Design and Space Exploration of NoC on FPGA

A Project Report

Submitted by

Sangeetha G S (14CO141)
Vignesh R (14CO153)
Matthew G Kallarackal (14CO225)

VIII Semester BTech

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL, MANGALORE - 575025
April 2018

DECLARATION

We hereby *declare* that the Project Work Report entitled **Design and Space Exploration of NoC on FPGA** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** for the award of the Degree of Bachelor of Technology in Computer Science and Engineering is a *bonafide report of the work carried out by us.* The material contained in this report has not been submitted to any University or Institution for the award of any degree.

14CO141, Sangeetha G S

_____

14CO153, Vignesh R

_____

14CO225, Matthew G Kallarackal

_____

Department of Computer Science and Engineering

Department of Computer Science and Engineering

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

Certificate

This is to certify that the project report entitled: **Design and Space Exploration of NoC on FPGA** submitted by:

Name of the student	Roll No
Sangeetha G S	14CO141
Vignesh R	14CO153
Matthew G Kallarackal	14CO225

as the record of the work carried out by them, is *accepted as the B.Tech. Project Work Report Submission* in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering.

Dr. Basavaraj Talawar
Project Guide

Dr. Alwyn Roshan Pais
(Course Coordinator)

Dr. Basavaraj Talawar
(Course Coordinator)

Place:

Date:

Acknowledgement

We hereby gratefully acknowledge the support and inspiration provided by our esteemed guide Prof. Basavaraj Talwar, Department of Computer Science and Engineering, NITK Surathkal, which was instrumental for this major project. We are forever indebted to him for giving us the wonderful opportunity of working under his guidance, thereby helping us gain immense knowledge and experience. We would like to express our sincere gratitude to him for his insightful advice, encouragement, guidance, critics and valuable suggestions, throughout the course of our major project.

We would also like to express our deeply felt gratitude to Dr. P Santhi Thilagam, Associate Professor and Head, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal, for her constant co-operation, support and for providing necessary facilities throughout the B.Tech program.

We would also like to take this opportunity to thank Mr. Prabhu and Mr. Khyamalin, for introducing us to the field and continuously supporting us throughout the course of the project, without which we would not have been able to progress.

We also take this opportunity to thank the teaching and non-teaching staff in the Department of Computer Science, National Institute of Technology Karnataka, Surathkal, for their invaluable help and support.

We also like to express our gratitude for our classmates who have been a constant source of motivation and who have made the time spent at NITK, truly memorable.

14CO141, Sangeetha G S

14CO153, Vignesh R

14CO225, Matthew G Kallarackal

Abstract

Networking On Chips is now becoming an extremely important part of the present and future of electronic technology. It is extensively used in Multiprocessor System-on-Chips and in Chip Multiprocessors. Using an NoC, the backend wiring involved has drastically reduced in an SoC. Further, SoCs with NoC interconnect operates at a higher operating frequency, mainly because the hardware required for switching and routing are simplified. The NoC researchers have relied on simulators based on performance and power to study the different factors of NoC such as algorithm in place, the topology, the buffer management and location schemes, the flow control and routing among others. Through this project, we aim is to explore this design space using FPGAs for some topologies.

Contents

Abbreviations

FPGA	Field Programmable Gate Array
NoC	Networking on Chip
SoC	System on Chip

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background

Today configuration challenges push the designers to convey interconnect subsystems to an exhibitions level step by step higher. Network on Chip models fame comes straightforwardly from the developing enthusiasm around System-on-Chip and Multi-Processor-System-on-Chip technologies. In a SoC-situated approach the creator coordinates in a similar chip distinctive Intellectual Property cores, with various functionalities (ALUs, peripherals controllers, RAM squares).

The outline logic arranged to the concept of Multi-Processor-System-on-Chip is significantly more a la mode, and without a doubt pushes specialists to examine and to enhance the interconnect innovations accessible today .

To comprehend the requirement for NoC, it is helpful to consider the difficulties postured by SoC and MPSoC. Innovative upgrades in the field of silicon innovation (as clarified in the International Roadmap for Semiconductors, in ten years it will be conceivable, for advanced systems architects, to acknowledge multi-billion transistors chips with clock frequencies around 10 GHz) open the street for tremendous execution enhancements. Tragically this mechanical change to be completely abused requires a reexamined of current VLSI configuration stream.

One conceivable solution to these issues requires a stronger adoption of

equipment reuse approaches, that lead the path to the creation of adaptable stages, while limiting the outline exertion for building new systems. Tomorrow mind boggling on-chip systems will be quite often made collecting an incredible number of IP modules, created in house and also by outsider accomplices.

Besides the development of the quantity of components that should be interconnected is beginning to build the negative symptoms of traditional "shared bus" structures, and represent the requirement for an interconnection system that enables more than one IP to utilize the communication assets in the meantime.

For every one of these reasons the NoC approach has all the earmarks of being the most encouraging solution nearby.

1.2 Motivation

The Network-on-Chip (NoC) is now an integral component in Multiprocessor System-on-Chips (MPSoCs) and in Chip Multiprocessors (CMPs) [1]. The communication time can influence the total turnaround time of the application significantly [2]. NoCs can span synchronous and asynchronous clock domains or use unclocked asynchronous logic. NoC technology applies networking theory and methods to on-chip communication. NoC improves the scalability of SoCs, and the power efficiency of complex SoCs compared to other designs, in the following aspects:

• It brings notable improvements over conventional bus and crossbar interconnections. The wires in the links of the NoC are shared by many signals. A high level of parallelism is achieved, because all links in the NoC can operate simultaneously on different data packets.

• As the complexity of integrated systems keeps growing, a NoC provides enhanced performance (such as throughput) and scalability in comparison with previous communication architectures (e.g., dedicated point-to-point signal wires or shared buses).

NoC researchers have relied on cycle accurate power and performance simulators (viz. Orion[3], Garnet[4], SICOSYS[5], Booksim[6]) to explore the

micro-architectural design space of on-chip networks. Design space exploration of NoCs in hardware environment such as on an FPGA provides the realistic details about the resource utilization. Additionally, the programs can be executed parallely in the hardware which make simulations faster, indicating a need for hardware acceleration.

1.3 Problem Statement

We aim to develop a platform that gives the user access to realistic details about the resource utilization of Noc architectures and their individual components. This includes exploration of various design decision parameters of NoC by modeling them on a FPGA. Different topologies will be considered for the experimentation purposes with different routing algorithms. Also, parameters such as buffer depth, traffic pattern and flit width will be varied to observe the effect on the NoC behavior.

Ultimately, we aim to develop a GUI which facilitates the user to build customized NoC routers and topologies. The simulator will enable users to test their architecture quickly and efficiently, with the help of an FPGA.

1.4 Objectives

We will use Verilog, a hardware description language, to implement an NoC router and the topology of the on-chip network. A NoC router consists of a buffer- to store incoming messages, a routing logic- determine the outgoing port for the packet based on the destination address, an arbiter- to resolve conflict between two packets competing for the same output port and a crossbar switch. The crossbar switch, which is connected to the output ports, sends packets from the buffer to the respective output port. Each router is connected to a processing element through its local port. The routers which are connected to the processing elements are interconnected to form a topology. Processing elements can be chips or any memory element. To simulate traffic generation from processing elements, we will implement shift registers. The routing logic

can be changed to measure the performance of different algorithms on the same topology.

The features that we plan to implement, in order, are as follows:

- Implementation of the oblivious, deflective and adaptive routing algorithms, along with other components of a NoC router.
- Implementation and Parameterization of Virtual Channels in NoC routers.
- Reduction of the depth of pipeline from 5 stages to 3-stage and 4-stage pipelines followed by elimination of buffer from pipeline. Comparison of performance of the each of these architectures.
- Measurement of link delay (as a function of the wire length).
- Implementation of variable header size (in bits) that will be specified by the user.
- Development of a software tool capable of automating the route generation for all standard topologies. It should populate routing tables appropriately. A GUI tool must also be provided to create any new topology and generate routing table for data packets in that topology.
- Generation of network traffic using that of GEM5. Interfacing of the GEM5[14] traffic with our NoC simulator.

1.5 Thesis Outline

This thesis report is organized into six chapters.
Chapter 1: Introduction to Network on Chips
Chapter 2: Deals with the survey of related research literature
Chapter 3: Presents the work already done with regards to the project
Chapter 4: States a conclusion of the project and proposes future work

Chapter 2

Literature Survey

2.1 History of NoC Development

The Network-on-Chip (NoC) is now an integral component in Multiprocessor System-on-Chips (MPSoCs) and in Chip Multiprocessors (CMPs). The communication time can influence the total turnaround time of the application significantly. NoC researchers have relied on cycle accurate power and performance simulators (viz. Orion, Garnet, SICOSYS, Booksim) to explore the microarchitectural design space of on-chip networks.

The modern SoCs require NoCs for various reasons. Many signals can share the wires in the links of NOC networks as opposed to the point-to-point connections in traditional networks. Any increase in complexity of the chip has only a slight impact on the throughput and efficiency of the network. These features help the network in achieving a high degree of parallelism and scalability. NoC simplifies the circuit design, since the hardware required for switching and routing is minimized. It provides the opportunity to place pipeline registers along speed-sensitive paths. These factors help in achieving high operating frequencies that typically results in higher through- put. NoCs form a independent subsystem. Therefore, the IP blocks associated with them can be changed. Such changes are quite common and this capability provided by NoC makes the system for configurable and rapid design of architecture feasible. Although the concept of NoC is quite new, there have been significant developments in their topologies and routing algorithms. To explore further design possibilities and

enhancements, we want to achieve the following in this project and compare their performances.

2.2 Software Simulators

To explore the microarchitectural design space of NoCs, researchers are relied on simulators to evaluate the power and performance. SICOSYS [5] is a general-purpose interconnection network simulator that allows the modeling a wide variety of message routers in a precise way. The parameters such as traffic pattern, applied load, message length etc., can be provided as input for simulation. Noxim [7] is another NoC simulator which is implemented in SystemC. Booksim2.0 [6] is a cycle-accurate simulator. It is flexible in terms of modeling network components. A large set of network parameters which are configurable such as topology, routing algorithm, flow control and router microarchitecture are implemented. Orion [3] a set of architectural power models for on-chip interconnection routers. A classic five-stage pipelined router with virtual channel flow control has been modeled in GARNET [4].

2.3 FPGA Based Simulators

In [8], an NoC emulation environment on FPGA called AcENoCs has been proposed. AcENoCs utilizes both software and hardware components of the FPGA. Traffic generators, clock generation and traffic sinks are implemented on the Microblaze softcore processor. The clock generation on software is flexible but potentially slow. The hardware platform is the network-on-chip to be emulated. Also, it consists of a register bank acting as an interface between the NoC under verification and software. An FPGA-based NoC emulator has been proposed in DART [9]. The synthetic and trace-driven workloads are supported. Global interconnect across all the nodes is provided. Any topology can be emulated by DART leaving out resynthesis of design utilizing these global interconnects and employing a software tool to configure the routing tables 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. [10] proposed FIST - a fast and simple packet latency estimator that can replace time-consuming detailed NoC models in full-system performance simulators. Analytical network modeling and execution-driven simulation models were combined to build FIST. Two variations of FIST in the context of software-based, cycle-level CMP are studied. Accuracy and performance were evaluated against full system CMP simulator's NoC mesh topology. The static approach trained offline using uniform random traffic achieved 6% average error in packet latency and average speedup of 43× was observed for 16×16 topology. Another FPGA emulation platform called Ultra-Fast [11] proposes two methods enabling swift emulations of larger NoC architectures on a single FPGA. Synthetic workloads are modeled accurately on FPGA by decoupling time of network being emulated from traffic generation units time. The TDM approach has been employed to emulate entire network utilizing more physical nodes. Authors have considered mesh topology of size 128 × 128 with look ahead XY routing and credit based flow control. The speedup of 5490× has been achieved over Booksim simulator. [12] implements bufferless customized unidirectional Torus topology with Deflective routing. By incorporating bufferless deflective routing, the hardware required by buffers can be saved which in turn reduces the power consumption also. The cost for Crossbar gets reduced considerably by doing so as the unidirectional Torus topol- ogy accepts packets only from two neighbouring ports and a local port thus reducing the crossbar complexity. Hardware requirements will be 3.5× smaller 2× faster than conventional 2D Mesh.

2.4 Conclusions

As per the literature survey conducted it is found that there exist several techniques to attain optimal NoC performance on a variety of microprocessor architectures. However, features or tools useful for setting up quick simulation environment and independent components for use in integration are not provided to a satisfactory level. In this project, we aim to address these issues and aim to provide a clean and simple user experience with the simulation software.

Chapter 3

Work Done

3.1 Router Design

The architecture of single router is as follows: It is connected to an Intellectual Property (IP) and other routers in the topology. Fig 3.1 shows the typical router architecture in a 2D mesh. It consists of buffers, routing logic, arbiters and crossbars. There are buffers, one for each input port. These buffers store the packet as and when they arrive at the router. The routing algorithm extracts the header of each packet and determines the ports through which the packets must be routed so as to reach their next immediate destination. There are arbiters, one for each output port. All packets that must be sent via the same output port are passed to the corresponding arbiter, which grants access to one of them. There is a single crossbar, common to all the ports. If a packet has been granted access to a port by the arbiter, the crossbars sends the packet to the corresponding output register. Thus, the arbiter's grant signals act as select signals for the crossbar. If the register is local, the packet is then forwarded to the network interface between the IP and router. Otherwise, the packet is forwarded to the next router.

3.1.1 Implementation Details

We have used a linear feedback shift register (lfsr) to generate network packets in a pseudo-random manner. The input to lfsr is the result of xor

operation of its output bits, which is considered as the feedback. This lfsr acts as the IP in all our topologies. Buffers store the packets that arrive at each router. We have used FIFO implementation for the buffer by using as a linear queue. The routing algorithm is implemented as a separate module. We have implemented two routing algorithms viz. XY Routing and Adaptive Routing. The arbiter resolves conflict between the packets. If there is a contention for the same output port by two or more packets simultaneously, the arbiter ensures that only one of these packets are granted access to the output port at any given time. The crossbar is a multiplexer that redirects packets from input ports to output ports based on the value of select signals.

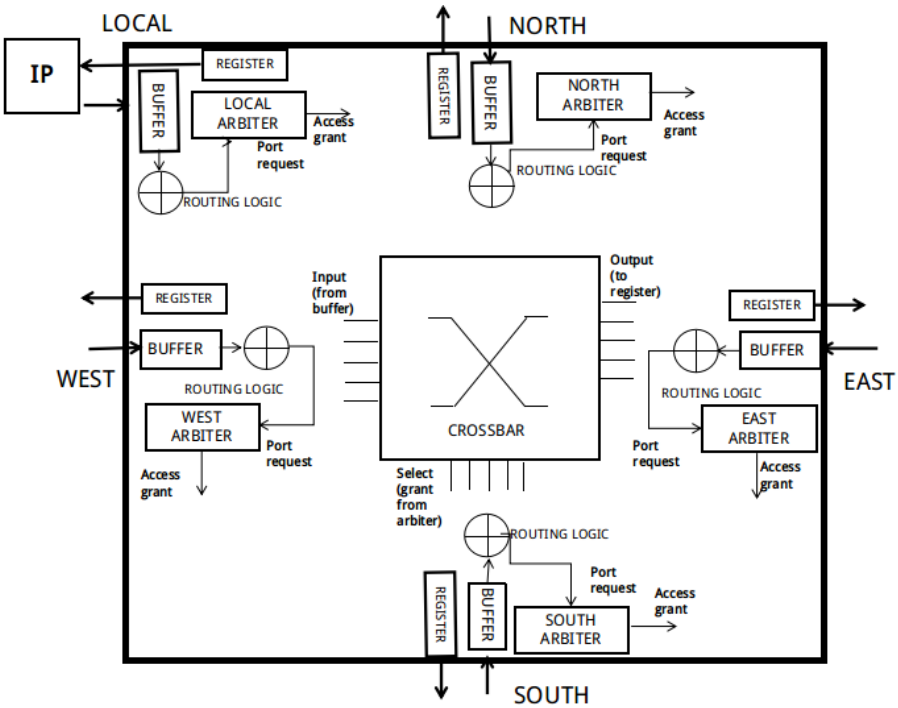

Figure 3.1: NoC Router used in 2D Mesh Topology

3.2 Topologies and Routing Algorithms

This section explains different NoC topologies implemented in Verilog, and routing algorithms used in the corresponding topologies.

3.2.1 Mesh Topology : Dimension Order Routing (XY)

In mesh topology, nodes are placed in a mesh-like formation. Every node, except the nodes in the border rows, has 4 neighboring nodes, one in the north-south-east-west directions respectively. The topology in shown in Fig 3.2. We have implemented 4x4 2D mesh in Verilog (with 16 nodes). We have implemented XY routing to route packets in this topology. Once the header of the packet is extracted, the routing logic determines the source and destination addresses. The first log(n) bits in each address represent the numbering of nodes in the X and Y directions. The algorithm used in XY routing is given in ALgorithm 1.

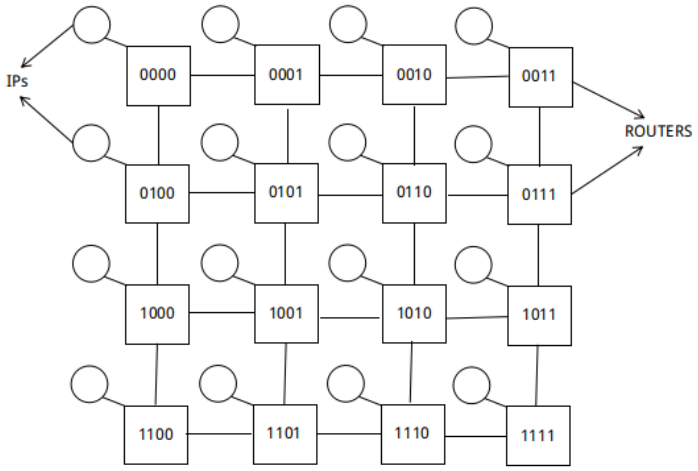

Figure 3.2: 4x4 Mesh Topology

3.2.2 Ring Topology : Shortest Path Routing

In ring topology, nodes are placed in a ring-like formation. Every node has 2 neighboring nodes, one in the east-west directions respectively. The topology in shown in Fig 3.3. We have implemented 8 node ring topology in Verilog. We have implemented Shortest Path Routing to route packets in this topology. Once the header of the packet is extracted, the routing logic determines the source and destination addresses. The algorithm used to find the shortest path is given in Algorithm 2.

Algorithm 1: XY Routing Algorithm

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 $xd > 0 \&\& ydiff < 0$ **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

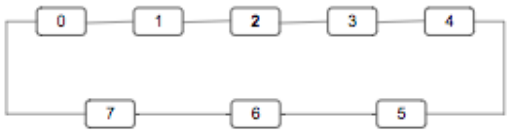

Figure 3.3: Ring Topology with 8 nodes

Algorithm 2: Shortest Path routing algorithm for Ring based topology

Input : Coordinates of current node (xc), destination node (xd), n is number of nodes in Ring topology
Output: Selected output port(East, West, Local)
Begin
if $xd == xc$ **then**
  return $Local$;
else if $xd >= (((n-1)/2+1) \&\& xc == 0$ **then**
  return $West$;
else if $xd <= (((n-1)/2+1) \&\& xc == 0$ **then**
  return $East$;
End

3.2.3 Star Topology : Routing

In star topology, nodes are placed around a central node. Every node is connected to the central node via a router. Every router is connected to the

central node and one of the leaf nodes. The central node also has a router. The topology in shown in Fig 3.4. We have implemented 8 node star topology in Verilog. Once the header of the packet is extracted, the routing logic determines the source and destination addresses. The algorithm used to route packets in leaf router is different from that of the central router. Algorithm 3 and Algorithm 4 illustrate the same.

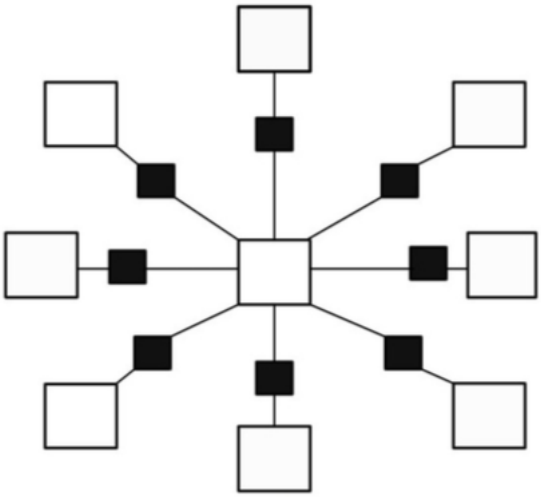

Figure 3.4: Star Topology with 8 leaf nodes and 1 central node

Algorithm 3: Leaf Node routing algorithm for Star topology
Input : Coordinates of current node (xcurrent), destination node (xdest), n is number of nodes in Star topology
Output: Selected output port(Local, Out)
Begin
if $xdest == xcurrent$ **then**
| return *Local*;
else
| return *Out*;
End

3.2.4 Butterfly Topology : Butterfly Routing

In butterfly topology, nodes are connected using interconnection switches. 2x1 switches are used to connect a pair of nodes to a pair of routers. In a topology with n nodes, *log n* levels of routers are present. In each level *n/2*

Algorithm 4: Central router routing algorithm for Star topology

Input : Out, n is number of nodes in Star topology
Output: Selected output port(DestinationPort)
Begin
for `i =1; i<(n-1); i++` **do**
  if $DestinationPort == out[i]$ **then**
    return $DestinationPort[i]$;
  else
    return $InvalidPort$;
end
End

routers are required. Overall, *(log n) * (n/2)* are needed in a butterfly topology of n nodes. In this topology, there exists a unique path between any 2 pair of nodes. The topology in shown in Fig 3.5. We have implemented 8 node butterfly topology in Verilog. Once the header of the packet is extracted, the routing logic determines the source and destination addresses. The algorithm used to find the route between nodes is given in Algorithm 5.

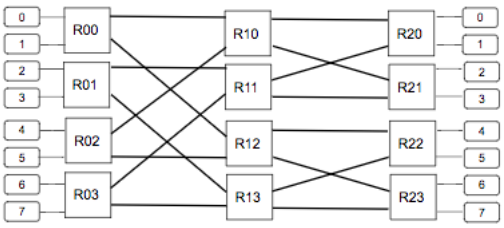

Figure 3.5: Butterfly Topology with 8 nodes

Algorithm 5: Routing in Butterfly topology

Input : Coordinates of current node (xc), destination node (xd), n is the level of the router in butterfly topology
Output: Selected output port(up, down)
Begin
if $xor(xd[n], xc[n]) == 0$ **then**
  return up;
else if $xor(xd[n], xc[n]) == 1$ **then**
  return $down$;
End

3.2.5 Trace Driven Simulation

The Linear Feedback Shift Register used in the router architecture does not provide a realistic utilization of the network. In order to get a realistic traffic, we have implemented Trace Driven Simulation. Through trace driven simulation, we can analyze the performance of the network in real-time. We tried using GEM5[14] and Multi2sim[15] simulators for collecting traces. However, we were unable to produce multicore traces required for the NoC topologies designed by us.

With the help of Netrace[16], a software tool to read and process traces, we were able to generate traces for our NoC. The Netrace tool can be run to parse any trace file and extract information such as packet injection cycle, source address, destination address, type of request etc.... We used the tracefiles that the developers of Netrace collected by running the PARSEC benchmark on the M5 simulator and we ran the tool on them. These traces are 64 node traces. Every tracefile has 5 regions. Region 0 is a short period of execution before the benchmark starts running. Region 1 is the warm-up portion of the benchmark, during which it is reading inputs and setting up threads. Region 2 is the PARSEC defined region of interest (ROI) which represents the parallel portion of the application. Region 3 is the end of the benchmark, during which it is writing results and cleaning up. Finally, Region 4 is between the end of the benchmark and simulation exit. The output of the tool is shown below:

```
Inject: 0   ID:0 CYC:0 SRC:4 DST:12 ADR:0x1fd0c840 TYP:ReadReq NDEP:1 1
Eject: 3   ID:0 CYC:0 SRC:4 DST:12 ADR:0x1fd0c840 TYP:ReadReq NDEP:1 1
Inject: 24   ID:1 CYC:24 SRC:12 DST:4 ADR:0x1fd0c840 TYP:ReadResp NDEP:2 2 8
Eject: 27   ID:1 CYC:24 SRC:12 DST:4 ADR:0x1fd0c840 TYP:ReadResp NDEP:2 2 8
Inject: 151   ID:2 CYC:151 SRC:4 DST:62 ADR:0x1df7e000 TYP:ReadReq NDEP:1 3
Eject: 178   ID:2 CYC:151 SRC:4 DST:62 ADR:0x1df7e000 TYP:ReadReq NDEP:1 3
```

The 'TYP' field signifies the type of packet being sent. There are many types of packets such as ReadReq, ReadResp, RespResp, WriteBack. For instance, ReadReq packets are sent when a miss occurs at the L1 cache. For this experiment, we have considered only packets of types 'ReadReq' or 'ReadResp' or 'RespResp'.

To process this trace and use it for our NoC, we wrote a python script that parses each line and extracts selective information. Out of the fields shown above, we require, CYC (cycle number), SRC (source node), DEST (destination node) and TYP (type of packet). The python script first filters files based on the TYP field. Since the trace is for a 64 node topology, and our NoC has 16 nodes, we take source address to be (SRC)MODULO 16. This does not alter the traffic behavior. All address and cycle numbers are written into a MEM file (that can be used in Verilog) in Hexadecimal (since there are 16 nodes). Out of the 6640073 present in the initial trace, our trace had 5328305 packets, after processing. The information contained in this processed trace is shown below:

```
C400000018
4E00000097
04000001CA
49000001EF
```

Each file in the processed trace corresponds to a packet in the NoC. Consider the packet - C400000018. As mentioned earlier, the information is written in Hexadecimal. Thus, the first entry C (corresponds to decimal value 12) is the source node (1100 in binary) and the second entry 4 is the destination address (0100) in binary. The rest of the bits, 00000018 represent the cycle at which the node has to prepare the packet and inject it into the network. This is also considered as payload in our NoC architecture.

3.2.6 Traffic Generator in NoC

To utilize the processed trace in the NoC, we made a traffic generator module inside a node. The traffic generator module of a node with source address X, reads the tracefile line by line. For each line, if the source address extracted from the line is same as X, and cycle number extracted is the ongoing cycle in the NoC, the traffic generator extracts the destination address from the line and sends the 3 values to the node. The node generates a packet containing Source and Destination addresses and a payload (in a particular format) and injects the packet into the network or in other words, sends it to the local input buffer of its router. The routing module then makes sure the packet reaches its destination

node. Once the packet reaches its destination node, the latency is calculated by subtracting packet's cycle number from the current cycle number. This has been implemented in verilog for a 16-node mesh topology, with XY routing and for a 16-node mesh topology, with XY routing in VC-based routers(VC-based router adopted from ProNoc[17]).

Chapter 4

Conclusion

4.1 Summary and Conclusion

Simulating large networks on software simulator takes considerable amount of time to provide the results. We emulated 16-Node Mesh-based topology, 3-Node Ring and 6-Node Star topologies on Xilinx Artix 7 FPGA board. We designed and implemented the router components in Verilog. All the components of router were integrated to build the desired topology. We explore the key differences between the mesh, star, ring and butterfly topology and have compared their performance in terms of traffic, resource and area usage, latency and other important factors. Ring and Star topologies with shortest path routing algorithm were evaluated under Uniform random traffic, Ring performs better compared to Star topologies. We observed extensive reduction in execution time with our FPGA framework with little variation in latency compared to Booksim simulator. In near future, we plan to enhance our framework by including the features such as different link latency, support for custom topology, support for 3D NoCs. We have identified that there does not exist any dynamic routing technique for flattened butterfly networks. We also plan to incorporate trace-driven simulation for all the above topologies and compare the performance of these networks, under real-time traffic.

4.1.1 Overall result analysis

We have computed the resource utilization and generated utilization reports using Vivado's built-in tools.

Site Type	Used	Fixed	Available	Util %
LUT as Logic	263	0	63400	0.41
LUT as Memory	24	0	19000	0.13
LUT as Distributed RAM	24	0		
LUT as Shift Register	0	0		
Register as Flip Flop	262	0	126800	0.21
Register as Latch	30	0	126800	0.02
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Table 4.1: Slice Logic for Star Topology

Site Type	Used	Fixed	Available	Util %
LUT as Logic	72	0	63400	0.11
LUT as Memory	56	0	19000	0.29
LUT as Distributed RAM	56	0		
LUT as Shift Register	0	0		
Register as Flip Flop	259	0	126800	0.20
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Table 4.2: Slice Logic for Butterfly Topology

Site Type	Used	Fixed	Available	Util %
LUT as Logic	825	0	63400	1.30
LUT as Memory	192	0	19000	1.01
LUT as Distributed RAM	56	0		
LUT as Shift Register	0	0		
Register as Flip Flop	1449	0	126800	1.14
Register as Latch	0	0	126800	0.00
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Table 4.3: Slice Logic for Ring Topology

Site Type	Used	Fixed	Available	Util %
LUT as Logic	1719	0	63400	2.71
LUT as Memory	320	0	19000	1.68
LUT as Distributed RAM	320	0		
LUT as Shift Register	0	0		
Register as Flip Flop	1861	0	126800	1.47
Register as Latch	280	0	126800	0.22
F7 Muxes	0	0	31700	0.00
F8 Muxes	0	0	15850	0.00

Table 4.4: Slice Logic for Mesh Topology

4.2 Future work

The project can be further extended to include the following:

1. Parameterizable topologies : To improve ease of use, the current modules can be modified to use parameters and generate required number of registers to accommodate all the components of the NoC.

2. Reduction in pipeline stages : Currently, the NoC routers follow a 5-stage pipeline. We can do look-ahead route computation and reduce number of pipeline stages to 3 or 4, thus decreasing latency.

3. Latency Comparison : After adding the trace-driven traffic generator module in ring, star and butterfly topologies, we can study and compare packet latencies for these networks.

Bibliography

References

[1] W. J. Dally and B. Towles, *Route Packets , Not Wires : On-Chip Interconnection Networks*, pp. 0–5.

[2] P. P. Pande *et al.*, *Performance evaluation and design trade-offs for network-on-chip interconnect architectures*, IEEE Transactions on Computers, vol. 54, no. 8, pp. 1025–1040, 2005.

[3] A. B. Kahng *et al.*, *ORION 2 . 0 : A Power-Area Simulator for Interconnection Networks*, Tvlsi, vol. XX, no. 1, pp. 1–5, 2010.

[4] N. Agarwal *et al.*, *Garnet: A detailed on-chip network model inside a full-system simulator*, ISPASS 2009, April 2009, pp. 33–42.

[5] V. Puente *et al.*, *Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems*, Parallel, Distributed and Network-based Processing, 2002, 2002, pp. 15–22.

[6] N. Jiang *et al.*, *A detailed and flexible cycle-accurate network-on-chip simulator*, Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on, April 2013, pp. 86–96.

[7] Davide Patti. Noxim. [Online]. Available: https://github.com/davidepatti/noxim

[8] S. Lotlikar *et al.*, *Acenocs: A configurable hw/sw platform for fpga accelerated noc emulation*, VLSID 2011, Jan 2011, pp. 147–152.

[9] D. Wang *et al.*, *Dart: A programmable architecture for noc simulation on fpgas*, NOCS 2011, ser. NOCS '11. New York, NY, USA: ACM, 2011, pp. 145–152.

[10] M. K. Papamichael *et al.*, *Fist: A fast, lightweight, fpga-friendly packet latency estimator for noc modeling in full-system simulations*, Fifth ACM/IEEE International Symposium on NoC, ser. NOCS '11. New York, NY, USA: ACM, 2011, pp. 137–144.

[11] T. V. Chu *et al.*, *Ultra-fast noc emulation on a single fpga*, FPL 2015, Sept 2015, pp. 1–8.

[12] N. Kapre and J. Gray, *Hoplite: Building austere overlay nocs for fpgas*, FPL 2015, Sept 2015, pp. 1–8.

[13] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[14] Nathan Binkert, Bradford Beckmann. - *The Gem5 Simulator.* ACM SIGARCH Computer Architecture. May 2011.

[15] Rafael Ubal, Byunghyun Jang, Perhaad Mistry. *Multi2Sim: A simulation framework for CPU-GPU computing.* 21st International Conference on Parallel Architectures and Compilation Techniques (PACT). Sept. 2012.

[16] J. Hestness, B. Grot, S. W. Keckler. *Netrace: Dependency-Driven, Trace-Based Network-on-Chip Simulation.* 3rd International Workshop on Network on Chip Architectures (NoCArc). Dec. 2010.

[17] Alireza Monemi, Jia Wei Tang, Maurizio Palesi, Muhammad N Marsono. *ProNoC: A low latency network-on-chip based many-core system-on-chip prototyping platform.* Microprocessors and Microsystems, Elsevier. Oct. 2012.