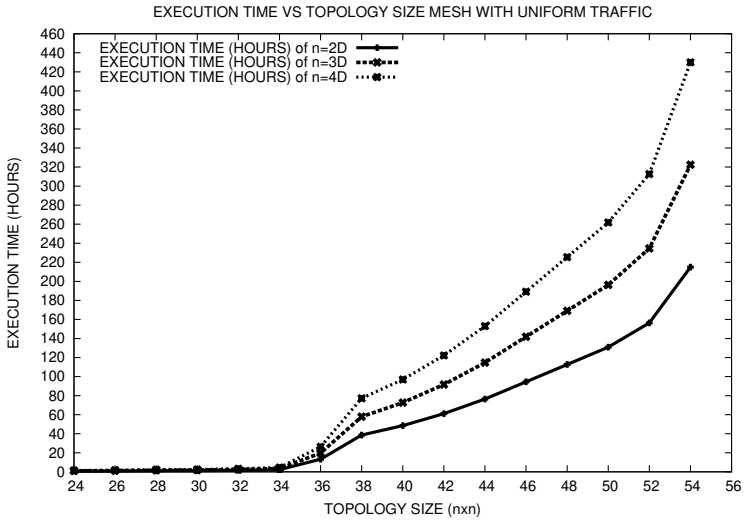


Figure 1.1: Booksim2.0 Execution Time Of k-ary n-dimensional Mesh Networks(n=2,3 and 4) Parane et al. (2016).

and precise 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.

The main objective of this thesis work is to improve the performance of modern computer systems by use of extensive hardware-level parallelism. A programmable, reconfigurable, hardware accelerator has the potential to provide an efficient alternative for the improvement of system simulation performance. Field Programmable Gate Arrays(FPGAs) contain an array of configurable logic blocks with programmable interconnects. The logic blocks can be configured to perform any arbitrary functions. The logic block inputs and outputs are connected by employing the programmable interconnection to form complex circuits(Ciletti (2011)). FPGAs have a short period of time to market and offer reconfigurability as compared to application-specific circuits (ASICs) in the design and evaluation of digital systems. Wolkotte (Wolkotte et al. (2007)) was an FPGA based simulator allow performance/area trade-offs by virtualizing a single router on an FPGA. In (Lotlikar et al. (2011)), an NoC emulation environment on FPGA called AcENoCs has been proposed. Both of the software and hardware components of FP-

GAs have been utilized by AcENoCs. A synthesizable NoC RTL generator on FPGA called CONNECT in (Papamichael and Hoe (2015)). AdapNoC, a fast and flexible FPGA-based NoC simulator in (Kamali and Hessabi (2016)). The router microarchitectural parameters are reconfigurable in AdapNoC. All these FPGA based simulators (Wolkotte et al. (2007); Lotlikar et al. (2011); Papamichael (2011); Papamichael and Hoe (2015); Kamali and Hessabi (2016)) have been proposed to improve the performance of the simulations. All the system events execute in parallel in an FPGA during a system simulation. Component models from the software simulators are effectively parallelized and executed on the FPGA to achieve a better simulation speed without compromising accuracy.

1.1 PROBLEM DESCRIPTION

In systems with a large number of components, bus based communication will not be efficient in terms of scalability, efficiency, and performance. The Network-on-Chip (NoC) has become a tangible on-chip communication technique. There is a need to model and evaluate large scale NoC designs fast and accurately in order to explore the performance characteristics along with the effect on overall system. NoC researchers have relied on cycle accurate power and performance software simulators to explore the microarchitectural design space. The NoC parameters such as topology, routing algorithm, flow control, and router microarchitecture, including buffer management and allocation schemes can be analyzed using these simulators. The simulation of large NoC architecture takes too much time. The slow speed of NoC simulators is a significant bottleneck in the study and analysis of interconnection architecture for next generation computing systems.

1.2 RESEARCH OBJECTIVES

The significant capabilities and the flexibility of FPGAs make them an ideal vehicle for accelerating and addressing the challenges of NoC architecture simulation.
Following aspects have been identified as the research objectives:

1. Review and performance optimization of NoC architecture simulators.

- Study and performance enhancement of software NoC simulator.
- Comparison and analysis of improved software simulator with existing hardware based NoC simulator.

2. Implementation and optimization of scalable FPGA based accelerator for NoC architecture simulator.
    - The efficient mapping techniques are employing for effectively utilize the dedicated FPGA resources.
    - Design of NoC router to optimize area with an improved network performance.
    - FPGA based hardware/software codesign of NoC simulation.

1.3 THESIS CONTRIBUTIONS

This thesis presents several key ideas to implement NoC architecture in FPGA platform. The contributions of this thesis with a brief summary are as follows:

1. To speedup the simulations, it is necessary to investigate and optimize the hotspots in the simulator source code. Among several software based simulators available, Booksim2.0 (Jiang et al. (2013)) has been chosen for the experimentation as it is being extensively used in the NoC community. We analyze and optimize the cache and memory system behaviour of Booksim2.0 to accurately monitor input dependent performance bottlenecks. We also employ the thread parallelization and vectorization to improve the overall performance of Booksim2.0.

2. The parallelization of software based simulator has high synchronization overhead, this reduces the speed of simulation, to overcome this we present an FPGA based NoC simulation framework-YaNoC. It supports the creation of standard and custom topologies, design of routing algorithms, generation of various synthetic traffic patterns, and exploration of a full set microarchitectural parameters. The efficient routing algorithm is required to reduce the congestion in the network and proper usage of the communication bandwidth. Also, the reliable NoC architecture require for tolerate the fault that causes the serious issues such as deadlock, packet loss, increased packet latency, erroneous messages in the networks. All of which result in on-chip performance degradation. We present an Adaptive and Reliable Network on Chip router architecture using FPGA to improve the performance of NoC architecture.

3. To build the large NoC architecture on FPGA, various components of the FPGA have to be utilized to their full extent to achieve the high performance. The Soft logic (CLBs) of the FPGAs are used implicitly to implement the design. However, the final mapping of the design on the FPGAs is in the control of backend tools. This results in inefficient use of the high performance Hard blocks of FPGAs such as DSP slices and BRAMs. The DSP slices of the modern Xilinx FPGAs offers the feature of dynamic reconfiguration in which a single DSP slice can be used

for various computations in each clock cycle.

4. Next, an FPGA-based parameterized framework for analyzing the performance of NoC architectures based on various design decision parameters has been proposed. An investigation on the key trade-offs in the organization of the router, including the design of the router virtual channel, input/output ports and evaluate the overhead and performance of NoC architecture has been presented.

5. Finally, a low latency router with the look-ahead bypass(LBNoC), with reduced area, latency and improved performance has been presented. Techniques such as single cycle router bypass, parallel virtual channel and switch allocation, combined virtual cut through and wormhole switching have been employed in the design of the LBNoC router.

1.4 THESIS ORGANIZATION

The organization of the thesis is as follows:

Chapter 1: In this chapter, The research work objectives are listed. A glimpse of the research contributions are also given.

Chapter 2: This chapter introduces some basic concepts about the on-chip network and Field Programmable gate array(FPGA). A pertinent related work in which state-of-art and recent development of software and FPGA based NoC simulator are introduced. This culminates into motivation for the present research work.

Chapter 3: Presents the analysis of cache and memory system behavior of Booksim2.0 software based NoC simulator. Also, the optimization techniques are employed for enhancing the performance of Booksim2.0 have been presented.

Chapter 4: In this chapter, a modular and configurable NoC simulation acceleration framework called YaNoC on FPGAs is presented. The novel adaptive and reliable router architecture has been proposed which is capable of detecting the congestion and faults in the NoC networks.

Chapter 5: Discusses the efficient mapping techniques to effectively utilize FPGA dedicated resources. The NoC router components such as crossbar and input buffer are mapped on the DSP48E1 and BRAM dedicated blocks of FPGA.

Chapter 6: This chapter describes the performance evaluation of NoCs architecture using P-NoC framework. The P-NoC framework consists of Topology, Router and Traffic modules. These modules support the implementation of NoCs architecture for design space exploration.

Chapter 7: In this chapter, an optimized, low cost, high performance NoC router microarchitecture is presented.

Chapter 8: Summarises the research work reported in the thesis.

CHAPTER 2

BACKGROUND AND REVIEW OF RELATED WORK

This chapter presents the fundamentals of NoCs with special focus on performance parameters. A general description of FPGAs is also provided. Finally, an overview of the existing software and FPGA based NoC simulators have been reviewed.

2.1 NETWORK-ON-CHIP: AN OVERVIEW

NoCs are the communication networks that connect multiprocessors on CMP and the system components in SoCs. NoC architecture is composed of three main building blocks, viz, links, routers, network adapter(NA) or network interface(NI) (Guerrier and Greiner (2000); Dally and Towles (2001); Benini and De Micheli (2002); Dally and Towles (2004)). Links are the communication fabric that physically connects nodes. The router implements the communication protocol. The router of NoC plays the role of a smart buffer in interconnection networks(Bjerregaard and Mahadevan (2006)). The router receives packets from the shared link and forwards the packet to its locally connected Processing elements(PEs) or to other shared links based on the node identification information in the packet header. The logical connection between the nodes and the network is through the network interface(NI). Usually a node has a unique policy or protocol to interface with the network. The topology, the configuration of the router microarchitectural components, and routing algorithms affect the performance of an NoC architecture.

All the major entities of an NoC are introduced below:

2.1.1 Data Packet

The packet is a group of network data, fixed and variable length packets available in the network. Packets are usually divided into several flow control units(flits). A flit size is same as the link width and is the largest amount of data. It is transferred in parallel by allocating the small amount of resources.

2.1.2 Routing Algorithms

Typically, there are multiple possible paths in an NoC between every source and destination pair. The selection of the best path is done by deterministic, oblivious or adaptive routing algorithms (Enright and Peh (2009)). The packet always uses the same path between source and destination nodes in the deterministic routing. The DoR XY and source routing are common deterministic routing algorithm. Oblivious routing determine the path between each pair of nodes, without being aware of the traffic condition in the network. Oblivious algorithms are easy to design and make deadlock free. In the adaptive routing, if the local link or original path is congested, the routing algorithm determines alternative paths.

2.1.3 Flow Control

The resource allocation and the resolution of conflicts can be done through flow control. It 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. Various kinds of flow control mechanisms are in use, viz., store-and-forward packet switching (Dally and Towles (2004)), virtual cut-through (Kermani and Kleinrock (1979)), and wormhole switching (Dally and Seitz (1986)). The Virtual Channels (VCs) mechanism has been implemented in (Dally (1992)) to prevent the deadlock and to achieve better performance.

2.1.4 Links

A link is a communication link, that is composed of a set of wires connecting two routers in the network. The link, also called a channel, is a group of wires connecting two entities. Typically, an NoC link has two physical channels making a full-duplex

connection between the routers.

2.1.5 Router Architecture

The NoC router is consists of a number of input/output ports, switching matrix connecting the input to output port, and a local port to access the processing element connected to this router. Fig. 2.1 depicts the microarchitecture of the router.

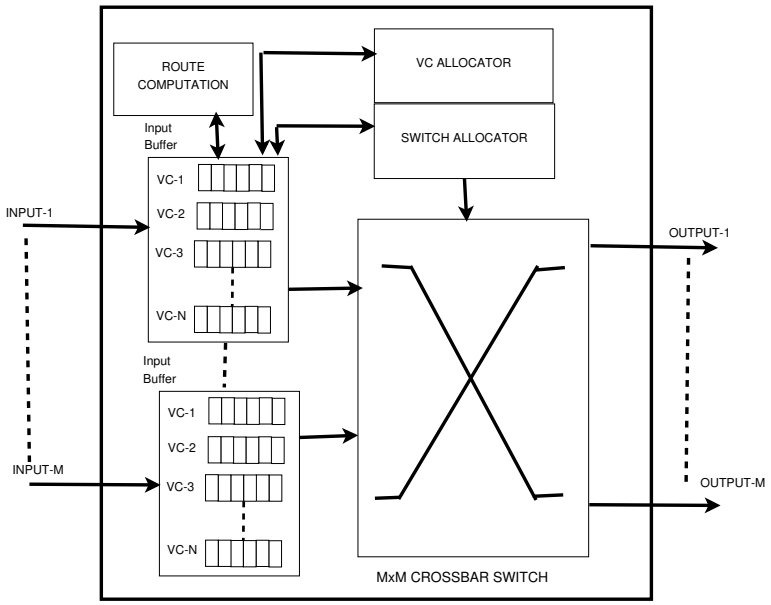


Figure 2.1: An NoC router microarchitecture with M input/output ports, N-virtual channels at each input port and M × M Crossbar switch (Enright and Peh (2009)).

2.1.5.1 Router Pipeline

The number of pipeline stages affects the latency and throughput of the router. Following are the pipeline stages of router architecture (Dally and Towles (2001, 2004); Partha Pratim Pande et al. (2005)).

Buffer Write: The router input ports have Input Buffers to store the incoming flits from the neighboring nodes, before transferring flit to the next nodes. To improve the performance and for avoiding the contentions in the NoC router, the virtual channel(VC) buffers are introduced. Each header flit contains a virtual channel identifier(VCID) to write flit in corresponding virtual channel buffer.

Route computation: The Route computation logic computes the correct output port and the set of candidate output VCs for the given incoming header flit according to the rout-

ing algorithm. This pipeline stage only needs to be performed for the header flit of each packet.

Virtual Channel allocation: The header flit of each packet arbitrates for the available output VCs corresponding to its output port. Similar to route computation, this pipeline stage only needs to be performed for the header flit of each packet.

Switch allocation: Once a header flit of each packet has obtained an output port and VC, it can proceeds to the switch allocation stage. Here, the header flits are arbitrated for accessing the output port.

Switch traversal: After getting the grant from the switch allocator, proceeds to the this stage, where the head flit traverses the crossbar.

Link traversal: Finally, the flit is travel to the next node in this stage.

2.1.6 Network Interface

A Network interface (NI) is the third building block of the NoC. The Processing Elements (PEs) and the NoC architecture are connected logically, employing the NI. Separation of computation and communication is done with the help of an NI.

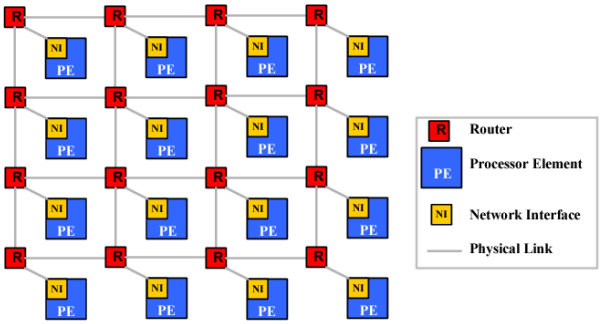


Figure 2.2: 4×4 NoC Mesh topology, Each PEs connects to a local port of router through NI, other ports of router connects to North, East, South and West neighbours using links (Partha Pratim Pande et al. (2005)).

2.1.7 Topology

The topology of an on-chip network determines the physical layout and connections between nodes and links in the network. The nodes(PEs core and router) in the NoC can be interconnected in various topologies such as Mesh-based, Tree-based, and user-specific architectures(Bjerregaard and Mahadevan (2006); Partha Pratim Pande et al. (2005);

Balfour and Dally (2006); Kim et al. (2007)). Fig. 2.2 depicts 4×4 2D Mesh NoC topology, where the PEs generate and receive the packtes, the routers are responsible for forwarding packets between the PEs and other neighbour routers using communication link.

2.2 NOC PERFORMANCE PARAMETERS

The performance of NoC architectures can be evaluated considering the standard set of metrics (Partha Pratim Pande et al. (2005); Enright and Peh (2009)).

2.2.1 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 equation 2.1

$$Avg_{lat} = 1/N \sum_{i=1}^{N} Li \tag{2.1}$$

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

2.2.2 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 2.2) :

$$TP = (\textit{Total Messages completed} * \textit{Message length})/(\textit{Number of PE blocks}*\textit{Total Time}) \tag{2.2}$$

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.

2.2.3 Area

In the NoC architecture design, the presence of the input buffers, Switch allocator, crossbar switch and the interfaces can result in the silicon area overhead. The area of

NoC architecture is given by equation 2.3 and 2.4

$$NoC_{Area} = Routers_{Area} + Links_{Area} \tag{2.3}$$

$$Router_{Area} = IB_{Area} + RCL_{Area} + Crossbar_{Area} \tag{2.4}$$

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

2.2.4 Power

The total power consumed by the NoC architecture can be breakdown 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 equation 2.5 and 2.6

$$P_{NoC} = P_{router} + P_{link} + P_{Interfaces} + P_{clk} \tag{2.5}$$

$$P_{router} = P_{FIFO} + P_{routelogic} + P_{allocator} + P_{crossbar} \tag{2.6}$$

2.3 FIELD PROGRAMMABLE GATE ARRAY

A brief overview of the FPGA architecture to highlight some key characteristics of that make it well suited to accelerate NoC simulations are presented. An FPGA is a massively parallel architecture that implements computation using a large number of configurable logic blocks connected to each other through a programmable interconnect fabric. An FPGA mainly contains:

- Configurable Logic blocks - which implement logic functions.
- Programmable Interconnects - which implement routing.
- Programmable Input/output(I/O) blocks - which connect with external components.
- Embedded hard blocks - which used to implement the specific functions.

Fig. 2.3 shows the generic architecture of an FPGA. The CLBs are interconnected through the programmable interconnect. The programmable I/O blocks are placed at

the borders of the grid to connect with external devices. 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.

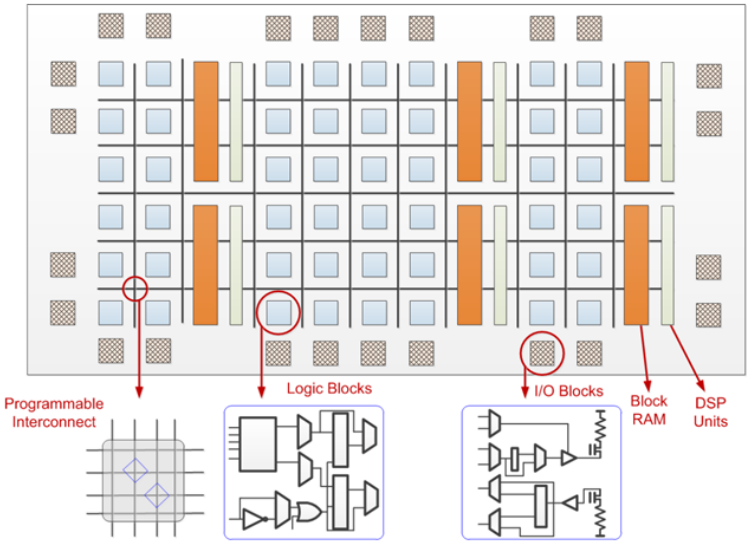


Figure 2.3: A generic architecture of Xilinx FPGA (Xilinx Inc (2019b))

2.3.1 Configurable Logic Blocks(CLB)

Any logic functions can be implemented using CLBs. The 7 series FPGA from Xilinx provides advanced, high-performance programmable logic. In the Xilinx 7 series FPGAs, each CLB contains a pair of slices which can work independently (Xilinx Inc (2016)). The slice in CLB may be SLICEL and SLICEM, which can be configured as logic and memory respectively. Each slice is made up of four Lookup Tables (LUTs), eight flip-flops, multiplexers and the carry logic. Various functions can be generated by combining LUTs through the multiplexers. Fig. 2.4 shows the structure of a CLB.

2.3.2 Programmable Interconnects

The programmable Interconnects provide the routing paths used to connect the inputs and output of I/O blocks and CLBs into logic networks. The various components of the FPGAs are connected through the use of programmable interconnects or routing fabric. The routing resources comprise segments of wire and consists of the switch modules

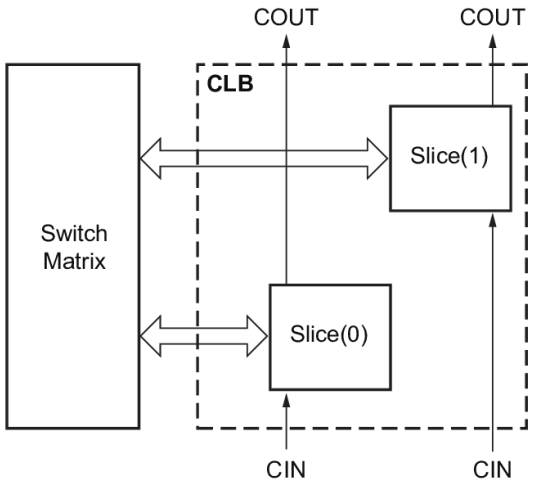


Figure 2.4: Architecture of the Configurable Logic Block(Xilinx Inc (2016))

and connection modules, which contain programmable switches (Chang et al. (1996)).

2.3.3 Embedded hard block

The embedded hard block of FPGAs are seen to be more area, power efficient and provide higher performance. These hard blocks are used to design the logic functions, which improves the performance and reduces the power consumption in comparison with the same functions designed 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 (2019a)) and DSP blocks (Xilinx Inc (2018)) are discussed in detail in Chapter 5.

2.4 RELATED WORK

The design space exploration of NoCs in modern communication centric designs need the fast simulator for exploration has resulted for the development of many architectural simulators. The architectural simulators are classified into software simulators and FPGA based emulators. This chapter provides an overview of state-of-art software based simulators and FPGA based emulators.

2.4.1 Software based Simulators

The performance evaluation of NoC architecture can be performed at different abstraction levels. The most of software simulators are designed in a high level abstraction.

The most important feature of software simulators is flexibility. They can be developed and modified simply. The software simulators are need to estimate the NoC area, performance and power consumption in the early stages of the design. These are the important design decision parameters in designing NoCs. Many of the full-system simulators such as Gem5 (Binkert et al. (2011)), Simplex(Hardavellas et al. (2004)) 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 (Agarwal et al. (2009)). GARNET has been integrated with (Binkert et al. (2011)) full system simulator. NoC microarchitectural 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.

Booksim2.0 (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 microarchitecture, are implemented. Noxim (Catania et al. (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.

2.4.2 FPGAs based NoC simulators

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 virtual-

ized implementation has been proposed in (Papamichael (2011)). The NoCs to be emulated are directly implemented on the FPGA. A Time Division Multiplexing 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 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.

An FPGA emulation platform called Ultra-Fast has been proposed in (Thiem Van 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 microarchitectural parameters are config-

urable 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 8×8 can be emulated in (Drewes et al. (2017)).

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. Also, reliable NoC router architecture is designed and evaluated. Various hard blocks of FPGA, such as DSP48E1 and the BRAMs, are used efficiently to map the NoC router microarchitectural 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			Reliable Router	FPGA	HW/SW	FPGA components for mapping router components		
	Mesh based based	Tree based based	Cutsom	DoR	Table based based	Adaptive	No. of Ports Ports	VC	Pipeline stages stages				CLB	DSP	BRAM
AcENoCs (Lotlikar et al. (2011))	Yes	No	No	Yes	No	No	5	2	1	No	V5	RS232	Yes	No	No
Papamichael (Papamichael (2011))	Yes	No	No	Yes	No	No	4/8/12/16	2/4/8	1	No	V5	UART	Yes	No	No
Zhang (Zhang et al. (2013))	Yes	No	No	Yes	No	No	5	2/4	3+	No	V6	-	Yes	No	No
DART (Wang et al. (2014))	Yes	Yes	No	Yes	No	No	upto 8	upto 4	5	No	V6	PCIe	Yes	No	Yes
UltraNoC (Thiem Van et al. (2015))	Yes	No	No	Yes	No	No	5	1/2	4/5	No	V7	-	Yes	No	Yes
AdapNoC (Kamali and Hessabi (2016))	Yes	No	No	Yes	No	Yes	5	upto 4	4/5	No	V6	PCIe	Yes	No	Yes
Hoplite DSP (Kapre and Gray (2015))	Unidirectional Torus	No	No	Yes	No	No	3	Bufferless	-	No	V7	-	Yes	Yes	No
SRNoC (Xu et al. (2019))	Yes	Yes	No	Yes	No	Yes	5	upto 8	5	No	V7	PCIe	Yes	No	Yes
This thesis work	Yes	Yes	Yes	Yes	Yes	Yes	3/5/7/8/9	2/4/8/16	2/5	Yes	Artix 7 Zynq 7000	USB-UART	Yes	Yes	yes

CHAPTER 3

ANALYSIS OF CACHE BEHAVIOUR AND SOFTWARE OPTIMIZATIONS FOR FASTER ON-CHIP NETWORK SIMULATIONS

The Network-on-Chip(NoCs) architecture based on the packet switched mechanism emerge as the most promising interconnection architecture for the modern CMPs and MPSoCs (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 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 Puente et al. (2002), Booksim2.0 Jiang et al. (2013)) to explore the microarchitectural design space of on-chip networks. Amongst these, Booksim2.0 has emerged as one of the prominent NoC performance analysis tools.

The execution performance of the applications running on computer systems depends on the cache configuration and memory access. Modern architectures face an ever-widening gap between the memory speed and the processor speed. Using a small but fast memory such as a cache reduces the memory access delay when the requested address is already stored in the cache. To enhance the execution performance of the applications running on a computer, it is necessary to understand the cache and memory system behavior of applications. The main idea was to identifythe hurdles associated with the slowness of the BookSim2.0's execution speed. One of the major factors af-

fecting the speed of simulation would be the hardware configuration of the computer system on which the BookSim2.0 is being executed. Most importantly, the memory configurations of the system play a vital role in deciding the speed of execution. In this chapter, we analyze cache and memory system behaviour of Booksim2.0 considering various cache and memory configuration. We used Valgrind and Cachegrind tools for analysis of cache and memory system behavior of Booksim2.0 running on the computing system. These tools take the Booksim2.0 binary as input and output the cache misses, memory access patterns. Based on the analysis, we identify the best cache configuration for the Booksim2.0 application running on a computing system, which improves the execution performance. The hotspots are identified where the simulation spends most of its execution time. Further, thread parallelization and vectorization have been employed to improve the overall performance of Booksim2.0.

3.1 METHODOLOGY

Booksim2.0 offers network parameters such as topology, routing algorithm, flow control, and router microarchitecture, 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 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 Booksim2.0 varies as topology size increases from 6 seconds to 10 days simulating 4x4 and 54x54 NoC architectures of Mesh topology. By increasing the dimension from 2 to 3 and 4, the time taken will 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 the potential problems. Profilers can identify the Hotspots where the program spends most of its execution time and the memory access patterns. Profiling Booksim2.0 will reveal 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

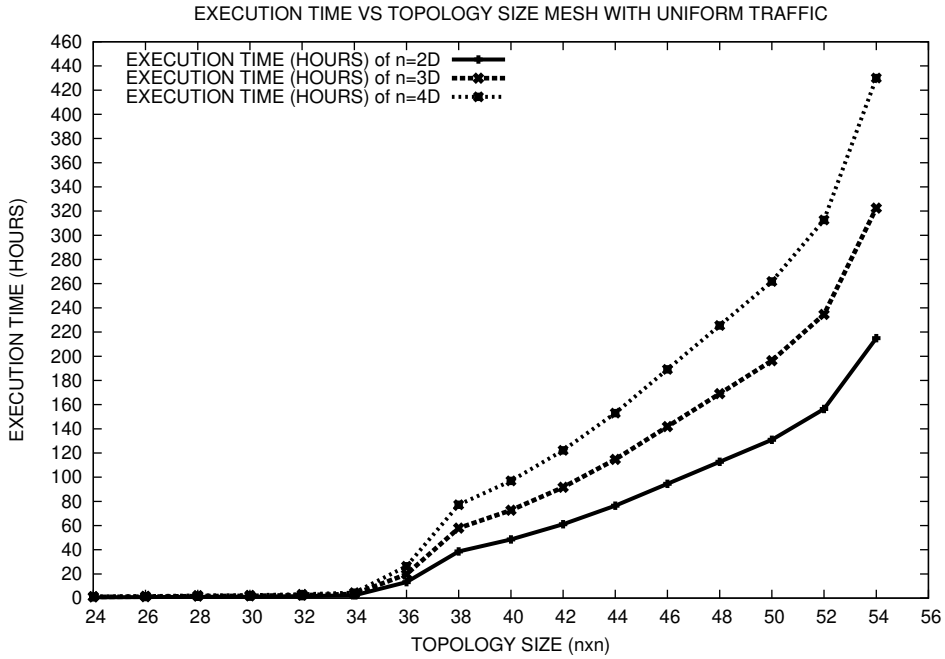


Figure 3.1: Booksim2.0 Execution Time Of k-ary n-dimensional Mesh Networks(n=2,3 and 4).

of Booksim2.0. Valgrind is a dynamic binary instrumentation framework for building dynamic binary analysis tools. Cachegrind tool of the Valgrind framework has been employed for profiling Booksim2.0. considering various cache configurations. Cachegrind simulates a system with independent first-level (L1) instruction and data caches(I1 and D1), backed by a unified last-level cache(LL). The Callgrind, another tool of Valgrind to record the call history of functions in Booksim2.0. KCachegrind, GUI based tool which is used for identifying the Hotspots of Booksim2.0

Based on profiling, the best cache configuration in which the cache misses are minimum and memory access patterns of Booksim2.0 which will help improve 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.

Booksim2.0 is a refactored version of Booksim's source code. When we performed the profiling to analyze the function callgraph of Booksim2.0, we observed that some of the functions being unused. The unused functions identified using profiling are never used in the execution of Mesh-based topologies of Booksim2.0. Unused functions and

variables affect performance since they occupy memory within a computer system. Hence, we removed the unused functions to improve the performance of Booksim2.0.

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.

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. 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 will improve the performance.

To reduce the execution time of Booksim2.0, optimization methodologies such as vectorization and thread parallelization are employed. These techniques have been employed to parallelize the portions of the Booksim2.0 source code. The most time-consuming loops were identified employing the Intel Advisor tool. Memory access patterns have an impact on improving the performance of an application.

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 applica-

tion is recorded as a function of input sizes to the routines of the application in the aprof tool. 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. (2010)) aims to reduce the cost of user-defined analysis tools in instrumentation by parallelizing dynamic program profiling in multi-core systems.

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 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 (Nguyen and Xu (2013)) run-time profiling tool 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 major 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 reports loops whose computation has repetitive and partially similar memory-access patterns across loop iterations. MemInsight (Jensen et al. (2015)) implements 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 will 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 opti-

mization 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 Booksim2.0. 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 Booksim2.0. KCachegrind - a GUI based tool is used for identifying the Hotspots of Booksim2.0. Employing these tools, the best cache configuration in which the cache misses are minimum is identified. Also, memory access patterns of Booksim2.0 which helps in improving the performance are identified.

Cppcheck (Daniel Marjamäki (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 Booksim2.0. Intel Advisor suite (Intel Corporation (2017)) has been employed to identify the top time-consuming loops of Booksim2.0. Based on these analyses, we employ the OpenMP programming model to parallelize the top time-consuming loops of Booksim2.0.

3.3.1 Experimental methodology

The cache design and Booksim2.0 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 Booksim2.0 considering various topology sizes. Based on these analyses, the best cache configuration with a minimum number of cache misses has been identi-

Table 3.1: Experimental Setup Details

System Configuration	
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
Network Configuration Input to Booksim2.0	
Topology Type	Mesh and Torus
Network size	4×4, 6×6, ..., 30×30
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

	L1 cache configuration			**LL cache configuration**		
Sl No.	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

fied. 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 Booksim2.0.

3.4 RESULTS AND DISCUSSION

3.4.1 Identifying the best cache configuration

The performance of cache memory has been studied with various cache and topology sizes. 12 different cache configurations have been 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. Booksim2.0 simulations were run for 2D Mesh topology of sizes ranging from 4×4, 6×6, ... , 30×30.

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 particular cache configuration. The values were obtained by averaging the MPKI of 14 experiments of Mesh topology from 4×4, 6×6, ... , 30×30 network size. All the other values were computed in similar way.

3.4.1.1 L1 instruction (I1) cache analysis

Effect of cache size on I1 cache misses: From Fig. 3.2, it can be seen that I1 cache misses were 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 were reduced by 22.47%.

As shown in Table 3.3, 2.73% to 45.24% reduction of misses were observed for all other cache configurations when I1 cache size has been increased from 32KB to 64KB.

Effect of associativity on I1 cache misses: From Fig. 3.2 it can be seen that when

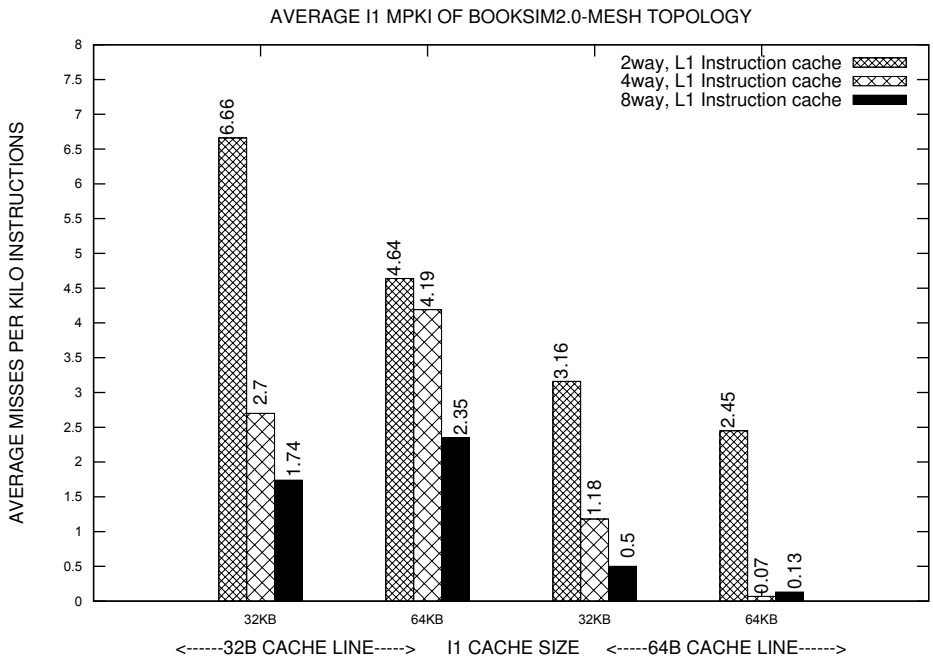


Figure 3.2: Average I1 MPKI of Booksim2.0 for Mesh topology.(MPKI is averaged over topology sizes mentioned in Table 3.1 And L1 cache configurations were varied as shown in Table 3.2)

the cache configurations are changed from 2-way to 4-way considering 32KB I1 cache with 32B line size, the misses were 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% was 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 were reduced by 30.33%. This is due to good spatial locality of reference.

Reduction of misses from 32.7% to 91.8% was observed for all 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

Reduction in I1 Misses		
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%
Reduction in D1 Misses		
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: Increase in the size of D1 cache in an incremental manner yields an incremental increase of D1 cache performance. From Fig. 3.3, for 2-way D1 cache with 32B line, when the configurations are changed from 32KB to 64KB cache, the misses were reduced by 4.18%. For 64B cache line, 7.78% reduction in misses was 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 were reduced by 2.12%. On changing the associativity from 2-way to 8-way, the misses were reduced by 0.77%.

From Table 3.3, 0.78% to 3.35% reduction of misses were observed for all 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.

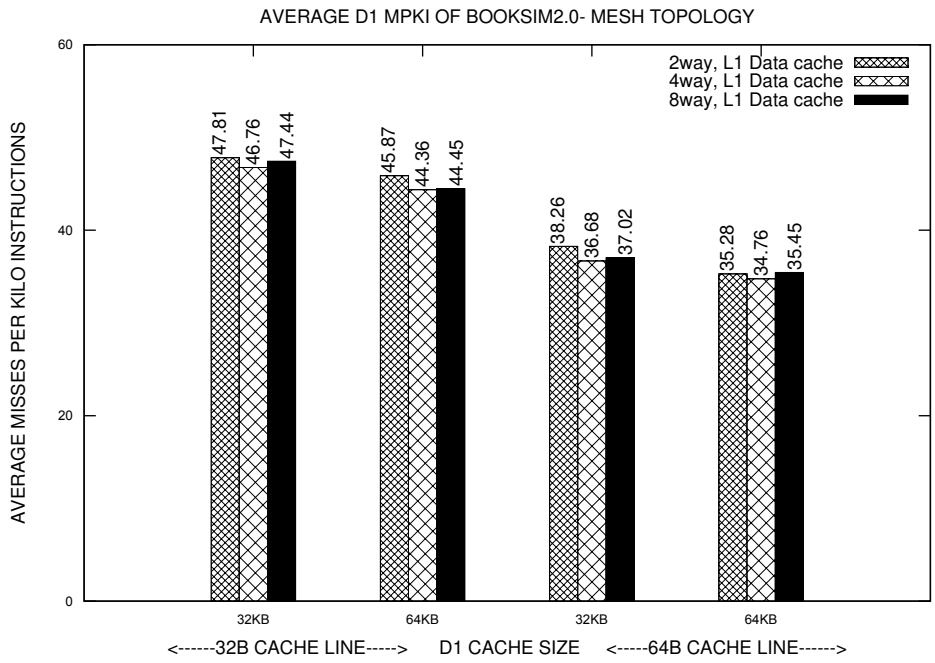


Figure 3.3: Average D1 MPKI of Booksim2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 And L1 cache configurations were varied as shown in Table 3.2)

Effect of cache 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% was 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 Booksim2.0 simulations.

3.4.1.3 Last level (LL) cache analysis

The last level cache size of 512KB, 4MB and 8MB were 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.

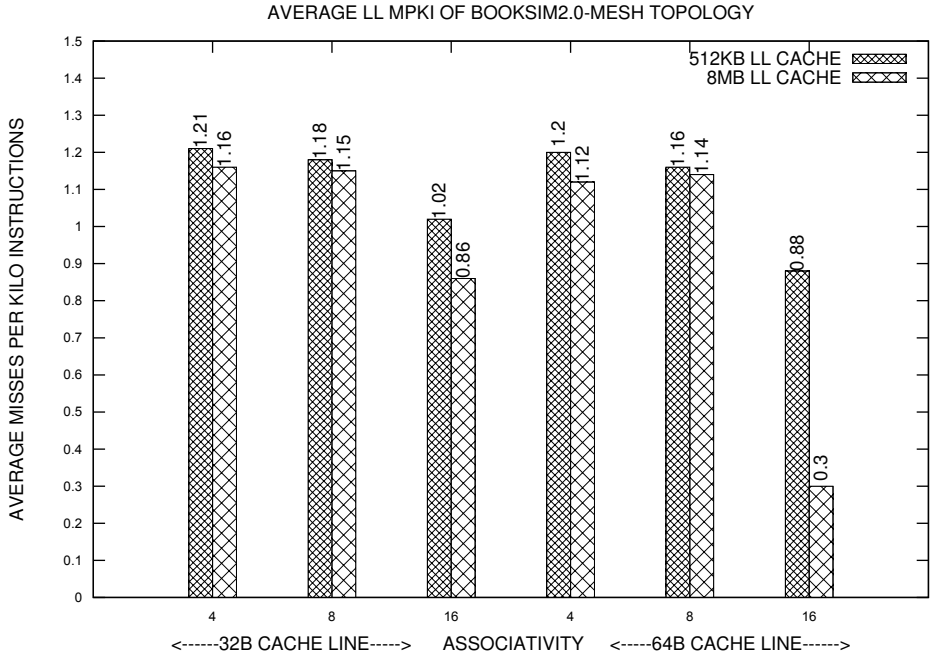


Figure 3.4: Average LL MPKI of Booksim2.0 for Mesh topology. (MPKI is averaged over topology sizes mentioned in Table 3.1 And LL cache configurations were varied as shown in Table 3.2)

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 were reduced by 4.13%. Similarly, for 64B cache line, 6.67% reduction in misses was 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 reduced by

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%

2.48%. On moving from 4-way to 16-way, the misses were reduced by 15.7%. Reduction of misses from 0.86% to 73.21% was observed for all the other cache configurations 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 were 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% was 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 larger cache line performs better than all other cache configurations. In our experiments, 16-way, 8MB LL cache with 64B line LL cache configuration is appropriate for running the Booksim2.0 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 Booksim2.0.

3.4.2 Hotspot and CPI analysis

The instruction references, L1(I1+D1) and LL cache misses for all the methods of Booksim2.0 were 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.

Table 3.5: Analysis Of Miss Rates Of Hotspot Methods In Booksim2.0

24×24 Mesh topology				
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%

The reduction of misses from 2.1% to 24.22% was observed on moving from 32KB+32KB L1 cache to 64KB+64KB L1 cache configuration for all the other hotspot methods of Booksim2.0 for topology size varied from 4×4 to 30×30.

CPI is one of the critical parameters to measure the performance of Booksim2.0 with 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 14x14 Mesh Topology. Employing the best cache configuration i.e., 4-way, 64KB+64KB L1 cache, 16-way 8MB LL cache with 64B line size, the CPI reduces to 1.39. Speedup of 4.1× is observed when the best cache configuration is used. For the smaller Mesh topology sizes, the cache misses will be lower as there will be less traffic generated. Hence, the speedup for the topology sizes 4×4 and 6×6 is in the range of 1.32× to 1.67×. As the topology size increases, the higher cache configuration yields the better performance. Hence, it can be observed from Fig. 3.5 that the speedup achieved for the higher topology sizes will be in the range of 2.66× to 4.04×. From these experiments, it is evident that increasing cache configuration improves the performance of Booksim2.0 simulations.

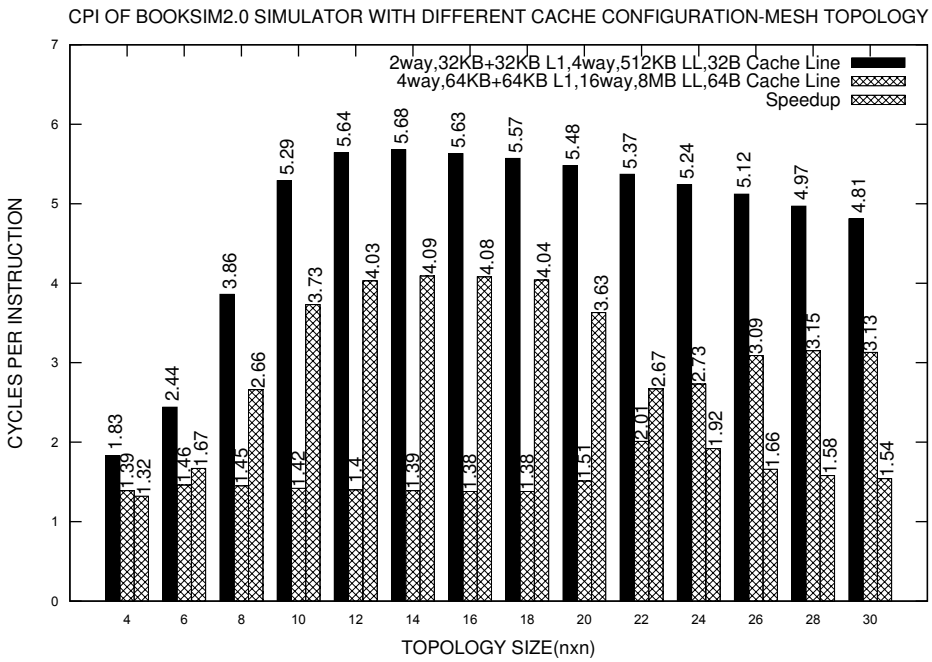


Figure 3.5: CPI For Booksim2.0 Running Various Sizes Of Mesh Topology

3.5 EXPERIMENTAL RESULTS BASED ON OPTIMIZATION STRATEGIES

In this section, the techniques which are used to optimize the cache misses have been explained. The best cache configuration for Booksim2.0 has been identified using Valgrind profiling tool. Further, the cache misses have been minimized and the performance of Booksim2.0 has been improved by considering 4-way, (64KB+64KB) L1 cache and 16-way, 8MB LL cache with 64B block size cache configuration.

3.5.1 Minimizing the cache misses and Performance analysis

The performance of Booksim2.0 has been improved by employing various optimization techniques such as the removal of unused functions, loop optimization, and pre-increment operator.

3.5.1.1 Removal of unused functions

Table 3.6 shows the list of unused functions in various source files of Booksim2.0.

These unused functions were removed from the source code. As shown in Fig. 3.6, the cache misses for 30×30 sized mesh network was 49.23M for the execution that contains the unused function. On removing the unused functions, the misses were

Table 3.6: Unused functions in Booksim2.0 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()

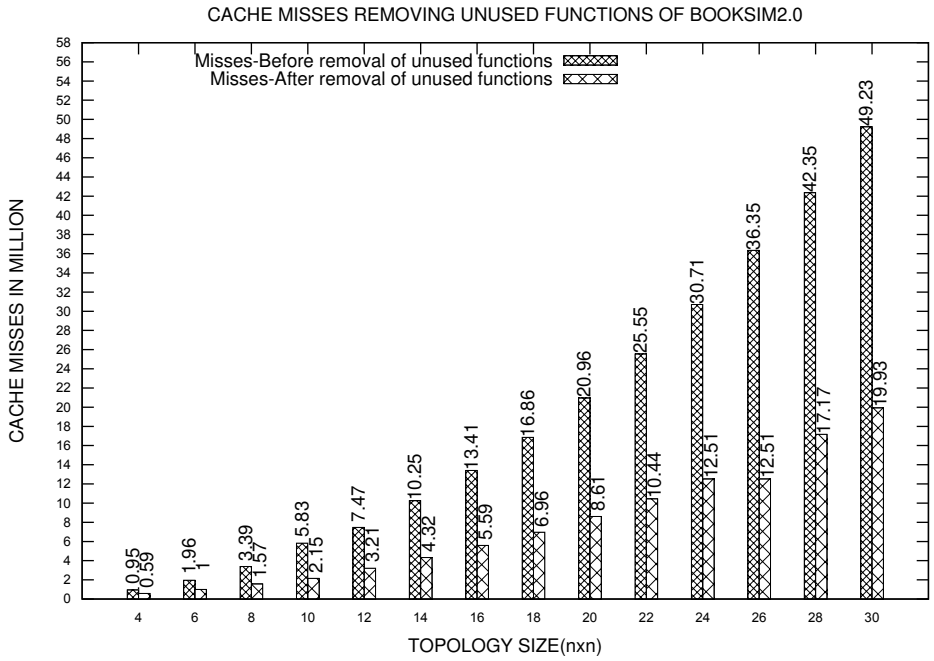


Figure 3.6: Cache misses before and after optimization

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

3.5.1.2 Loop optimization

The technique of loop reversal has been 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 2.47x was observed for 30×30

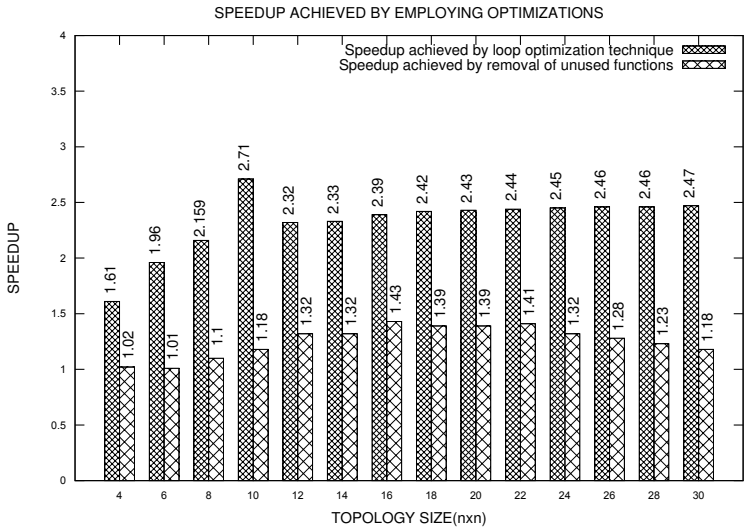


Figure 3.7: Speedup achieved before and after optimization

network topology. From 1.61x to 2.71x speedup has been observed for all the other topology sizes.

The below code snippet shows the technique that was 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

Cppcheck (Daniel Marjamäki (2011)), a static analysis tool has been used to identify the post-increment operators in the C++ source code of Booksim2.0.

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

Above command detects the post-increment operators in the source code of Booksim2.0. 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)
```

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 %

`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 Booksim2.0. As seen from the "Improvement" column of the table, the cache misses are reduced from 2.2% to 3.9%.

3.5.2 Improving the performance of Booksim2.0

Intel vectorization tool has been employed to identify the stride access patterns of Booksim2.0. 17% of memory instructions were unit stride, 34% of memory instructions were fixed non-unit stride and 49% of memory instructions were variable stride after annotating Singlesim method of TrafficManager class of Booksim2.0. Based on these observations, the performance of Booksim2.0 has been improved by changing unaligned memory accesses to aligned memory access. The compiler directive as shown below is inserted in the source files of Booksim2.0 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 Booksim2.0 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
		_RetireFlit	649
Variable stride	credit.cpp	Reset	47, 48,49
	credit.cpp	New	58
	network.cpp	WriteCredit	229
	new_allocator.h	construct	104
	stats.cpp	AddSample	114
	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 have been employed to parallelize and vectorize the most time-consuming portions of Booksim2.0. Execution times of sequential code with parallel code have been compared considering different network topology size of Booksim2.0 with Mesh topology as shown in Fig. 3.8. The speedup of 2.93x as shown in Fig. 3.9 has been achieved by parallelizing the sequential code of Booksim2.0 using OpenMP constructs considering 30×30 network size of Mesh topology. $1.07\times$ to $3.0\times$ speedup was observed for all the other sizes of Mesh topology.

Also, the SIMD construct was used with the OpenMP programming model to achieve fine-grain parallelization. By using SIMD with OpenMP model, the performance improvement of 3.97x was observed for 4×4 Mesh topology. And, speedup from 2.64x to 3.69x was observed for all the other sizes of Mesh topology. The overall traffic statis-

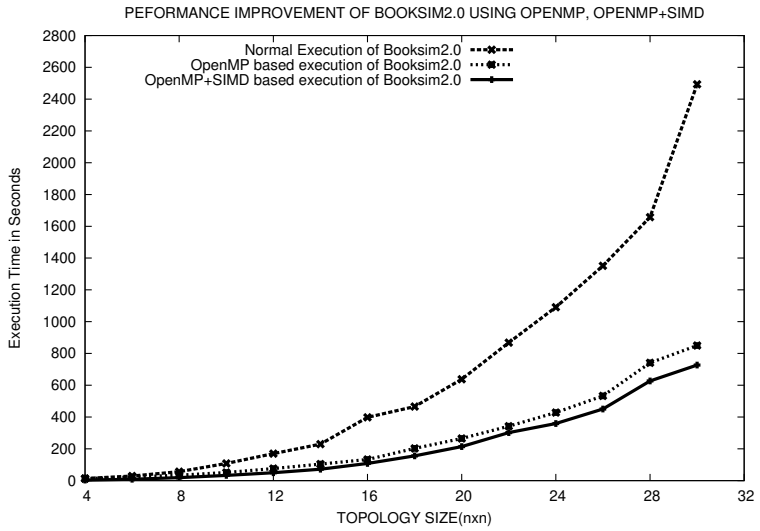


Figure 3.8: Simulation execution times before and after improvements

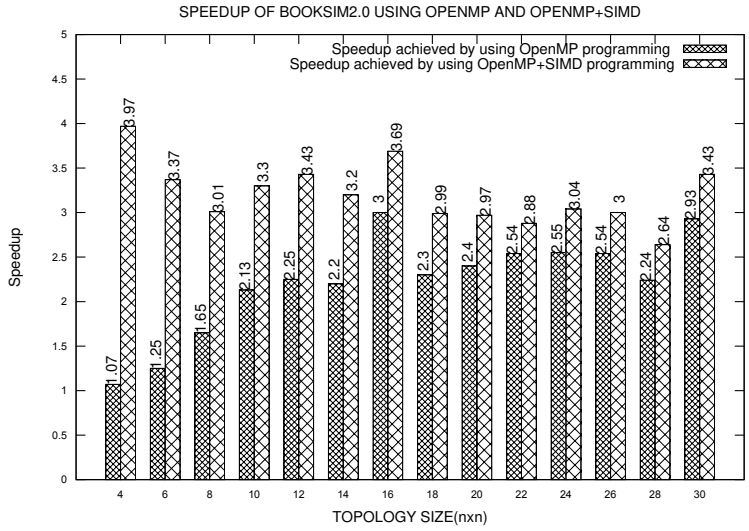


Figure 3.9: Speedups with Mesh topology of varying sizes

tics of Booksim2.0 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 30×30 Mesh topology from 60 to 14 minutes and 12 minutes, respectively.

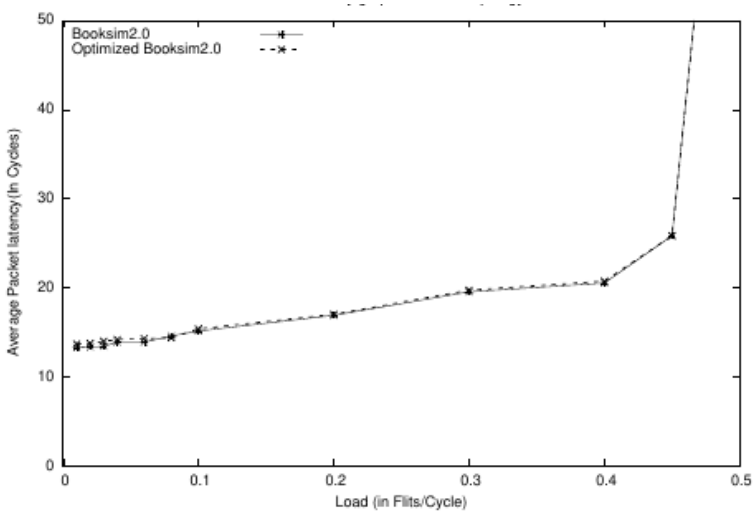


Figure 3.10: Average packet latency for Booksim2.0 and Optimized Booksim2.0 for a 3×3 Mesh topology

To verify the correctness of optimized Booksim2.0 against Booksim2.0. We simulate a 3×3 Mesh topology and compare the measured average packet latency reported by optimized and unoptimized Booksim2.0. Fig.3.10, shows the performance comparision of optimized Booksim2.0 and Booksim2.0 simulators. Overall, the simulations using Booksim2.0 and the optimized Booksim2.0 exhibits nearly identical performance.

3.6 SUMMARY

In this chapter, the profiling and software optimization strategies that can improve the performance of Booksim2.0 NoC simulator are discussed. The use of profiling to measure the performance, identify Hotspots and diagnose potential problems of Booksim2.0 simulators. The cache design and Booksim2.0 configuration parameters considered for the experiments. The Cachegrid tool of valgrind use to analyse the cache behaviour and memory usage of Booksim2.0 considering different topology size by configuring the cache design parameters. Hotspot methods of Booksim2.0 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 Booksim2.0 simulator. By using the Intel advisor tool, the top time-consuming loops in Booksim2.0 source code have been identified. The OpenMP programming model has been used to parallelize

the time consuming loops in the simulator.

The acceleration via FPGAs motivates researchers to implement FPGA based NoC simulators to provide better speedup and accuracy compared to the software simulators (Angepat et al. (2014)). An FPGA based NoC simulation acceleration framework is proposed in the subsequent chapters. The framework is capable of design space exploration of standard and custom NoC topologies considering a full set of microarchitectural parameters.

CHAPTER 4

YANOC - FPGA BASED SIMULATION ACCELERATION FRAMEWORK

The FPGA based NoC performance evaluation frameworks are important for early design simulation of the NoC architecture. Indeed, the FPGAs are quite good at exploiting parallelism and potential to run fast. An FPGA based NoC simulation acceleration framework has been proposed in this chapter. The framework support the design space exploration of standard and custom NoC topologies considering a full set of microarchitectural parameters. 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 Lookup tables (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.

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

The 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 traffic patterns, and exploration of a full set of microarchitectural 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 to YaNoC is 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 the design with flit width 32 bits and flit buffer depth to be 8 flits has to be evaluated, these parameters have to be specified in the configuration file. The Verilog code corresponding to this configuration will be generated by the Automated Verilog HDL Generator. 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. Au-

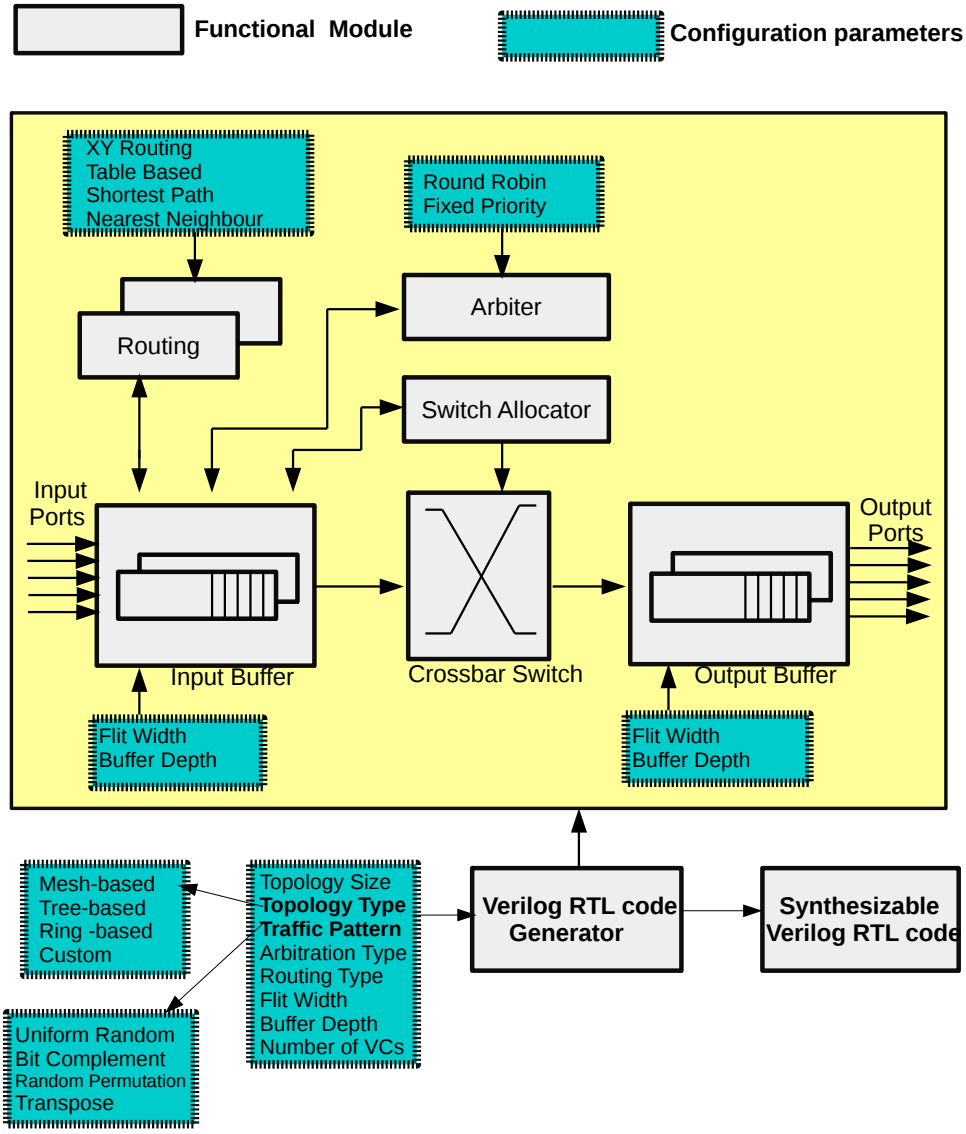


Figure 4.1: Architecture of the proposed YaNoC FPGA based NoC simulation acceleration framework

tomated 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 algorithm for Mesh topology, the parameters for routing algorithms can be modified accordingly.

4.2.2 Router Architecture

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

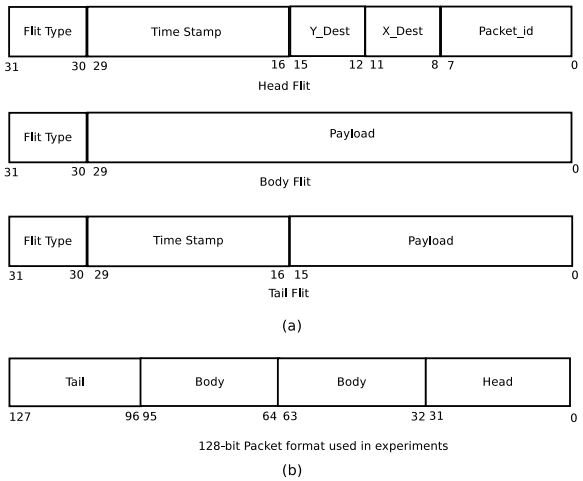


Figure 4.2: (a)Flit types and (b)Packet structure used in experiments. (Time stamp field is useful in calculating the latency of a packet)

4.2.2.1 Flit Buffer

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

4.2.2.2 Flit Structure

Flits of variable widths are supported by YaNoC. The structure of the head, body and tail flits have been shown in Fig. 4.2(a). 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. In order to calculate the latency of the network, the tail flit comprises of the 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 in between the neighboring cores and a remaining port is used to connect to the local processing

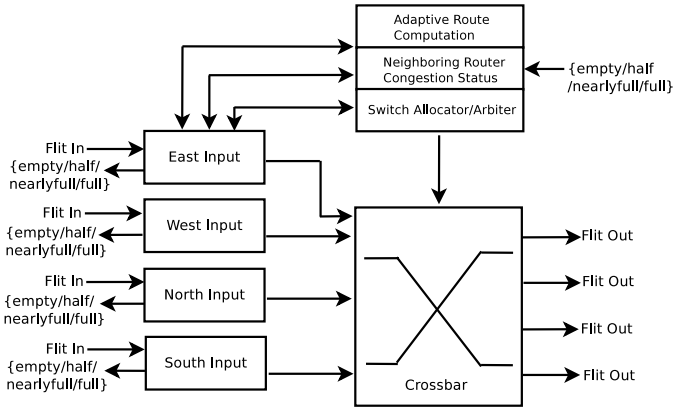


Figure 4.3: Modified Router architecture supporting Congestion-aware Adaptive routing

element of that core. Similarly, Mesh and DMesh topology have 5 and 9 ports for communication. The provision for variable ports has been provided to explore various custom topologies.

4.2.2.4 Routing Algorithms

The standard minimal routing algorithms (Dimension Order - XY) for conventional NoCs are supported. Table based routing has been implemented to support the creation of custom topologies. The output ports to all the destinations in the network are stored in the Look Up Tables (LUTs). The entries in routing LUTs will be large for large networks. Distributed RAMs (DRAM) of FPGA have been employed in the proposed architecture to implement the routing tables. A single DRAM will be typical of single-bit wide memory with 16-64 elements constrained to a specific FPGA family. As the entries in routing tables are maximum of 3 bits 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 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. Working of the proposed adaptive routing algorithm has been detailed in Section section 4.3.4.

4.2.2.5 Arbitration Schemes

To ensure the fairness in the allocation of resources, Round Robin and Priority based arbitration schemes have been implemented.

4.2.3 Router Datapath

Adjacent routers communicate with each other through input and output port interfaces depending on the configuration of the router. The port interfaces are bi-directional and can be used to connect other routers or to form network endpoints. The router microarchitecture is organized as a five-stage pipeline as shown in Fig. 4.4.

Buffer Write: In each clock cycle, flits enter the router through an input port and are stored in the flit buffers.

Route Computation: Routing logic monitors the incoming flits tagged with the destination address in order to forward the flits to a proper output port leading to the destination.

Switch Allocation: Once the route computation is completed by the routing logic, the flits will head for arbitration unit where they have to compete with the other flits to traverse through a particular outport port. The arbiter allocates a grant signal to incoming request based on round robin mechanism or priority of the flits.

Switch Traversal: After getting the grant from the arbiter, the head flit traverses to the crossbar unit. The crossbar unit maps the incoming head flit to the output port which was computed in route compute unit.

Link Traversal: Later, head flit traverses through the output port to next router in the network. The body and tail flits follow the route created by the header flit. It is made sure that the grant signal provided by arbiter is held until all the flits of the packet that is, body and tail flits are traversed from input port to output port.

Figure 4.4: 5 Stage Router pipeline

4.2.4 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 in each router. The LFSR modules generate the traffic according to the traffic pattern and the specified injection rate.

To calculate the latency of the network, 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.

4.2.5 Software Tools Supporting YaNoC

4.2.5.1 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.

4.2.5.2 Routing Table Generator

Along with the support for 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 containing the entries of the shortest path to every other node in the topology will be 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 will be populated depending on the selected configuration.

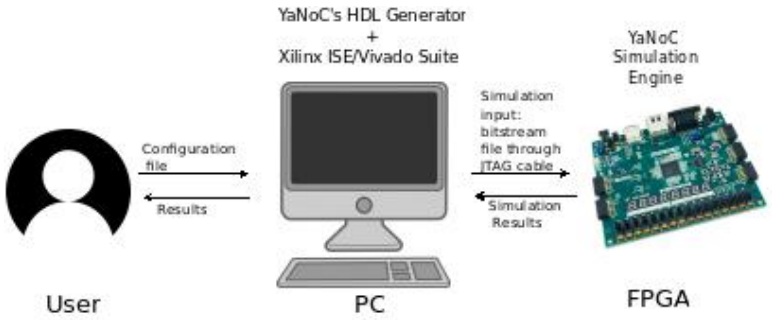


Figure 4.5: A High-level block diagram of YaNoC consisting of Host PC connected to an FPGA Board.

4.2.5.3 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. 4.5.

4.2.6 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 was thoroughly analyzed considering clock by clock transitions. The flit traversal through each pipeline stage (Fig. 4.4) was analyzed for the functional correctness. In the implementation phase, the HDL for required NoC design was generated with the help of Automated Verilog HDL Generator, and it was 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. 4.5). The following steps describe the flow of YaNoC framework shown in Fig. 4.6.

1. The configuration file is updated by the user to reflect the correct parameters of the NoC to be simulated.

2. Synthesizable Verilog HDL is generated by the YaNoC's automated HDL gener-

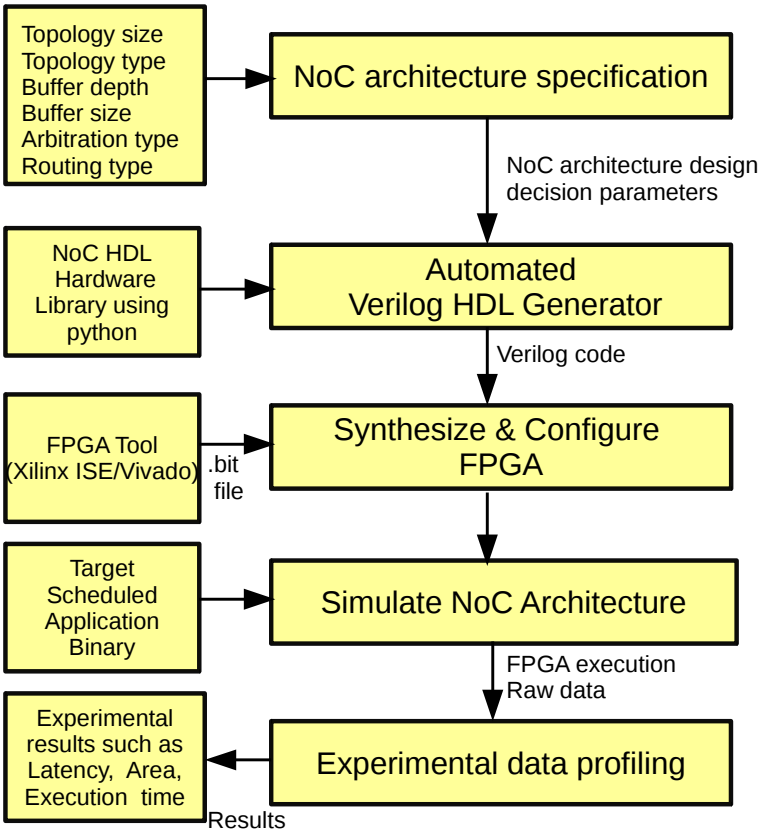


Figure 4.6: Simulation framework flow

ator. 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 the simulation are extracted from the portal developed for interaction with the FPGA through the 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. We simulate the two-dimensional (2D) mesh-based topologies, which are a popular choice for NoCs in tile-based architecture as they perfectly match the 2D silicon surface. The custom topology

called DMesh is designed by adding diagonal links to the Mesh topology, because of the emergence of X link architecture routing in chip manufacturing (Igarashi et al. (2002)). The custom topology design is possible because of the table based routing approach. Along with the support for Table based routing approach, the route computation template can be modified as per the user logic to route the packets in the network.

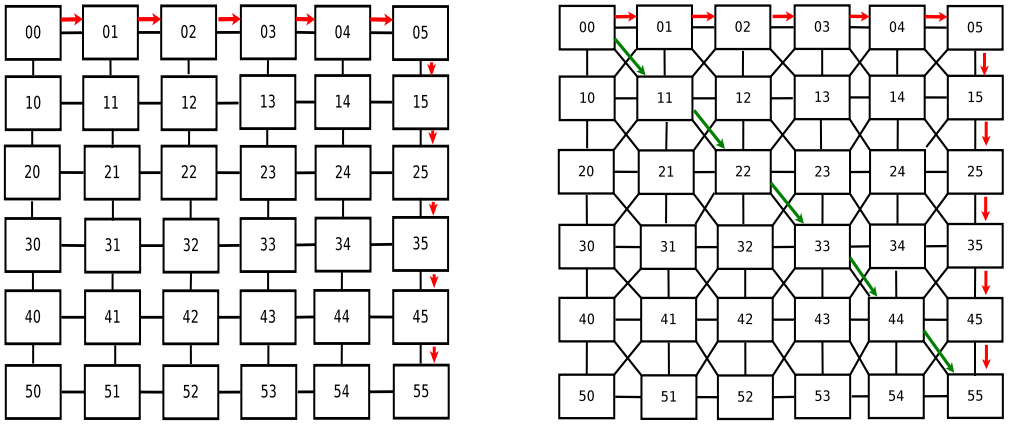
4.3.1 Design of DMesh topology in YaNoC

To design an application specific custom topology, interconnection in between nodes and routing tables along with the other router microarchitectural 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 Routing algorithm for DMesh topology in YaNoC

Table based routing can be 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 in appendix A.1. The route compute template has to be instantiated in the router module in order to route the packets. Fig. 4.7(a) and 4.7(b) show the 6×6 Mesh and DMesh topologies. The arrows in red of Fig. 4.7(a) indicate 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. 4.7(b) represent 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).



(a) 6x6 Mesh topology.

(b) 6x6 Diagonal Mesh topology.

Figure 4.7: Mesh and Diagonal Mesh topologies (Red and Green colors indicate the routes calculated by XY and novel shortest path XY routing algorithms)

4.3.2.1 Routing Tables

A routing table will be stored in each router. The routing table contains the route to all the other routers in the network. Below lines specify the routing table for a Router with ID "0" in a 2×2 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 Diagonal Mesh topology. The architecture of the Diagonal Mesh (DMesh) is shown in Fig. 4.7(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. 4.8 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
```

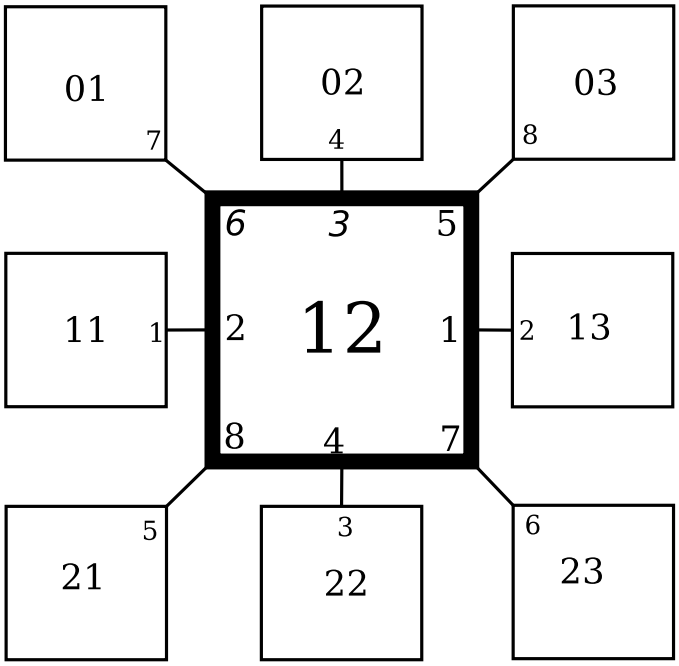


Figure 4.8: Interconnection of the Router 12 with other Routers in DMesh topology

```
#(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 Diagonal Mesh 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
#(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 6×6 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 will generate 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 The proposed Congestion-aware adaptive Routing algorithm

Employing the proposed adaptive routing algorithm, packets will be routed along the communication path which is minimally congested. In a given 2D Mesh topology, if the current and destination addresses are same ((xc=xd) and (yc=yd)), the flit has reached its destination and will be forwarded via the local port to the processing element. When the current and destination nodes are different ((xc!=xd) and (yc!=yd)) the flit will be 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. First priority is given to the West port if (xdiff<0) where xdiff is (xd-xc). In order to avoid the deadlock condition, second priority will be given to South port and the third priority will be shared among the East and the North ports. Similarly, when (xdiff>0), first priority is given to the East port. For the deadlock avoidance, the second priority will be given to North port and the third priority will be shared among the West and the South ports. The conditions discussed above are shown in the Equations 4.1 and 4.2.

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

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

The algorithm for calculating the priorities of all the ports of a router is shown in Algorithm 4.1.

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

Algorithm 4.1: Priority calculation for directions

Input : Coordinates of current node (xc,yc), destination node (xd,yd)
Output: Priority Matrix of all the ports P[E/W/N/S]

Begin
xdiff=xd-xc;
ydiff=yd-yc;
if $xdiff < 0$ **then**
  return $P[E/W/N/S] = \{3/1/3/2\}$;
else if $xdiff > 0 \&\& ydiff < 0$ **then**
  return $P[E/W/N/S] = \{1/2/2/3\}$;
else if $xdiff > 0 \&\& ydiff > 0$ **then**
  return $P[E/W/N/S] = \{2/3/3/1\}$;
else if $xdiff > 0$ **then**
  return $P[E/W/N/S] = \{1/3/2/3\}$;
else if $ydiff > 0$ **then**
  return $P[E/W/N/S] = \{2/2/3/1\}$;
else if $ydiff < 0$ **then**
  return $P[E/W/N/S] = \{2/2/1/3\}$;
End

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 in order 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 ϵ {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 calculation of the weight matrix. The port with minimal weight value will be chosen as the final output port. The adaptive routing algorithm is shown in Algorithm 4.2.

$$W[i] = P[i] + W_{empty} + W_{halffull} + W_{nearlyfull} + W_{full} \tag{4.3}$$

The current status information of the (i+1)th router is taken into consideration at the ith router in order to achieve the adaptiveness in the NoC router architecture. "Neighboring Router Congestion Status" monitors the congestion status of all neighboring router's ports. The "Adaptive Route Computation" unit calculates the priority for the ports in all direction as discussed above.

4.3.4.1 Deadlock and livelock avoidance

To ensure deadlock freedom, several assumptions have been defined for deadlock avoidance. A node can not send a packet to itself, a packet is ejected when it reached to the destination node, the source and destination nodes are in a connected region. The virtual channels(VCs) are employed in the YaNoC router architecture to avoid deadlock. The incoming packets are temporarily stored in virtual channels buffer while the output port is busy serving other packets. The VC implementation is one solution of deadlock avoidance. In NoC router architecture, the FIFO buffer is generally used for implementing the Virtual channels. The NoC system in Fig.4.9 is used to demonstrate how the VCs can be employed to avoid deadlock in the YaNoC framework. The NoC architecture has four nodes, there are four packets (p1-p4) destined for nodes 3, 4, 1, and 2, respectively. The packets are first forwarded through the X-axis path and then turn to Y-axis, for example, the path of packet p2 is 2-1-4. The different colors are used

Algorithm 4.2: Adaptive routing algorithm for Mesh based topologies

Input : Coordinates of current node (xc,yc), destination node (xd,yd) and the number of ports {E,W,N,S}
Output: Selected output port

Begin
W[E/W/N/S]=0;
for `i = E to S` **do**
  Calculate P[i] using Algorithm 4.1.
  if $empty == 00$ **then**
    return $W_{empty}[i] = 2$;
    return $W_{half\_full/nearly\_full/full}[i] = 0$;
  else if $half\_full == 01$ **then**
    return $W_{half\_full}[i] = 3$;
    return $W_{empty/nearly\_full/full}[i] = 0$;
  else if $nearly_full == 10$ **then**
    return $W_{nearly\_full}[i] = 5$;
    return $W_{empty/half\_full/full}[i] = 0$;
  else if $full == 11$ **then**
    return $W_{full}[i] = 10$;
    return $W_{empty/half_full/nearly_full}[i] = 0$;
  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
End

to mark the paths for all the packets. It can be seen that for the NoC architecture, the adaptive routing is deadlock free as the channel dependence graph is acyclic.

The livelock avoidance includes three scenarios- first, the adaptive routing is livelock-free as the packets will eventually reach the destination node although it will experience a longer path delay. Second, the adaptive routes packets along the edge, then turn direction at the region corner and finally arrive at the destination node. Third, in the other serious scenarios, a re-routing constraint mechanism (Wang et al. (2014)) can be employed. It constraints the no of re-routing performed and discards packets if re-routing exceeds a threshold number.

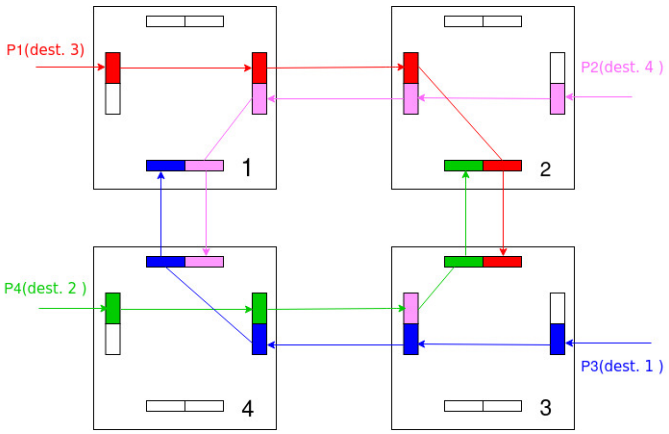


Figure 4.9: A deadlock avoidance in the YaNoC using VCs

4.4 THE PROPOSED RELIABLE NETWORK ON CHIP ROUTER

The 2-stage conventional router architecture has different pipeline stages, which have distinct and very specific role. A reliable router architecture has been designed to address multiple permanent faults in NoC system. Each pipeline stages of 2-stage conventional router are modified for fault detection and tolerance mechanism. We assume that faults can be detected by using one of the many existing fault detection mechanisms (Prodromou et al. (2012)). The difference between (Wang et al. (2014)) is that, we introduced novel fault tolerance mechanism for Input buffer and crossbar pipeline stages. The RC, VA and SA fault tolerance and detection mechanism employed from (Wang et al. (2014)). Fig. 4.10(c), shows that design of reliable fault tolerance router architecture.

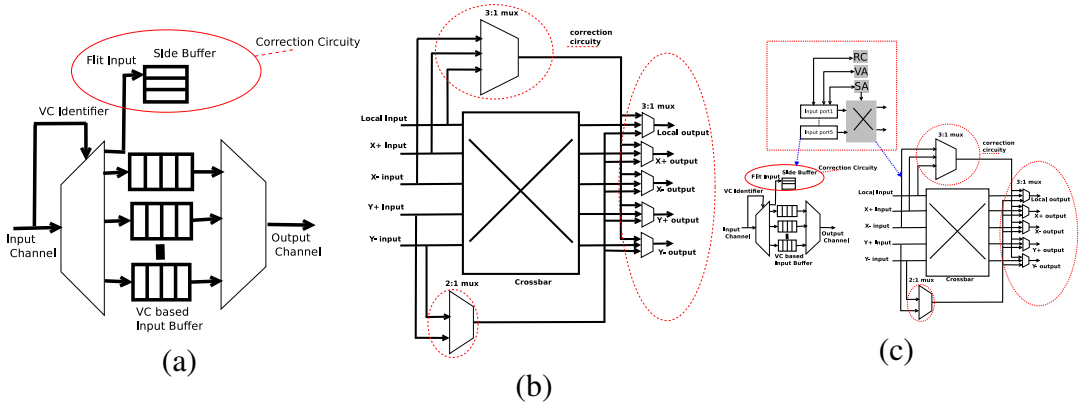


Figure 4.10: (a)Input buffer fault tolerance strategy, (b)Crossbar fault tolerance strategy and (c) proposed Adaptive and Reliable router architecture

4.4.1 Sideband Register buffer for input buffer(IB) faults

The input buffer is used to store the incoming flits from downstream routers. Improve the performance and avoid Head of line blocking in NoC system by introducing the virtual channel buffer. Each flit contains Virtual channel identifier, to store flit in corresponding virtual channel buffer. The faulty virtual channel leads to an error in incoming flits. We introduce a mechanism called sideband Register buffer. This is a simple circuit and less area overhead, but in (Wang et al. (2014)) has no fault tolerance mechanism for IB. The sideband buffer as shown in Fig. 4.10(a), used for storing incoming flit, when the corresponding virtual channel(VC) is faulty. The side buffer treats as new VC for that flit.

4.4.2 Crossbar fault strategy

The (Wang et al. (2014)) crossbar fault tolerate only for DoR XY routing algorithm. Modify the crossbar architecture to support both the DoR XY and Adaptive routing algorithm in the proposed method. From Fig. 4.10(b), it can be seen that each output port has a multiplexer that a flit from any input port needs to traverse through to reach the aforementioned output port. To provide fault tolerance to the generic crossbar architecture, we propose to have alternative paths to reach a specific output port of the crossbar by using smaller size multiplexers. Here, along with crossbar, the additional fault circuitry is composed of seven multiplexers(six 3:1 and one 2:1 multiplexer). Even when entire crossbar is faulty, the proposed strategy can work normally as compared to (Poluri and Louri (2014)). It has better performance than the Vicis router (Fick et al. (2009)).

4.5 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 were 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 used in this paper. The proposed YaNoC framework is capable of simulating the Mesh, Torus, and Fat tree

Table 4.2: Experimental Setup Details

Experimental Setup	
Topology	6×6 and 8×8 Mesh, Torus, 56 node Fat tree and 6×6 Diagonal Mesh
Buffer type	FIFO buffer
Buffer Depth	4, 8, 16, 32, 64
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 bit
Packet length	4 and 8 flits
Traffic pattern	Uniform random, Random Permutation, Bit complement, Transpose

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

4.5.1 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. The Verilog HDL code for all these topologies is generated by our Automated Verilog HDL Generator. The synthesis results of these NoC configurations

Table 4.3: Resource utilization of 6×6 (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 8×8 (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

have been obtained by employing the Xilinx Vivado 2016. The Xilinx Artix 7 FPGA (Device XC7A100T, Package-CSG324, speed grade-3) has been targeted in our experiments. In order to optimize the FPGA resource usage, the router microarchitectural parameters have been 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) has been varied between 4 to 64. Increase in the FPGA resources has been observed when we increase FW and BD parameters. The LUT and FF usage will be 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. The DRAM usage remains unchanged for the BD parameters till 32. When we increase the BD beyond 32, increase in the DRAM usage has been observed. The same behavior has been observed for both 6×6 Mesh and Torus topologies. The proposed design is optimized such that the DRAMs are capable of supporting the BD until 32 without any change in their usage. But, when we increase the BD beyond 32, the more number of DRAMs are needed to support the configuration.

Comparing the 6×6 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 6×6 Mesh topology consumes 32.33% LUTs and 12.62% of FFs. Whereas the 6×6 Torus topology consumes 40.15% LUTs and 15.54%

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

FFs. Similar behavior is observed for all the other configurations of BD and FW. The configuration of the boundary routers and the 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 8×8 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, increase in the FPGA resources has been observed. The behavior of DRAM resource consumption is similar to that of 6×6 Mesh and Torus topologies. Inferences similar to that of 6×6 Mesh and Torus topologies can be drawn with respect to 8×8 Mesh and Torus topologies. Considering the FW of 32 bits and the BD of 32 and 64, the 8×8 Mesh and Torus topologies exceeded the FPGA resources. Hence, the result for the same configurations has not been shown in the Table 4.4.

Table 4.5 shows the synthesis results for the 56 node Fat tree topology. Increasing the BD and FW parameters yield the increase in FPGA resource utilization. The LUT and FF usage will be increased from 41.21% to 56.96% and 16.06% to 26.06% considering for the BD of 4 and FW of 16 and 32 bits respectively. Comparing the 56 node Fat tree and 6×6 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 6×6 Mesh and Torus topologies.

4.5.2 FPGA synthesis results of Custom Topology

Table 4.6 shows the area utilization of 5-port and 9-port routers. 9-port 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 YaNoC router components on

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

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

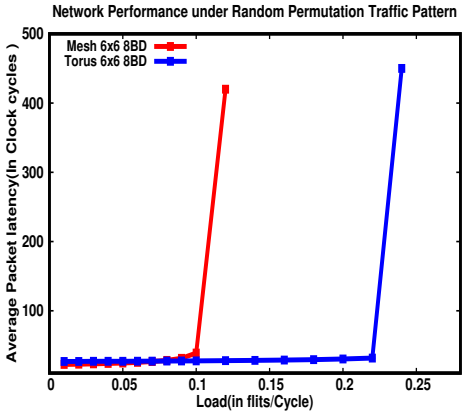
the Xilinx Artix7 XC7A100T device. Due to the more number of ports in the DMesh topology, its components consume $2\times$ the resources of Mesh topology.

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

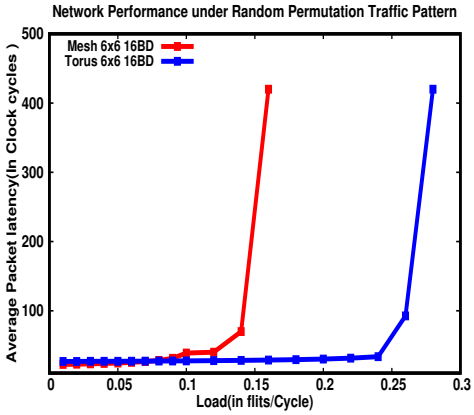
Our framework is also capable of supporting table based routing algorithm for custom topologies. Table 4.8 shows the synthesis results of Mesh and DMesh topologies considering the table based routing. The table based routing consumes 12% and 20% fewer LUTs compared to XY and novel shortest path version 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.

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
%LUTs	35.65	87.55	27.70	67.76
%DRAMs	11.37	20.46	11.12	20.02
%Flip Flops	15.14	20.62	13.08	19.89



(a) 6x6 Topology with Buffer Depth(BD) 8



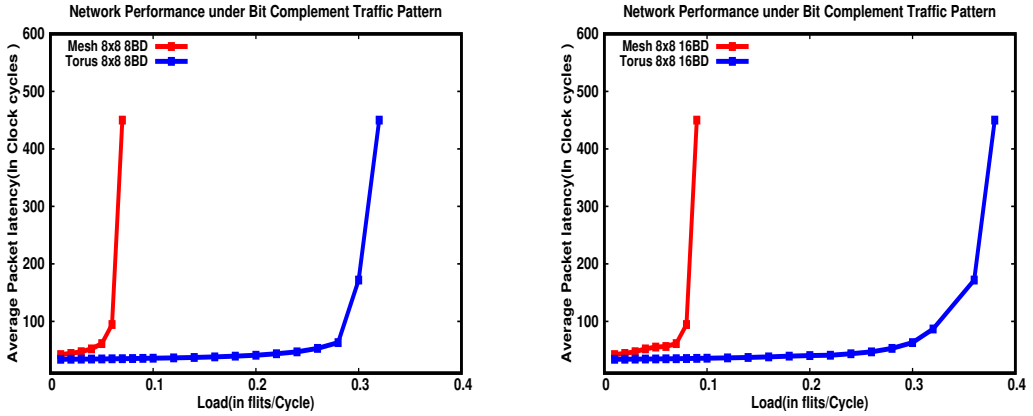
(b) 6x6 Topology with Buffer Depth(BD) 16

Figure 4.11: 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

4.5.3 Latency Analysis

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

Fig. 4.11 shows the average packet latency of 6×6 Mesh and Torus topologies under random permutation traffic pattern. From Fig. 4.11(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 a fairly good performance compared to Mesh topology. From Fig. 4.11(b), the 5%(i.e., from 10% to 15%) increase in saturation throughput of Mesh topology is observed. This is due to the effect of increasing the size of the BD. Mesh and Torus topologies saturated



(a) 8x8 Topology with BD=8 flits

(b) 8x8 Topology with BD=16 flits

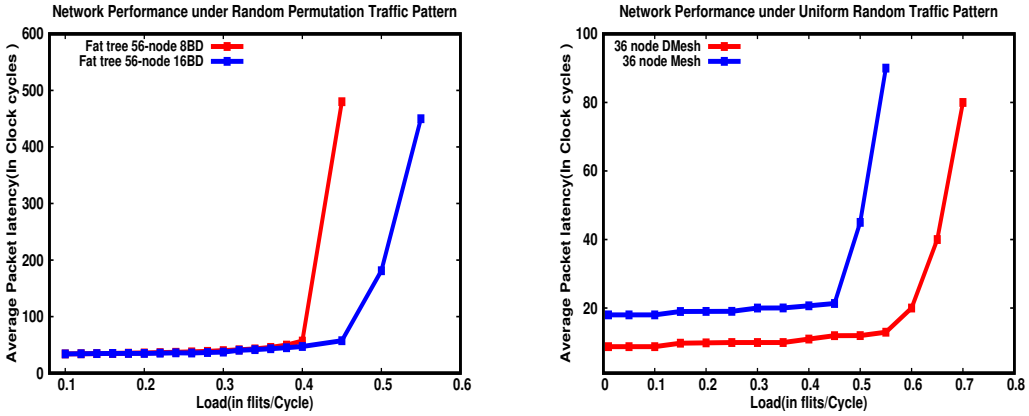
Figure 4.12: Load Delay graph of 8x8 Mesh and Torus Topologies under Bit Complement Traffic patterns (a)Buffer Depth=8flits and (b)Buffer Depth=16flits

at 15% and 24% respectively. The packet latency decreases significantly across all loads as we move from BD of 8 to 16 flits.

Fig. 4.12 shows the network performance of the 8×8 Mesh and Torus topologies under Bit complement traffic. From Fig. 4.12(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. 4.12(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. 4.13 (a). We observed that the 56-node Fat tree with BD of 16 has higher saturation throughput compared to BD of 8. The larger BD accommodates more packets resulting in 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.

Fig. 4.13 (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



(a) Fat Tree Topology with Buffer Depth(BD) 8 and 16

(b) Mesh and DMesh topologies

Figure 4.13: (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

of 55%. This is because of the higher Bisection bandwidth and connectivity of DMesh topology. The maximum hops 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 take 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.

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

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

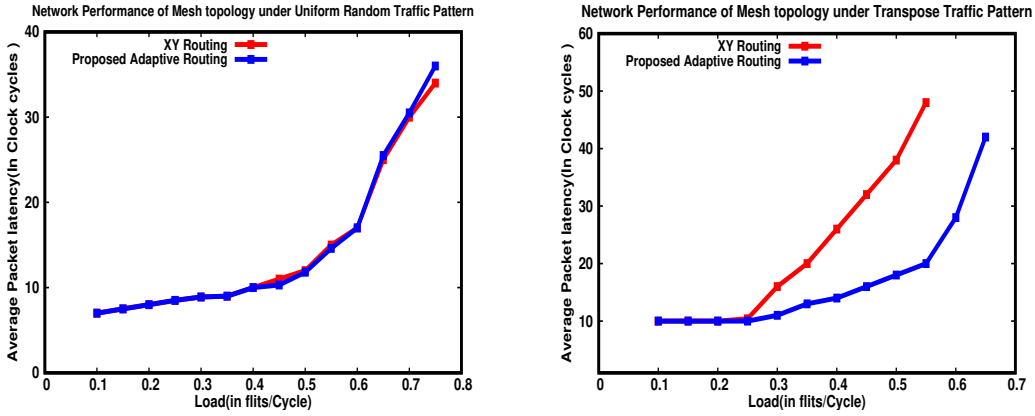
$$H_{max}(DMesh) = (N - 1) \qquad if N > M \tag{4.6}$$

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

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 utilization in %	XY routing	Proposed Adaptive routing
LUTs	35.65	37.31
DRAM	11.37	11.37
FFs	15.14	15.12



(a) Uniform traffic

(b) Transpose traffic

Figure 4.14: Load Delay graph of Mesh Topology under (a)Uniform and (b)Transpose traffic patterns

4.5.4 Analysis of Mesh Topology with Congestion-aware Adaptive Routing Algorithm

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

4.5.4.1 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 the algorithms consume the same amount of DRAMs and FFs. 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.

4.5.4.2 Network latency analysis

The conventional XY and proposed adaptive routing algorithms were evaluated considering Uniform and Transpose traffic patterns. In Uniform traffic pattern, each node sends a fixed size packet consisting of 8 flits to random nodes with Bernoulli distribution. From Fig. 4.14 (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. 4.14 (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%.

4.5.5 Analysis of Mesh Topology with proposed reliable NoC router

The 4×4 Mesh topology was evaluated considering the conventional and reliable router to analyse hardware resource utilization and the performance.

4.5.5.1 Area resource utilization

We implemented all pipeline stages of both the conventional and proposed router using Verilog HDL coding to study the impact. Synthesis results have been extracted from Xilinx Vivado 2016.2. Results include resource usage on the Xilinx Aritex 7 FPGA board(XC7A100T chip). Table 4.10 shows the detailed synthesis results of the conventional and proposed router for 4×4 Mesh topology. Based on the synthesis results, proposed router increases the area utilization by 24.34% with respect to the conventional router. The fault detection mechanism proposed in (Prodromou et al. (2012)), is chosen for faults detect. The area utilization increases 27.12% by incorporating fault detection mechanism into the proposed router.

4.5.5.2 Network latency analysis

We implemented 4×4 Mesh topology in Artix 7 FPGA, to study the effect of latency on proposed and conventional router architecture. We used uniform random synthetic traffic for NoC performance analysis. Analyze the fault tolerance of NoC architecture

Table 4.10: Area utilization of 4×4 Mesh Topology on Artix-7 FPGA device (XC7A100T, speed-3)

4x4 Mesh based topology		
H/W utilization in %	Conventional router	Proposed Adaptive and Reliable router
LUTs	41	52.12

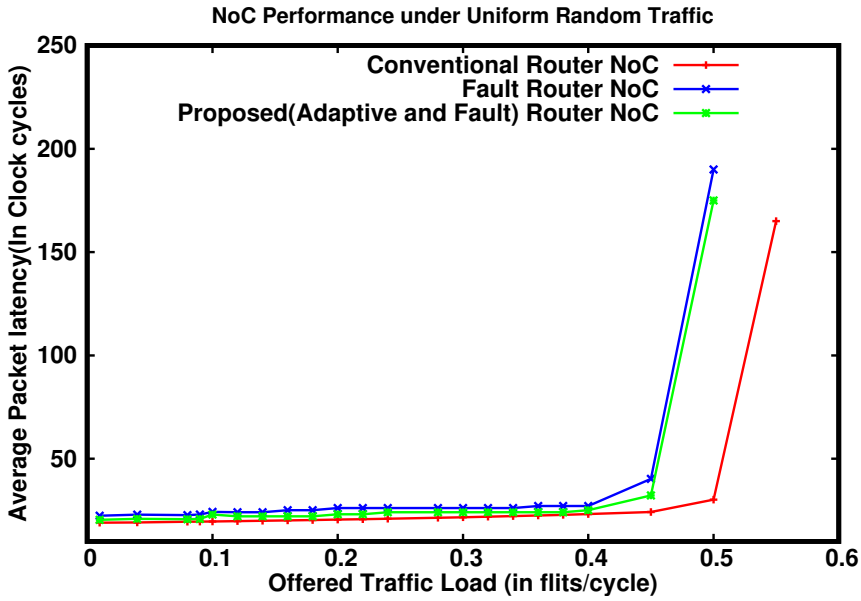


Figure 4.15: proposed Adaptive and Reliable router architecture

by injecting different faults into router pipeline stages of the 4×4 NoC. The additional circuitry is used to complete the execution of the pipeline stage, due to the faults in the pipeline stages. From Fig. 4.15, we observed that the average latency of the fault NoC has increased by 2.71% compared to the fault free NoC. The adaptive routing with fault tolerance reduced the 0.69% and increases by 2.0% average latency compared to fault NoC and fault-free NoC respectively as shown in Fig. 4.15.

4.5.5.3 Reliability Analysis

In this section, we study the reliability improvement of the proposed router in comparison to BulletProof (Constantinides et al. (2006), VicisFick et al. (2009), RoCoKim et al. (2006), PoluriPoluri and Louri (2014) and Wang Wang et al. (2014)) using Silicon Protection Factor(SPF). SPF is defined as the ratio of mean number of faults required to

cause a failure and the area overhead incurred due to the correction circuitry (Constantinides et al. (2006)). **The input buffer(IB) unit:** The faulty virtual channel is a input buffer virtual channel, which stores the input flits. It can introduce errors in the stored flits because of faulty, later flits turn to be invalid flits in the input buffer. The proposed router has 5-input port. Each port has 4 virtual channels. If all virtual channels are non-faulty, the router can operate well. The proposed strategy can tolerate 20 faults at most with 4 faulty virtual channels in each input port. A minimum of 5 faults can cause the failure of 4 VCs and one sideband register of input port are faulty. **The RC unit:** A maximum of 5 faults can tolerate router in this strategy. The functionality of one RC component per input port is affected by each fault. It requires that downstream router RC unit to be non-faulty. A minimum of 1 fault can cause failure the adjacent router has faulty RC. **The VA unit:** If each input arbiter becomes faulty, this unit can tolerate a maximum of 20 faults. A minimum of 2 faults can result in failure if the input VC arbiter and its default winner path both have faults. **The SA unit:** This unit can tolerate at most 10 faults, with all the arbiters in one switch allocator being faulty. The minimum of 2 faults can cause the failure of the SA. **The crossbar unit:** The faulty in the crossbar is defined as a faulty multiplexer. The crossbar unit can tolerate 5 faults at most. The minimum of 2 faults can cause failure with one additional bypass and one faulty multiplexer

The minimum number of faults to cause failure in individual stages of the pipeline router is calculated as

$$min(5(IB), 1(RC), 2(VA), 2(SA), 2(XB)) \tag{4.8}$$

The maximum number of faults tolerated by the router pipeline is calculated as the sum of the maximum faults tolerated by each individual stage, which results in.

$$20(IB), 5(RC), 20(VA), 10(SA), 5(XB) = 60faults. \tag{4.9}$$

An additional fault would result in failure. So the maximum number of faults to cause failure is 60. The mean number of faults to cause failure is (2+60)/2 . Area overhead is 27.12%. SPF of proposed Adaptive and Reliable router is 31/1.2712=24.4. Compare the proposed router with other faults tolerant router as shown in Table. 4.11. Other designs, the SPF value obtained from related work (Constantinides et al. (2006),

Fick et al. (2009), Kim et al. (2006), Poluri and Louri (2014) and Wang et al. (2014)). The proposed router achieves high reliability than the other fault tolerant designs based on SPF values.

Table 4.11: SPF comparison of the proposed router with other faults tolerant router designs.

Architecture	Area	Fault to cause failure	SPF
BulletProof	52%	3.15	2.07
Vicis	42%	9.3	6.55
RoCo	NA	5.5	<5.5
Poluri	31%	15	11.4
Wang	30%	28.5	21.9
Proposed	27.12%	31.5	23.95

4.5.6 Speedup

The simulation time of Booksim2.0 simulator was 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 Booksim2.0 to the simulation time of YaNoC. The simulation for a 6×6 network was run on both Booksim2.0 and YaNoC. A speedup of $2548\times$ is observed over Booksim2.0 simulator.

4.6 YANOC VS. STATE-OF-THE-ART

4.6.1 YaNoC and CONNECT

The Verilog HDL code of 6×6 Mesh and custom DMesh topologies are generated from CONNECT and YaNoC frameworks for comparing the hardware resource utilization.

Considering XY routing algorithm, it can be observed from the Table 4.12 that YaNoC's implementation of 6×6 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 a shortest path is chosen between the communicating routers.

Table 4.12: Resource utilization of CONNECT and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for 6×6 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

Table 4.13: Resource utilization of DART and YaNoC on Artix-7 FPGA device (XC7A100T, speed-3) for 3×3 Mesh topology

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

Also, for custom DMesh topology, the synthesis will not succeed as there will be a resource crunch while employing CONNECT's implementation. The Input Output Blocks (IOBs) will be exceeding the limit of Artix7 FPGA board. Whereas, synthesis of YaNoC's HDL code succeeds and the corresponding resource utilization is shown in the table. The 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. YaNoC is 2.55$\times$ faster than CONNECT NoC generator.

4.6.2 YaNoC and DART

3×3 network with XY routing algorithm of both DART Wang et al. (2014) and YaNoC are compared in Table 4.13. It can be observed that the % 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× speedup relative to Booksim. YaNoC is 25× times faster than DART.

4.7 SUMMARY

This paper presents YaNoC - a Network-on-Chip simulation acceleration framework using FPGAs. YaNoC supports the performance evaluation of various standard and custom NoC topologies. The router microarchitectural 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. 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 2.55x and 25x faster than CONNECT and DART frameworks, respectively.

CHAPTER 5

MAPPING THE NOC ROUTER COMPONENTS ON THE DSP48E1 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 crossbar switch has been embedded in 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 Configurable Logic Blocks (CLBs) utilization of NoC topologies on the FPGA has been observed by embedding the functionality of crossbar on the hard blocks of the FPGA.

5.1 INTRODUCTION

Modern FPGAs embed the ASIC-like hard blocks to perform the common functions more efficiently. The embedded hard blocks include Block RAMs (Xilinx Inc (2019a)), Embedded processors, DSP blocks (Xilinx Inc (2018)). These hard blocks, in addition to their functionality, can also be used to serve the other purpose. The BRAMs form dual-port memory blocks having independent write or read ports. Several Kilobits of data can be stored in BRAMs. The BRAMs can serve as an efficient on-chip memory as they can be configured as single or dual port memories with various port widths. The cascading support in latest FPGAs allow the BRAMs to be used as memory block with

large capacity. Also, the BRAMs can be employed as FIFOs, register files.

The primary goal of having DSP slices on the FPGAs is to efficiently perform the signal processing operations such as multiply and multiply-accumulate. The latest DSPs on the FPGAs support arithmetic operations, shift operations, logical operations and pattern matching operations. Multiple DSPs can be interconnected to perform the operations which are wider than the supported port widths. The DSP slices embedded in latest Xilinx 7 series FPGAs have an operating frequency of 740 MHz(Xilinx Inc (2018)).

All these features open up the opportunities for employing DSP blocks for the applications other than signal processing. The features of the five port NoC router crossbar switch have been embedded in a DSP tile consisting of two DSP48E1 slices. An FPGA based NoC simulation acceleration framework is proposed 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) 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.

Most of the state-of-the-art FPGA based NoC simulators utilize CLB components(LUTs and FFs) only for modeling the NoCs. 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 operating frequency of the whole circuit with the reduced power consumption. The modern FPGAs have a large number of DSP slices, due to which the higher frequency of operation and an increase in execution speed and lower power consumption can be achieved.
An FPGA-based NoC simulation acceleration framework is presented in this chapter.

The framework is capable of utilizing both the soft blocks (CLBs made up of LUTs and FFs) and the hard blocks (DSP48E1 slices) 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. The crossbar switch consumes 54% of the overall NoC router area (Kundu (2006)). This resource consumption can be reduced significantly by mapping the functionality of the crossbar to the hard blocks of the FPGA.
The contributions in this chapter are:

- A parameterized FPGA based NoC simulation acceleration framework employing the crossbar switch functionality of the NoC router efficiently on the DSP48E1 slices of the FPGA.
- Efficient utilization of DSP48 slices by employing the Time multiplexing technique at the inputs.

5.2 NOC ARCHITECTURE

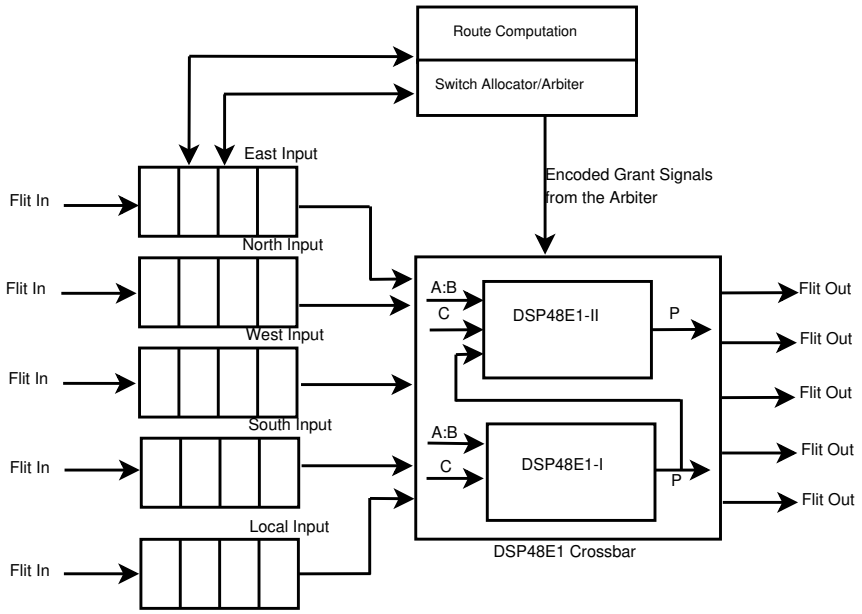


Figure 5.1: NoC router with the proposed DSP48E1 based crossbar

The framework is capable of embedding the functionality of the five port NoC router's crossbar switch by employing two DSP48E1 slices of a DSP tile. The proposed NoC router architecture is shown in Fig. 5.1.

5.3 DSP48E1 TILE AS THE CROSSBAR SWITCH

5.3.1 Xilinx DSP48E1 primitives

Fig. 5.2 shows the Xilinx DSP tile architecture. Two DSP48E1 slices are connected by the dedicated cascade links in each DSP tile (Xilinx Inc (2018)). A pre-adder, a multiplier and an ALU are included in each DSP48E1 slice. A 25-bit two-input adder/subtractor is included in the pre-adder. The output of pre-adder along with the asymmetric 18-bit and 25-bit inputs are provided as the input to the multiplier. The multiplier output together with the other 48-bit inputs form the ALU unit's input. With Xilinx DSP blocks, the data flow and the arithmetic operations can be dynamically reconfigured during runtime. Using this method, more complex data flow expressions can be implemented on the same DSP slice by employing the technique of time multiplexing. The dynamic operation of the DSP slice is accomplished by altering the OPMODE control signal of the multiplexer to select the right inputs and by modifying the ALUMODE signal per cycle to change the arithmetic/logic function.

5.3.2 Crossbar functionality on the DSP48E1 Multiplexers

ALU unit and the X, Y and Z multiplexers along with the control signals namely ALUMODE, INMODE and OPMODE account for configuring the DSP48E1 slices as the NoC router crossbar. By altering the OPMODE control signals according to the user needs, the multiplexers are controlled on a cycle basis. Likewise, the ALUMODE signals can be modified to configure the ALU unit to perform the logical or arithmetic operations. In the proposed work, the ALU unit is configured to perform the (X+Y+Z) operation through the ALUMODE control signal.

For mapping the functionality of the crossbar, a DSP tile containing two DSP48E1 slices is used. The cascaded links (PCOUT and PCIN) are used to interconnect 2 DSP48E1 slices present in a DSP tile. To achieve a similar behavior as the CLB based crossbar, the DSP48E1 slice inputs are provided with the correct router inputs at the right time.

The WEST and SOUTH output ports have been configured on the DSP48E1-I slice's P output port. And, the EAST and NORTH outputs are configured on the DSP48E1-II

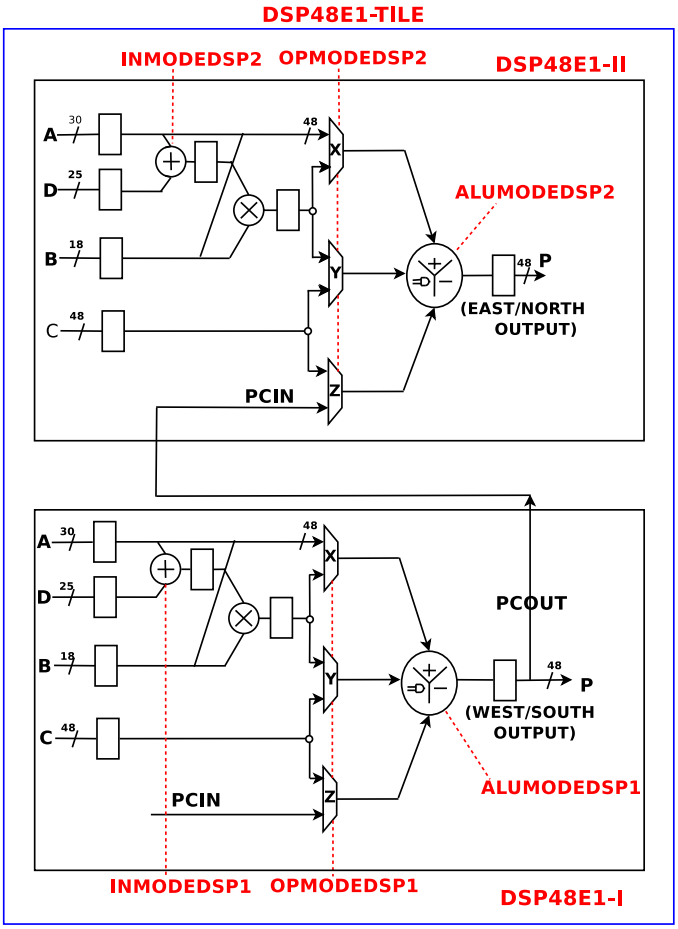


Figure 5.2: Two DSP48E1 slices connected by dedicated cascade links form a single DSP tile (Xilinx Inc (2018))

slice's P output port as shown in Fig. 5.3. Configuring the LOCAL output port to any of these DSP48E1 slices results in increased latency as the packets have to wait until the output port at the Crossbar is not allocated to any other ports. By employing the 4:1 Multiplexer, a new data path is introduced for bypassing the packets to sink at the LOCAL output port depending on the Arbiter grant signals.

The arbiter's encoded grant signals play an important role in choosing the OPMODE control signals of the DSP48E1 slice's X, Y and Z multiplexers. One-hot encoded signals are 10000, 01000, 00100, 00010, 00001 and respectively for South, West, Nort, East and Local input ports. Tables 5.2 and 5.3 indicate the DSP48E1-I and DSP48E1-II arbiter signals respectively. Table 5.1 shows the 4:1 Multiplexer operating signals based on the Arbiter grant signals. The framework supports round-robin, weighted round-robin and fixed priority arbitration schemes. In this work, the round-robin arbiter

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

is employed to select the output ports fairly when various input ports contend for a one output port. The 5-bit one-hot encoded signal is allocated to each input port that wins 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 this work: depending on these one-hot encoded signals from the arbiter, the 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.

In our routing logic, the turn models (Glass and Ni (1992)) are employed to overcome the turns leading to the network's deadlock condition. An assumption where the input port i sending the data to an output port j where $i \neq j$ has been made. The assumption made helps to avoid the critical path delay and the deadlock circumstances in the crossbar.

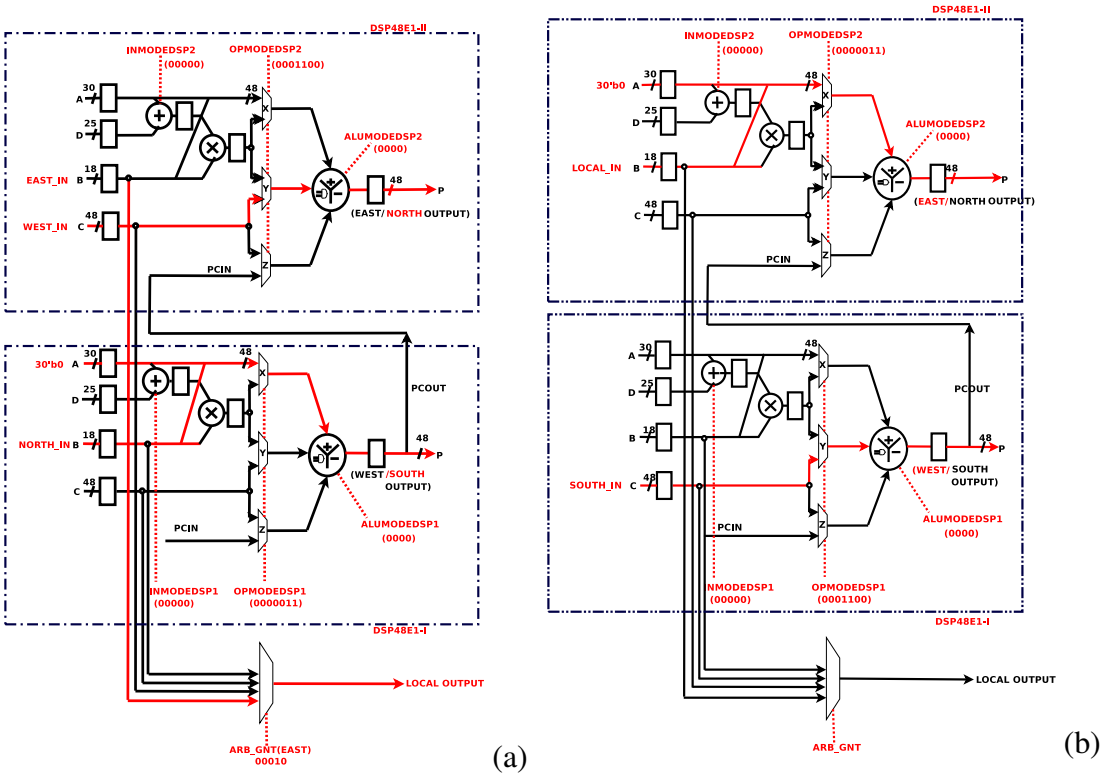


Figure 5.3: Illustration of mapping the input ports to the DSP48E1 based crossbar

The right inputs are configured at the right DSP48E1 slice depending on the Time multiplexing technique. Fig. 5.3 illustrates the embedding of NoC router crossbar functionality on the DSP48E1 slice. During the first clock cycle(Fig. 5.3(a)), if the flits

from the router's WEST and NORTH inputs are granted with the NORTH and SOUTH output ports, the arbiter grant signals are supervised to configure the input of the router to the DSP crossbar input. The inputs from WEST and NORTH ports are configured on the C and A: B inputs of the DSP48E1-II and DSP48E1-I slices as seen in Fig. 5.3(a) based on the arbiter grant signal, i.e. 01000 and 00100. Based on the arbiter grant signals, the OPMODEDSP2 and OPMODEDSP1 signals are configured to 0000011 and 0001100 to choose the WEST and NORTH inputs from the Y and X multiplexers of DSP48E1-II and DSP48E1-I slices to map the inputs to the NORTH and SOUTH output ports respectively. Also, when the LOCAL output port has been granted for the EAST input, the 4:1 Multiplexer configures the flits from EAST input to the LOCAL output port bypassing the DSP tile.

The flits from LOCAL and SOUTH inputs are mapped to the EAST and WEST output ports during the second clock cycle with the arbiter grant signals 00001 and 10000. LOCAL and SOUTH inputs are mapped to A:B and C inputs of DSP48E1-II and DSP48E1-I slices with the configuration of OPMODEDSP2 and OPMODEDSP1 signals to 0000011 and 0001100 as shown in Fig. 5.3 (b).

Table 5.4: Experimental Setup Details

Experimental Setup	
Topology	8×8 and 6×6 Mesh-based
Arbiter type	Fixed priority and Round-robin
Flow control	Wormhole switching
Crossbar mapping	DSP48E1 and CLB based crossbar
Routing Algorithm	Lookahead and Dimension-order (XY)
Router pipeline depth	5-stage
Buffer type	FIFO buffer
Buffer depth	4,8
Traffic pattern	Random Permutation, Hotspot, Nearest Neighbor, Tornado, Bit-complement
Packet length	4-flits
Flit size	16,32
FPGA Board	Xilinx Artix 7 (XC7A100T)

Table 5.5: Resource utilization of the DSP48E1 and CLB based Crossbar implementation ONLY on the Artix 7 (XC7A100T) board

	Proposed DSP CLB Crossbar	Crossbar
No. of FFs	148	243
No. of Slices Occupied	79	171
No. of LUTs	163	258
No. of DSP48E1 slices	2	0

5.4 RESULTS AND DISCUSSION

The Xilinx Artix 7 FPGA board (XC7A100 T chip) is employed for experiment. Verilog HDL has been used to implement the microarchitectural components of the NoC architectures. The findings of the synthesis results were extracted from Xilinx Vivado . Table 5.4 provides information on the experimental setup.

5.4.1 FPGA Utilization Results

5.4.1.1 Router implementation

The resource utilization of the Router architecture employing the CLB and DSP slices is shown in Table 5.5. The synthesis results for the topologies based on the CLB crossbar are obtained from YaNoC (Khyamling et al. (2019)). 171 slices are occupied by the CLB crossbar whereas 79 slices are occupied by the DSP48E1 crossbar. 258 and 163 LUTs, 243 and 148 FFs are occupied by the CLB and the DSP48E1 crossbars.A DSP tile containing 2 DSP48E1 slices along with a 4:1 multiplexer are used for mapping the crossbar switch with 5-ports. The DSP crossbar based NoC router occupies 53% fewer slices, 39% fewer LUTs and 36% fewer FFs compared to the CLB crossbar based NoC router. A DSP48E1 tile's height is the same as five CLB's (Xilinx Inc (2018)). Each CLB contains 8 LUTs and 16 FFs. Rounding to 40 LUTs and 80 FFs per DSP48E1 tile. The crossbar mapping on DSP will not constitute any loss in the silicon area for the applications which leave the DSP resources unused. We use the idle DSP48E1 slices to simulate the larger topologies on FPGAs.

5.4.1.2 Topology implementation

The hardware resource utilization of 6×6, 8×8 Mesh and Torus topologies considering the CLB and DSP based crossbar implementation can be seen in Tables 5.6, 5.7, 5.8 and 5.9. The XY and Look ahead routing algorithms have been employed for comparing the performance. The framework is capable of design space exploration of the NoC architectures by allowing the parameterized values of Input Buffer depth (BD) and Flit width (FW) parameters. The 'Configuration' column shows the configuration of Flit width and Buffer depth parameters. The 'CLB Crossbar' column shows the FPGA resource consumption in '%' of the resource used considering the CLB implementation of the crossbar, and the 'DSP Crossbar' column shows the FPGA resource consumption in '%' of the resource being used considering the DSP implementation of the crossbar.

It can be observed from both the tables that the CLB based implementation of the Topologies consume more LUTs and FFs than the DSP based implementation. This is because of the efficient mapping of the NoC Router's Crossbar component on the DSP48E1 blocks.

Table 5.6: Resource utilization of 6×6 and 8×8 Mesh topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with XY routing

Area Resource Utilization Under XY Routing									
Configuration			CLB Crossbar			DSP Crossbar			
Flit width	Buffer depth	Mesh Topology	LUT %	FF %	Occupied Slices%	LUT %	FF %	Occupied Slices%	DSP %
16	4	6x6	31	12	45	20	7	36	36
		8x8	64	24	87	36	12	50	53
	8	6x6	32	13	46	21	8	38	36
		8x8	65	25	88	37	13	51	53
32	4	6x6	45	18	60	22	9	40	36
		8x8	70	28	89	38	14	48	53
	8	6x6	46	19	60	24	10	43	36
		8x8	72	29	91	40	16	52	53

From Table 5.6, it can be observed that the increase in the Buffer depth or the Flit width parameters has an effect on the hardware usage of the NoC. 6×6 Mesh topology consumes 31% LUTs and 12% FFs, occupying 45% slices considering the FW of 16

bits and BD of 4 flits under the CLB crossbar implementation. The 8×8 Mesh topology consumes 64% LUTs, 24% FFs and occupying 87% slices under the same configuration. Increasing the BD to accommodate 8 flits with the same FW (i.e. 16 bits), an increase in the hardware resource consumption is observed. Likewise, when the FW is increased to 32 bits, 6×6 and 8×8 Mesh topologies consume 45% and 70% LUTs and 18% and 28% FFs, occupying 60% and 89% slices considering the BD of 4.

Considering the DSP48E1 crossbar implementation, fewer FPGA resources are consumed compared to the CLB crossbar implementation. The reduction in the resources is due to the effective mapping of crossbar on the DSP48E1 slice of the FPGA. It can be observed from Table 5.6 that, the 6×6 Mesh topology under DSP crossbar implementation consumes 20% LUTs, 7% FFs and 36% slices for the FW of 16 bits and BD of 4 flits. Similarly, 8×8 Mesh topology consumes 36% LUTs and 12% FFs with 50% slices occupied for the FW of 16 bits and BD of 4 flits. On average, the DSP crossbar implementation consumes 43%, 44% and 33% fewer LUTs, FFs and Occupied slices respectively compared to the CLB crossbar implementation.

Table 5.7: Resource utilization of 6×6 and 8×8 Mesh topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with LA routing

Area Resource Utilization Lookahead Routing									
Configuration			CLB Crossbar			DSP Crossbar			
Flit width	Buffer depth	Mesh Topology	LUT %	FF %	Occupied Slices%	LUT %	FF %	Occupied Slices%	DSP %
16	4	6x6	36	14	49	20	7	36	36
		8x8	65	25	86	44	14	58	53
	8	6x6	37	15	51	25	9	46	36
		8x8	66	27	87	45	15	59	53
32	4	6x6	51	20	66	26	10	49	36
		8x8	71	29	88	46	17	63	53
	8	6x6	52	20	64	27	11	50	36
		8x8	74	32	91	47	18	62	53

Table 5.7 shows the FPGA resource utilization of 6×6 and 8×8 Mesh topologies under the Look ahead (LA) routing algorithm. The LA routing implementation consumes more Hardware resources than the XY routing logic as it contains the hardware

Table 5.8: Resource utilization of 6×6 and 8×8 Torus topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with XY routing

Area Resource Utilization XY Routing									
Configuration			CLB Crossbar			DSP Crossbar			
Flit width	Buffer depth	Torus Topology	LUT %	FF %	Occupied Slices%	LUT %	FF %	Occupied Slices%	DSP %
16	4	6x6	33	13	48	20	7	37	36
		8x8	66	25	88	43	14	57	53
	8	6x6	34	14	49	22	9	39	36
		8x8	67	26	89	44	15	58	53
32	4	6x6	49	19	64	23	10	41	36
		8x8	Exceed	-	-	45	17	60	53
	8	6x6	50	20	65	25	12	46	36
		8x8	Exceed	-	-	46	18	61	53

to compute the route of a given flit in the next router. The 6×6 and 8×8 Mesh topologies under CLB crossbar implementation consume 36% and 65% LUTs, 14% and 25% FFs, occupying 49% and 86% slices for the FW of 16 bits and BD of 4 flits. Increasing the BD configuration to accommodate 8 flits with the same FW (i.e. 16 bits), increase in the hardware resource consumption has been observed. Similarly, the DSP crossbar implementation consumes 20% and 45% LUTs, 20% and 45% FFs, occupying 7% and 14% slices. On average, the DSP crossbar implementation consumes 37%, 44% and 20% fewer LUTs, FFs and slices compared to the CLB crossbar implementation. Similar behavior can be observed for the BDs of 8 flits with FW of 16 and 32.

Tables 5.8 and 5.9 show the synthesis results of the 6×6 and 8×8 Torus topologies under the XY and Look ahead routing respectively. From Table 5.6, 5.7, 5.8 and 5.9 it can be observed that the Torus topology consumes more hardware resources than the Mesh topology due to the presence of wrap around links. For example, from the Tables 5.6 and 5.8, considering the FW of 16 bits and BD of 4 flits under the CLB crossbar implementation, the 8×8 Mesh and Torus topologies consume 65% and 67% LUTs, 25% and 26% FFs, 88% 89% of the Slices respectively.

From Table 5.8 it can be observed that the 6×6 and 8×8 Torus topologies consume 33% and 66% LUTs, 13% and 25% FFs, 48% and 88% slices considering the FW of

16 bits and BD of 4 flits respectively under the CLB crossbar implementation. When the FW is increased to 32, the 8×8 Torus topology resource usage exceeds the number of LUTs present in the Xilinx Artix 7 FPGA. Same has been indicated by 'Exceed' in Table 5.8. This is because of the wrap around links present in the Torus topology. On the other hand, considering the DSP crossbar implementation of the 6×6, 8×8 Torus topologies consume fewer resources than the CLB crossbar implementation. With the DSP crossbar implementation, the mapping of 8×8 Torus topology succeeds effectively on the Xilinx Artix 7 FPGA. Considering the FW of 32 bits and BD of 8 flits, the DSP48E1 crossbar based 8×8 Torus topology consumes 46% LUTs, 18% FFs, occupying 61% slices and 53% DSP slices.

Table 5.9 shows the FPGA resource utilization of 6×6, 8×8 Torus topologies employing the LA routing algorithm. The LA routing implementation consumes more FPGA resources than the XY routing implementation due to the complexity of LA routing logic needed to compute the route. From Table 5.9 it can be observed that the 6×6 and 8×8 Torus topologies consume 38% and 68% LUTs, 15% and 26% FFs, 53% and 89% Slices considering the FW of 16 bits and BD of 4 flits respectively under the CLB crossbar implementation. When the FW is increased to 32, the 8×8 Torus topology resource usage exceeds the number of LUTs present in the Xilinx Artix 7 FPGA. The same has been indicated by 'Exceed' in Table 5.9. Contrarily, with DSP48E1 crossbar implementation, the 6×6 and 8×8 Torus topologies consume fewer resources than the CLB crossbar implementation and the mapping of 8×8 Torus topology succeeds effectively on the Xilinx Artix 7 FPGA. Considering the FW of 32 bits and BD of 8 flits, the DSP48E1 crossbar based 8×8 Torus topology consumes 49% LUTs, 19% FFs, occupying 74% slices and 53% DSP slices.

5.4.2 Latency and Saturation Throughput Analysis

The latency performance of CLB and DSP48E1 based crossbar implementation of 6×6 and 8×8 Mesh and Torus topologies is analyzed injecting various synthetic traffic patterns. Following are the synthetic traffic patterns employed in the experiments: (i)Nearest Neighbor(NN), (ii)Random Permutation(RP), (iii)Hotspot(HS), (iv)Tornado(TO)

Table 5.9: Resource utilization of 6×6 and 8×8 Torus topologies with CLB and DSP48E1 crossbar mapping on Artix 7(XC7A100T) FPGA with LA routing

Area Resource Utilization Under Lookahead Routing									
Configuration			CLB Crossbar			DSP Crossbar			
Flit width	Buffer depth	Torus Topology	LUT %	FF %	Occupied Slices%	LUT %	FF %	Occupied Slices%	DSP %
16	4	6x6	38	15	53	25	8	45	36
		8x8	68	26	89	45	14	60	53
	8	6x6	39	17	54	26	9	48	36
		8x8	70	28	90	46	15	61	53
32	4	6x6	55	21	70	27	10	51	36
		8x8	Exceed	-	-	48	17	65	53
	8	6x6	56	23	69	28	11	53	36
		8x8	Exceed	-	-	49	19	74	53

and (v)Bit-complement(BC).

The latency behavior of the 6×6 and 8×8 Mesh and Torus topologies under DSP and the CLB based crossbar implementations considering the above-mentioned traffic patterns is shown in Fig. 5.4 and 5.5. The average packet latency and injection load are denoted in clock cycles and flits per clock cycle per node.

It can be seen from Fig. 5.4 that the DSP48E1 crossbar based implementation performs similar to the CLB crossbar implementation of the 6×6 Mesh and Torus topologies with minimum error. Under the NN traffic (Fig. 5.4(a)) considering the XY routing algorithm, the Mesh topology reaches saturation at 0.22 injection rate. An improvement of 36% (0.22 to 0.30)in saturation throughput has been observed employing the LA routing algorithm. This behavior is because the congestion observed using the LA routing algorithm is lower than the congestion observed using the XY routing algorithm. Torus topology is capable of accepting more traffic as it has the higher bisection bandwidth than the Mesh topology. The 6×6 Torus topology saturates at an injection rate of 0.48 under the XY routing algorithm. Employing the LA routing algorithm, an improvement of 25% (from 0.48 to 0.60) has been observed.

With the RP traffic (Fig. 5.4(b)), the 6×6 the Mesh topology reaches saturation at 0.12 injection rate employing the XY routing algorithm. Considering the LA routing

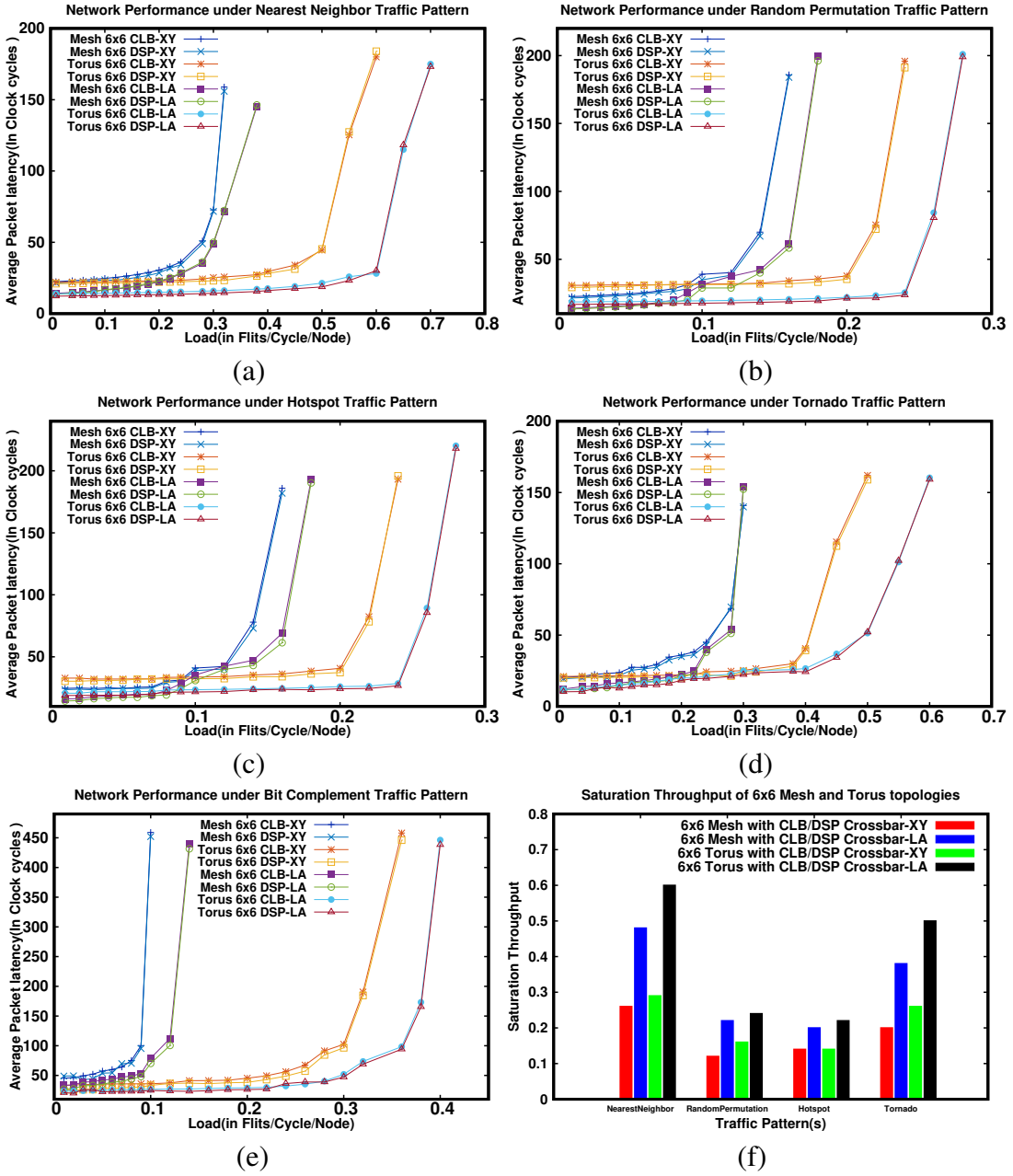


Figure 5.4: (a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the 6×6 Mesh and Torus topologies under CLB and DSP48E1 based crossbar implementation under Nearest Neighbor, Random Permutation, Hotspot, Tornado and Bit complement traffic patterns employing the XY routing and LA routing

algorithm, an improvement of 33% (from 0.12 to 0.16) has been observed. An improvement in saturation throughput of 30% (from 0.20 to 0.26) while employing the LA routing instead of XY routing has been observed with the 6×6 Torus topology. Similar behavior has been observed employing the HS traffic.

Considering the TO traffic (Fig. 5.4(d)), the Torus topology saturates earlier than the Mesh topology. This behavior is observed as the TO traffic stresses the Torus topology compared to the Mesh topology. Considering the XY routing algorithm, the 6×6 Torus topology saturates at the injection rate of 0.22. With the LA routing algorithm, an improvement of 36% has been observed (from 0.22 to 0.30). The 6×6 Mesh topology reaches saturation at 0.4 injection rate. Employing the LA routing, the 6×6 Mesh topology reaches saturation at 0.5 injection rate. An improvement of 25% (from 0.4 to 0.5) has been observed employing the LA routing algorithm.

Under the BC traffic pattern, the 6×6 Mesh topology saturates earlier than the Torus topology. The 6×6 Mesh topology saturates at an injection rate of 0.07 and 0.09 considering the XY and LA routing algorithms. An improvement of 28% (from 0.07 to 0.09) has been observed employing the LA routing algorithm. Similarly, the 6×6 Torus topology saturates at the injection rates of 0.28 and 0.37 considering the XY and LA routing algorithms. An improvement of 32% (from 0.28 to 0.37) has been observed employing the LA routing algorithm.

Fig. 5.5 (a) - (e) show the behavior of 8×8 Mesh and Torus topologies under the traffic patterns mentioned in Table. 4.2. Improvement in the saturation throughput of the 8×8 Mesh and Torus topologies has been observed in case of all the traffic patterns employing the LA routing over the XY routing. Improvement of 19% (from 0.25 to 0.3), 44% (from 0.09 to 0.13), 16% (from 0.12 to 0.14), 19% (from 0.2 to 0.24) and 28% (from 0.07 to 0.09) has been observed with respect to the 8×8 Mesh topology under the NN, RP, HS, TO and BC traffic patterns employing the LA routing. Similarly, improvement of 25% (from 0.4 to 0.5), 22% (from 0.18 to 0.22), 11% (from 0.18 to 0.20), 11% (from 0.34 to 0.38) and 28% (from 0.28 to 0.36) have been observed with respect to the 8×8 Torus topology under the NN, RP, HS, TO and BC traffic patterns employing the LA routing.

Fig. 5.4(f) and 5.5(f) show the saturation throughputs of 6×6, 8×8 Mesh and Torus topologies under the afore-mentioned traffic patterns considering the XY and LA routing algorithms. The topologies perform better under the LA routing algorithm due to the pre-computation of the preferred output port. It can be observed that the Torus

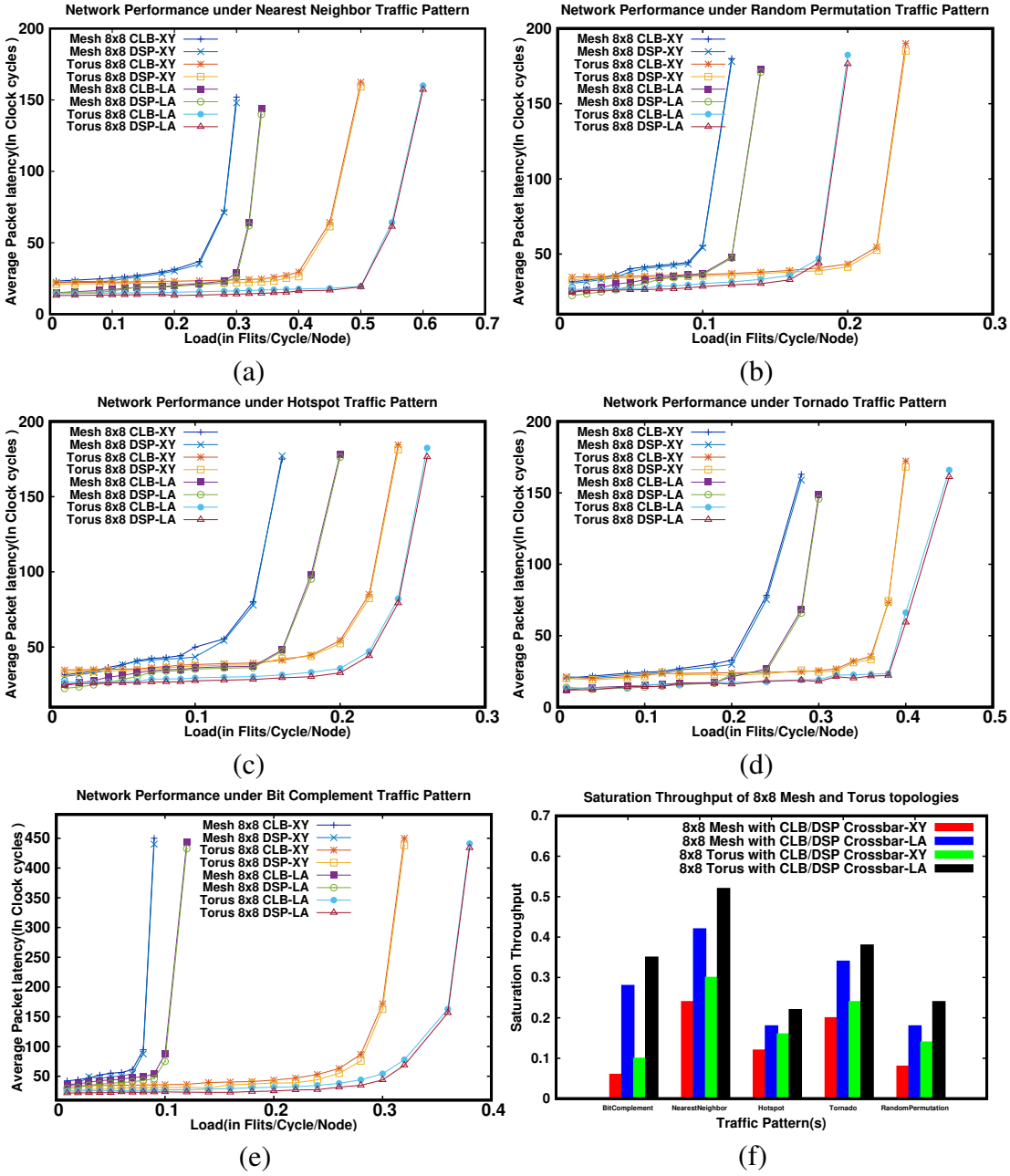


Figure 5.5: (a), (b), (c), (d), (e) - Load injected vs Observed Latency curves and (f) - Saturation Throughput for the 8×8 Mesh and Torus topologies with CLB and DSP48E1 based crossbar implementation under Nearest Neighbor, Random Permutation, Hotspot, Tornado and Bit complement traffic patterns employing the XY routing and LA routing

topology has the capability of sustaining the higher injection rates compared to the Mesh topology because of the higher bisection bandwidth.

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

A comparison of the proposed work is made with state-of-the-art CONNECT framework (Papamichael and Hoe (2015)). HDL code for 6×6 Mesh topology has been obtained from the CONNECT tool with the same configurations as shown in Table 4.2. The resource utilization results of the 6×6 Mesh topology designed with the proposed DSP48E1 crossbar architecture and the CONNECT NoC generation are shown in Table 5.10. 23% fewer slices, 41% fewer LUT are consumed by the 6×6 Mesh topology with proposed DSP48E1 based crossbar compared to the 6×6 Mesh topology obtained from CONNECT NoC framework. The operating frequencies of the 6×6 Mesh topology under the proposed work and the CONNECT implementation are 442MHz and 146MHz respectively. The CONNECT architecture employs a single stage router pipeline due to which it consumes fewer FFs compared to the proposed work employing the five stage pipeline. 6×6 Mesh topology with the proposed DSP48E1 crossbar implementation consumes 44% less power compared to the CONNECT architecture.

Table 5.10: FPGA synthesis results of the 6×6 Mesh topology considering the proposed DSP48E1 crossbar implementation and CONNECT's implementation on Artix 7 (XC7A100T) board

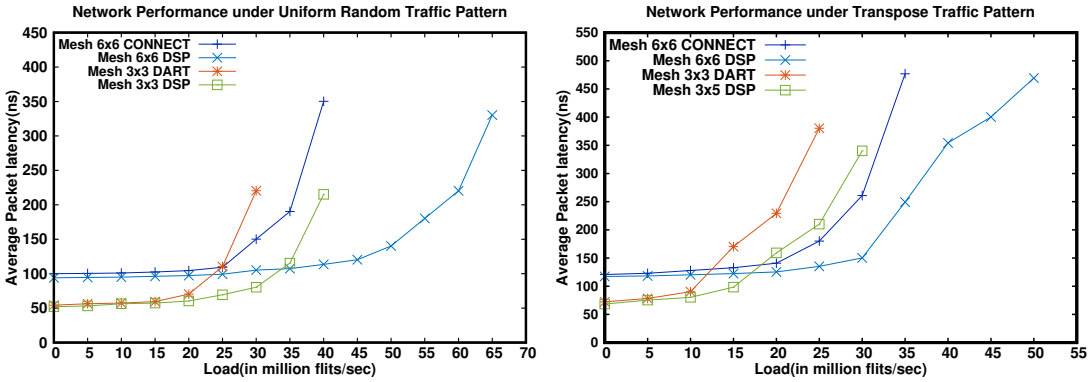
	CONNECT (Papamichael and Hoe (2015)) 6×6 Mesh		6×6 Mesh with Proposed DSP Crossbar	
	Number	(%)	Number	(%)
FFs	3127	2	8797	7
LUTs	22158	34	12873	20
Slices Occupied	7608	47	5779	36
DSP48E1s	0	0	72	30
Clock(ns)	6.85	-	2.26	-
Power(mW)	412	-	229	-

3×3 Mesh topology considering XY routing algorithm of both the DART (Wang et al. (2014)) and the proposed DSP48E1 based crossbar are compared in Table 5.11. DART consumes 9% FFs, 30% LUTs and occupies 40% slices. The 3×3 Mesh topology with the proposed DSP48E1 based crossbar consumes 1% FFs, 4% LUTs, 8% slices and 7% DSP48E1 slices available on the FPGA. Employing the proposed DSP48E1

Table 5.11: Hardware utilization results of the 3 × 3 Mesh topology with proposed DSP48E1 based crossbar and DART's implementation on Artix 7 (XC7A100T) FPGA

	DART (Wang et al. (2014)) 3 × 3 Mesh		3 × 3 Mesh with Proposed DSP Crossbar	
	Number	(%)	Number	(%)
FFs	12034	9	2173	1
LUTs	19210	30	3159	4
Slices Occupied	6596	40	1420	8
DSP48E1s	0	0	18	7
Clock(ns)	5.82	-	1.9	-
Power(mW)	234	-	59	-

based crossbar architecture in the 3 × 3 Mesh topology, a reduction of 88%, 86% and 80% are observed with respect to FFs, LUTs and the occupied slices. The 3 × 3 Mesh topology with the proposed DSP48E1 based crossbar consumes 74% less power than the DART's implementation. Also, the operating frequency of DART and the proposed implementation were found to be 171MHz and 526MHz respectively.



(a) (b)

Figure 5.6: (a), (b) - Load vs Latency comparison of the Mesh topologies employing proposed DSP48E1 crossbar architecture and CONNECT, DART topologies

The latency performance comparison of the proposed DSP48E1 crossbar based Mesh topology with CONNECT and DART Mesh topologies is illustrated in Fig. 5.6 (a) and (b) considering the Uniform random and Transpose traffic patterns. As the injection rate increases, the Mesh topology employing the proposed crossbar offers lower latency than the CONNECT and DART Mesh topologies. The Mesh topology em-

ploying the proposed DSP48E1 based crossbar has the capability of accepting twice the injection rate compared to the CONNECT and DART architectures. CONNECT and DART Mesh NoC topologies saturate much earlier compared to the Mesh topology employing the proposed DSP48E1 based crossbar architecture.

5.5 SUMMARY

The unused hard blocks of the FPGA, such as DSP48E1 blocks are employed to map the functionality of NoC router's Crossbar components. A reduction of soft logic has been observed employing the proposed technique. 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. The flexible communication and traffic generation model has been proposed in the next chapter.

CHAPTER 6

P-NOC: PERFORMANCE EVALUATION OF NOCS ARCHITECTURE USING FPGA

In this chapter, P-NoC: an FPGA-based parameterized framework for analyzing the performance of NoC architectures based on various design decision parameters is presented. The Mesh and a Multi-Local port Mesh(ML-Mesh) topologies have been considered for the study. Experiments show that the flit width(FW), Buffer Depth(BD), virtual channels(VCs) parameters have a significant impact on the FPGA resources. Along with various router microarchitectural configurations, the hop count analysis has been performed to understand the effect of hop count on the latency and throughput of the NoC under consideration.

6.1 P-NOC: FPGA-BASED PARAMETERIZED FRAMEWORK

The P-NoC:FPGA-based parameterized framework for designing and evaluating the performance of NoC architectures as shown in Fig. 6.1. P-NoC is implemented with highly parameterized, modular and it provides easy of reconciling new Network-on-Chip architecture. P-NoC consists of Topology, Router and Traffic modules. The topology module supports the implementation of standard and customized NoC networks. The router module is highly parameterized with Buffer depth(BD), Flit width(FW), Virtual Channel(VC), routing logic, Channel width and I/O ports being configurable. The traffic module consists of a traffic receptor and a generator. The generation of packets and injection into the NoC is done by the traffic generator. The framework supports

the generation of various types of synthetic patterns such as Uniform random, Bit complement, Bit shuffle, Transpose, Random Permutations and Nearest Neighbour. The traffic receptor ejects the packet from NoC architecture and collects the Area, Latency, Throughput, Hop count, Average Hop count and Power results. The NoC architecture is generated combining all these modules based on the configuration of parameters.

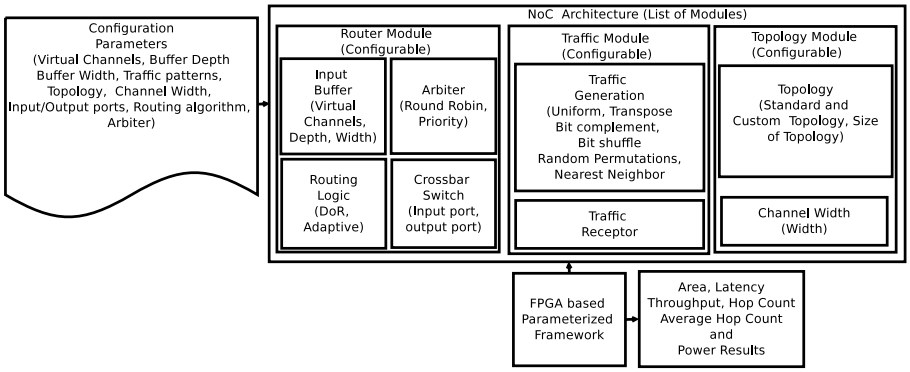


Figure 6.1: Layout of P-NoC: An FPGA based Parameterized framework for design space exploration and performance analysis of NoCs for Chip Multiprocessor architecture

6.1.1 Topology Module

This module generates various NoC topologies based on the given input parameters. Mesh and Multi-local port Mesh(ML-Mesh) topologies have been considered for the experiments in this work.

6.1.1.1 The Mesh based NoC Architectures

Fig. 6.2(a), illustrates the 16-Node Mesh topology for on-chip communication. From the Fig. 6.2(a), it can be seen that each processing element(PE) is connected to a router. The bi-directional links are used for interconnecting the routers. The two dimensional mesh topology is the primary choice for tiled architectures due to ease of implementation, routing and it's closely matches with the physical layout of the die.

6.1.1.2 The Multi-local Port Mesh(ML-Mesh) NoC architecture

Fig. 6.2(c) shows the 16-node ML-Mesh topology with four local ports. Nodes are interconnected using cluster-based design. This improves the performance of NoC architecture by reducing the distance between two communicating elements as the nature

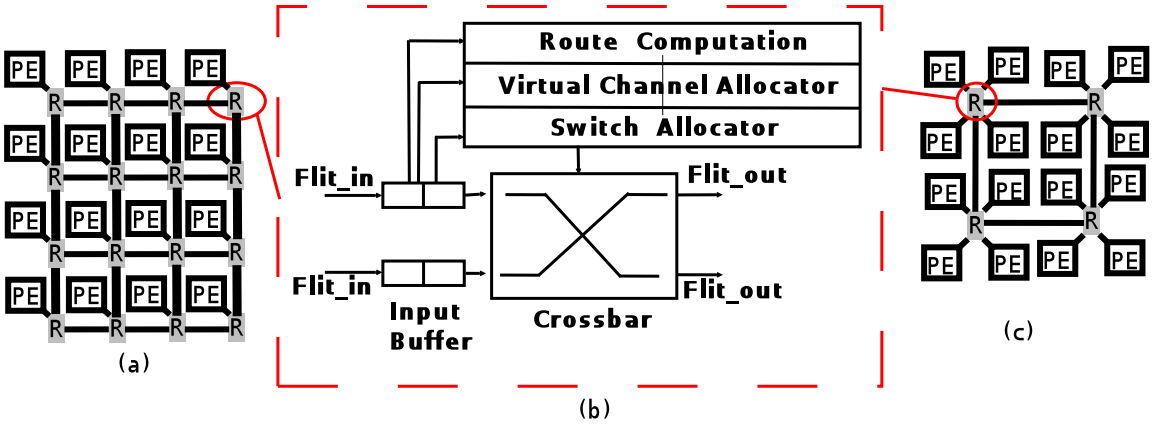


Figure 6.2: (a)16-Node Mesh Topology(16-nodes and 16-routers), (b) parameterized router architecture and (c) 16-Node ML-Mesh topology(16-nodes and 4-routers)

of traffic is localized. In the ML- Mesh topology, each router consumes more area, but the overall chip size is reduced. This allows a larger number of input buffers in the routers reducing the maximum effective distance across the network. The multi-local port NoC architecture provides a more compact physical layout and reduces the wires allowing wider channel widths. In this NoC architecture, less number of routers permit a lower hop count without increasing wiring complexity. From Fig. 6.2(c), it can be seen that the router can have four local ports connected to the neighboring processing elements(PEs). Therefore, the hop count and communication latency of neighboring PEs are reduced by exchanging data through a single router.

Six hops are needed for the communication between first and last corner nodes in the 16-node Mesh topology as seen in Fig. 6.2(a). Considering the ML-Mesh topology, the hop distance has been reduced to two hops as seen in the Fig. 6.2(c). The parameterized router architecture used for communication fabric in Chip Multiprocessors(CMPs) design is shown in Fig. 6.2(b).

6.1.2 NoC Router Microarchitecture

The NoC router performs the forwarding of packets from the source PEs to the destination PEs. The router comprises of Input Buffers, a Route computation unit, Virtual allocation(VC), Switch allocation(SA) and Switch travels(Crossbar) as shown in Fig. 6.2(b).

6.1.2.1 Input Buffers

The router input ports have Input Buffers to store the incoming flits from the neighboring nodes, before transferring flit to the next nodes. Fig. 6.3 shows the implementation of the Input Buffer at each input port. The Input Buffer has two status signals:*full* and *empty* to control the flit transfer between routers. The Input Buffer can be re-configured considering the Flit width and Buffer depth parameters with various sizes. This identifies the impact of performance and silicon area of NoC architecture for specific applications.

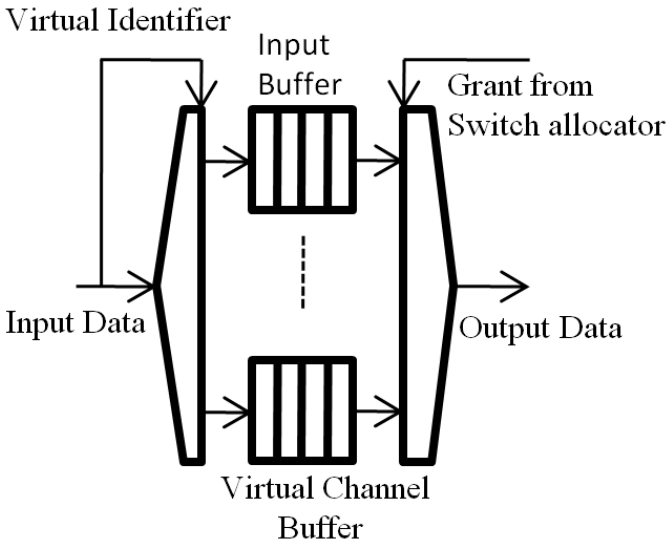


Figure 6.3: Architecture of the Parameterized VC based Input buffer employed in designing the router(Virtual Channel, Input Buffer Depth, Width can be configurable)

6.1.2.2 Route Computation Unit

The Route computation unit computes the correct output port for the given incoming packets. For every source and destination pair, there exist multiple paths. The selection of best path is done by the routing algorithm in NoCs. The routing algorithm impacts the latency and throughput experienced by the traffic. Various routing algorithms are implemented to balance the load in the NoC networks. In this work, we implement oblivious routing algorithms such as dimension ordered routing(DoR) and adaptive routing algorithm to route the packets in the networks.

6.1.2.3 Virtual channel(VC) and Switch Allocator

The Virtual channel(VC) allocator performs a quality matching between input VCs requested to the output VC resources which is subjected to the constraint that, at any given time slot, many input VCs request to any of the output VCs of the same output port. The VC allocator is required to generate the best match for given input VCs. Otherwise, the generation of best matchings later can affect the average waiting time of head flits before being assigned to an Output VC. This will prevent flit movement of the same packet and thereby increasing average buffer utilization. After successful allocation of VCs, the Switch Allocator is required to arbitrate individual flits for accessing physical channels. We implemented round robin arbitration for the fair sharing of each resources in the NoC architecture.

6.1.2.4 Crossbar

All input and output ports are interconnected by a crossbar block. After a grant signal from the VC and Switch Allocation block, the head flit traverses to the crossbar unit. The crossbar unit maps the incoming head flit to the output port which was computed in route compute unit.

6.1.3 Traffic Pattern Module

We implemented Traffic pattern modules for the NoCs performance analysis using a traffic generator. This module consists of a Traffic generator and receptor.

6.1.3.1 Traffic Generation

The Traffic Generation (TG) modules are used to generate the various traffic patterns. Each router of the NoC topology are associated with traffic generation module. The TG modules are responsible for generating the packets and storing them in the source queue. Also, the flit generation logic has been incorporated in the TG module. The synthetic traffics such as Uniform, Bit complement, Bit shuffle, Transpose, Random Permutations and localized(Nearest Neighbour) traffic are generated by TG Module.

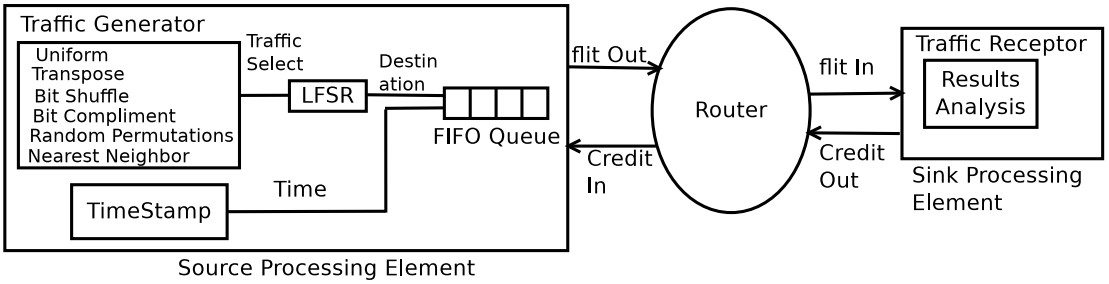


Figure 6.4: Packet generator and receptor

6.1.3.2 FIFO Queue

The FIFO queues are implemented to store the flits generated by the TG module prior to the injecting the packet in the network. The source queues operate in the FIFO fashion. The FIFO queue takes care of the injection of the flits into the NoC topology based on the emulation cycles. The flits are read in the FIFO fashion from the non-empty queues and transmitted to the respective router through local input port after data validation.

6.1.3.3 Traffic Receptor

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

Fig. 6.4, shows the router with the Source and Sink processing element. The Source Processing element is responsible for Traffic generation. The randomness traffic has been generated by employing Linear Feedback Shift Register(LFSR) mechanism and to add the TimeStamp module in the traffic being generated. The Sink processing element accepts flits from the router and it has a Traffic Receptor module to keep a record of the number and time of incoming flits. The Traffic receptor module performs the latency, throughput analysis.

6.1.4 Flit Format

The channel width matches the width of a flit by dividing packet into Head, Body and Tail flits. Fig. 6.5 shows the structure of 32-bit flit. The flit size can be configured to any flit size varying from 32 and 64 bits. The 2-bit ‘Type’ field is used to represent the

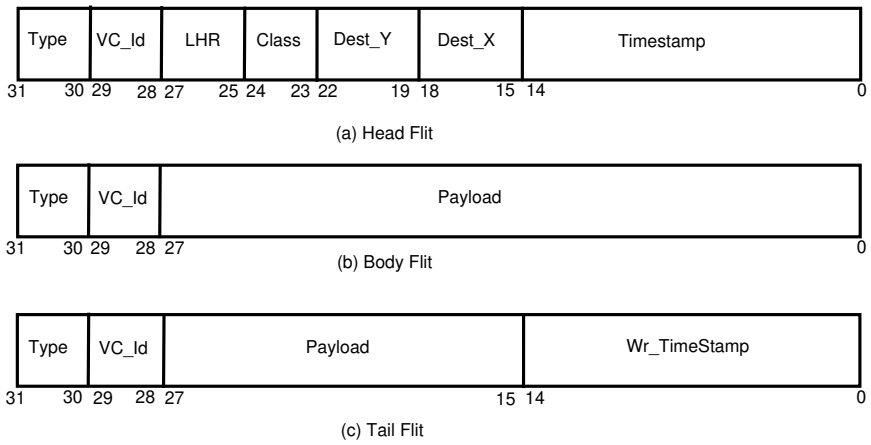


Figure 6.5: The configurable Flit Structure. 32 and 64-bit flit have been used in this work. We show the example of only 32-bit Flit Structure, each field in Flit Structure can been configurable according the size of Topology, Packets and Number of virtual channels

type of the 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 router computed using the look-ahead routing algorithm. The 'Class' field specifies the message class. Destination Y and X address fields are specified in 'Dest_Y' and 'Dest_X' fields. The 'TimeStamp' field has been used to specify flit creation time at the Source. The Body flit contains '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 time at each Tail flit received by the destination node. Table 6.1 shows the detail of each fields in the flit structure.

6.2 DESIGN COST AND PERFORMANCE ANALYSIS

To compare and contrast different NoC architectures, standard set of metrics have been considered (Partha Pratim Pande et al. (2005)). The wormhole switching is employed for the data transmission where the packet is divided into flits of a small length of flow control units. The Head flits of a packet hold the routing and control information based on that path can be established. This path is follow the Body and Tail flits. The comparative analysis focuses on well-established benchmarks of latency, throughput, silicon area and Power.

Table 6.1: Flit structure employed in the experiments. All the fields are configurable. 32-bit flit structure has been employed in this work for reference.

Flit Structure(32-bit)	
Field	Size
Type	2-bit(00-Body, 10-Head, 01-Tail)
VC_ID	Virtual Channel Identifier(2-bit, if VC=2, 01 for VC1 and 10 for VC2)
LHR	Lookahead Routing(3-bit for 5-port)
Class	Message Class(2-bit)
Dest_Y	Destination Y Coordinate address(2-bit for 16-node topology)
Dest_X	Destination X Coordinate address(2-bit for 16-node topology)
TimeStamp	Time(19-bit)
Payload	Actual Data(28-bit)
Wr_TimeStamp	Tail flit is created(19-bit)

6.2.1 Hop count

The Hop count is defined as the number of routers the packet would traverse from the source processing element to the destination processing element. The Hop count is calculated assuming dimension-ordered(DoR) routing for Mesh, ML-Mesh topologies. For Mesh topology, if the address coordinates of source and destination routers are S_x, S_y, D_x, D_y, respectively, then $Diff_x$ and $Diff_y$ are the x and y directions hops respectively are obtained as follow.

$$Diff_x = |D_x - S_x| \tag{6.1}$$

$$Diff_y = |D_y - S_y| \tag{6.2}$$

The Hop count can be calculated as:

$$Hopcount = Diff_x + Diff_y; \tag{6.3}$$

6.2.2 Link utilization

In the NoC architecture, two neighboring nodes are interconnected using communication links. In general, the number of bidirectional links in Mesh based NoC is given as follows:

$$Links = M_1(M_2 - 1) + M_2(M_1 - 1) + PE \tag{6.4}$$

where M_i represents the number of routers in the i^{th} dimension and PE is the number of processing elements. For instance, in an 4x4 2D Mesh based NoC, this yield 40 links. In 16-node ML-Mesh topology, this yield 20 links.

6.2.3 Average Hop count

The network throughput depends on the number of links and the average number of hops of the NoC. In the Mesh based NoC, the average number of hops is given by

$$Averagehop_{Mesh} = \frac{M_1 M_2 (M_1 + M_2) - M_1 M_2}{2(M_1 M_2 - 1)} \tag{6.5}$$

Where M_i is the number of routers in the i^{th} dimension.

In the ML-Mesh based NoC, the average number of hops is given by

$$Averagehop_{ML-Mesh} = \frac{(\frac{D_x}{M_x} + \frac{D_y}{M_y})}{3} \tag{6.6}$$

where D_x and D_y are the number of routers x and y dimensions. M_x and M_y are the number of processing elements in x and y dimensions.

6.2.4 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:

$$Avg_{lat} = 1/N \sum_{i=1}^{N} Li \tag{6.7}$$

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.

6.2.5 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:

$TP = (\textit{Total Messages completed * Message length})/(\textit{Number of PE blocks*Total Time})$ (6.8)

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.

6.2.6 Area

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

$$NoC_{Area} = Routers_{Area} + Links_{Area} \tag{6.9}$$

$$Router_{Area} = IB_{Area} + RCL_{Area} + Crossbar_{Area} \tag{6.10}$$

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

6.2.7 Power

The total power consumed by the NoC architecture can be breakdown 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 as follow

$$P_{NoC} = P_{router} + P_{link} + P_{Interfaces} + P_{clk} \tag{6.11}$$

$$P_{router} = P_{FIFO} + P_{routelogic} + P_{allocator} + P_{crossbar} \tag{6.12}$$

6.3 RESULTS AND DISCUSSION

All the modules in P-NoC such as Topology, Router microarchitecture and Traffic have been implemented in Verilog. We customize the P-NoC for the Mesh and ML-Mesh NoC topologies as shown in Fig. 6.2. The performance of 16-node Mesh and ML-Mesh NoC topologies analyze using various configurations of the input buffer, virtual channel and standard synthetic traffic patterns. The experimental setup employed in

evaluating the NoC architecture is shown in Table 6.2.

Table 6.2: Experimental Setup Details

Experimental Setup Details	
Topology	16-Node Mesh & ML-Mesh Topology
Buffer type	FIFO Buffer
Virtual Channel(VC)	2 & 4
Arbiter type	Round robin
Router Pipeline depth	3-stage
Flit size	32,64
Buffer depth	4, 8, 16 and 32
Packet length	4 flits
Flow control	Credit based
Routing algorithm	Dimensional order routing(DOR)
Synthetic traffic patterns	Uniform, Random Permutations, Bit complement, Transpose, Bit shuffle and Nearest Neighbor

6.3.1 FPGA Utilization Results

The NoC area utilization includes the area of the routers and links(wires). The router area includes the area of the input FIFO buffer, route compute logic, allocator and the crossbar. To evaluate the NoC architecture cost, we consider the FPGA logic resources such as LUT(Look Up Table) and FF(Flip-flop) as the cost metrics. To implement any logical function in FPGA, LUT and FFs have been used as the preliminary resources. The Table.6.3 and 6.4 shows the FPGA resource utilization results of the 16-node Mesh and ML-Mesh topologies. Here, we tuned the parameters such as Flit width(FW), Buffer Dept(BD) and Virtual channel to obtain FPGA area utilization. Flit Width (FW) of 32 bits, Virtual channel of 2 and Buffer Depth (BD) has been varied between 4 to 32 in these experiments. Table. 6.3, record the usage of FPGA resources by the Mesh and ML-Mesh topologies. An increase in the LUT utilization of 15%, 17.4%, 7.4% and 3.44% are observed when BD is increased from 2 to 4, 4 to 8, 8 to 16, 16 to 32 respectively for the 16-node Mesh topology. An increment in FF usage by 40% has been observed when we increased BD from 2 to 32 for 16-node Mesh topology. Similar behavior can be observed for 16-node ML-Mesh topology. An increase in the number of VCs in NoC architecture, we observed an increase in the FPGA area

utilization. When increasing the VCs from 2 to 4 with the FW 32-bits and BD of 2, we observed that the LUT and FF utilization increased by 75%, 60% and 60%, 10% for the Mesh and Mesh and ML-Mesh topologies respectively. The 16-node ML-Mesh topology utilizes lesser FPGA resources for identical configuration than that of the 16-node Mesh. This is because of sharing of the channels, router resources and reduced network diameter in 16-node ML-Mesh topology. Increasing FW from 32 to 64-bits results in an increase of FPGA resources from 20% to 26% for Mesh and from 5% to 6% for ML-Mesh topology with BD and VC of 2. This is due to the wider width channel required for flit transmission. The flit width also influences the FPGA area utilization in NoC architecture.

Table 6.3: Synthesis results of 16-node Mesh and ML-Mesh topologies with FW 32 bits, Virtual channels 2,4VCs and BD of 2 to 32 on Artix-7 FPGA board

Flit Width(FW)		32-bits									
Virtual Channels(VCs)		2					4				
Buffer Depth(BD)		2	4	8	16	32	2	4	8	16	32
16-Node Mesh	LUT(%)	20	23	27	29	30	35	41	51	57	60
	FF(%)	5	6	6.3	6.6	7	8	10	10.5	11	12
16-Node ML-Mesh	LUT(%)	5	6	7	8	10	8	10	12	14	16
	FF(%)	1	1.2	1.3	1.4	1.6	1	2	2.3	2.5	3

Table 6.4: Synthesis results of 16-node Mesh and ML-Mesh topologies with FW 64 bits, Virtual Channels 2,4VCs and BD of 2 to 32 on Artix-7 FPGA board

Flit Width(FW)		64-bits									
Virtual Channels(VCs)		2					4				
Buffer Depth(BD)		2	4	8	16	32	2	4	8	16	32
16-Node Mesh	LUT(%)	26	29	33	35	38	41	47	57	63	66
	FF(%)	6	7	8	9	10	9	11	12	13	14
16-Node ML-Mesh	LUT(%)	6	7	8	10	11	10	11	13	15	17
	FF(%)	1	1.3	1.6	2	2.3	2	2.3	2.5	3	3.5

6.3.2 Clock Frequency Analysis

In this section, we compare the clock frequency of the Mesh and ML-Mesh Topologies as shown in Fig. 6.6. The FW is fixed to 32-bit, 4 VCs have been employed and the BD

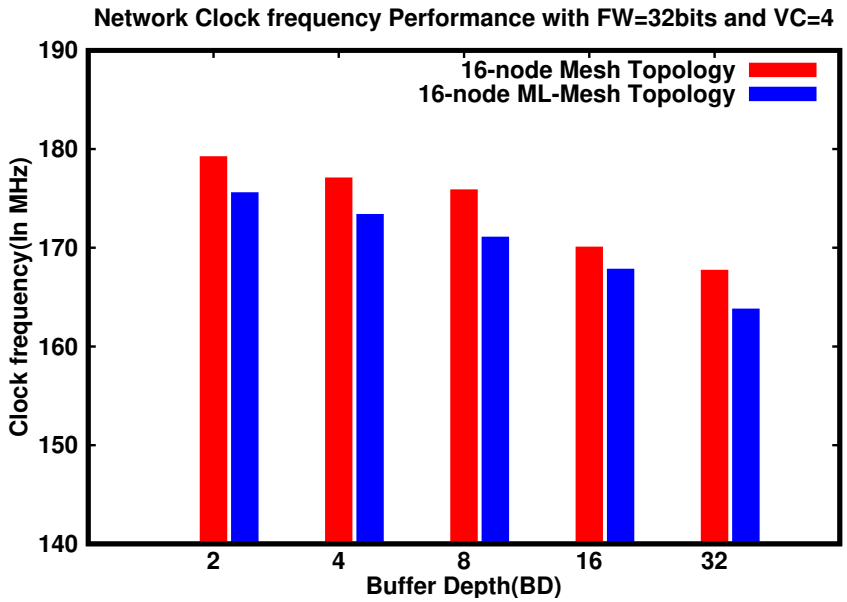


Figure 6.6: Clock frequency performance for Mesh and ML-Mesh topologies

has been varied from 2 to 32. The maximum clock frequency of Mesh is higher than ML-Mesh topology for this configuration due to the size of the router. The Mesh topology has a 5-port router compared to 8-port router of ML-Mesh topology. This influence in Mesh topology has lesser transistor switching and routing delay compared to ML-Mesh topology. Varying the BD from 16 to 32, the operating frequency reduces slightly from 169.99MHz to 168.65MHz and from 168.75 to 159.720 for Mesh and ML-Mesh topologies. This reduction in frequency is due to the higher Buffer Depth value which makes the use of extra memory and the routing and logical resource consumption increases along the critical path. Similar behavior is observed for all other configurations of FW, BD, and VC.

6.3.3 Critical Path Delay Analysis

Two NoC architecture are compared using critical path delay, the Xilinx timing analyzer tool is used for the timing analysis. From Fig. 6.7, it can be seen that the critical path delay of ML-Mesh topology is more than the Mesh topology due to the large routing and logic delay. From our findings, the total critical path delays up to 75% are from routing and the remaining 25% is due to the logic. In the Fig. 6.7 where the FW is 32-bits, the number of VCs being 4 and increasing the BD parameters from 2 to 32,

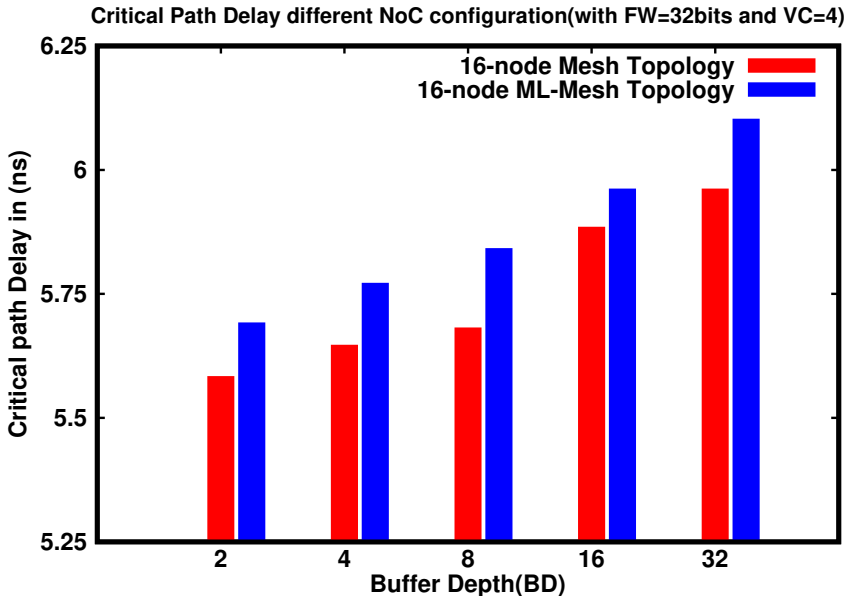


Figure 6.7: Critical path delay for Mesh and ML-Mesh topologies

a slight increase in the critical path delay has been observed. The Mesh topology has lesser critical path delay compared to ML-Mesh topology. This is due to the Mesh topology structure looks more similar to the perfect square grid that matches nearly with the real on-chip core structure compared to ML-Mesh topology. This agrees with the findings by (Genko et al. (2005)).

6.3.4 Latency Analysis

In Fig. 6.8, the latency variation with injection load under Uniform random and Transpose traffic patterns can be observed. The packet latency is directly affected by the injection load. The ML-Mesh topology has lower average packet latency compared to Mesh topology, this is due to the reduction of average hop count between two communicating nodes in ML-Mesh topology. Lower the average packet latency, faster the response time for the application running on the source processing element. Under Uniform traffic pattern, ML-Mesh topology saturates earlier than the Mesh topology, due to less path diversity and higher congestion in ML-Mesh topology. Similar behavior can be observed in the Transpose traffic pattern.

From Fig. 6.8(c) and (d), it can be seen that the Mesh topology performs better compared to the ML-Mesh topology in terms of latency under Random Permutation

and Bit complement traffic patterns. From Fig. 6.8(c) and (d), it can be seen that the saturation throughput of Mesh and ML-Mesh topology increases when the number of VCs is varied from 2 to 4. To reduce the packet contention in the NoC architecture, more VCs are needed.

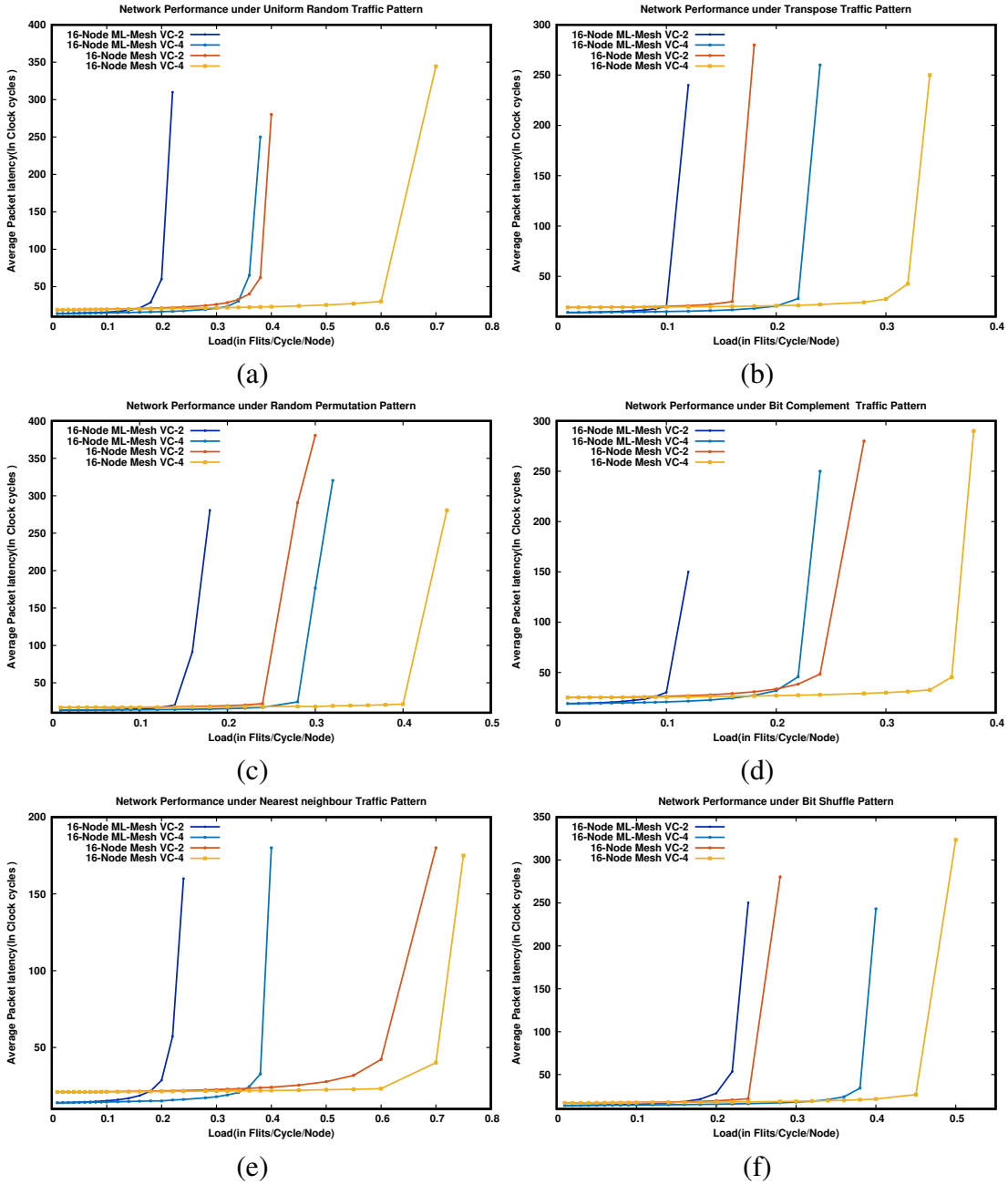


Figure 6.8: Performance comparison of 16-Node ML-Mesh and Mesh NoC topologies under (a)Uniform Random, (b)Transpose, (c)Random Permutation, (d)Bit Complement, (e)Nearest neighbor and (f)Bit Shuffle Traffic Patterns.

Fig. 6.8(e) shows the performance of Mesh and ML-Mesh topologies under the Nearest neighbor traffic pattern. The average packet latency increases as the injection rate increases. This is due to more number of packets injected into the network. The ML-Mesh topology has lower average packet latency compared to the Mesh topology due to more intracluster traffic than on-chip traffic. Under the Nearest neighbor traffic, the communicating nodes are just two hops away. In the ML-Mesh topology, an application running on the source processing elements has a faster response time. From Fig.8(f), it can be seen that the ML-Mesh topology with 4 VC consumes 55% fewer FPGA resources than Mesh topology with 2 VC. The ML-Mesh topology has lower average packet latency and higher saturation throughput than Mesh topology under Bit shuffle traffic pattern.

It can be seen from Fig. 6.8 that the ML-Mesh topology has lower average packet latency than Mesh topology. The Mesh and ML-Mesh topologies saturate at the higher injection rates as the VC is varied from 2 to 4 as shown in Fig. 6.8.

6.3.5 Throughput Analysis

From Fig. 6.9 it can be seen that under the uniform traffic, the ML-Mesh topology provides a lower saturation throughput than Mesh topology. This is due to the fact that the Mesh topology has more number of communication links between two pair of nodes. We observed that Mesh and ML-Mesh topologies saturate at 36% and 20% of traffic loads respectively. Under Transpose, Bit complements, Bit Shuffle and Random permutations traffic patterns, the Mesh topology saturate at 16%, 24%, 25%, and 24% of traffic loads respectively. ML-Mesh topology saturates at the injection rates of 10%, 10%, 20%, and 14% with respect to the Transpose, Bit complements, Bit Shuffle and Random permutations traffic patterns. Under the neighbor traffic pattern, the Mesh and ML-Mesh topologies saturate at 55% and 22% of traffic loads respectively. Under the neighbor traffic the ML-Mesh topology performs better compared to Uniform random traffic. The Mesh and ML-Mesh topologies have $0.53\times$ and $0.1\times$ improvement in performance under neighbor traffic compared to uniform random traffic. Fig. 6.10 shows the network performance of the topologies under all six traffic patterns considering 4

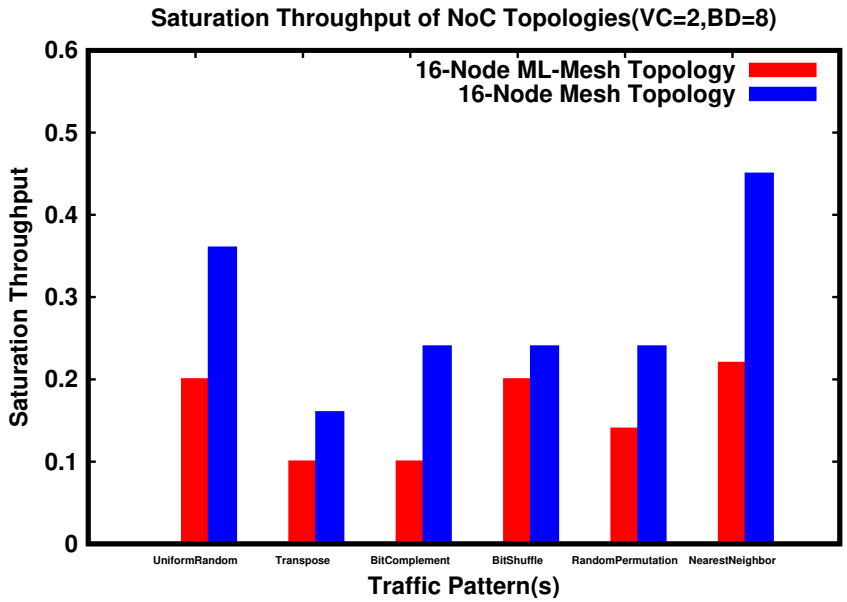


Figure 6.9: Saturation Throughput comparison between 16-node Mesh, ML-Mesh topologies with network configurations of 2 VCs and BD of 8

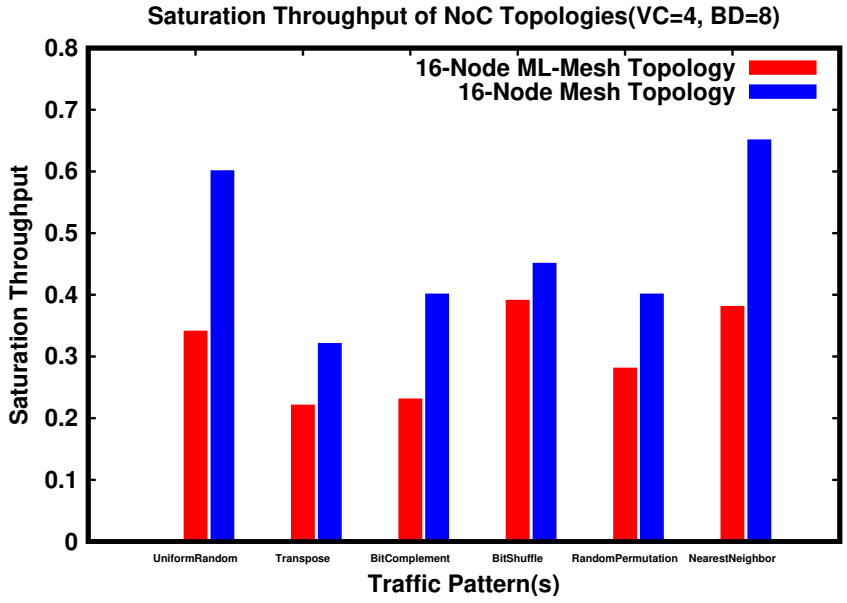


Figure 6.10: Saturation Throughput comparison between 16-node Mesh, ML-Mesh topologies with network configurations of 4 VCs and BD of 8

virtual channels. Under all the six traffic patterns, both the Mesh and ML-Mesh topologies have the higher sustainable capacity as the number of VCs are varied from 2 to 4. Under the uniform random traffic, saturation throughput of Mesh and ML-Mesh topologies are 60% and 34% of traffic load respectively. The Mesh and ML-Mesh topologies have $0.67\times$ and $0.7\times$ improvements as the VCs are varied from 2 to 4. Similar behavior

Table 6.5: Average Hop count of Mesh and ML-Mesh topologies

Number of Nodes 16	
Topology	Average Hop count
Mesh	2.7
ML-Mesh	0.333

can be observed for all the other traffic patterns. In Fig. 6.9 and 6.10, as the VCs are varied from 2 to 4, the increase in saturation throughput is observed for both the topologies. This is due to the effect of VC parameter as more VCs are needed for efficient routing of packets in the network.

6.3.6 Average Hop Count Analysis

For the ML-Mesh, the per hop router latency increases, but the overall latency of the NoC system decreases due to the reduction of average hop count. The hop count decreases due to multiple local ports in the ML-Mesh architecture. The hop count is calculated from Eq. 6.5 and 6.6 for Mesh and ML-Mesh topologies, as shown in Table 6.5. On an average, a flit in Mesh topology traverses 87% more hops than a flit in ML-Mesh topology. The ML-Mesh topology routers have higher connectivity with the increased number of ports in the router. All these features account for the throughput improvement in intracluster traffic load. As shown in Fig. 6.10, under the Nearest neighbor traffic, an improvement in the throughput of ML-Mesh topology has been observed compared to Uniform random traffic. This is due to the reduction in average hop count in ML-Mesh topology. But, the ML-Mesh topology saturates earlier than the Mesh topology due to the contention in the single router connected multiple processing elements.

6.3.7 Power Analysis

The total power consumption of the NoC architecture includes the power consumed by the router, links, interface and other components. Table 6.6 shows the power consumption of the ML-Mesh topology with different configurations of BD and VCs. We observed that the power consumption of NoC architecture increases as we increase

Table 6.6: Power analysis of Mesh and ML-Mesh topologies configurations with(FW 32 bits, VCs 2 to 4 and BD of 2 to 32) on Artix-7 FPGA board

Flit Width(FW)		32-bits									
Virtual Channels(VC)		2					4				
Buffer Depth(BD)		2	4	8	16	32	2	4	8	16	32
16-Node Mesh Mesh	Power(mW)	227	244	251	473	525	396	430	673	699	779
16-Node ML-Mesh	Power(mW)	126	132	134	146	171	161	166	193	203	235
Power Ratio	Reduction(%)	44.49	45.9	46.6	69	67.4	59.3	61.4	71.3	70.9	69.8

the buffer depth from 2 to 32. This is due to the large size buffers at the input ports. When the number of VCs are varied from 2 to 4, increase in the power consumption of the NoC architecture has been observed as the more number of VCs consume large hardware resources. Similar behavior is observed in Mesh topology with various NoC configurations. We compare the power consumption of Mesh and ML-Mesh topologies as shown in Table 6.6. From Table 6.6, we noticed that the Mesh topology consumes 44.5% more power than the ML-Mesh topology. It can be concluded that the ML-Mesh topology is best suitable for designing the power efficient MPSoCs.

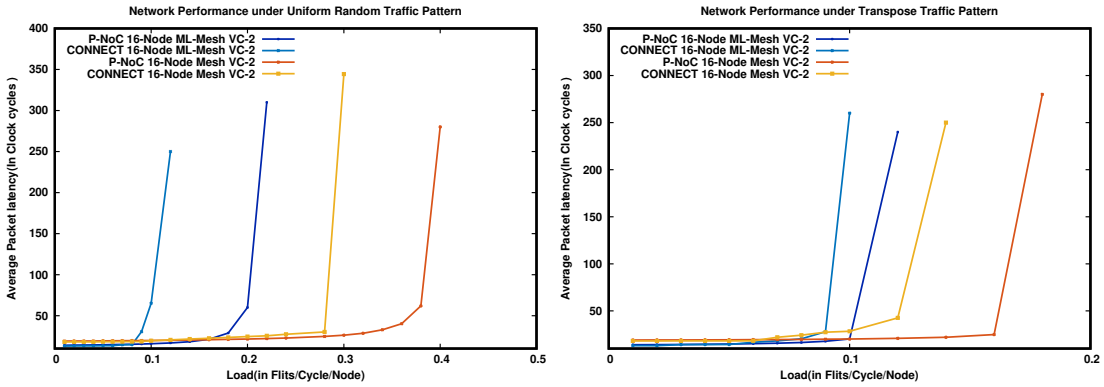
6.3.8 Comparison with the state-of-the-art CONNECT(Papamichael and Hoe (2015))

Table 6.7: Synthesis results of 16-node Mesh and ML-Mesh topologies on Artix-7 FPGA board

Topology	P-NoC Mesh		CONNECT Mesh		P-NoC ML-Mesh		CONNECT ML-Mesh	
Flit width	32	64	32	64	32	64	32	64
Virtual Channel	2	2	2	2	2	2	2	2
Buffer Depth	4	4	4	4	4	4	4	4
Resource Utilization	%	%	%	%	%	%	%	%
Number of Slice LUTs	23	29	25	32	5	7	5	8

The proposed work has been compared with the state-of-the-art CONNECT NoC generator tool (Papamichael and Hoe (2015)). The 16-node ML-Mesh and Mesh topologies HDL code were generated from the CONNECT online NoC generation tool by configuring the router microarchitecture components. The CONNECT design implements

all the input memory buffers on soft logic elements. This causes high resource utilization and increases the critical path delay of the CONNECT design, which reduces the performance of the architecture. The P-NoC framework maps and routes NoC architecture efficiently on the FPGA to consume fewer resources. This reduces the critical path delay improving the performance of P-NoC compared to CONNECT. Table 6.7 shows the area utilization results for the CONNECT and P-NoC topologies by considering of 16-node ML-Mesh and Mesh. The proposed P-NoC architecture consumes 9.3% and 13.33% fewer FPGA LUT resources compared to the CONNECT architecture of Mesh and ML-Mesh by configuring router microarchitecture of VC 2, BD 4 and FW of 64-bit. For all configuration of 16-node ML-Mesh and Mesh topologies of P-NoC architecture consumes fewer hardware resource than CONNECT architecture, this is due to the efficient mapping of NoC topology on FPGA resources.



(a) (b)

Figure 6.11: Performance comparison of P-NoC and CONNECT topologies under (a)Uniform Random, (b)Transpose.

The network performance comparison of the ML-Mesh and Mesh topologies of P-NOC and CONNECT are illustrated in the Fig. 6.11 (a) and (b) considering the Uniform random and Transpose traffic pattern. The ML-Mesh has lower average packet latency than Mesh topology, this is due to the cluster based communications in ML-Mesh. The P-NoC's ML-Mesh and Mesh has 54.2% and 30% higher saturation throughput than CONNECT's ML-Mesh and Mesh topologies under uniform traffic pattern. For transpose traffic, P-NoC has 22.2% and 28.57% higher saturation throughput than CONNECT for the ML-Mesh and Mesh topologies respectively.

6.4 SUMMARY

P-NoC: an FPGA-based paratermized framework for design space exploration of NoC architecture is presented in this chapter. The performance analysis and design trade-off of two NoC architectures namely Mesh and ML-Mesh topologies using FPGA. P-NoC contains a fully parameterized Topology, Router and Traffic generation and Receptor modules. The experimental results show that the Virtual channel(VCs), Flit width(FW) and Buffer depth(BD) parameters contribute to higher FPGA resource utilization and influence the performance of NoC architectures. The FW, BD and VCs have significant impact on FPGA resource utilization.

CHAPTER 7

DESIGN OF LOW LATENCY AND AREA EFFICIENT ROUTER ARCHITECTURE FOR NOC USING FPGA

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.

7.1 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. In order to overcome this issue, Virtual Channel (VC) buffers have been introduced

(Becker (2012),Dally (1992)). The express virtual channels have been employed in the applications demanding high throughput which leads to complex router microarchitecture (Kumar et al. (2007)).

The bufferless router has been 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, 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 affects 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 has been 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 has been proposed in this work. 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.

7.2 RELATED WORK

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

The efficient NoC router microarchitecture has been proposed in (Becker (2012)). The nonatomic VC reallocation method employed in router architecture. The full crossbar architecture has been employed in the router microarchitecture. This results in the increased area overhead of the router. In the a look-ahead routing technique (Galles (1997)), the route computation is done advance in a neighboring router and this routing

information will be 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 precomputed output port. The switch allocation with speculation (Peh and Dally (2001)) has been 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 will be an increase in unsuccessful speculation which leads to the inefficient use of the speculation technique. The technique of precomputing arbitration has been proposed by Mullins et al. (Mullins et al. (2004)). Employing this technique, the critical path delay has been reduced for the separable input-first VC allocator. In the pipeline stages, the switch allocation stage has been 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 microarchitecture has been presented in (Lu et al. (2011)). The two clock cycle router architecture is designed by combining the VC and switch allocation. The atomic VC allocation has been 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 the higher priority for the non-header flit requests. There can be a possibility where it may not be able to assign the OVC for a header flit granted by the switch allocation. A prototyping platform for many-core SoCs employing the low latency network-on-chip has been proposed in (Monemi et al. (2017)). ProNoC is supports emulation of Torus and Mesh topologies. ProNoC employs the full crossbar architecture to implement routers. This will lead to a higher area overhead and increased latency.

The express VCs (Kumar et al. (2007)), dynamic allocation of VC (Nicopoulos et al. (2006)) and flit reservation flow control (Peh and Dally (2000)) have been proposed for high throughput. These designs are more complex which leads to more area utilization and increased dynamic power. (Nicopoulos et al. (2006)) improves the 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 has been proposed in (Ramanujam et al. (2010),Ramanujam et al. (2011),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),Soteriou et al. (2009)) has higher zero load latency than a virtual channel router(VCR). This is due 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 has been proposed in (Becker et al. (2012)). This implementation leads to an increase in the hardware cost, delay and power. In (Yan et al. (2015), Yan and Sridhar (2018)), predefined priority cooperation and centralized priority management based round-robin arbiter has been proposed. These design 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.

7.3 LBNOC-FPGA BASED BYPASS NOC FRAMEWORK

LBNoC framework contains components on both the software and the hardware partitions of the FPGA. The operations that are being 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 calculating. The hardware side includes the NoC router microarchitecture, programmable logic, memory and interfaces, flow control and the packet traversal. Fig. 7.1 shows the architecture of the LBNoC framework in which the Xilinx Zynq 7000 ZC702 SoC has been 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. 7.1, USB-UART driver has been employed to establish the communication between the host PC and the FPGA. This guarantees the performance

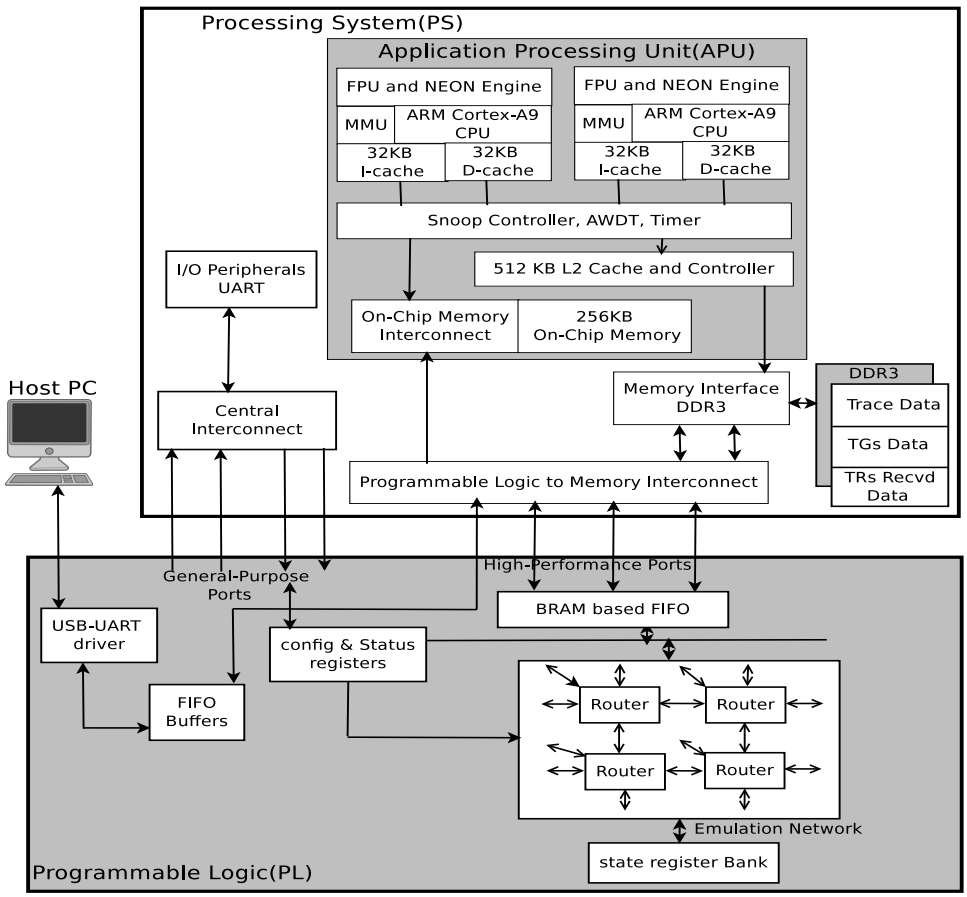


Figure 7.1: The overall architecture of LBNoC-framework implemented on Xilinx Zynq 7000 ZC702 SoC. The PS consists of two core ARM Cortex-A9 processors and the PL has Artix-7 FPGA

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 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.

7.3.1 Hardware Components

The Network-on-chip(NoC) architecture has been implemented on the PL side of the Zynq 7000 SoC. The two stage pipeline router architecture has been designed. The router microarchitectural parameters such as flit width, buffer depth, virtual channel,

ports, routing algorithm, link width and topology are configurable in the LBNoC framework.

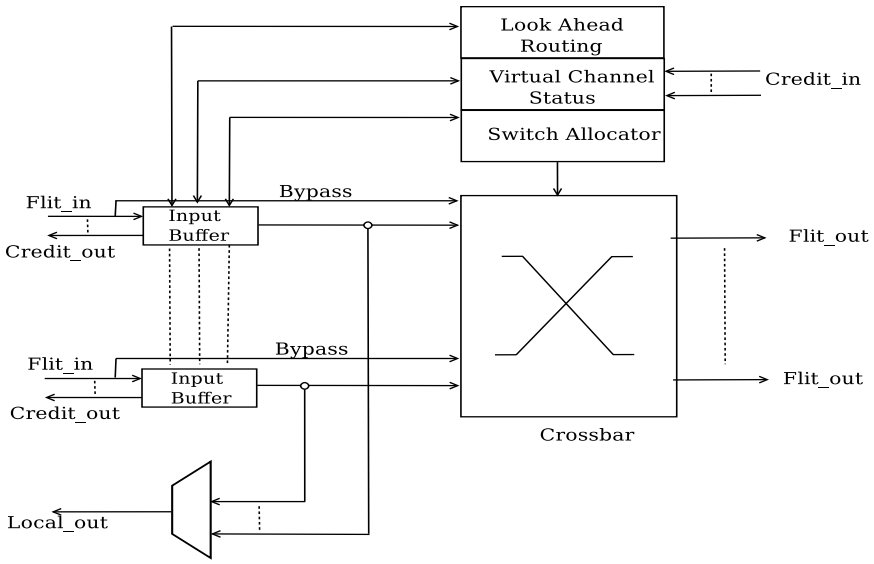


Figure 7.2: Two clock cycle Low latency router architecture implemented in LBNoC framework(The router is highly parameterized with combined VC and Switch allocation stages)

7.3.1.1 NoC Router

Fig. 7.2 shows the LBNoC router architecture. The two-stage pipelined router architecture employing the lookahead bypass technique comprises of 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.

7.3.1.2 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 will be a need for large width multiplexers (Monemi et al. (2017)) that leads to large area utilization. In LBNoC design, large width multiplexers are replaced with a single, dual port Block RAM(BRAM)

memory and two multiplexers with narrow width as shown in Fig. 7.3. 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.

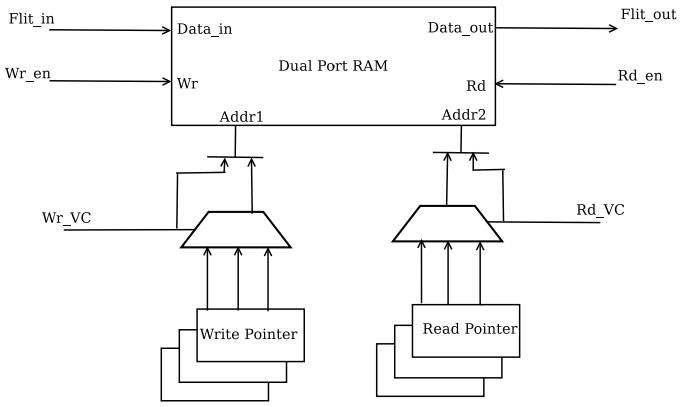


Figure 7.3: The architecture of Input buffer employed in designing low latency router

Fig. 7.3 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(Block RAM(BRAM)). When the size of the input buffer is small, LBNoC utilizes the DRAMs to map input buffers. In case of large input buffers, the BRAMs are used instead of the DRAMs thus improving the performance of the LBNoC.

Table 7.1: 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)

7.3.1.3 Routing Algorithms

The incoming flits will be sent to the route computation module to determine the output port. The framework supports deterministic, table based and minimal adaptive routing for regular and custom topologies. Table based routing has been implemented for implementing 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 have been used in the LBNoC work to store the routing tables. A single DRAM will be typical of 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.

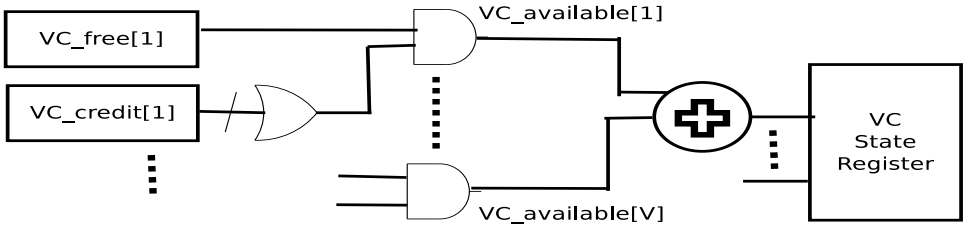


Figure 7.4: Free VC availability check and count

7.3.1.4 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, also influence the performance and area overhead significantly. From Table 7.1, 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. 7.4. As the queues of free VCs selection require a single arbitration stage, it results in faster execution and less area overhead. The conventional 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.

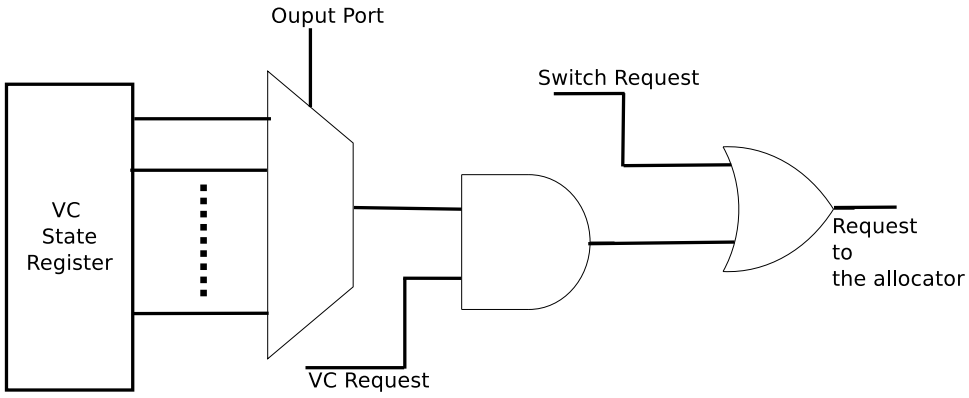


Figure 7.5: Request filter logic

In order 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 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. 7.5. The request for output channel is filtered, where 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 has been implemented as shown in Fig.7.6. 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 rest of the requests(Virtual

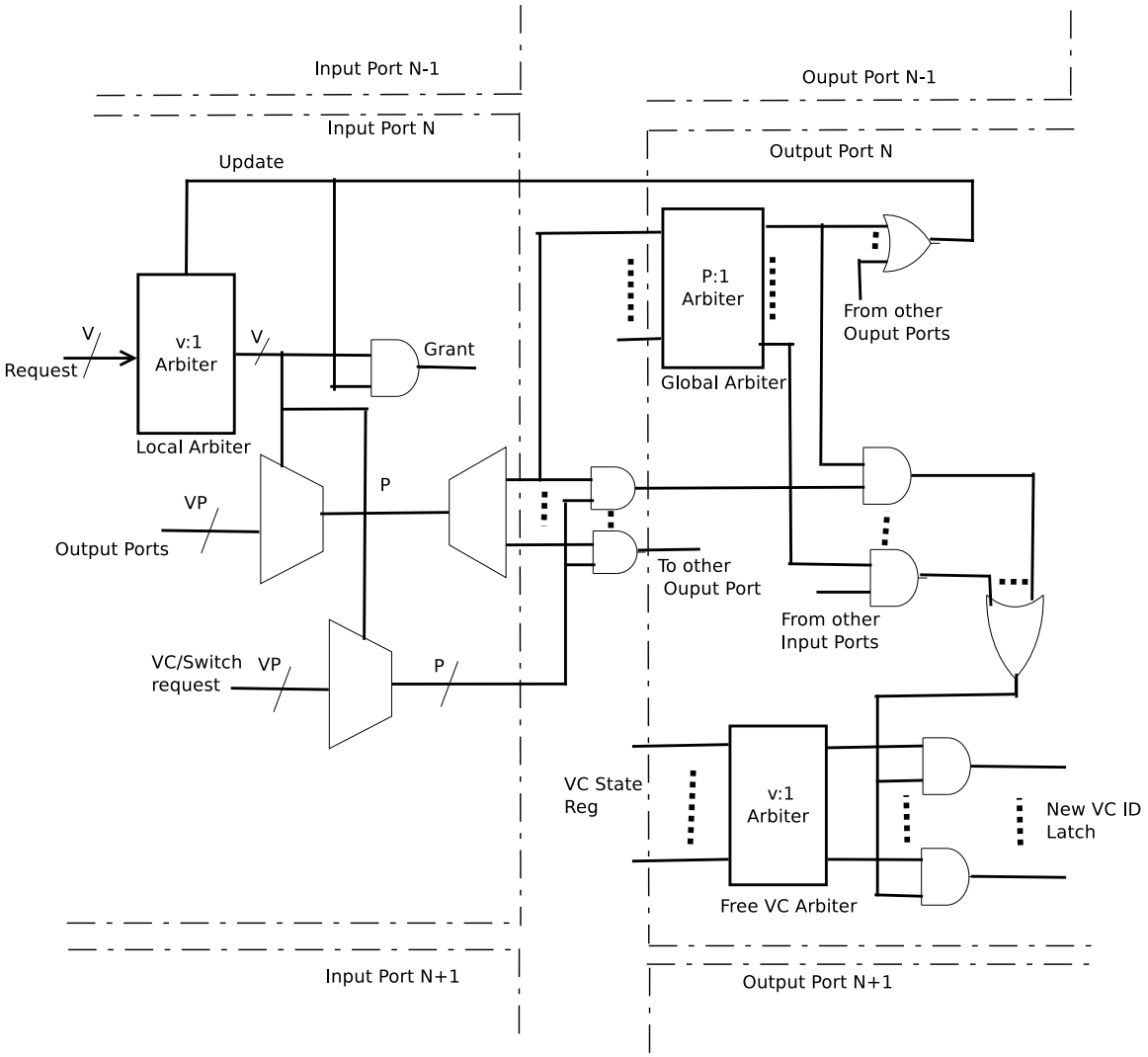


Figure 7.6: Parallel Virtual Channel and Switch allocator

channel or switch). The header flit requests both VC and SW allocator. The non-header flit requests only switch allocator, as the VC allocation has already been done by the header flit. The VC allocation done for the header flit is reserved for the entire duration of the packet. But, 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 signal, the VC and switch requests are designated with “1” and “0” bits respectively. The selected output port will be 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 arbi-

tration 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 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 and free VC has been eliminated by the request monitoring logic. The resultant will be at most one grant can be made available at the output channel, always successful result is returned by VC allocation. The free VC selection is not performed on the critical path, therefore it is performed parallel 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.7.2. The LBNoC supports round-robin, weighted round-robin and fixed priority arbitration schemes.

Table 7.2: 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)

7.3.1.5 Crossbar

The decomposed crossbar architecture has been 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 the 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 has been reduced.

7.3.1.6 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 has been established employing the hybrid flow control. As the packet stays across a lesser number of routers, less number of network resources such as Virtual channels 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.

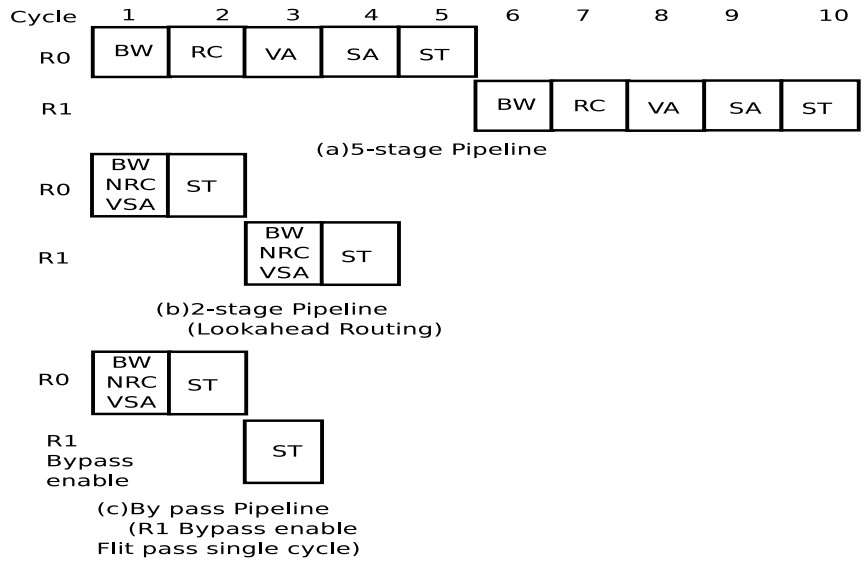


Figure 7.7: Pipeline stages of conventional and LBNoC router architecture

7.3.1.7 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. 7.7 (b) when the above conditions are not satisfied. If the above conditions are satisfied, the pipeline delay of the flit is just the single clock cycle.

7.3.1.8 Pipeline Architecture

Fig.7.7 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 architecture, the pipeline structure of the router is optimized by employing lookahead routing and combining the Nonspeculative Virtual channel and switch allocation stages. Also, the Buffer write, Lookahead Route compute(NRC) 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.

7.3.1.9 Adaptive Routing Algorithm

Fig. 7.8 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

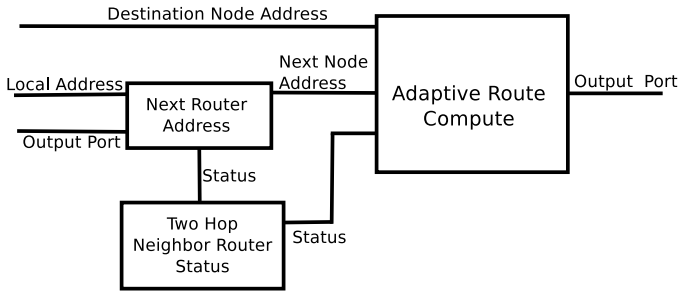


Figure 7.8: Proposed adaptive look-ahead routing module

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 flipflops (Papamichael and Hoe (2015)). The adaptive routing employs 2-bit values for indicating the congestion 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 in order to make the correct routing decisions. The adaptive routing algorithm (Parane et al. (2018)) with negligible area overhead compared to XY routing algorithm of 1.66% is used in the proposed LBNoC architecture.

7.3.2 Software Components in the LBNoC

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

7.3.2.1 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 has been incorporated in the TG module. The TG module is responsible for generating the traffic. Various types of synthetic traffic patterns such as uniform random, bit complement, transpose, bit reverse, hotspot and tornado are supported by the framework.

7.3.2.2 Source Queue

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

7.3.2.3 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 head flit. Also, it monitors the number of total packets received, number of packets transmitted, average packet latency.

7.3.2.4 Global Clock Generation

The Global Clock Generation (GCG) module is responsible to maintain 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.

7.4 RESULTS AND DISCUSSION

The microarchitectural components of the NoC architecture have been implemented in Verilog. Synthesis results have been extracted from Xilinx Vivado 2016.2. Results include resource usage on the Xilinx Zynq 7000 SoC (ZC 702 board). The hardware side has been implemented on the Artix 7 FPGA and the software side has been implemented on the ARM Cortex 9 soft processors. The software side accounts for the traffic generators and traffic receptors. The UART interface has been employed for transceiving the traffic from the software to the hardware side of the framework. Another USB-UART interface has been employed for communication between the host PC and the PL side of the Zynq SoC.

The experimental setup employed in evaluating the LBNoC work is shown in Table 7.3. Various configurations of Buffer depth, Number of VCs, Flit width, Number of I/O ports, traffic patterns and different topologies have been employed to evaluate the

Table 7.3: Experimental Setup Details

Experimental Setup Details	
Topology	4×4 & 5×5 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

LBNoC framework.

7.4.1 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 Block RAM(BRAM). Table 7.4 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 7.4, we observed that mapping memory logic on the Register RAM consumes more FPGA resource 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 7.5, we fixed the Buffer depth to 15 flits and flit width has been 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 block with flit size of 256 bit compared to Register RAM and LUTRAM.

Tables 7.6 and 7.7 show the FPGA BRAM resource utilization for various configurations of NoC router input buffers targetting Zynq 7000 SoC. The results in the Tables

Table 7.4: 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 7.5: 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

7.6 and 7.7 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 7.6 and 7.7 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 will play a major role in the performance of NoC architecture.

Table 7.6: 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

In designing the low latency router architecture in LBNoC, we employed merging of all input VCs buffer at input ports. Table 7.8 shows the resource utilization of merged buffer implementation and CONNECT's (Papamichael and Hoe (2015)) conventional buffer implementation. It can be seen from Table 7.8 that the merged implementation

Table 7.7: 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

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. Increase in the number of LUTs increases the critical path thus reducing the operating frequency.

Table 7.8: 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 queues of free selector have been considered for implementing the VC allocator in low latency router architecture. Table 7.9 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)).

The decomposed crossbar architecture has been implemented to directly route the packet which is destined for the local input port based on the look-ahead routing. Table 7.10 shows the resource utilization of Full crossbar (Monemi et al. (2017)) and the Decomposed crossbar. The Decomposed implementation consumes 10.64 % fewer LUTs

Table 7.9: 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

than the Full crossbar.

Table 7.10: 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

7.4.1.1 Topology Implementation

The 4×4 and 5×5 prototypes have been implemented on LBNoC to demonstrate the resource utilization.

Table 7.11: Synthesis results of Mesh topology of size 4×4 and 5×5 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	
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

Table 7.11 shows the synthesis results for the designed topologies. Increasing the flit width increases the FPGA resource utilization. Considering the 4×4 Mesh topology, when the flit width is increased from 32 to 64 bits, the LUT utilization will be increased from 26% to 35% for buffer depth 4 and 2 VCs. Similar behaviour is observed for the 5×5 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 is used. When the 64-bit flits are used, 36Kb BRAMs are utilized. This is because, the Xilinx 18Kb BRAM can be configured to accommodate the 512 flits of each of 32 bit width. As we increase the flit width to 64 bit, the 18Kb BRAM will not be sufficient to map the required width. Hence, the 64-bit flits will occupy the 36Kb BRAMs by configuring 512 of flits each of 64 bits. As the topology size increases, the FPGA resources also increase. The 5×5 with 4 VCs and 64 bit flit width consumes 65.30 % and 40 % more LUTs and 36Kb BRAMs respectively than the 4×4 Mesh topology.

7.4.2 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 7.3 shows the basic experimental setup of latency analysis. From Fig. 7.9, 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. 7.9(a), the 4×4 Mesh topology has a lower average packet latency compared to 5×5 Mesh topology. This is due to the lower diameter of the 4×4 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 4×4 Mesh topology. The 5×5 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 4×4 Mesh topology. For the 5×5 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,

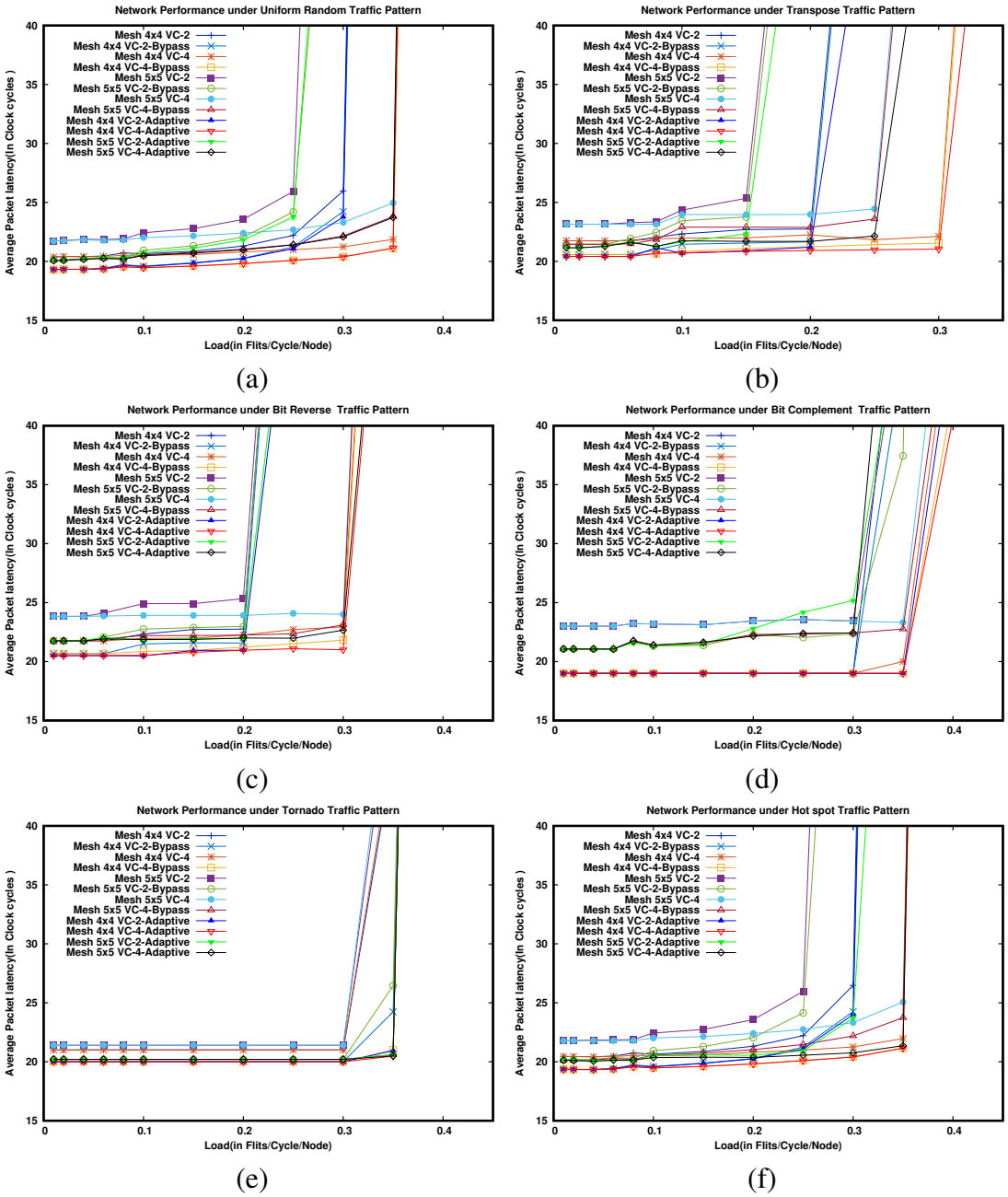


Figure 7.9: Performance comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns.

Bit complement, Tornado and Hot-spot respectively have been observed. 4×4 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. 7.9(d), it can be observed that 4×4 with 2 VCs considering the bypass and baseline two clock cycle architecture saturates

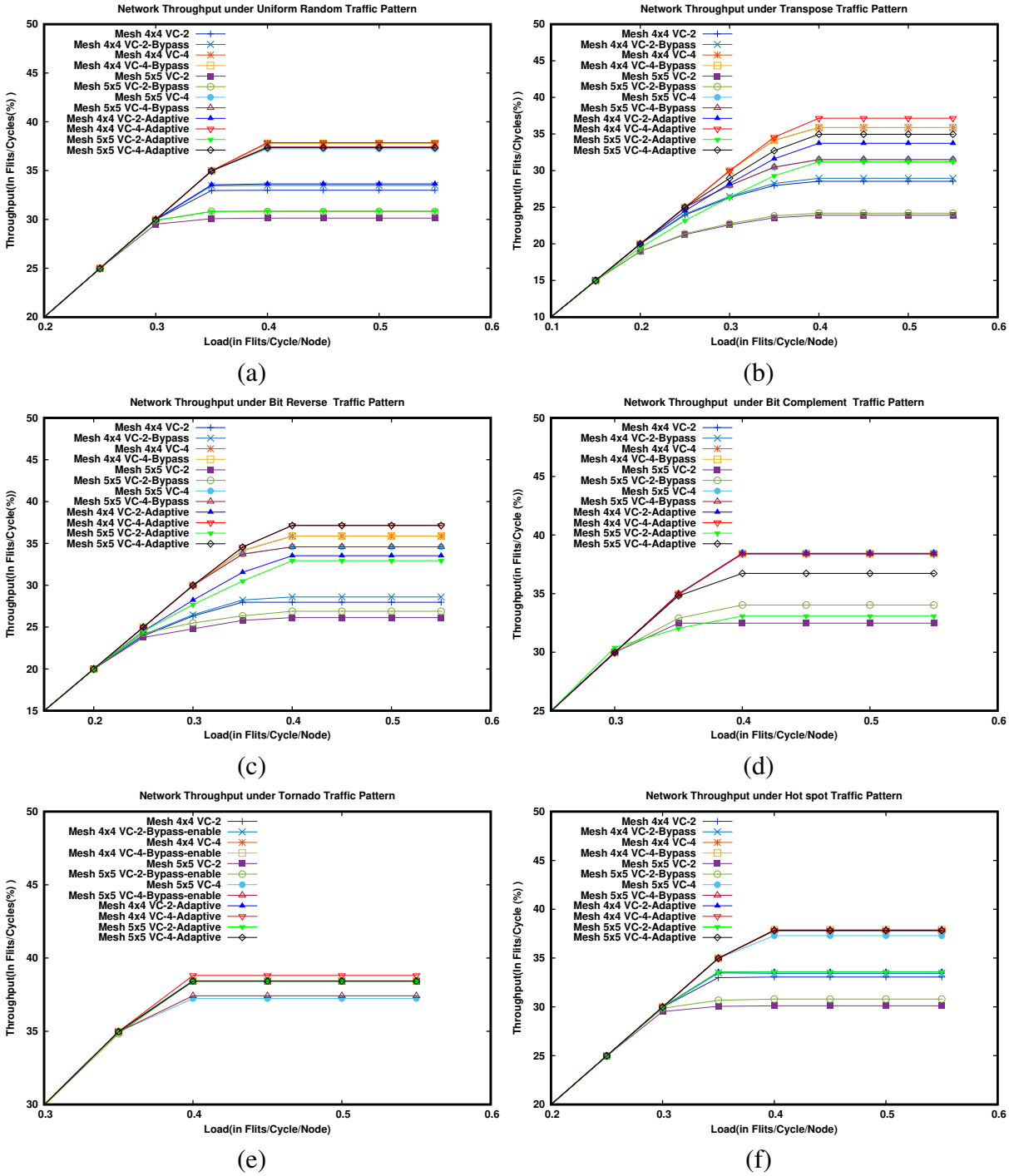


Figure 7.10: Throughput comparison of 4x4 and 5x5 NoCs topologies with various configurations under a different type of traffic patterns.

early compared to all other configurations. Fig. 7.9(e) shows that 5×5 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 4×4 Mesh topol-

ogy with 4 VCs and the Bypass path yields better performance compared to all other configuration as shown in Fig. 7.9(e).

7.4.3 Throughput Analysis

Fig. 7.10 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 the more number of flit in the network. The 4×4 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.7.10(a), (b),(c) and (f). From Fig. 7.10(d) it can be seen that 4×4 Mesh topology with 2 VC baseline architecture has lower saturation throughput compared to all other configuration. The 5×5 Mesh topology with 2 VC single cycle Bypass path has lower saturation throughput compared to all other configuration as shown in Fig.7.10(e).

7.4.4 Power Analysis

Table 7.11 shows power consumption for various configurations of 4×4 and 5×5 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 virtual channel, flit width, buffer depth and size of the topology. For example, from 374mW for a low configuration (4×4 Mesh topology with VCs of 2, Buffer Depth of 2 and Flit width of 32) to 1481mW for a highest configuration (5×5 Mesh topology with VCs of 4, Buffer Depth of 4 and Flit width of 64) as shown in Table 7.11. 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.

7.5 COMPARISON WITH THE STATE-OF-THE-ART NOC ARCHITECTURES

7.5.1 Comparison with FPGA state-of-the-art

7.5.1.1 Area, Frequency and Power

Table 7.12 shows the resource utilization and maximum operating frequency of different NoC architectures. 4×4 Mesh topology implemented employing the LBNoC NoC

Table 7.12: Resource utilization and Maximum operating frequency of Different NoC configurations considering 4×4 Mesh topology

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

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 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 less FPGA resource utilization.

7.5.1.2 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 have been conducted. Fig. 7.11 shows the load vs delay graph for various synthetic traffic patterns in 5×5 Mesh topology. The average latency increases with an increase in the packet injection rate.

Fig. 7.11(a) shows the load vs delay graph of 5×5 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. 7.11(b), it can be seen that LBNoC architecture has 30% and 15% lesser average packet latency than CONNECT and ProNoC architectures under Transpose traffic pattern. Fig. 7.11(c),(d),(e) and (f) shows the load vs delay graph of 5×5 Mesh topology under Tornado, Hot-spot, Bit complement and Bit Reverse traffic patterns. The LBNoC has less average packet latency compared to CONNECT and ProNoC. It can be seen that the LBNoC architec-

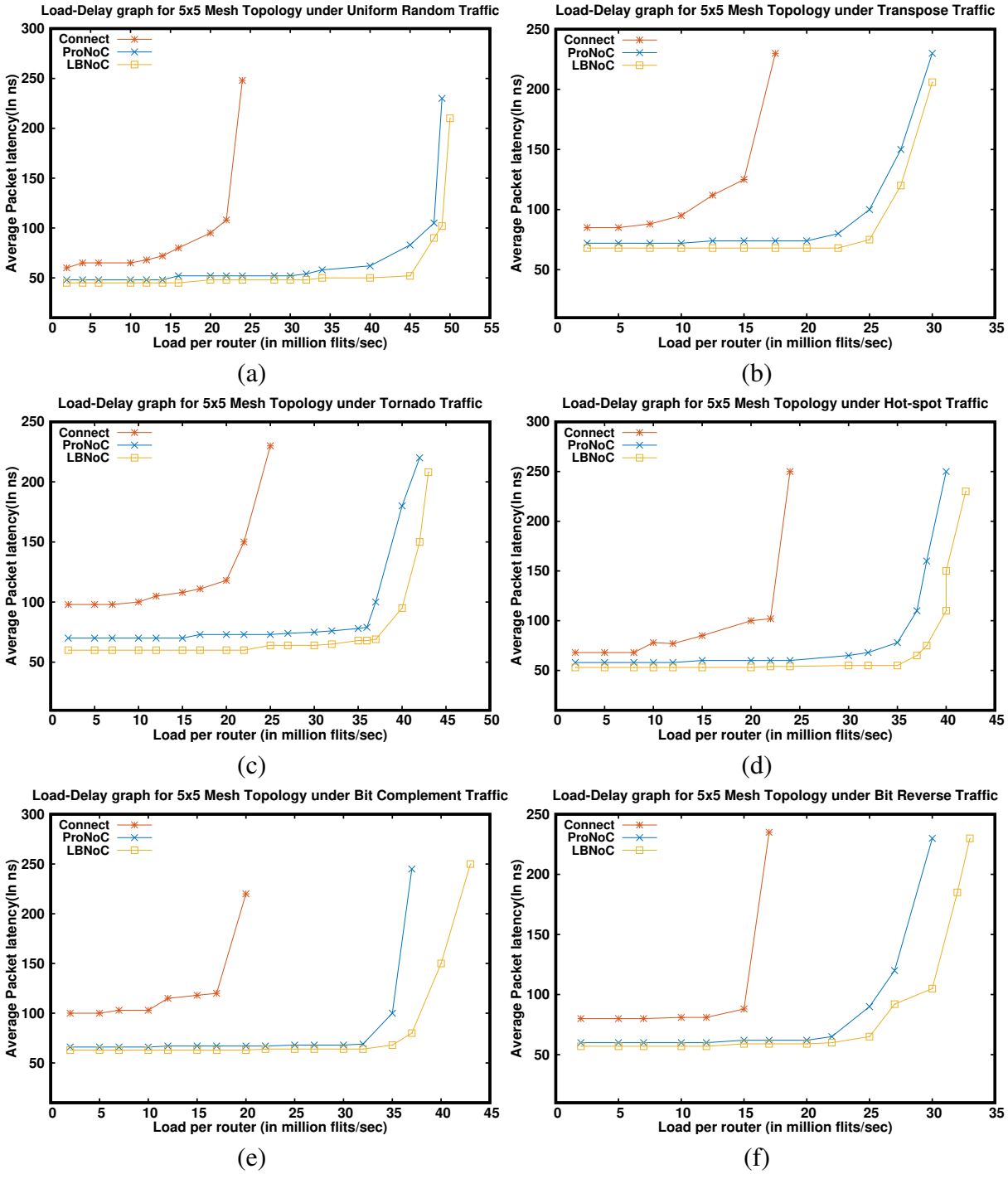


Figure 7.11: Average packet latency comparison between LBNoC, CONNECT (Papamichael and Hoe (2015)) and (ProNoCMonemi et al. (2017)) considering different types of traffic patterns

ture 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.

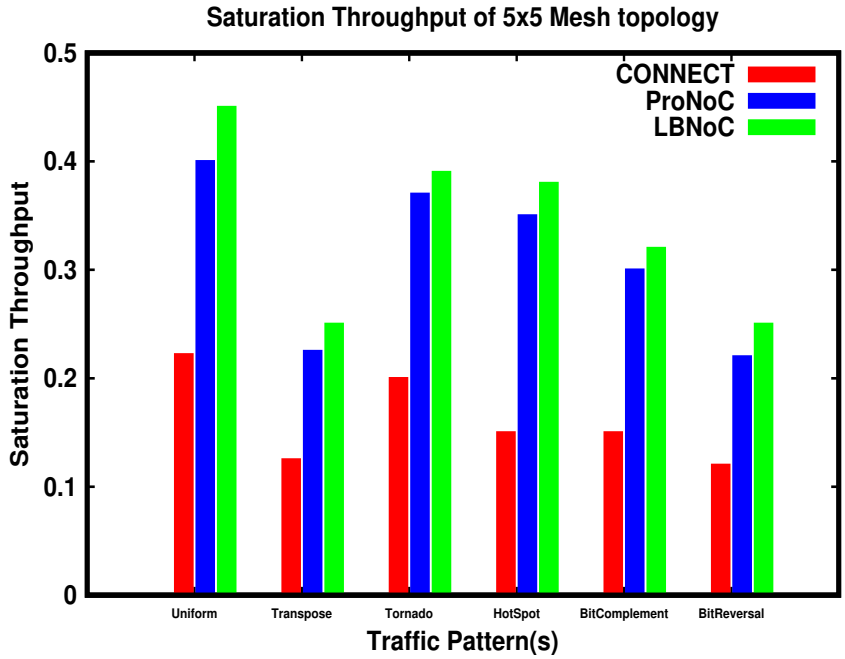


Figure 7.12: Throughput comparison of LBNoC, Pronoc and Connect NoC architecture

Fig. 7.12 shows the saturation throughput of the CONNECT, ProNoC and LBNoC architectures under Uniform, Transpose, Tornado, Hot-spot, Bit complement and Bit 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 2.16x and 1.06x with respect to CONNECT and ProNoC architectures. Similarly, under the Transpose traffic pattern, LBNoC achieves an improvement of 2.6x and 1.16x with respect to CONNECT and ProNoC architectures. LBNoC has higher saturation throughput of 0.95x,1.5x,1.13x and 1.08x under Tornado, Hot spot, Bit complement and Bit reversal traffic patterns respectively compared to CONNECT. LBNoC has 0.05x, 0.08x, 0.07x and 0.13x higher saturation throughput under Tornado, Hot spot, Bit complement and Bit reversal traffic patterns respectively compared to ProNoC architecture.

7.5.2 Comparison with ASIC targetted state-of-the-art NoC architectures

7.5.2.1 Area, Frequency and Power

Fig. 7.13 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 synthesized 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 (Group. (2012)), 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 upto 10.5%, 5.3% and 15.84% can be observed in proposed router with respect to SOTA, Priority and shared buffer NoC router architectures. The proposed NoC architecture operates at higher frequency than the state-of-the-art NoC router architectures.

Table 7.13 shows the resource utilization, maximum operating frequency and power consumption of different NoC architectures. 4×4 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 higher frequency of 205MHz compared to SOTA(101.72MHz), Priority(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 architec-

ture.

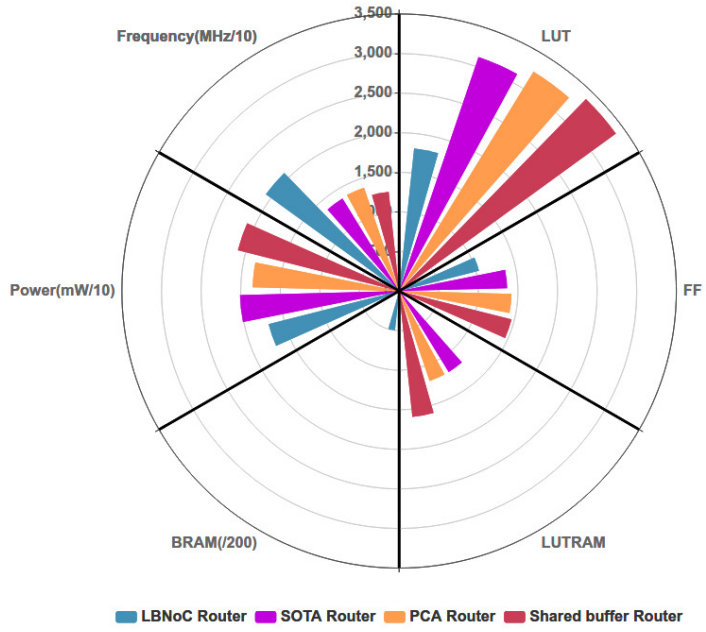


Figure 7.13: Area, Frequency and Power utilization of various router architectures

Table 7.13: Resource utilization and Maximum operating frequency of Different NoC configurations considering 4×4 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

7.5.2.2 Latency and Throughput Analysis

The network performance of LBNoC is compared with publicly available state-of-the-art RTL of VC-based router (Group. (2012), Yan et al. (2015)) NoC architectures. The results for Shared buffer architecture (Soteriou et al. (2009)) have been obtained by modifying SOTA (Group. (2012)) RTL code. And, the results for PCA have been obtained from (Yan et al. (2015)). Fig. 7.14 shows the average latency comparison between LBNoC, SOTA and PCA under various traffic patterns. 5×5 Mesh topology has been considered for the experiments. An increase in the average latency can be

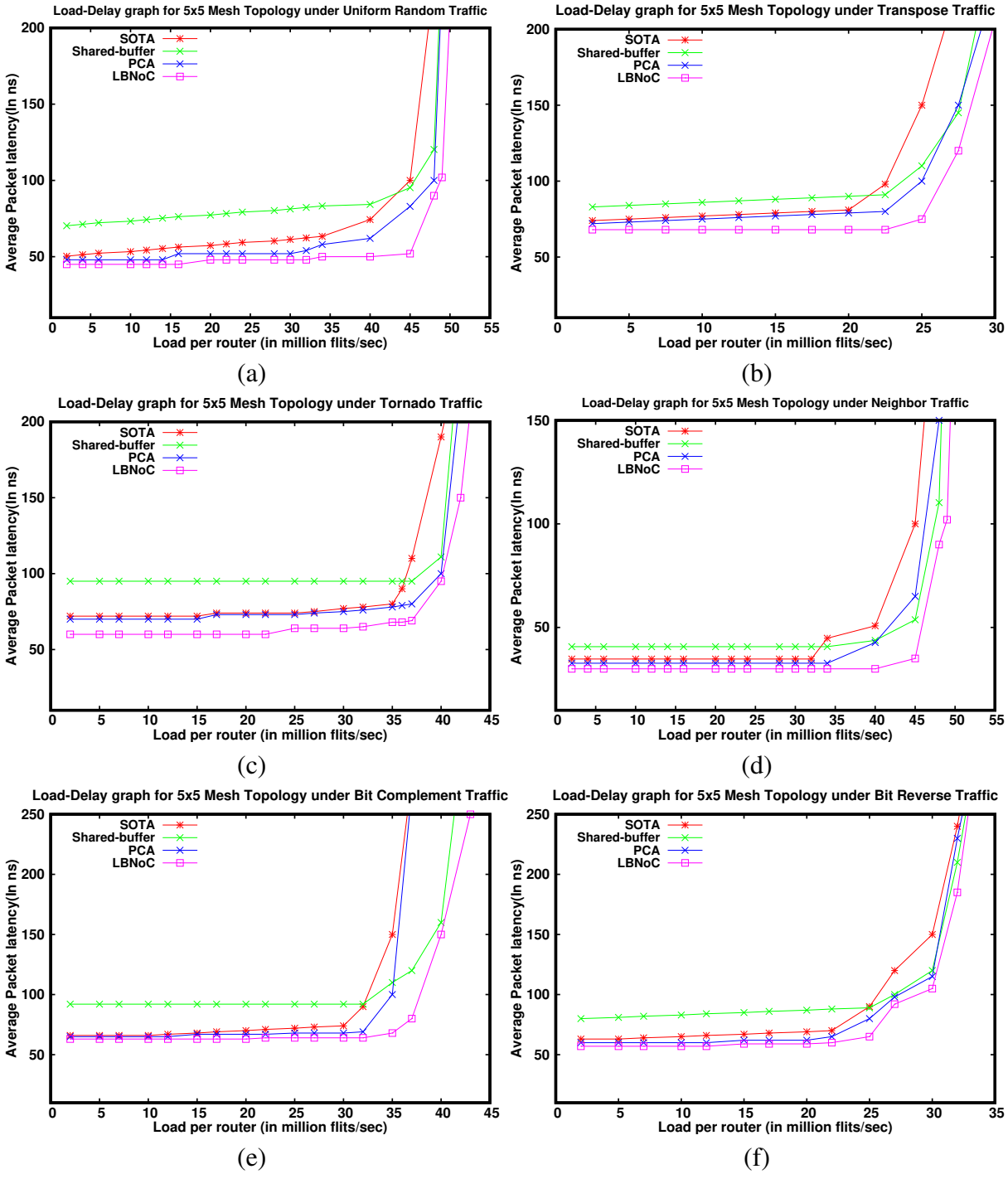


Figure 7.14: Average packet latency comparison between LBNoC, SOTA(Group. (2012)), Shared-buffer (Soteriou et al. (2009)) and PCA (Yan et al. (2015)) considering different types of traffic patterns

observed with the increase in the packet injection rate. Fig. 7.14(a) shows the load vs latency behavior of the 5×5 Mesh topology considering It can be observed that the LBNoC architecture outperforms all the other NoC architectures consistently offering

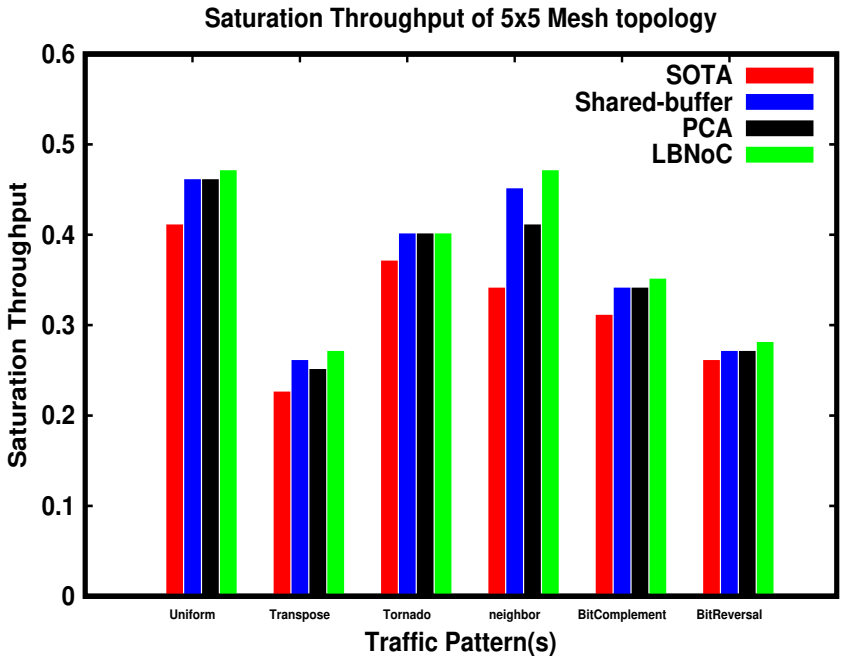


Figure 7.15: Throughput comparison of LBNoC, SOTA, Shared-buffer and PCA NoC architectures

lower packet latency considering all the traffic patterns.

In Fig. 7.14, considering the 5×5 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 traffic patterns compared to SOTA NoC architecture. Similarly, considering the 5×5 Mesh topology LBNoC router architecture, a reduction in average latency by 30.02%, 18.01%, 32.8%, 26.3%, 28.75% and 31.5% have been observed with respect to Shared buffer 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. 7.15 shows the saturation throughputs of 5×5 Mesh topology under the afore-mentioned traffic patterns considering the LBNoC, Shared-buffer and PCA NoC architectures. The 5×5 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.

7.6 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× and 1.18× 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 8

CONCLUSIONS AND FUTURE WORKS

The contributions of this thesis are: The optimization and profiling of Booksim2.0 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. An FPGA-based parameterized framework with flexible communication and traffic generation model for NoC architectures evaluation. And, design of lightweight router architecture for NoCs using FPGAs to achieve high performance.

The thesis initial work involves profiling and software optimization of the Booksim2.0 simulator to analyze the performance. Various software mechanisms have been employed to improve the performance of Booksim2.0 NoC simulator. The OpenMP programming models have been used for parallelizing the sequential code of Booksim2.0. This results 2.93$\times$ speedup over sequential code by simulating the 30$\times$30 Mesh topology on Booksim2.0. The parallelization and vectorization reduced the simulation time of 30$\times$30 Mesh topology from 60 minutes (normal Booksim2.0 simulation time) to 14 minutes and 12 minutes. The average simulation time reduced by 67.31$\times$ for all network sizes of Mesh topology in Booksim2.0 employing the software optimizations.

An FPGA based NoC simulation acceleration framework called YaNoC has been proposed to speed up the NoC simulations. YaNoC supports the design space exploration of various standard and custom NoC topologies. The router microarchitectural 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. A congestion-aware adaptive routing has been proposed to route the packets along the minimally congested path. A reliable router architecture has been designed for NoC systems. Employing the congestion-aware adaptive routing, network latency is reduced by a factor of 55× and negligible area overhead compared to the XY routing algorithm. The reliable router adds extra area cost for the design of fault correction architecture to provide low latency and better fault tolerance. YaNoC consumes 9.29% fewer resources and is 2.5× faster than the CONNECT framework. Also, YaNoC consumes 17.59% fewer resources and 25× faster than the DART simulator. The speedup of 2548× compared to the Booksim2.0 software simulator has been observed using YaNoC.

Only the CLB components(LUTs and FFs) of the FPGA are utilized for mapping the NoC architecture in the YaNoC. The unused DSP tiles of the Xilinx FPGAs are used for mapping the NoC router's functionality. The wide multiplexers of the Xilinx DSP48E1 slices have been used to support the crossbar functionality of the 5-port, buffered NoC router. A reduction of soft logic has been observed employing the proposed technique. By employing the proposed DSP48E1 based crossbar architecture, the topology sizes which exceed the CLB resources of an FPGA can be implemented successfully on the same FPGA without any extra cost of the CLB resources. The implementation of the topologies with DSP crossbar consumes 43%, 44%, and 33% fewer LUTs, FFs, and occupied slices, respectively, compared to the topologies with CLB crossbar implementation. The 6×6 Torus topology saturates at an injection rate of 0.48 under the XY routing algorithm. Employing the Look-ahead routing algorithm, an improvement of 25% has been observed compared to XY routing. The 6×6 Mesh topology with proposed router architecture consumes 23% fewer slices, 41% fewer LUT resources than the CONNECT implementation. The 3×3 Mesh topology with the proposed router architecture consumes 88%, 86%, and 80% fewer FFs, LUTs, and the occupied slices respectively than the DARTs implementation.

An FPGA based parameterized framework called P-NoC for analyzing the performance of NoC architectures based on various design decision parameters has been pro-

posed. P-NoC contains a fully parameterized Topology, Router, and Traffic generation and Receptor modules. The experimental results show that the Virtual channel(VCs), Flit width(FW) and Buffer depth(BD) parameters contribute to higher FPGA resource utilization and influence the performance of NoC architectures. The Mesh and Multi-Local port(ML) Mesh topologies have been considered for the experiments. As the BD is varied from 4 to 8 with a fixed FW of 32-bits and 2 VCs, the FPGA area utilization increased by 17.4% and 16.67% for Mesh and ML-Mesh topologies respectively. The Mesh and ML-Mesh topologies have 0.53× and 0.1× higher saturation throughput under Nearest neighbor traffic compared to uniform random traffic. The ML-Mesh topology yields 75% lesser utilization of FPGA resources compared to the Mesh. The ML-Mesh topology shows an improvement of 33.2% in network latency under localized traffic patterns.

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 4×4 Mesh topology implemented employing the LBNoC architecture consumes 4.5% and 27.1% fewer hardware resources than 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× and 1.18× have been observed for LBNoC architecture concerning ProNoC and CONNECT NoC architectures.

The 3-Dimensional (3D) NoC architectures have been emerging as an area, power-efficient, high-performance communication framework for next-generation computing systems compared to 2-Dimensional (2D) NoC. The presented FPGA based NoC simulation framework supports for design space exploration of 2D NoC architectures. Future efforts can be directed towards the development of a fast and flexible FPGA based NoC simulator, which supports for design space exploration of 3D NoC architectures. The design and evaluation of novel routing algorithms and optimization of router architecture for 3D NoC to achieve higher performance along with the reduction of the area and network latency.

Appendix A

A.1 SHORTEST PATH ROUTING ALGORITHM FOR DMESH TOPOLOGY

The novel routing algorithm for DMesh topology as shown in below code snippet. The shortest path between a source and destination pairs has been achieved in the DMesh topology make use of novel routing algorithm. The detail description is given in the section 4.3

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
ports correspoding 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, NEo will
be the output port.
    if (xc[2:0]<xd[2:0]&&yc[2:0]>yd[2:0])
    begin
        port_num_next = NEo;
    end
    //Condition for SouthEast Output port.  If true, SEo will
be the output port.
    else if(xc[2:0]<xd[2:0]&&yc[2:0]<yd[2:0])
    begin
        port_num_next = SEo;
    end
    // Conditions for rest of the ports can be included as
shown above.
end
endmodule
```

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." *Multim. Tools Appl.*, 28(1-2), 239–256.

Ax, J., Sievers, G., Daberkow, J., Flasskamp, M., Vohrmann, M., Jungeblut, T., Kelly, W., Porrmann, M. and Rückert, 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.

Balfour, J. D. and Dally, W. J. (2006). "Design tradeoffs for tiled CMP on-chip networks." In Egan, G. K. and Muraoka, Y., editors, *Proceedings of the 20th Annual*

International Conference on Supercomputing, ICS 2006, Cairns, Queensland, Australia, June 28 - July 01, 2006, ACM, 187–198.

Balkind, J., McKeown, M., Fu, Y., Nguyen, T., Zhou, Y., Lavrov, A., Shahrad, M., Fuchs, A., Payne, S., Liang, X., Matl, M. and Wentzlaff, D. (2016). "Openpiton: An open source manycore research framework." In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 217–232.

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), 1.

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.

Chang, Y., Wong, D. F. and Wong, C. K. (1996). "Universal switch modules for FPGA design." *ACM Trans. Design Autom. Electr. Syst.*, 1(1), 80–101.

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 *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 249–261.

Chung, E. S., Papamichael, M., 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), 15:1–15:32.

Ciletti, M. D. (2011). chapter Advanced Digital Design with the Verilog HDL, 117–132. Prentice Hall, 2nd Edition, USA., Prentice Hall, First edition, USA.

CMU-SAFARI (2018). "*NOCulator*." http://access.ee.ntu.edu.tw/noxim/index.html).

Constantinides, K., Plaza, S., Blome, J. A., Zhang, B., Bertacco, V., Mahlke, S. A., Austin, T. M. and Orshansky, M. (2006). "Bulletproof: a defect-tolerant CMP switch architecture." In *12th HPCA*, 5–16.

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 Kaeli, D. R. and Moseley, T., editors, *12th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2014, Orlando, FL, USA, February 15-19, 2014*, ACM, 230.

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. and Seitz, C. L. (1986). "The torus routing chip." *Distributed Comput.*, 1(4), 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 Marjamäki (2011). ""Cppcheck, A tool for static C/C++ code analysis".").

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.

Enright, N. and Peh, L. (2009). *On-Chip Networks*.

Fick, D., DeOrio, A., Hu, J., Bertacco, V., Blaauw, D. T. and Sylvester, D. (2009). "Vicis: a reliable network for unreliable silicon." In *Proceedings of the 46th DAC*, 812–817.

Galles, M. (1997). "Spider: a high-speed network interconnect." *IEEE Micro*, 17(1), 34–39.

Genko, N., Atienza, D., De Micheli, G., Mendias, J. M., Hermida, R. and Catthoor, F. (2005). "A complete network-on-chip emulation framework." In *Design, Automation and Test in Europe*, 246–251 Vol. 1.

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.

Group., S. C. V. A. (2012). *Open Source Network-on-Chip Router RTL.* [online]. available:https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/resources/route, (Accessed: October 2014).

Guerrier, P. and Greiner, A. (2000). "A generic architecture for on-chip packet-switched interco840047nnections." In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, 250–256.

Hardavellas, N., Somogyi, S., Wenisch, T. F., Wunderlich, R. E., Chen, S., Kim, J., Falsafi, B., Hoe, J. C. and Nowatzyk, A. (2004). "Simflex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture." *SIGMETRICS Perform. Evaluation Rev.*, 31(4), 31–34.

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.

Ho, R., Mai, K. W. and Horowitz, M. A. (2001). "The future of wires." *Proceedings of the IEEE*, 89(4), 490–504.

Igarashi, M., Mitsuhashi, T., Le, A., Kazi, S., Yang-Trung Lin, Fujimura, A. and Teig, S. (2002). "A diagonal-interconnect architecture and its application to risc core design." In *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, volume 1, 210–460 vol.1.

Infante, A. (2014). "Identifying caching opportunities, effortlessly." In Jalote, P., Briand, L. C. and van der Hoek, A., editors, *36th International Conference on Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*, ACM, 730–732.

Intel Corporation (2017). "Intel Advisor XE.").

Jang, H., Han, K., Lee, S., Lee, J. and Lee, W. (2019). "Mmnoc: Embedding memory management units into network-on-chip for lightweight embedded systems." *IEEE Access*, 7, 80011–80019.

Jensen, S. H., Sridharan, M., Sen, K. and Chandra, S. (2015). "Meminsight: platform-independent memory debugging for javascript." In Nitto, E. D., Harman, M. and Heymans, P., editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, ACM, 345–356.

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.

Joardar, B. K., Kim, R. G., Doppa, J. R., Pande, P. P., Marculescu, D. and Marculescu, R. (2019). "Learning-based application-agnostic 3d noc design for heterogeneous manycore systems." *IEEE Transactions on Computers*, 68(6), 852–866.

Kahng, A. B., Li, B., Peh, L. and Samadi, K. (2012). "Orion 2.0: A power-area simulator for interconnection networks." *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(1), 191–196.

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 Trans. 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." *Comput. Networks*, 3, 267–286.

Khyamling, P., Prasad, P. B. M. and Talawar, B. (2019). "Yanoc: Yet another network-on-chip simulation acceleration engine supporting congestion-aware adaptive routing using fpgas." *Journal of Circuits, Systems, and Computers*, 28(12), 1950202:1–1950202:31.

Kim, J., Balfour, J. D. and Dally, W. J. (2007). "Flattened butterfly topology for on-chip networks." *IEEE Comput. Archit. Lett.*, 6(2), 37–40.

Kim, J., Nicopoulos, C., Park, D., Narayanan, V., Yousif, M. S. and Das, C. R. (2006). "A gracefully degrading and energy-efficient modular router architecture for on-chip networks." In *33rd ISCA*, 4–15.

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.

Kundu, P. (2006). "On-die interconnects for next generation CMPs." *2006 Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems*.

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 *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 1–12.

Lotlikar, S., Pai, V. and Gratz, P. V. (2011). "Acenocs: A configurable hw/sw platform for fpga accelerated noc emulation." In *2011 24th Internatioal 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. (2017). "Dadiannao: A neural network supercomputer." *IEEE Transactions on Computers*, 66(1), 73–88.

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. H. (2013). "Cachetor: detecting cacheable data to remove bloat." In Meyer, B., Baresi, L. and Mezini, M., editors, *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, ACM, 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 Pasareanu, C. S. and Marinov, D., editors, *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, ACM, 282–292.

Nistor, A., Song, L., Marinov, D. and Lu, S. (2013). "Toddler: Detecting performance problems via similar memory-access patterns." In *2013 35th International Conference on Software Engineering (ICSE)*, 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, B. M. P. and Talawar, B. (2016). "Cache analysis and software optimizations for faster on-chip network simulations." In *2016 11th International Conference on Industrial and Information Systems (ICIIS)*, 83–88.

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)*.

Partha Pratim Pande, 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.

Patel, A., Afram, F., Chen, S. and Ghose, K. (2011). "MARSS: a full system simulator for multicore x86 cpus." In Stok, L., Dutt, N. D. and Hassoun, S., editors, *Proceedings of the 48th Design Automation Conference, DAC 2011, San Diego, California, USA, June 5-10, 2011*, ACM, 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)*, 1–11.

Poluri, P. and Louri, A. (2014). "An improved router design for reliable on-chip networks." In *28th IPDPS*, 283–292.

Porterfield, A. K. (1989). *Software Methods for Improvement of Cache Performance on Supercomputer Applications*. PhD thesis, Rice University.

Prodromou, A., Panteli, A., Nicopoulos, C. and Sazeides, Y. (2012). "Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures." In *45th MICRO*, 60–71.

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 Euromicro 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 Eidt, C., Holler, A. M., Srinivasan, U. and Amarasinghe, S. P., editors, *10th Annual IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2012, San Jose, CA, USA, March 31 - April 04, 2012*, ACM, 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.

Suboh, S., Bakhouya, M., Gaber, J. and El-Ghazawi, T. (2010). "Analytical modeling and evaluation of network-on-chip architectures." In *2010 International Conference on High Performance Computing Simulation*, 615–622.

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, 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.

Wee, S., Casper, J., Njoroge, N., Teslyar, Y., Ge, D., Kozyrakis, C. and Olukotun, K. (2007). "A practical fpga-based framework for novel CMP research." In DeHon, A. and Hutton, M., editors, *Proceedings of the ACM/SIGDA 15th International Symposium on Field Programmable Gate Arrays, FPGA 2007, Monterey, California, USA, February 18-20, 2007*, ACM, 116–125.

Wolkotte, P. T., Holzenspies, 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).

Xilinx Inc (2018). "*7 Series DSP48E1 Slice User Guide*." https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf).

Xilinx Inc (2019a). "*7 Series FPGAs Memory Resources*." https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdfa).

Xilinx Inc (2019b). "*Introduction to FPGA Design*." https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdfb).

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. H., Bond, M. D., Qin, F. and Rountev, A. (2011). "Leakchaser: helping programmers narrow down causes of memory leaks." In Hall, M. W. and Padua, D. A., editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, ACM, 270–282.

Yan, D., Xu, G. H. and Rountev, A. (2012). "Uncovering performance problems in java applications with reference propagation profiling." In Glinz, M., Murphy, G. C. and Pezzè, M., editors, *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, IEEE Computer Society, 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., Li, X., Shi, X. and Zheng, R., editors, *Network and Parallel Computing - 10th IFIP International Conference, NPC 2013, Guiyang, China, September 19-21, 2013. Proceedings*, volume 8147 of *Lecture Notes in Computer Science*, Springer, 167–178.

Zhao, Q., Cutcutache, I. and Wong, W. (2010). "Pipa: Pipelined profiling and analysis on multicore systems." *TACO*, 7(3), 13:1–13:29.

Publications based on the research work

Journal Publications

1. **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

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, *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

4. 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

5. **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 Ciruits Systems and Computers, World Scientific, 2019, 28:12, https://doi:0.1142/S0218126619502025

6. Prabhu Prasad B M, **Khyamling Parane**, and Basavaraj Talawar, *FPGA friendly NoC simulation acceleration framework employing the Hard Blocks,* Computing Journal, Springer, (Revision submitted).

Conference Publications

1. **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.

2. 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.

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, 31st International Conference on VLSI Design (VLSID), 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), Roorkee, India:IEEE, Dec 2016.

Bio-Data

Name:	Khyamling
Date of Birth:	01/06/1987
Email Id:	pkhyamling@gmail.com
Contact No:	+91-9482723110
Present Address:	Khyamling, Research Scholar, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, Mangalore - 575 025
Permanent Address:	Khyamling s/o Abrutappa parane, At Post: Hattarga(s), Tq: Basavakalyan, Dist: Bidar, Karnataka-585 419
Educational Qualifications:	B.Tech in Computer Science and Engineering from Visvesvaraya Technological University, Belgaum, Karnataka, India M.Tech in Computer Network Engineering from Visvesvaraya Technological University, Belgaum, Karnataka, India
Previous Work experience:	Assistant Professor in BKEC Basavakalyan, Karnataka from Aug, 2009 to Sep, 2010 Assistant Professor in RIT Sangli, Maharashtra from June, 2012 to June, 2015