DESIGN AND FPGA IMPLEMENTATION OF RING NOC USING VHDL

By

Lopamudra Baruah R790211023

Amit Sabu R790211038

College of Engineering

University of Petroleum & Energy Studies

Dehradun

April, 2015

DESIGN AND FPGA IMPLEMENTATION OF RING NOC USING VHDL

A project report submitted in partial fulfillment of the requirements for the Degree of Bachelor of Technology (Electronics Engineering)

By

Lopamudra Baruah R790211023

Amit Sabu R790211038

Under the guidance of

Dr. Adesh Kumar `

Assistant Professor (Senior)

Department of Electronics, Instrumentation and Control Engineering

Approved

Dr. Kamal Bansal

Dean, College of Engineering Studies

College of Engineering

University of Petroleum & Energy Studies

Dehradun

April, 2015

CERTIFICATE

This is to certify that the work contained in this report titled "Design and FPGA Implementation of Ring NoC using VHDL" has been carried out by Lopamudra Baruah (R790211023) and Amit Sabu (R790211038) under my supervision and has not been submitted elsewhere for a degree.

Dr. Adesh Kumar
Assistant Professor (Senior)
Department of Electronics, Instrumentation and Control Engineering

Date

ACKNOWLEDGEMENT

We would like to thank our mentor, Dr. Adesh Kumar for providing us the guidance without which it would have been impossible to complete the project.

We extend our gratitude to the Dean of College of Engineering Studies, Dr. Kamal Bansal and the whole department of Electronics Instrumentation and Control Engineering, especially our Head of Department Mr. S. Choudhury for providing us the opportunity to learn, explore, and work on such a good topic for our final year project in our undergraduate study.

Many people have given us their help and support during the course of our Final year project for completion of our Bachelor of Technology degree. We extend our gratitude to all of them for helping us in their own individual ways in completing the project.

Lopamudra Baruah R790211023
Amit Sabu R790211038
B.Tech Electronics Engineering

TABLE OF CONTENTS

	PAGE NO.
LIST OF FIGURES	vii-viii
LIST OF TABLES	ix
NOMENCLATURE	x-xi
ABSTRACT	xii
1. INTRODUCTION	1-3
1.1. Problem Statement	2
1.2. Objectives	2-3
1.3. Scope of the research	3
2. LITERATURE REVIEW	4-7
2.1. General Survey	7
3. THEORETICAL DEVELOPMENT	8-19
3.1. Ring NoC Structure	8-10
3.2. Token Passing Local Area Network	10-18
3.2.1. Token Ring Local Area Network	10-11
3.2.1.1. Media Access Control in Local Area Network	11
3.2.1.2. IEEE 802.5 Frame Format	12-16
3.2.2. Token Bus LAN	16-17
3.3. Data Path Architecture	18-19
4. RESEARCH METHODOLOGY	20-22
4.1. FPGA and Project Design Flow	22-21
4.1.1. Project Design Flow	22
4.2. Software Development Tools	22-24
4.2.1. Project navigator Application Version 6.1i or	22

ISE 14.2 of Xilinx Company
4.2.2. Model SimEE 10.1 B 23
4.2.3. ModelSim: Simulation and Debug 23
4.2.4. ModelSim EE Features 23
4.2.5. ModelSim EE Benefits 23

5. RESULTS AND DISCUSSION 25-27

6. SYNTHESIS AND EXPERIMENTAL ANALYSIS 28-33
6.1. Device Utilization and Timing summary 29-30

7. CONCLUSION 34

REFERENCES 35-36

APPENDIX 37-65

LIST OF FIGURES

PAGE NO.

Fig. 1 Ring topological structure 9

Fig. 2 FIFO logic to address node 9

Fig. 3 Token ring LAN 11

Fig. 4 Media Access Control using Token 12

Fig. 5 Token Frame 13

Fig. 6 Data/Control Frame 13

Fig. 7 IEEE 802.4 Token ring bus structure 17

Fig. 8 IEEE 802.4 frame structure 17

Fig. 9 Data path architecture 18

Fig. 10 FPGA View 21

Fig. 11 FPGA Project Design Flow 22

Fig. 12 Chip Design Process Flow in ring NoC 24

Fig. 13 RTL view of RoC 26

Fig.14 Modelsim simulation for intercommunication 26

Fig. 15 FPGA synthesis block diagram 28

Fig. 16 FPGA Results 29-31

LIST OF TABLES

PAGE NO.

Table 1	Node selection scheme in ring NoC	9
Table 2	Pin details of ring NoC for (N=65536)	27
Table 3	Device utilization in ring NoC, for N = 65536	32
Table 4	Timing parameters for ring NoC, for N = 65536	32

NOMENCLATURE

AC	Access Control
ASIC	Application Specific Integrated circuits
DA	Destination Address
DDR	Double Data Rate
ED	End of Delimiter
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
FIFO	First input First output
FC	Frame Control
FCS	Frame Check Sequence
IEEE	Institute of Electrical and Electronics Engineering
IP	Intellectual Property
LUT	Look Up Table
LAN	Local Area Network
MPSoC	Multiprocessor system on chip
NoC	Network on Chip
PE	Processing Elements
RTL	Register Transfer Level
RoC	Rotator on Chip
SA	Source Address
SD	Staring Delimiter
SoC	System on Chip
SDRAM	Synchronous Dynamic Random Access Memory

THT	Token Holding Time
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale of Integration

ABSTRACT

The project is based on the ring based Network on Chip (NoC) structure design and modeling in Hardware Description Language (HDL). The network configuration is chosen for 65536 nodes, which is synchronized with same clock pulse. The functionality of each node is checked in Modelsim 10.1b software. The inter-process communication among nodes in verified using Virtex-5 FPGA. The priority of the nodes is assigned using FIFO logic, which is integrated with the NoC chip. The NoC architecture is based on token ring based network concept, called Rotator-on Chip (RoC). The design and modeling is done in Xilinx 14.2 ISE using VHDL programming language and synthesized on Digilent manufactured FPGA, with the target device, xc5vlx20t-2-ff323, Virtex-5. Hardware and timing parameters are extracted from the synthesized results and maximum frequency is found 535.733MHz and memory utilization is 263208 Kb.

CHAPTER 1
INTRODUCTION

This chapter gives a brief idea about Network-on-Chip and ring based NoC, and further illustrating the problem statement and objectives of the project.

1.1.Introduction to NoC

Network on Chip (NoC) is the latest approach to overcome the limitations of bus based communication network [1]. NoC is a set of routers employed in a network, in which different nodes are interconnected with their cores and can communicate with each other. In a network data comes in packets and sent to the destination with IP via routers and links [4]. When a packet reaches its destination address, it means packet is switched [5] to the IP attached to the router. On-chip communications among different networks is possible using interconnection network topology [5], switching, routing, queuing, flow control [11] and scheduling. Research is going on for three dimensional topological structures network on chip design. The idea of NoC is derived from distributed computing and large scale computer networks. There are different routing techniques used in NoC design considerations to meet high throughput. The routing methods for NoC should be very simple,due to constraints on hardware and memory resources utilization can have 2D and 3D network configurations.

NoCs support efficient on-chip communication [6] [7] potentially leads to NoC based multiprocessor systems characterized by high structural complexity and functional diversity. Multiprocessor System On-Chip (MPSoCs) [8] consists of complex integrated components, which communicates with each other at very high speed rates. A single shared bus or hierarchy of buses is not feasible so much for intercommunication. Intercommunication requirements of MPSoCs made of hundreds of cores having poor scalability with their shared bandwidth between all the attached cores, system size and the energy efficient requirements of final products. Networks-on-Chip (NoCs) is a promising solution to the scalability problem of forthcoming MPSoCs [5] [10]. NoC is an approach for designing the telecommunication subsystem between IP cores in a System-on-a-Chip (SoC) [7] [10]. The software and application layer is a very critical aspect on the NoC communication stack [11]. Secured transmission among nodes is possible if it follows the

NoC layer protocol [11] [12], mostly physical and network layers. All networked layers follow the pipeline and parallel processing that validate the optimized hardware parameters on chip development.

Rotator on chip (RoC) architecture [3] is based on token ring concept, which guarantees of packet arrival in sequence. It is based on scalable network and control mechanism that is helpful in the implementation of low latency on chip communication NoC. The performance of the ring NoC depends on the time slots and addressing schemes for the intercommunication among nodes in ring topological network. The NoC is applicable in telecommunication systems and is a great solution for rotator switches used in telecommunications systems. The topologies used for NoCs are tree, ring, crossbars, bus and meshes [3] [12]. Crossbar networks are larger in size and have low latency, poor scalability and greater cost. Tree topology structures [8] have good performance in terms of good latency but the networks based on tree topology have high wiring costs and router designing is also costly. There are also the chances of blocking several links in tree topology and messages delivery may fail. Mesh topological structures offers good latency and higher bandwidth but the latency can vary with the traffic intensity.

Ring topological structures achieved very good energy efficiency at low to medium core counts [3]. For a single ring system router, the routing logic needs to handle only two processes, injecting traffic into a ring structure, when there is space, and ejecting traffic when it is addressed to the current router [3]. In the simple structure of routers, most of the energy consumed in the interconnects is due to link traversal in ring NoC.

1.1. Problem Statement

Design and FPGA implementation of ring Network-on-Chip (NoC) using VHDL programming.

1.2. Objectives

- Design of Ring Network-on-Chip using VHDL Programming on Xilinx 14.2 software.
- Modeling and simulation of same Network-on-Chip (NoC) on Modelsim 10.1B software.
- FPGA Synthesis of NoC for the same using Spartan-3E FPGA or Virtex 5 FPGA.

- Analysis of the NoC with different test cases.

1.3.Scope of the research

Speed and size are two important factors while designing the electronic system. It's Speed of operation and flexibility to modify, measures the performance of the system operation. Traffic handling capacity is an important element of service quality and will therefore play a basic role in this choice Microprocessor/microcontroller (MPMC) system can handle sequential operations with high flexibility and use of Field Programmable Gate Array (FPGA) can handle concurrent operations with high speed in small size area. So combined features of both these systems can enhance the performance of the system. High Performance Hybrid Network System (HTSS) is designed using combination of stored program control (SPC) and VLSI technology.

CHAPTER 2
LITERATURE REVIEW

This chapter gives a description of the work that is related to the project and its current research status. A number of research papers have been studied which are described in the section.

2.1. General Survey

Technology in present world is always striving to be at par with the demands of the people and even surpass their expectations. In each and every field, for example automobiles, robotics, electronics etc., the contest to make the technology better always continues. Speed is considered as a priority in every aspect. In the field of electronics attaining great speed along with the increased complexity in the physical connections is always challenging.

Nowadays in the nanometer technology, the concept of multiple processors integrated in a single chip is thriving. This is known as MPSoC, Multi-processor System on Chip. It really is an achievement how a small chip can do such complex works. Since the MPSoC comprises of complex integrated components which need to communicate with each other at high speed. So to achieve this high speed communication, a single shared bus is not feasible for intercommunication. Network on Chip (NoC) is a promising solution. Noc consists of point-to-point links between routers that are arranged in a planar grid technology.

The major challenge in NoC is to achieve high speed using parameters such as latency, bandwidth etc. The concept of Network on Chip is a recent topic on which still experiments are going on. The aim is to achieve good performance. Both topology and scheduling algorithm of the NoC plays a role to achieve the desired performance with respect to latency and average bandwidth.

Various research papers were studied to get a clearer concept about the topic. The following papers give an idea of the research status on the topological NoCs.

1. **A Fast, Source-synchronous Ring-based Network-on-Chip Design Ayan Mandal, Sunil P. Khatri , Rabi N. Mahapatra, *EDAA,*2012 PP(1-7).**

Mandal et. al.[8] presented a fast source synchronous ring-based Network-on-Chip design. It is the first paper to demonstrate this concept using a resonant clock. Globally Asynchronous Locally Synchronous (GALS) communication approach was studied which is very in large multi-processor chip. As we know that as the size of the MPSoC increases there arises more difficulty to share a single synchronous clock over the entire chip. This led to the use of asynchronous communication between the different regions in the chip, while each region is clocked in a synchronous manner.

The most common and simple topology of a NoC is the 2D Mesh topology, which was first proposed by Dally and Towles as a network-on-chip architecture. The desirable features in a network-on-chip are:

- Minimal power consumption
- Low interconnect latency
- Scalability
- High link data-rates

The main disadvantage of mesh and torus topology, which is inherent, is the large communication radius they require, which results in large number of interconnects leadind to a large power consumption. The paper by Mandal et.al. provides some important conclusions. The ring based NoC is better in the following areas than a mesh topology NoC:

- POWER: Achieved 35% gain in total power.
- AREA: Achieved 11% gain in total area.
- LATENCY: Improved the average contention free latency by 7.4X
- BANDWIDTH: Available bandwidth is 4.5X more than a regular mesh.

These above results were achieved because the rings are unidirectional, which reduces the number of ports per intersection of horizontal and vertical links.

2. **An FPGA implementation of a scalable network on chip based on token ring concept Hadjiat K, St-Pierre F, Bois G, Svariya Y, Langevin M, Paulin P , 14th IEEE *International Conference on Electronics, Circuits and Systems, (ICECS) 2007,*pp(995 – 998)**

Karim et.al. presented a paper on an FPGA implementation of a scalable Network-on-Chip based on token ring concept. This paper helped us learn an insight about the token ring concept and FPGA implementation of ring Network-on-Chip. The trend of SoC requires us a need to integrate the complex components to attain efficient communication between the multiple blocks. An FPGA prototype implementation of a ring network-on-chip is shown which is based on token ring concept. They have calculated area complexity with respect to the number of nodes.

Multiple routers are required in a network-on-chip between the source and the destination, thus resulting in multiple queuing points. This affects the overall traffic performance. Recently, high performance products emerged in ring topologies using only one queuing point between the source and destination.

The following results were attained by Karim et.al:

- For about 64 nodes or less, hardware complexity is lower than mesh implementation.
- Ring network-on-chip latency is about 5 times smaller than mesh implementation.
- A traffic load of 70% and above can be supported with 32 node version of ring based network-on-chip using 12 channels.

The implementation has been validated by simulation on ISE tool and implemented on a Xilinx Virtex-II Pro FPGA.

3. A Tree-Based Topology Synthesis for On-Chip Network Jason Cong, Yuhui Huang, and Bo Yuan Computer Science Department University of California, Los Angeles Los Angeles, USA (page 9)

The Network-on-Chip (NOC) interconnect network of future multi-processor system-on-a-chip (MPSOC) needs to be efficient in terms of energy and delay. The custom on-chip network, which targets a given application, has proved to be more efficient than the regular structure on-chip network design in. The reason is that the communication requirement for each data flow is available in the design time, so the power consumption and packet latency are predictable once the links of networks are determined. The problem of topology synthesis is to determine the number of routers, the location of newly added routers, and the connectivity between them. Power consumption and packet latency are two trades off factors for any ASIC design which are met in the research.

4. **International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue-6, January 2012 Bursty Communication Performance Analysis of Network-on-Chip with Diverse Traffic Permutations, Naveen Choudhary (page 1)**

Network on Chip proposes to establish a communication infrastructure for the complex VLSI circuit in such a way that communication between any nodes in the circuit is possible even if the circuit blocks are not directly connected by a direct channel. Each circuit block of the whole circuit can be assumed as an Intellectual Property (IP) which may be a microprocessor, memory or ASIC, etc. In this paper the performance of standard 2D mesh NOC is analyzed for bursty communication traffic for various traffic or topology mapping patterns such as butterfly, transpose etc over a NOC simulation framework. The routing for the NoC is assumed to be XY and OE.

CHAPTER 3

THEORETICAL DEVELOPMENT

This chapter shows the topics studied under this project and the detailed description of each topic. This gives a brief idea about the theoretical development related to the project.

3.1. Ring NoC Structure

The designing block diagram of ring topological NoC is shown in fig. 1. It has 65536 nodes, arranged in a ring. Each node has its Processing Elements (PEs) and addressed by their node addresses. Each Processing Element (PE) [5] operates in a synchronous manner, and is assumed to operate at same frequency. The operation of the ring NoC can be understood with the help of table 1. All 65536 nodes are counted from N_0 to N_{65535} having a node address of 16 bit starting from “0000000000000000” to “1111111111111111” Let node N_0 is assigned a source_address “0000000000000000”, Node N_1 has address “00000000000000001”. Similarly all the nodes can be assigned their 16 bit of address and node N_{65535} is assigned source_address “1111111111111111”. In the full duplex mode, any pair can communicate to each other and vice versa. Let N_0 want to communicate in ring network. It has the probability to be routed via any link targeting any node N_1 to N_{65535} as shown in fig. 2. Each PE can process intercommunication to any other PE. Table 1 list the possibility to access other nodes. The node_address can be configured as source_address and destination_address for the source and destination nodes respectively. Similarly all the nodes can process intercommunication to each other. There is also the first input first output (FIFO) logic in case of multiple nodes try to communicate to the single node. It decides the priority of communicating nodes, based on the token structures. The data is transferred in the form of packets to each node. A data packet is carrying the information of source_address, destination_address and data.

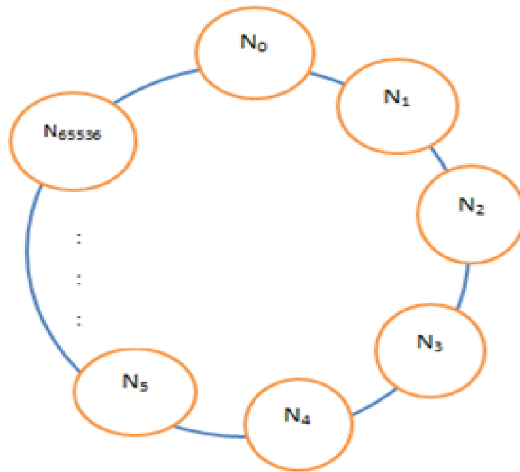


Fig.1 Ring topological structure

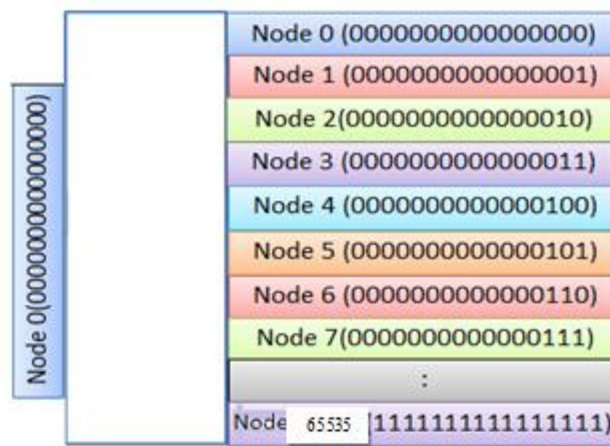


Fig. 2 FIFO logic to address node

Table 1 Node selection scheme in ring NoC

Source Address (16 bit)	Destination Address (16 bit)	Node Selection
0000000000000000	0000000000000000 : 1111111111111111	Node 0 : Node 65535
0000000000000001	0000000000000000 : 1111111111111111	Node 0 : Node 65535
0000000000000010	0000000000000000 : 1111111111111111	Node 0 : Node 65535

0000000000000011	0000000000000000	Node 0
	:	:
	1111111111111111	Node 65535
0000000000000100	0000000000000000	Node 0
	:	:
	1111111111111111	Node 65535
0000000000000101	0000000000000000	Node 0
	:	:
	1111111111111111	Node 65535
0000000000000110	0000000000000000	Node 0
	:	:
	1111111111111111	Node 65535
:	:	:
:	:	:
1111111111111111	0000000000000000	Node 0
	:	:
	1111111111111111	Node 65535

3.2. Token Passing Local Area Networks

In these LANs the access to media is controlled by circulating a token among the stations connected to a LAN. A station that has the token has got the authority to use the media and transmit its frame. Token passing local area networks (LANs) can have two kinds of topologies: ring and bus. The following are the token passing LANs:

- IEEE 802.5 token ring LAN
- IEEE 802.4 token bus LAN

3.2.1. Token Ring Local Area Network

A token passing ring consists of many stations connected in the form of a ring through point-to-point links as shown in Fig 3. Each station acts as a repeater and regenerates the signals it receives on one link and sends them onto the next link after a delay of atleast one bit. The ring is unidirectional.

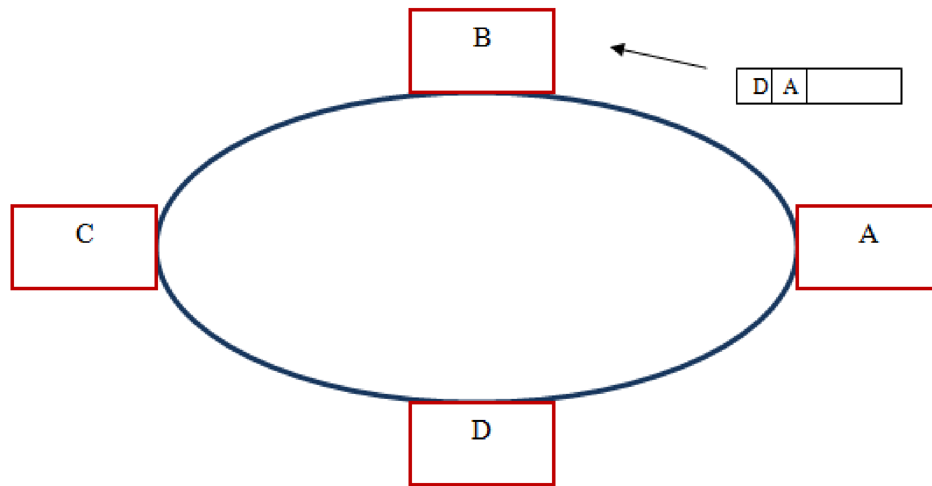


Fig 3 Token ring LAN

Suppose station A in Fig 3 wants to send a frame to station D. It breaks the ring and inserts its frame with the destination (D) and source (A) addresses. The frame passes through the stations B and C which act as repeaters and forward the frame to next link, while not copying the frame as it is not addressed for them. Station D copies the frame as it finds its address on the frame. The frame then returns back to station A continuing anticlockwise. The frame does not circulate again as the ring is broken at station A and then Station A removes the frame from the ring.

Thus an active station on the ring is always in one of the three modes: repeater mode, insert mode, or copy mode.

- REPEATER MODE: The received signals are regenerated and transmitted on the outgoing link. There is atleast one bit delay in the shift register.
- INSERT MODE: The ring is broken and the station sends its own frame on the outgoing link. The incoming signals are received but are not sent on the ring again.
- COPY MODE: The station regenerates the received signals and sends them on the outgoing link. It also copies the received signals for its use.

3.2.1.1.Media Access Control in Token Ring LAN

Access to the ring for passing a frame is controlled by the use of a token. The token is passed from station to station. The physical locations of the stations of the stations on the

ring determine the sequence of passing the token. When a station has frames to transmit, it seizes the token. It holds the token and sends one or more frames and then releases the token for the next station on the ring. Fig 4 shows in detail the sequence of events when station A sends a frame to station C and releases the token which is then picked by station D.

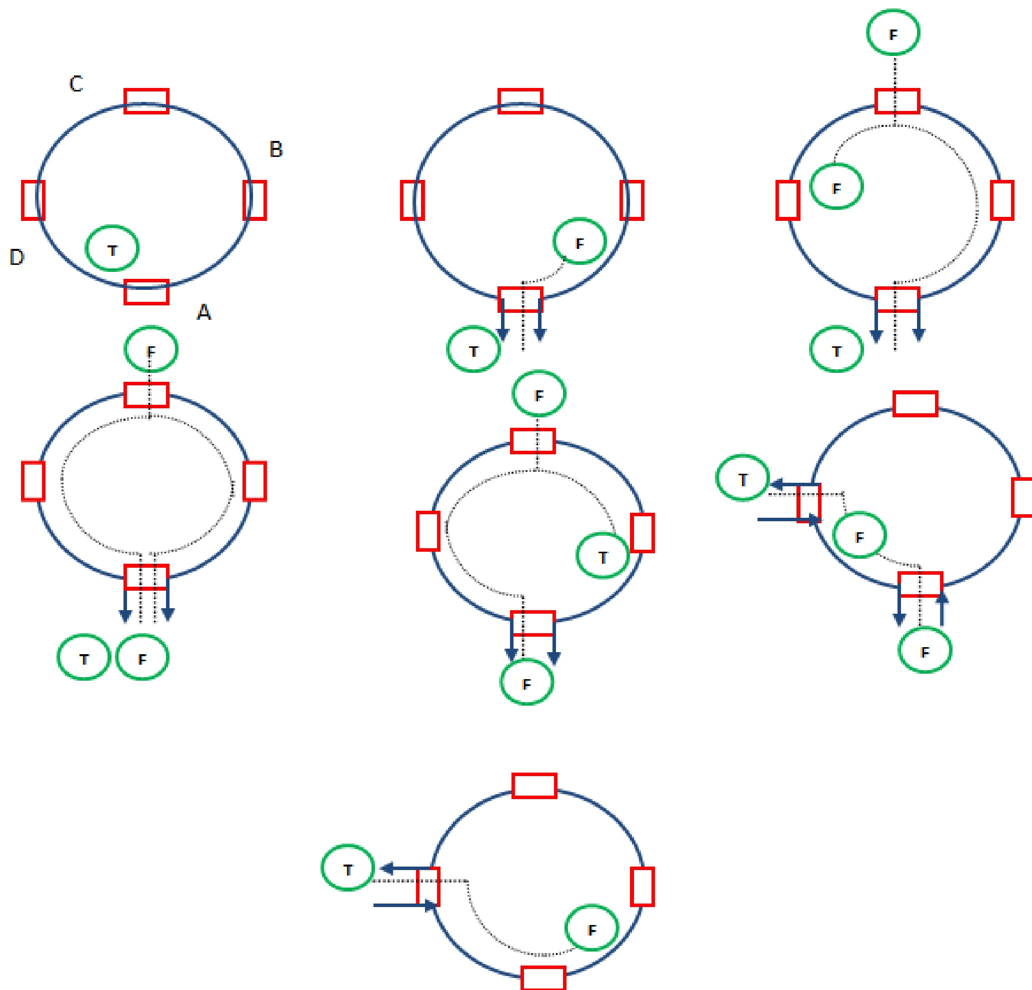


Fig 4 Media Access Control using Token

3.2.1.2.IEEE 802.5 Frame Format

IEEE 802.5 specifies two basic MAC frame format types:

- Token frame
- Data/Control frame

Token frame is three byte long and consists of start delimiter byte, access control byte, and end delimiter byte. Data Frames vary in size depending on the size of data field. Control frames do not have data field.

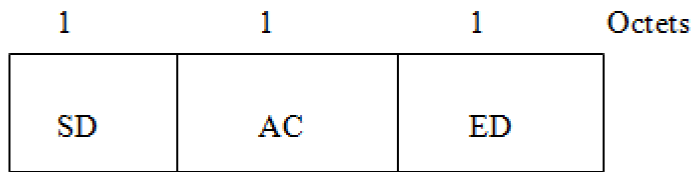
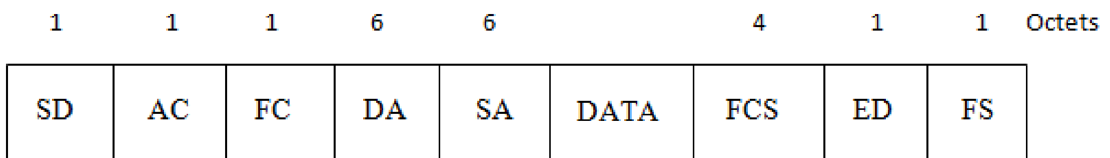


Fig 5 Token Frame



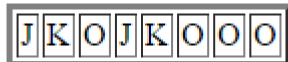
SD: Start Delimiter
SA: Source Address
DA: Destination Address
ED: End Delimiter
FC: Frame Control
AC: Access Control
FCS: Frame Check Sequence
FS: Frame Status

Fig 6 Data/Control Frame

Preamble is not required in these frames because a station receives regenerated signals from the neighbor and its internal clock is locked to a master clock of the ring. The frame format is described as follows:

- START DELIMITER (SD)

  It is one octet long unique symbol pattern that marks the start of the frame. J and K are special signals that enable identification of the start delimiter. Code violation property of J and K bits is used for their detection.

  J = Code Violation
  K = Code Violation

- ACCESS CONTROL (AC)

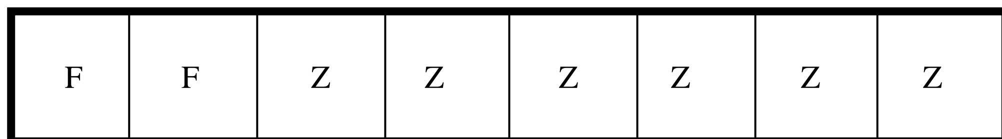
  It is one octet long field containing priority bits (P), token bit (T), monitoring bit (M), and reservation bits (R) as shown below:

T=Token
T = 0 for Token
T = 1 for Frame

Priority bits (P) indicate the current priority level of the data or token frame. Reservation bits (R) are used for reserving the next priority. The token bit (T) distinguishes a token frame from a data/control frame. A station in repeater mode which wants to send a frame waits for the token. If it finds that the token bit is 1, it seizes the token by changing over to insert mode and breaking the ring. Minimum one bit delay ensures that the token bit is not passed to the output port before the ring is broken. The station then inserts 1 in place of 0 in the token bit position, and then continues transmission of the rest of its data/control frame. Thus a token frame is converted into data/control frame. The priority bits (P) remain unchanged in this process.

- FRAME CONTROL (FC)

It is one octet long and differentiates between data and control frames. If FF bits are 01, then it is a data frame that contains LLC in the data field. If the FF bits are 00, then it is a control frame. The six Z bits indicate the control function.



Some of the control functions are as follows:

Z BITS	CONTROL FUNCTION
000011	Claim Token (CT)
000010	Beacon
000100	Purge
000101	Active Monitor Present (AMP)
000000	Standby Monitor Present (SMP)

- DESTINATION ADDRESS

It is 2 octets (local address) or 6 octets (global address) long and contains the address of the destination station to which the source station wants to transmit the frame.

- SOURCE ADDRESS

  The source address field is 2 bytes or 6 bytes long address structure. It contains the address of the source station which wants to transmit the frame that to the specified destination.

- DATA FIELD

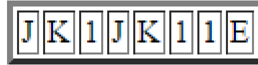
  It can have 0 or more bytes. There is no maximum size, but the station can hold the token for a limited time. The maximum size of data frame (and therefore maximum size of data field) is determined by bit rate and the Token Holding Time (THT). Typical maximum length of field is:
  - 4500 bytes for 4 Mbps LAN
  - 18000 bytes for 16 Mbps LAN

- FRAME CHECK SEQUENCE (FCS)

  It is 4 bytes long and contains CRC code. It checks on DA, SA, FC, and data fields. The source computes and sets this value and then destination too calculates this value. If the two are different, it indicates an error, otherwise the data may be correct.

- END DELIMITER (ED)

  It is one byte long and contains an unique symbol pattern as below that marks the end of a token or data frame. J and K are special symbols that violate the differential Manchester code and identify the end delimiter. E bit is for error bit. When a frame passes by a station, the station carries out FCS check on fly and if error is detected, it sets E-bit to 1. I-bit is set to 1 by the sending station if there are more frames to follow. It is set to 0 in the last frame.



J = Code Violation
K = Code Violation
I = Intermediate Frame Bit
E = Error Detected Bit

- FRAME STATUS (FS)

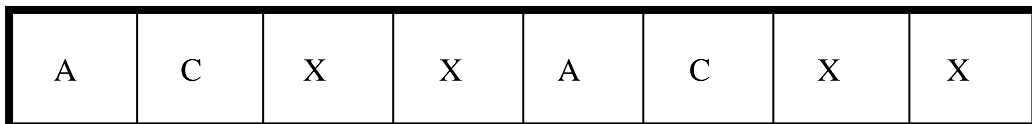
  This field is 1 byte long. It contains two address recognized bits (A-bits) and two frame copied bits (C-bits). Every frame is sent with AC=00. When a frame passes by a station having address same as in the DA field, the station sets A-bit to 1 indicating to the frame originating station that the destination station is alive in the ring. If the destination station is also able to copy the frame, it sets C-bit also to 1. Thus possible combinations of AC bits are as follows:

  AC = 00, Addressed station is not on the ring.

  AC = 01, Frame has been copied by the addressed station.

  AC = 10, Addressed station is on the ring but the frame is not copied.

  AC = 11, Invalid value of the AC field



A bit set to 1: destination recognized the packet.
C bit set to 1: destination accepted the packet.

3.2.2. Token Bus LAN

In the ring based NoC system, all the nodes are physically connected as a bus trough a coaxial cable as shown in Fig.7, but logically form a ring. The node (station), which is outside the ring, is considered as outer node. According to IEEE 802.4 token bus network frame has the following fields. The frame structure of IEEE 802.4 is shown in Fig.8.

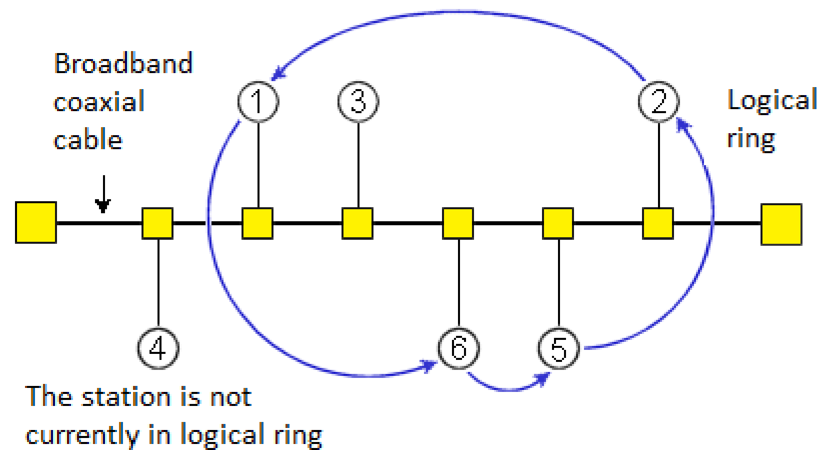


Fig. 7 IEEE 802.4 Token ring bus structure

Preamble: It is used to synchronize the receiver's clock.

Starting Delimiter (SD) and End Delimiter (ED): The Starting Delimiter and Ending Delimiter fields contain analog encoding of symbols other than 1 or 0 and are used to mark frame boundaries. The representation of symbols is because both cannot occur accidentally in the user data. Therefore, no length field is needed.

Frame Control (FC): The field is used to distinguish data frames from control frames. The field carries the frame's priority for data frames and a bit which the destination can set as an acknowledgement. Similarly, the Frame Control field is used to specify the frame type, for control frames. The data frame and control frame include token passing and various ring maintenance frames.

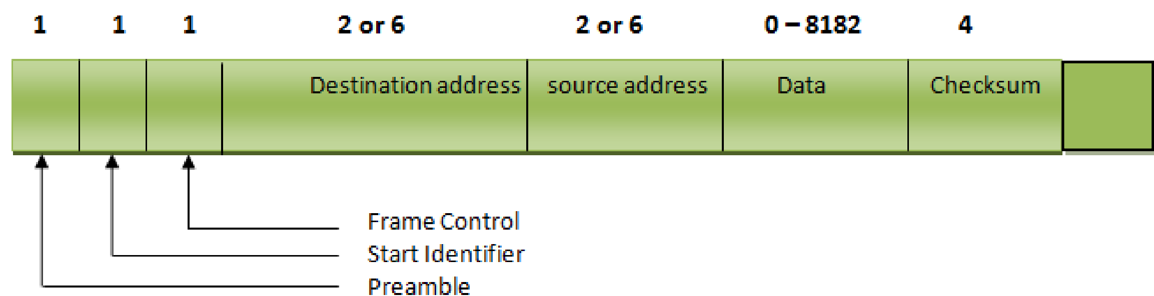


Fig. 8 IEEE 802.4 frame structure

Destination and source address: The field is used to provide the Destination node and source node address fields. The field may be l and global address. It is of bytes for a local address and 6 bytes for a global address.

Data: The Data field is always variable carries the actual data. It may be 8182 bytes when 2 byte addresses are used and 8174 bytes for 6 byte addresses.

Checksum: Checksum field is used in error detection. A 4-byte checksum can be used in the calculations of data.

3.3. Data Path Architecture

The data path architecture of the ring NoC is shown in fig. 9. It consists of (16 x 65536) decoder and (65536 x 1) demultiplexer. The source address and destination address are configured with the help of 16 address lines (A_0–A_{15}).

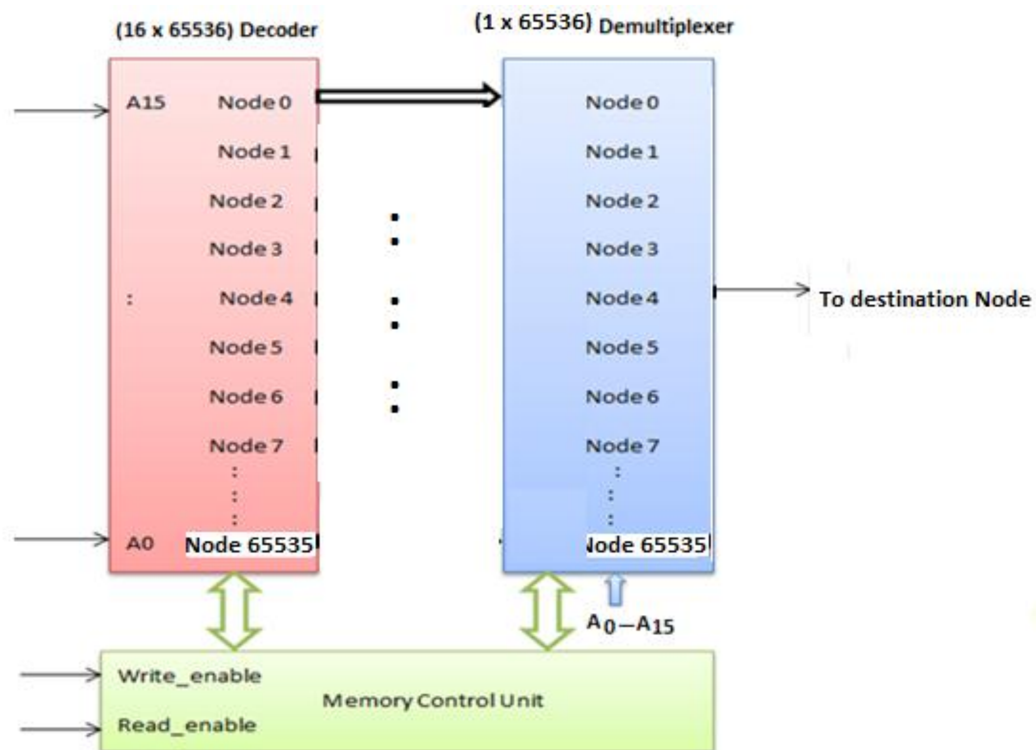


Fig. 9 Data path architecture

The 16 bit input decoder can have 65536 output lines, which are be assigned as number of nodes. Each node has control signals read_enable and write_enable to perform read and

write operations. The nodes are carrying data of 8182 bytes. The data transfer is possible to one node only at a time, based on the priority logic and FIFO selection. The logic to select the same functionality is demultiplexer. Demultiplexer (1 x 65536) has 65536 inputs with 16 bit address bus (A_0 – A_{15}) and one output. The memory control unit performs write and read operations.

CHAPTER 4
RESEARCH METHODOLOGY

FPGA design methodology supports both top-down and bottom-up design methodology. FPGA design flows support modular design approaches for bottom-up methodology and hierarchical design partitioning for top-down design methodology, similar to the process used for traditional standard cell ASIC devices. Xilinx design software also supports newer standards such as VHDL/ Verilog HDL that are becoming a part of traditional standard cell ASIC design methodology. For team-based designs that are typical of ASIC-size designs, the ISE Design Suit software supports incremental compilation methodology. For designs using hard copy ASICs to achieve higher performance than the companion FPGA, it is recommended that you use the hard copy first flow. Flight information controller design development includes the following steps for both FPGAs and traditional standard cell ASICs.

- Top-down or bottom-up methodology selection
- RTL coding in Xilinx tool
- Hardware optimization using in Xilinx tool
- Simulation and functional verification in Modelsim
- Specification of the external and internal memory Synthesis
- View synthesis report

4.1. FPGA and Project Design Flow

FPGA stand for Field-Programmable Gate Array and is an integrated circuit that can be programmed after it is manufactured, hence its name “Field Programmable”. There are two different types of FPGAs, one that only can be programmed once called One-Time Programmable, OTP, and one that is reprogrammable. OTP technology is much more common in space qualified designs because it is in general more robust than the reprogrammable one. But as Bonacini et al. (2006) writes, the in-system reprogram ability of FPGAs is of great importance giving extreme flexibility to the application, which can be updated in case of changing requirements or failure recovery.

To be programmable a FPGA contains numerous configuration switches that after programming routes the different Logic Blocks, LB, together. Described by Pellerin and

Thibault (2005), a typical FPGA contains Logic Blocks that make up the bulk of the device and they are based on Lookup Tables, LUT, (of perhaps four or five binary inputs) combined with one or two single-bit registers and additional logic elements such as clock enables and multiplexers. These Logic Blocks and LUTs look differently depending on the technology used, different companies use different technologies, but it usually also differ between product series within a company. These Logic Blocks are then connected through a grid surrounding them, which also connect with the I/O pins at the edges of the chip.

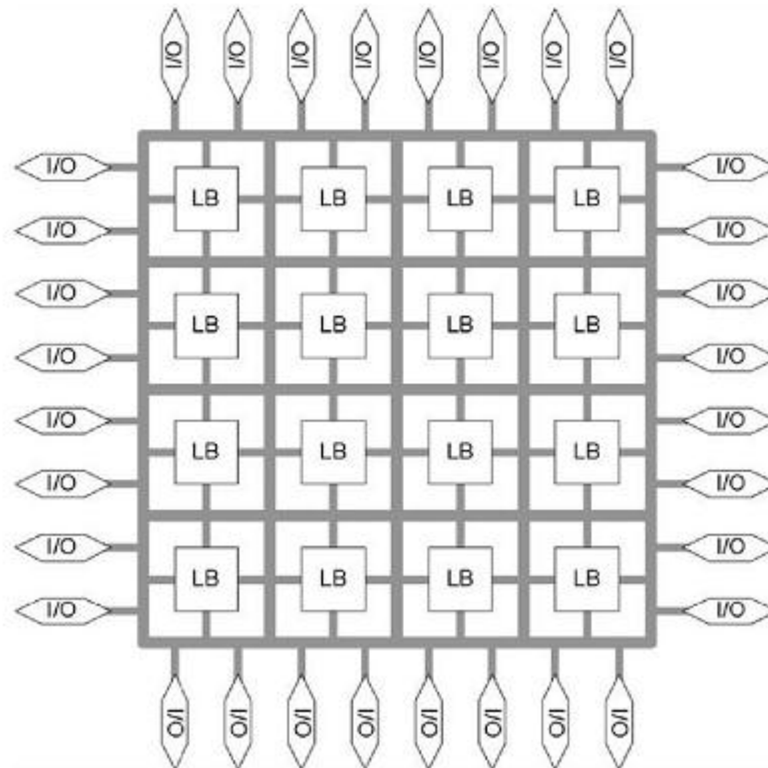


Fig. 10 FPGA View

Many FPGAs provide internal SRAMs located between the Logic Blocks to greatly improve functionality. This feature opens up to new possibilities although it also brings about another trait that is usually sensitive to radiation. But despite that the use of SRAM-based FPGA is growing in space based applications because of low application development cost, short time to market, and the reprogramming flexibility that they offer, the need is recognized and today there are FPGAs with radiation tolerant SRAM designs. Other FPGAs also include additional internal blocks to increase the performance in different applications: like internal DLLs/PLLs; multipliers or even more advanced DSP block (DSP Slices,); hard-macro processors (Power PCs,); high speed serial links; etc. But these bring about even more radiation issues to overcome.

4.1.1. Project design flow: Fig. 14 shows a flow chart over the design process when a design is implemented into an FPGA. This flow was followed with all designs in this project and so became an important structure in the project plan. For those that is not familiar with these concepts a short description will follow. For more case specific see all the steps listed below at front end design.

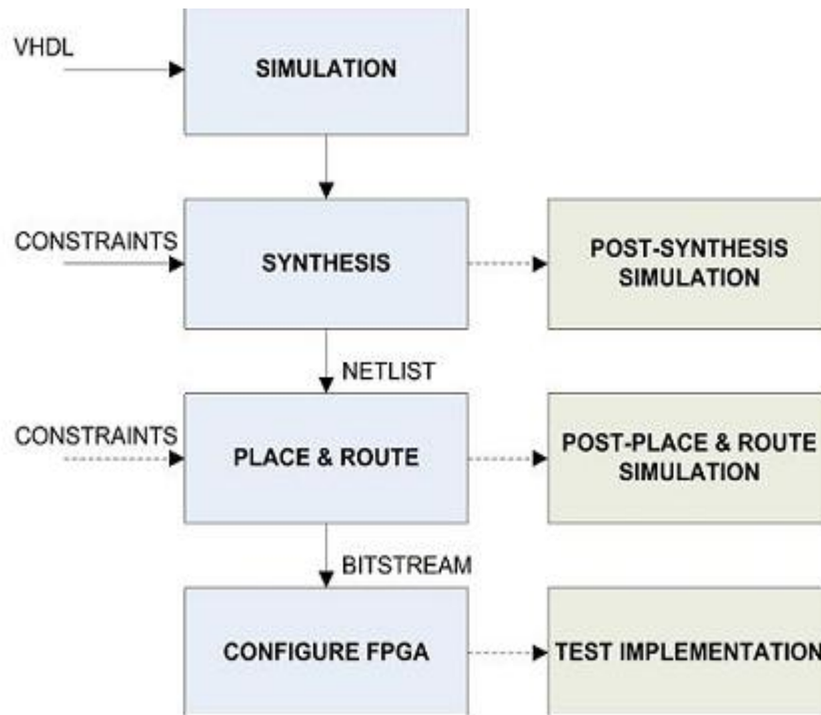


Fig. 11 FPGA Design project flow

4.2. Software Development Tools

FPGA implementation includes the following software development tools

4.2.1. Project navigator Application Version 6.1i or ISE 14.2 of Xilinx Company

Xilinx has been a semiconductor industry leader at the forefront of technology, market and business achievement. It is a tool to design the IC and to view their RTL (Register Transfer Logic) schematic .It is a tool to test the code on FPGA environment and we can get the all parameters details required to implement the chip.

4.2.2. Model SimEE 10.1 B

The combination of industry-leading, native SKS performance with the best integrated debug and analysis environment make ModelSim the simulator of choice for both ASIC and FPGA design. The best standards and platform support in the industry make it easy to adopt in the majority of process and tool flows.

4.2.3. ModelSim: Simulation and Debug

ModelSim EE is the industry-leading, Windows-based simulator for VHDL, Verilog, or mixed-language simulation environments.

4.2.4. ModelSim EE Features:

- Partial VHDL 2008 support
- Transaction wlf logging support in all languages including VHDL
- Windows7 a 32 Support
- SecureIP support
- SystemC option
- RTL and Gate-Level Simulation
- Integrated Debug
- Verilog, VHDL and SystemVerilog Design
- Mixed-HDL Simulation option
- Code Coverage option

4.2.5. ModelSim EE Benefits:

- Cost-effective HDL simulation solution,
- Intuitive GUI for efficient interactive debug,
- Integrated project management simplifies managing project data,
- Easy to use with outstanding technical support,
- Sign-off support for popular ASIC libraries,
- Hardware debugging
- Functional simulation

The following diagram shows the basic steps for simulating a design in ModelSim**.**

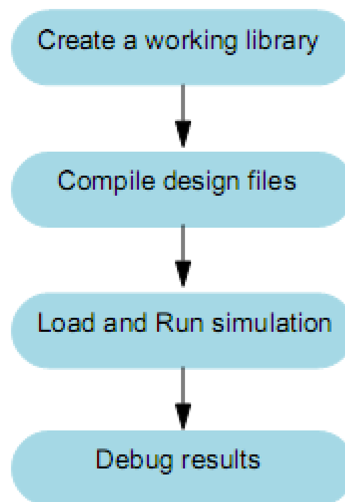


Fig. 12 Chip Design Process Flow

• Creating the Working Library: In ModelSim, all designs are compiled into a library. Typically start a new simulation in ModelSim by creating a working library called "work," which is the default library name used by the compiler as the default destination for compiled design units.

• Compiling Design: After creating the working library, design is being compiled into it. The ModelSim library format is compatible across all supported platforms.

• Loading and Running the Simulator with the Design: With the design compiled, we load the simulator with design by invoking the simulator on a top-level module (Verilog) or a configuration or entity/architecture pair (VHDL). Assuming the design loads successfully, the simulation time is set to zero, and you enter a run command to begin simulation.

- Debugging: ModelSim’s robust debugging environment is used to track down the cause of the problem.

CHAPTER 5

RESULTS AND DISCUSSIONS

The modeling and design of the RoC is done in Xilinx 14.2 ISE and function simulation is carried out in Modelsim 10.1. The data transfer is carried out for the different test cases and it has shown 100 % successful data transmission form source to destination node. The RTL view of developed chip is shown in the Fig. 11 and the functional details of RTL pins is discussed in table 2.The functional simulation of NoC is shown in Fig.12, shows the data transfer scheme from node N_4 to node N_{32}. The source_address = "000100" and destination_address = "100000". The input data is transferred from node N_4input_data_packet = 1'h 12345678ABCDABCD……AD (hex data) and same data is received at node N_{32} = 1'h 12345678ABCDABCD….AD (hex data). The data is received at the positive edge of the clock pulse. The clock signal is synchronized with the reset. FIFO_EMPTY and FIFO_FULL show the status of, FIFO logic or the priority of the source node. The destination node is free for intercommunication, then FIFO_EMPTY= '1' and FIFO_FULL = '0'.

The functional simulation depends on the following steps input:

Step input 1: Reset = '1' and run, output_data_packet will contains zero output.

Step input 2: Reset = '0', Apply rising edge clock pulse, source_address and destination_address value and 64 bit data of destination node with input_data_packet, then run.

Step input 3: Apply the source address and destination address of another nodes and data on input_data_packet.to run

Step input 4: check the output_data_packet of destination node and status of FIFO logic using FIFO_FULL and FIFO_EMPTY inputs.

Fig. 13 RTL view of RoC

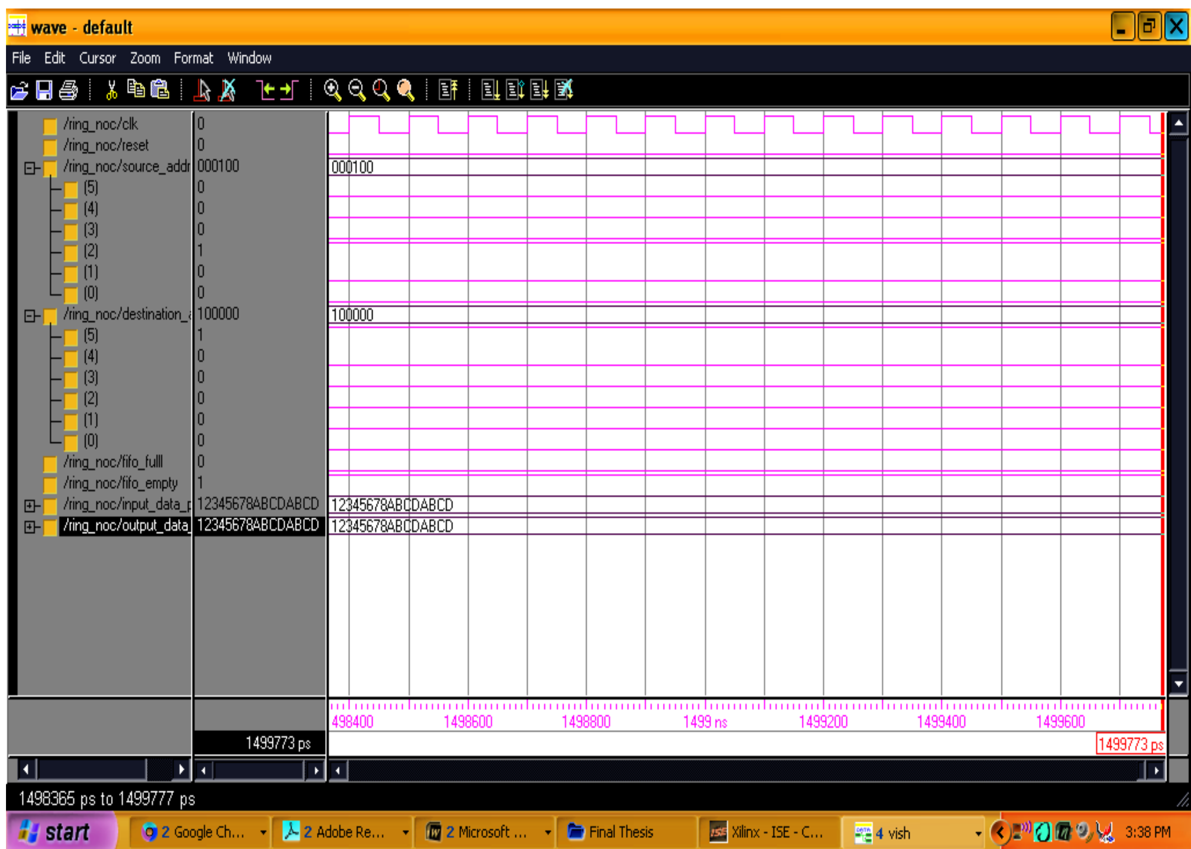


Fig.14 Modelsim simulation for intercommunication in ring NoC

Table 2 Pin details of ring NoC for (N=65536)

Pins	Description
Reset	Used to reset the memory contents zero for synchronization of the components by using clk of std_logic (1 bit)
Clk	Default input for sequential logic to work on rising edge of clock pulse of std_logic.(1 bit)
source_address [15:0]	Address of source node of std_logic_vector (16 bit)
destination_address [15:0]	Address of destination node of std_logic_vector (16 bit)
input_data_packet [8182x 8:0]	Input data of the source node of std_logic_vector (8182bytes)
output_data_packet [8182x 8:0]	Output data of the source node of std_logic_vector (8182bytes)
FIFO_EMPTY	Status of FIFO priority logic, signifies that node is free to communicate of std_logic (1 bit)
FIFO_Full	Status of FIFO priority logic, signifies that commutating node is not free, source subscriber is in assigned priority based on FIFO logic of std_logic (1 bit)
write_en	Control signal to perform memory write operation with respect to individual node of std_logic(1 bit)
read_en	Control signal to perform memory read operation with respect to individual node of std_logic(1 bit)

CHAPTER 6

SYNTHESIS AND EXPERIMENTAL ANALYSIS

The node data transfer is also verified with the experimental set up carried with the help of Digilent manufactured Virtex FPGA. The parameters supports to FPGA synthesis are discussed as device utilization and timing parameters. The block diagram of an experimental set up is shown in fig. 8. The experiment is carried out to validate the data transfer among inlets/outlets using Virtex -5 FPGA. Two 9-pin RS-232 ports assist in the transmission of serial data to and fro from the FPGA board. 50 MHz clock oscillator is the system clock provides the clock signal to the various events taking place within the FPGA and the various programs that require clock for their working. The input switches are given the source_address [5:0] and destination_address [5:0], reset and clock input to the FPGA board, the logged data can be shown on the motherboard of the PC. The onboard Xilinx Virtex-5 user FPGA is directly coupled to multiple independent banks of DDR2 SDRAM and QDR-II SRAM memory providing up to 13GBytes/sec of sustained memory bandwidth to the FPGA. Optimized memory controller IP cores and reference designs are included as part of the product deliverables along with VHDL source code and API for Windows operating systems. The DATA-V5 is tightly integrated to the Host computer via an 8-lane PCI Express connection supporting sustained bandwidths of up to 2.38GBytes/sec.

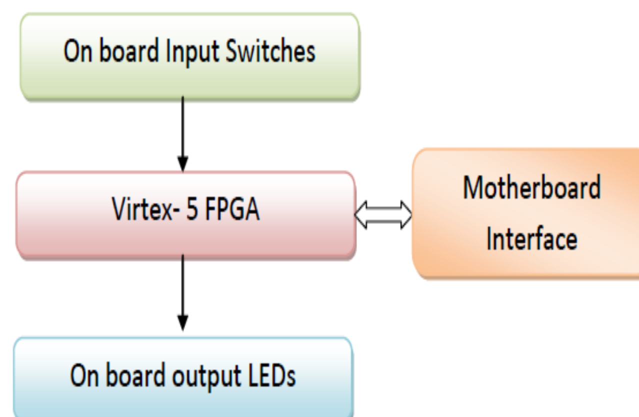


Fig. 15 FPGA synthesis block diagram

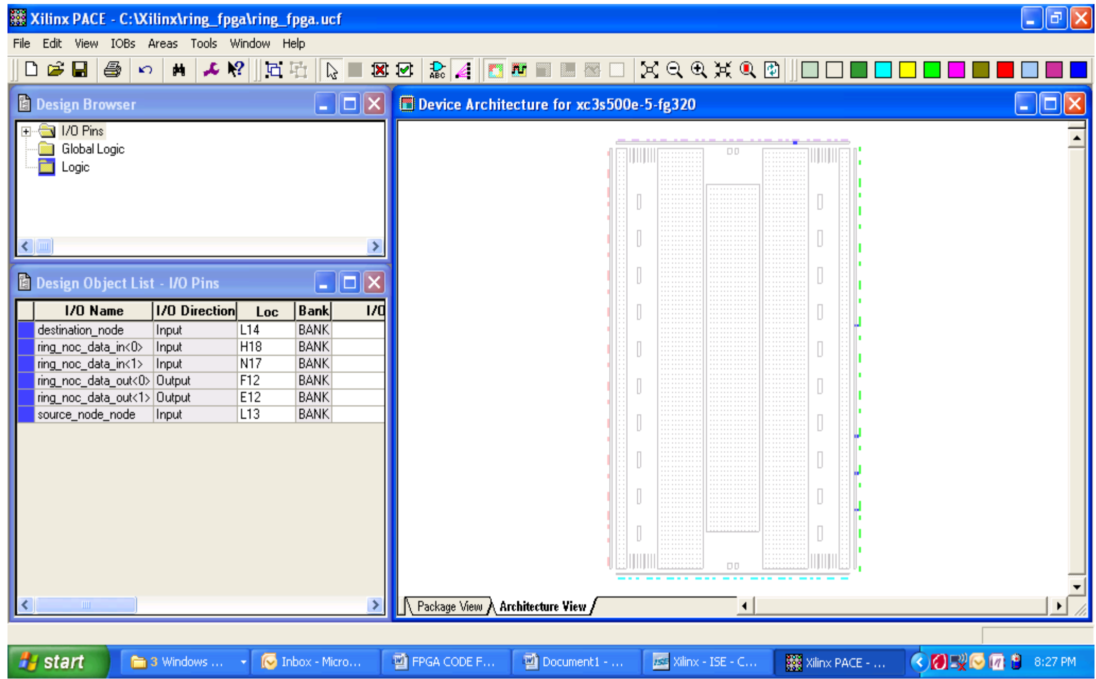


Fig.16 (A)

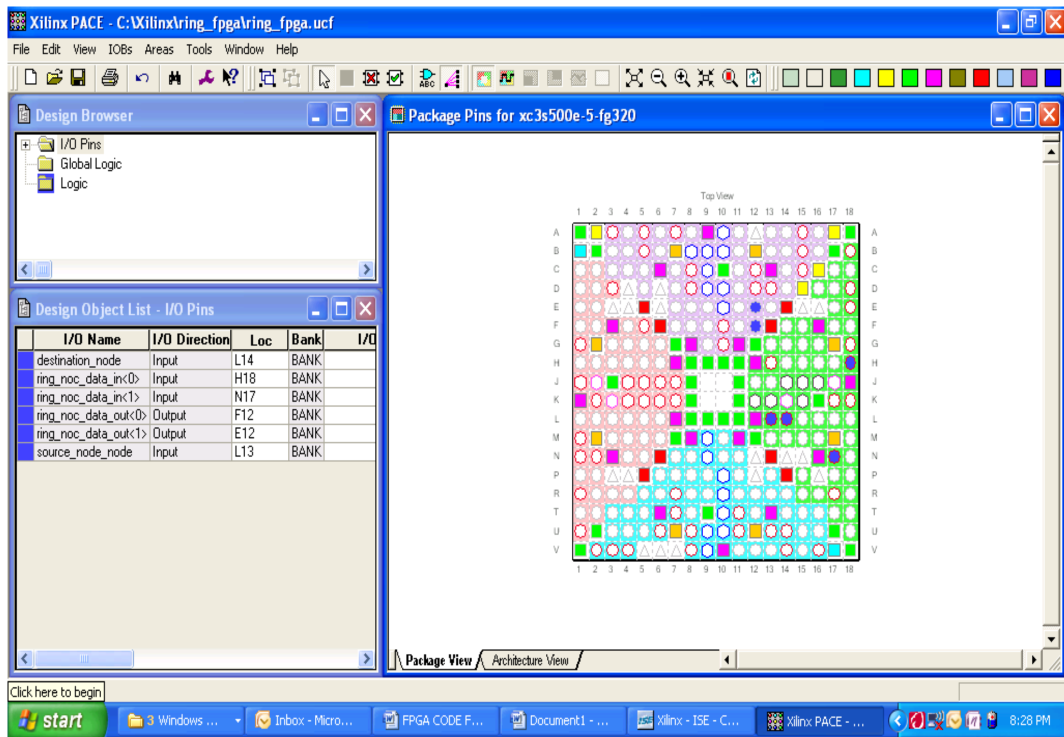


Fig. 16(B)

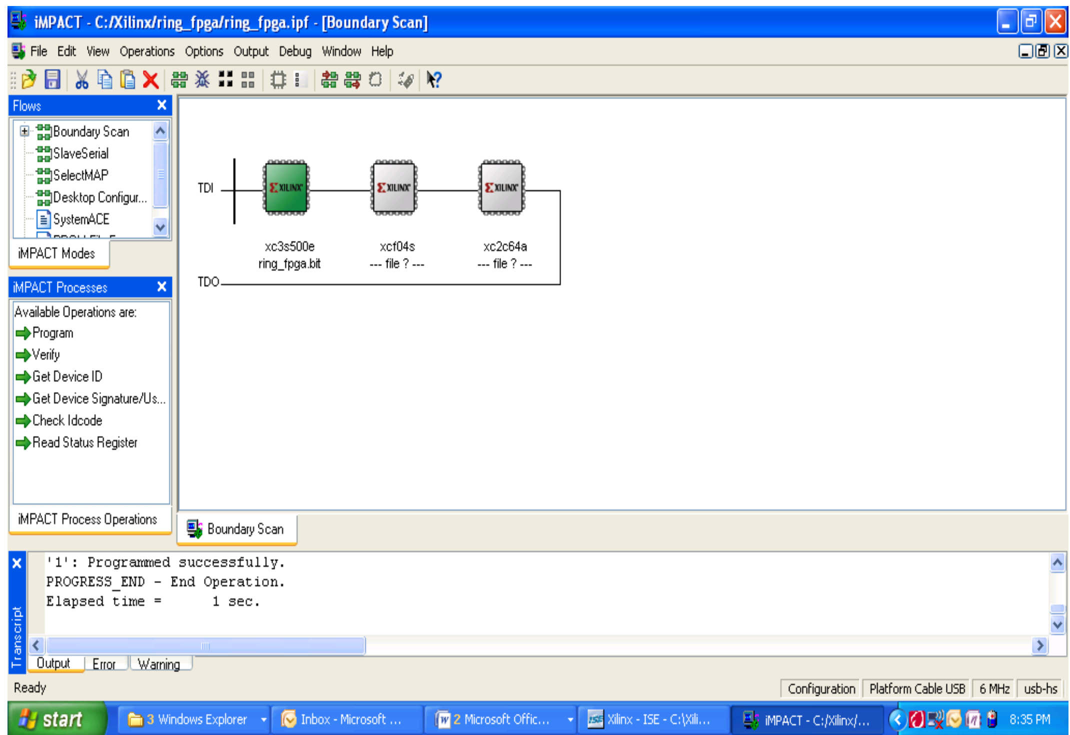


Fig. 16(C)

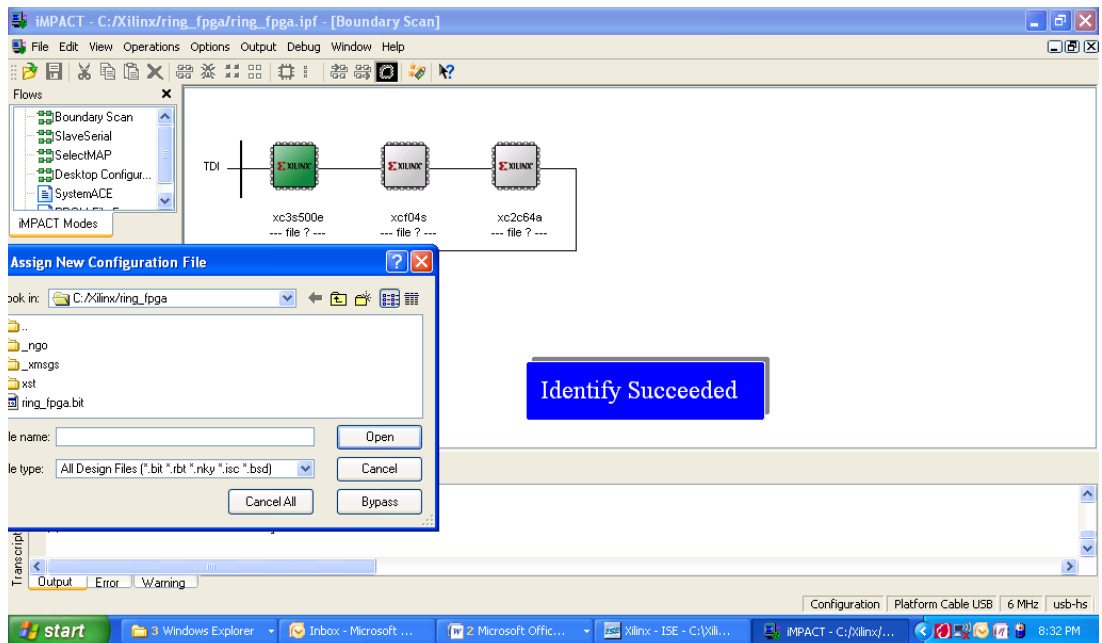


Fig. 16(D)

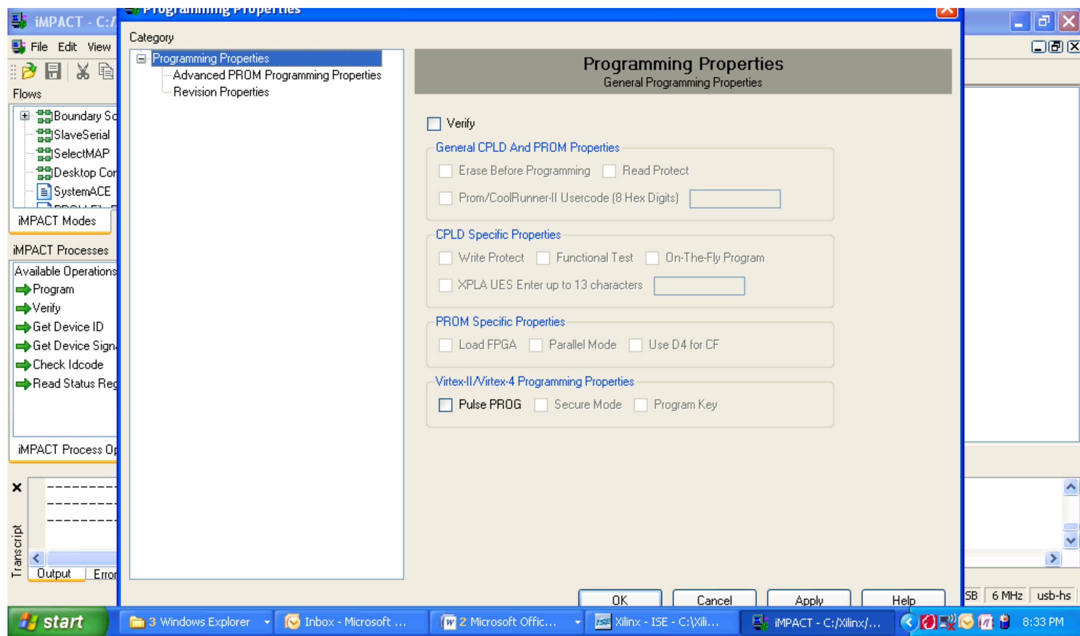


Fig. 16(E)

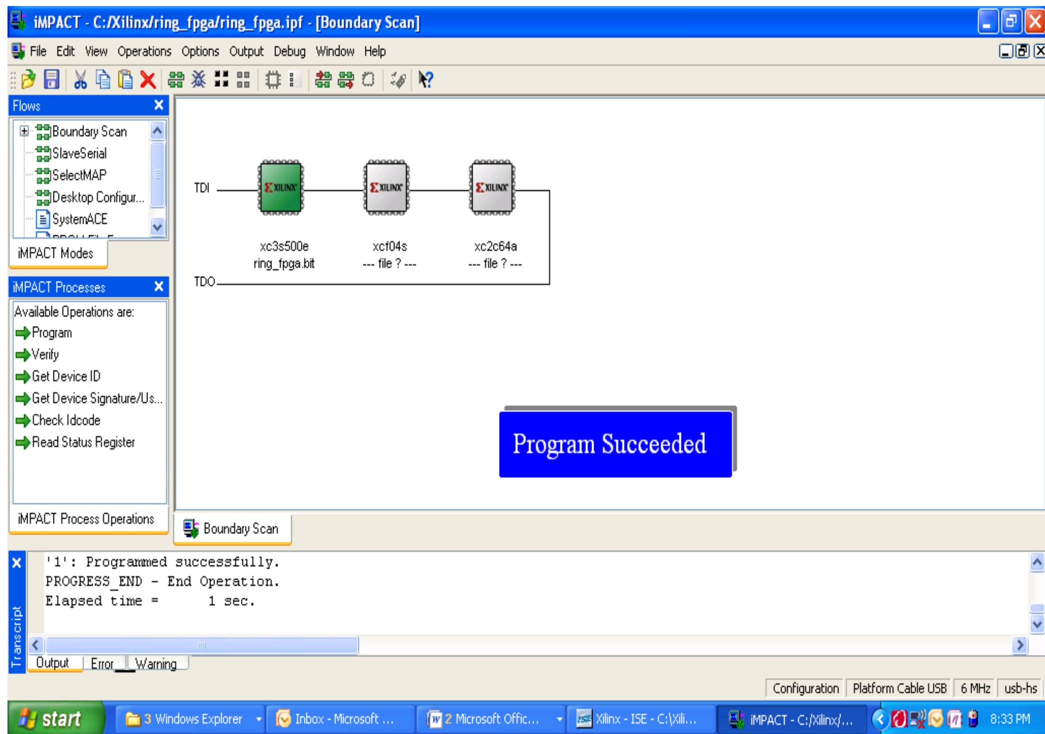


Fig. 16(F)

Fig 16 FPGA Results

6.1.Device Utilization And Timing Summary

Device utilization report gives the percentage utilization of device hardware for the chip implementation. Device hardware includes no. of slices, no. of flip flops, no. of input LUTs, no. of bounded IoBs, and no of gated clocks (GCLKs) used in the implementation of design. Timing details provides the information of delay, minimum period, maximum frequency, minimum input arrival time before clock and maximum output required time after clock. Table 3 and table 4 show the synthesis results as device utilization and timing parameters for ring NoC. Total memory utilization required to complete the design is also listed for individual stage. The target device is: xc5vlx20t-2-ff323 synthesized with Virtex-5 FPGA.

Table 3 Device utilization in ring NoC, for N = 65536

Device	Utilization	
Number of Slices	128 out of 12480,	1%
Number of Slice Flip Flops	493 out of 12480,	3%
Number of 4 input LUTs	128 out of 493,	25%
Number of bonded IOBs	156 out of 172,	90%
Number of GCLKs	1 out of 32,	3%

Table 4 Timing parameters for ring NoC, for N = 65536

Timing parameter	Utilization
Minimum period	1.867ns
Maximum frequency	535.733MHz
Minimum input arrival time before clock	4.090ns
Number of bonded IOBs	2.830ns
Total memory usage	263208 kB

The results are compared with ref [3], and ref [7]. In the ref. paper [3] the ring NoC was designed for 16 nodes, support to 2 GHz clock frequency. The ref [7], the ring NoC configuration was designed for 32 nodes on a Xilinx VP100, supports an aggregate bandwidth of about 12 GB/s. The design developed by us supports ring NoC structure for

65536 with the integration of FIFO priority logic, 535.733MHz frequency and synthesized on Virtex-5 FPGA. Moreover, Virtex -5 support the developed design for 50 MHz clock frequency and 13GBytes/sec bandwidth. Hence the developed design is an optimal solution in comparison to the existing solutions, support larger no. of nodes.

CHAPTER 7

CONCLUSION

The RoC or ring NoC chip design and modeling is done in Xilinx 14.2 and functionally simulated in Modelsim 10.1 b software. The NoC chip is designed to communicate 65536 nodes and intercommunication is checked with the data packet arrival on the destination node. There is an integration of token ring concept designed as FIFO logic, to assign the priority of communicating nodes at same instance. The nodes are identified with their node addresses and hardware parameters such as no. of logic gates, no of LUTs, memory utilization and minimum and maximum time values to route the packets are extracted from the Xilinx synthesis results. The results are validated with the data transfer among nodes with the help of Virtex-5 FPGA. The results present an optimal solution over existing ring NoC configured structures. NoC design and FPGA synthesis is a significant effort to design the programmable and reconfigurable structure for ring topological network. In future same structure can be configured for more number of nodes. It is also possible to integrate the concept of cryptographic techniques of encryption and decryption for transferring data among nodes.

REFERENCES

[1] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Evaluation of current QoS mechanisms in networks on chip," in Proceedings of the *International Symposium on System-on-Chip, (SOC'06),* pp. 1–4, Tampere, Finland, November 2006.

[2] Andreas Hansson, Kees Goossens and Andrei Radulescu "A Unified Approach to Mapping and Routing on a Network-on-Chip for Both Best-Effort and Guaranteed Service Traffic"*VLSI Design, Hindawi Publishing Corporation* Vol. 2007, pp (1-16).

[3] Ayan Mandal, Sunil P. Khatri , Rabi N. Mahapatra "A Fast, Source-synchronous Ring-based Network-on-Chip Design" *EDAA,*2012 PP(1-7).

[4] David Atienzaa, Federico Angiolini, Srinivasan Murali, Antonio Pullinid Luca Beninic, Giovanni De Michelia, "Network-on-Chip design and synthesis outlook" *Integration The VLSI Journal Elsevier*, Vol. 41 , pp(340-359), 2008.

[5] Ganghee Lee, Kiyoung Choi, and Nikil D. Dutt, "Mapping Multi-Domain Applications onto Coarse-Grained Reconfigurable Architectures" *IEEE Transaction on Computer Aided Design of Integrated Circuits and Systems*, Vol. 30, No. 5, pp (637-650) , May 2011.

[6] Francois Deslauriers, Michel Langevin, Guy Bois, Yvon Savaria, Pierre Paulin, RoC: A Scalable Network on Chip Based on the Token Ring Concept, *IEEE Xplore*, 2006, pp(157-158).

[7] Hadjiat K, St-Pierre F, Bois G, Svariya Y, Langevin M, Paulin P "An FPGA implementation of a scalable network on chip based on token ring concept" 14th IEEE *International Conference on Electronics, Circuits and Systems, (ICECS) 2007,*pp(995 – 998)

[8] HaoTian, Ajay K. Katangur, JilingZhong Yi Pan "A Novel Multistage Network Architecture with Multicast and Broadcast Capability" *The Journal of Supercomputing, Springer*, Vol.35, 2006, pp (277–300)

[9] Jason Cong, Yuhui Huang, and Bo Yuan "A Tree-Based Topology Synthesis for On-Chip Network" Computer Science Department, University of California, Los Angeles, USA, *IEEE Conference Proceedings,* pp (650-658), 2011.

[10] Muhammad Aqeel Wahlah, Kees Goossens, "A test methodology for the non-intrusive online testing of FPGA with hardwired network on chip" *Microprocessors and Microsystems*, Elsevier (2012), pp (1-18)

[11] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila "Online Reconfigurable Self-Timed Links for Fault Tolerant NoC" *VLSI Design, Hindawi Publishing Corporation* (2007), pp (1-13)

[12] Vasilis F. Pavlidis, Eby G. Friedman "3-D Topologies for Networks-on-Chip" *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 15, No. 10, October 2007,pp (1081-1091),

[13] Naveen Choudhary "Bursty Communication Performance Analysis of Network-on-Chip with Diverse Traffic Permutations", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, Issue-6, January 2012, (page 1)

APPENDIX

1. FINAL CODE

i. ROM decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
entityROMdecoder is
   Port ( A : in std_logic_vector(0 to 3);
                Write: in std_logic;
                            read: in std_logic;
D : out std_logic_vector(0 to 15));
endROMdecoder;
architecture Behavioral of ROMdecoder is
begin
process(A, write,read)
begin
if(write = '1' and read = '0') then
Case A is
when "0000" =>
D <= "1000000000000000";
when "0001" =>
D <= "0100000000000000";
when "0010" =>
D <= "0010000000000000";
when "0011" =>
D <= "0001000000000000";
when "0100" =>
D <= "0000100000000000";
```

```
when "0101" =>
D <= "0000010000000000";
when "0110" =>
D <= "0000001000000000";
when "0111" =>
D <= "0000000100000000";
when "1000" =>
D <= "0000000010000000";
when "1001" =>
D <= "0000000001000000";
when "1010" =>
D <= "0000000000100000";
when "1011" =>
D <= "0000000000010000";
when "1100" =>
D <= "0000000000001000";
when "1101" =>
D <= "0000000000000100";
when "1110" =>
D <= "0000000000000010";
when "1111" =>
D <= "0000000000000001";
when others =>          null ;
end case ;
end if;
end process ;
end Behavioral;
```

ii. ROM demux

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity  ROMdemux is
   Port ( Enable : in std_logic;
add_line : in std_logic_vector(0 to 3);
                               write: in std_logic;
                               read : in std_logic;
dest_node : out std_logic_vector(0 to 15));
end  ROMdemux;

architecture Behavioral of  ROMdemux is

begin
process( enable ,add_line,write,read )
begin
if (enable='1')then
if(write = '0' and read ='1') then
caseadd_line is
when "0000"=>
dest_node<= "1000000000000000" ;
when "0001"=>
dest_node<= "0100000000000000" ;
when "0010"=>
dest_node<= "0010000000000000" ;
when "0011"=>
dest_node<= "0001000000000000" ;
when "0100"=>
dest_node<= "0000100000000000" ;
when "0101"=>
dest_node<= "0000010000000000" ;
when "0110"=>
```

```
dest_node<= "0000001000000000" ;
when "0111"=>
dest_node<= "0000000100000000" ;
when "1000"=>
dest_node<= "0000000010000000" ;
when "1001"=>
dest_node<= "0000000001000000" ;
when "1010"=>
dest_node<= "0000000000100000" ;
when "1011"=>
dest_node<= "0000000000010000" ;
when "1100"=>
dest_node<= "0000000000001000" ;
when "1101"=>
dest_node<= "0000000000000100" ;
when "1110"=>
dest_node<= "0000000000000010" ;
when "1111"=>
dest_node<= "0000000000000001" ;
when others => null;
end case ;
end if ;
end if;
end process ;
end Behavioral;
```

iii. Token RING

```
Library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
packagering_pack is
typet_noc is array (0 to 15) of std_logic_vector(63 downto 0);
```

```
end package ring_pack;
--end of package
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
usework.ring_pack.all;
entityToken_Logic is
Port(data_in : in t_noc;
data_out : out t_noc;
source_ring_address : in std_logic_vector(0 to 3);
destination_ring_address : in std_logic_vector(0 to 3));
endToken_Logic;
architecture Behavioral of Token_Logic  is
begin
process(data_in,source_ring_address,destination_ring_address)
begin
casesource_ring_address is
when "0000" =>
Case destination_ring_address is
--Destination node 0
when "0000"=>
data_out(0) <= data_in(0);
when "0001" =>
data_out(1) <= data_in(0);
when "0010" =>
data_out(2) <= data_in(0);
when "0011" =>
data_out(3) <= data_in(0);
when "0100" =>
data_out(4) <= data_in(0);
when "0101" =>
data_out(5) <= data_in(0);
when "0110" =>
```

```
data_out(6) <= data_in(0);
when "0111" =>
data_out(7) <= data_in(0);
when "1000" =>
data_out(8) <= data_in(0);
when "1001" =>
data_out(9) <= data_in(0);
when "1010" =>
data_out(10) <= data_in(0);
when "1011" =>
data_out(11) <= data_in(0);
when "1100" =>
data_out(12) <= data_in(0);
when "1101" =>
data_out(13) <= data_in(0);
when "1110" =>
data_out(14) <= data_in(0);
when "1111" =>
data_out(15) <= data_in(0);
when others=>
null;
end case;
when "0001" =>
Case destination_ring_address is
--Destination node 1
when "0000"=>
data_out(0) <= data_in(1);
when "0001" =>
data_out(1) <= data_in(1);
when "0010" =>
data_out(2) <= data_in(1);
when "0011" =>
data_out(3) <= data_in(1);
when "0100" =>
```

```
data_out(4) <= data_in(1);
when "0101" =>
data_out(5) <= data_in(1);
when "0110" =>
data_out(6) <= data_in(1);
when "0111" =>
data_out(7) <= data_in(1);
when "1000" =>
data_out(8) <= data_in(1);
when "1001" =>
data_out(9) <= data_in(1);
when "1010" =>
data_out(10) <= data_in(1);
when "1011" =>
data_out(11) <= data_in(1);
when "1100" =>
data_out(12) <= data_in(1);
when "1101" =>
data_out(13) <= data_in(1);
when "1110" =>
data_out(14) <= data_in(1);
when "1111" =>
data_out(15) <= data_in(1);
When others=> null;
end case;
--Destination node 2
when "0010" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(2);
when "0001" =>
data_out(1) <= data_in(2);
when "0010" =>
data_out(2) <= data_in(2);
```

```
when "0011" =>
data_out(3) <= data_in(2);
when "0100" =>
data_out(4) <= data_in(2);
when "0101" =>
data_out(5) <= data_in(2);
when "0110" =>
data_out(6) <= data_in(2);
when "0111" =>
data_out(7) <= data_in(2);
when "1000" =>
data_out(8) <= data_in(2);
when "1001" =>
data_out(9) <= data_in(2);
when "1010" =>
data_out(10) <= data_in(2);
when "1011" =>
data_out(11) <= data_in(2);
when "1100" =>
data_out(12) <= data_in(2);
when "1101" =>
data_out(13) <= data_in(2);
when "1110" =>
data_out(14) <= data_in(2);
when "1111" =>
data_out(15) <= data_in(2);
When others=> null;
end case;

--Destination node 3
when "0011" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(3);
```

```
when "0001" =>
data_out(1) <= data_in(3);
when "0010" =>
data_out(2) <= data_in(3);
when "0011" =>
data_out(3) <= data_in(3);
when "0100" =>
data_out(4) <= data_in(3);
when "0101" =>
data_out(5) <= data_in(3);
when "0110" =>
data_out(6) <= data_in(3);
when "0111" =>
data_out(7) <= data_in(3);
when "1000" =>
data_out(8) <= data_in(3);
when "1001" =>
data_out(9) <= data_in(3);
when "1010" =>
data_out(10) <= data_in(3);
when "1011" =>
data_out(11) <= data_in(3);
when "1100" =>
data_out(12) <= data_in(3);
when "1101" =>
data_out(13) <= data_in(3);
when "1110" =>
data_out(14) <= data_in(3);
when "1111" =>
data_out(15) <= data_in(3);
When others=> null;
end case;

--Destination node 4
```

```
when "0100" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(4);
when "0001" =>
data_out(1) <= data_in(4);
when "0010" =>
data_out(2) <= data_in(4);
when "0011" =>
data_out(3) <= data_in(4);
when "0100" =>
data_out(4) <= data_in(4);
when "0101" =>
data_out(5) <= data_in(4);
when "0110" =>
data_out(6) <= data_in(4);
when "0111" =>
data_out(7) <= data_in(4);
when "1000" =>
data_out(8) <= data_in(4);
when "1001" =>
data_out(9) <= data_in(4);
when "1010" =>
data_out(10) <= data_in(4);
when "1011" =>
data_out(11) <= data_in(4);
when "1100" =>
data_out(12) <= data_in(4);
when "1101" =>
data_out(13) <= data_in(4);
when "1110" =>
data_out(14) <= data_in(4);
when "1111" =>
data_out(15) <= data_in(4);
```

```
When others=> null;
end case;
--Destination node 5
when "0101" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(5);
when "0001" =>
data_out(1) <= data_in(5);
when "0010" =>
data_out(2) <= data_in(5);
when "0011" =>
data_out(3) <= data_in(5);
when "0100" =>
data_out(4) <= data_in(5);
when "0101" =>
data_out(5) <= data_in(5);
when "0110" =>
data_out(6) <= data_in(5);
when "0111" =>
data_out(7) <= data_in(5);
when "1000" =>
data_out(8) <= data_in(5);
when "1001" =>
data_out(9) <= data_in(5);
when "1010" =>
data_out(10) <= data_in(5);
when "1011" =>
data_out(11) <= data_in(5);
when "1100" =>
data_out(12) <= data_in(5);
when "1101" =>
data_out(13) <= data_in(5);
when "1110" =>
```

```
data_out(14) <= data_in(5);
when "1111" =>
data_out(15) <= data_in(5);
When others=> null;
end case;

--Destination node 6
when "0110" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(6);
when "0001" =>
data_out(1) <= data_in(6);
when "0010" =>
data_out(2) <= data_in(6);
when "0011" =>
data_out(3) <= data_in(6);
when "0100" =>
data_out(4) <= data_in(6);
when "0101" =>
data_out(5) <= data_in(6);
when "0110" =>
data_out(6) <= data_in(6);
when "0111" =>
data_out(7) <= data_in(6);
when "1000" =>
data_out(8) <= data_in(6);
when "1001" =>
data_out(9) <= data_in(6);
when "1010" =>
data_out(10) <= data_in(6);
when "1011" =>
data_out(11) <= data_in(6);
when "1100" =>
```

```
data_out(12) <= data_in(6);
when "1101" =>
data_out(13) <= data_in(6);
when "1110" =>
data_out(14) <= data_in(6);
when "1111" =>
data_out(15) <= data_in(6);
When others=> null;
end case;
--Destination node 7
when "0111" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(7);
when "0001" =>
data_out(1) <= data_in(7);
when "0010" =>
data_out(2) <= data_in(7);
when "0011" =>
data_out(3) <= data_in(7);
when "0100" =>
data_out(4) <= data_in(7);
when "0101" =>
data_out(5) <= data_in(7);
when "0110" =>
data_out(6) <= data_in(7);
when "0111" =>
data_out(7) <= data_in(7);
when "1000" =>
data_out(8) <= data_in(7);
when "1001" =>
data_out(9) <= data_in(7);
when "1010" =>
data_out(10) <= data_in(7);
```

```
when "1011" =>
data_out(11) <= data_in(7);
when "1100" =>
data_out(12) <= data_in(7);
when "1101" =>
data_out(13) <= data_in(7);
when "1110" =>
data_out(14) <= data_in(7);
when "1111" =>
data_out(15) <= data_in(7);
When others=> null;
end case;

--Destination node 8
when "1000" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(8);
when "0001" =>
data_out(1) <= data_in(8);
when "0010" =>
data_out(2) <= data_in(8);
when "0011" =>
data_out(3) <= data_in(8);
when "0100" =>
data_out(4) <= data_in(8);
when "0101" =>
data_out(5) <= data_in(8);
when "0110" =>
data_out(6) <= data_in(8);
when "0111" =>
data_out(7) <= data_in(8);
when "1000" =>
data_out(8) <= data_in(8);
```

```
when "1001" =>
data_out(9) <= data_in(8);
when "1010" =>
data_out(10) <= data_in(8);
when "1011" =>
data_out(11) <= data_in(8);
when "1100" =>
data_out(12) <= data_in(8);
when "1101" =>
data_out(13) <= data_in(8);
when "1110" =>
data_out(14) <= data_in(8);
when "1111" =>
data_out(15) <= data_in(8);
When others=> null;
end case;

--Destination node 9
when "1001" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(9);
when "0001" =>
data_out(1) <= data_in(9);
when "0010" =>
data_out(2) <= data_in(9);
when "0011" =>
data_out(3) <= data_in(9);
when "0100" =>
data_out(4) <= data_in(9);
when "0101" =>
data_out(5) <= data_in(9);
when "0110" =>
data_out(6) <= data_in(9);
```

```
when "0111" =>
data_out(7) <= data_in(9);
when "1000" =>
data_out(8) <= data_in(9);
when "1001" =>
data_out(9) <= data_in(9);
when "1010" =>
data_out(10) <= data_in(9);
when "1011" =>
data_out(11) <= data_in(9);
when "1100" =>
data_out(12) <= data_in(9);
when "1101" =>
data_out(13) <= data_in(9);
when "1110" =>
data_out(14) <= data_in(9);
when "1111" =>
data_out(15) <= data_in(9);
When others=> null;
end case;

--Destination node 10
when "1010" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(10);
when "0001" =>
data_out(1) <= data_in(10);
when "0010" =>
data_out(2) <= data_in(10);
when "0011" =>
data_out(3) <= data_in(10);
when "0100" =>
data_out(4) <= data_in(10);
```

```
when "0101" =>
data_out(5) <= data_in(10);
when "0110" =>
data_out(6) <= data_in(10);
when "0111" =>
data_out(7) <= data_in(10);
when "1000" =>
data_out(8) <= data_in(10);
when "1001" =>
data_out(9) <= data_in(10);
when "1010" =>
data_out(10) <= data_in(10);
when "1011" =>
data_out(11) <= data_in(10);
when "1100" =>
data_out(12) <= data_in(10);
when "1101" =>
data_out(13) <= data_in(10);
when "1110" =>
data_out(14) <= data_in(10);
when "1111" =>
data_out(15) <= data_in(10);
When others=> null;
end case;

--Destination node 11
when "1011" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(11);
when "0001" =>
data_out(1) <= data_in(11);
when "0010" =>
data_out(2) <= data_in(11);
```

```
when "0011" =>
data_out(3) <= data_in(11);
when "0100" =>
data_out(4) <= data_in(11);
when "0101" =>
data_out(5) <= data_in(11);
when "0110" =>
data_out(6) <= data_in(11);
when "0111" =>
data_out(7) <= data_in(11);
when "1000" =>
data_out(8) <= data_in(11);
when "1001" =>
data_out(9) <= data_in(11);
when "1010" =>
data_out(10) <= data_in(11);
when "1011" =>
data_out(11) <= data_in(11);
when "1100" =>
data_out(12) <= data_in(11);
when "1101" =>
data_out(13) <= data_in(11);
when "1110" =>
data_out(14) <= data_in(11);
when "1111" =>
data_out(15) <= data_in(11);
When others=> null;
end case;

--Destination node 12
when "1100" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(12);
```

```
when "0001" =>
data_out(1) <= data_in(12);
when "0010" =>
data_out(2) <= data_in(12);
when "0011" =>
data_out(3) <= data_in(12);
when "0100" =>
data_out(4) <= data_in(12);
when "0101" =>
data_out(5) <= data_in(12);
when "0110" =>
data_out(6) <= data_in(12);
when "0111" =>
data_out(7) <= data_in(12);
when "1000" =>
data_out(8) <= data_in(12);
when "1001" =>
data_out(9) <= data_in(12);
when "1010" =>
data_out(10) <= data_in(12);
when "1011" =>
data_out(11) <= data_in(12);
when "1100" =>
data_out(12) <= data_in(12);
when "1101" =>
data_out(13) <= data_in(12);
when "1110" =>
data_out(14) <= data_in(12);
when "1111" =>
data_out(15) <= data_in(12);
When others=> null;
end case;

--Destination node 13
```

```
when "1101" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(13);
when "0001" =>
data_out(1) <= data_in(13);
when "0010" =>
data_out(2) <= data_in(13);
when "0011" =>
data_out(3) <= data_in(13);
when "0100" =>
data_out(4) <= data_in(13);
when "0101" =>
data_out(5) <= data_in(13);
when "0110" =>
data_out(6) <= data_in(13);
when "0111" =>
data_out(7) <= data_in(13);
when "1000" =>
data_out(8) <= data_in(13);
when "1001" =>
data_out(9) <= data_in(13);
when "1010" =>
data_out(10) <= data_in(13);
when "1011" =>
data_out(11) <= data_in(13);
when "1100" =>
data_out(12) <= data_in(13);
when "1101" =>
data_out(13) <= data_in(13);
when "1110" =>
data_out(14) <= data_in(13);
when "1111" =>
data_out(15) <= data_in(13);
```

```
When others=> null;
end case;

--Destination node 14
when "1110" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(14);
when "0001" =>
data_out(1) <= data_in(14);
when "0010" =>
data_out(2) <= data_in(14);
when "0011" =>
data_out(3) <= data_in(14);
when "0100" =>
data_out(4) <= data_in(14);
when "0101" =>
data_out(5) <= data_in(14);
when "0110" =>
data_out(6) <= data_in(14);
when "0111" =>
data_out(7) <= data_in(14);
when "1000" =>
data_out(8) <= data_in(14);
when "1001" =>
data_out(9) <= data_in(14);
when "1010" =>
data_out(10) <= data_in(14);
when "1011" =>
data_out(11) <= data_in(14);
when "1100" =>
data_out(12) <= data_in(14);
when "1101" =>
data_out(13) <= data_in(14);
```

```
when "1110" =>
data_out(14) <= data_in(14);
when "1111" =>
data_out(15) <= data_in(14);
When others=> null;
end case;

--Destination node 15
when "1111" =>
Case destination_ring_address is
when "0000"=>
data_out(0) <= data_in(15);
when "0001" =>
data_out(1) <= data_in(15);
when "0010" =>
data_out(2) <= data_in(15);
when "0011" =>
data_out(3) <= data_in(15);
when "0100" =>
data_out(4) <= data_in(15);
when "0101" =>
data_out(5) <= data_in(15);
when "0110" =>
data_out(6) <= data_in(15);
when "0111" =>
data_out(7) <= data_in(15);
when "1000" =>
data_out(8) <= data_in(15);
when "1001" =>
data_out(9) <= data_in(15);
when "1010" =>
data_out(10) <= data_in(15);
when "1011" =>
data_out(11) <= data_in(15);
```

```
when "1100" =>
data_out(12) <= data_in(15);
when "1101" =>
data_out(13) <= data_in(15);
when "1110" =>
data_out(14) <= data_in(15);
when "1111" =>
data_out(15) <= data_in(15);
When others=> null;
end case;
when others=> null;
end case;
end process;
end  Behavioral;
```

iv. MAIN PROGRAM

```
--program
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
usework.ring_pack.all;
entityring_noc is
   Port ( node_data_in : in t_noc;
node_data_out : out t_noc;
clk : in std_logic;
reset : in std_logic;
read : in std_logic;
write : in std_logic;
source_node_address : in std_logic_vector(0 to 3);
                              destination_node_address : in std_logic_vector(0 to 3);
enable : in std_logic);
endring_noc;
architecture Behavioral of ring_noc is
```

```
componentROMdecoder is
Port (A : in std_logic_vector(0 to 3);
  Write: in std_logic;
read: in std_logic;
D : out std_logic_vector(0 to 15));
end component ;
component  ROMdemux is
Port(enable : in std_logic;
add_line : in std_logic_vector(0 to 3);
Write: in std_logic;
read: in std_logic;
dest_node : out std_logic_vector(0 to 15));
end component ;
componenttoken_Logic is
    Port (data_in : in t_noc;
data_out : out t_noc;
source_ring_address : in std_logic_vector(0 to 3);
                              destination_ring_address : in std_logic_vector(0 to 3));
end component;

SIGNAL M : std_logic_vector(0 to 15);
signal N : std_logic_vector(0 to 15);
begin

U1: ROMdecoder port map(A(3)=>source_node_address(3),
A(2)=>source_node_address(2),
                                        A(1)=>source_node_address(1),
                                        A(0)=>source_node_address(0),
                                        D(0)=> M(0),
                                        D(1)=> M(1) ,
                                        D(2)=> M(2),
                                        D(3)=> M(3),
                                        D(4)=> M(4),
```

```
D(5)=> M(5),
D(6)=> M(6),
D(7)=> M(7),
D(8)=> M(8),
D(9)=> M(9),
D(10)=> M(10),
D(11)=>        M(11),
D(12)=>        M(12),
D(13)=> M(13),
D(14)=> M(14),
D(15)=> M(15),
        write => write,
        read => read);
U2 :ROMdemux port map (enable=> enable ,
add_line(0)=>destination_node_address(0),
        add_line(1)=>destination_node_address(1),
        add_line(2)=>destination_node_address(2),
        add_line(3)=>destination_node_address(3),
        dest_node(0)=> N(0),
        dest_node(1)=>  N(1),
        dest_node(2)=>  N(2),
        dest_node(3)=>  N(3),
        dest_node(4)=>  N(4),
        dest_node(5)=>  N(5),
        dest_node(6)=>  N(6),
        dest_node(7)=>  N(7),
        dest_node(8)=>  N(8),
        dest_node(9)=>  N(9),
        dest_node(10)=> N(10),
        dest_node(11)=> N(11),
        dest_node(12)=> N(12),
        dest_node(13)=>  N(13),
        dest_node(14)=>  N(14),
        dest_node(15)=>  N(15),
```

```
                              read => read,
                              write => write);

U3: token_Logic port map(data_in =>node_data_in,
                    data_out =>node_data_out ,
                    source_ring_address(0) =>source_node_address(0),
                    source_ring_address(1) =>source_node_address(1),
                    source_ring_address(2) =>source_node_address(2),
                    source_ring_address(3) =>source_node_address(3),
                    destination_ring_address(0) =>destination_node_address(0),
                    destination_ring_address(1) =>destination_node_address(1),
                    destination_ring_address(2) =>destination_node_address(2),
                    destination_ring_address(3)=>destination_node_address(3));

end Behavioral;
```

2. FPGA CODE FOR TWO NODES

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ring_noc_fpga is
   Port (ring_noc_data_in : in  STD_LOGIC_VECTOR (1 downto 0);
         ring_noc_data_out : out  STD_LOGIC_VECTOR (1 downto 0);
         source_node_node : in  STD_LOGIC;
         destination_node : in  STD_LOGIC);
end ring_noc_fpga;

architecture Behavioral of ring_noc_fpga is
begin
process(source_node_node,destination_node,ring_noc_data_in)
begin
case source_node_node is
when '0' =>    -- node 0
case destination_node is
when '0' =>
ring_noc_data_out <= ring_noc_data_in;
when '1' =>
ring_noc_data_out <= ring_noc_data_in;

when others=>
null;
end case;

when '1' =>    -- node 1
case destination_node is
when '0' =>
ring_noc_data_out <= ring_noc_data_in;
```

```
when '1' =>
ring_noc_data_out <= ring_noc_data_in;

when others=>
null;
end case;

when others=>
null;
end case;

end process;

end Behavioral;
```

3. RESEARCH PUBLICATION

The work done in our final year project of our B.Tech study was published in the form of research paper tiltled "ROTATOR ON CHIP (ROC) DESIGN BASED ON RING TOPOLOGICAL NETWORK ON CHIP (NOC)" in Elsevier- Procedia Computer science Journal, Vol.45, March, 2015. The paper was also presented in a paper conference ICACTA-2015 organized by Dwarkadas J. Sanghvi College of Engineering, Mumbai held on 26th and 27th of March, 2015.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
Vile Parle (W) Mumbai - 400 056.

DJ ICACTA

INTERNATIONAL CONFERENCE ON ADVANCED COMPUTING TECHNOLOGIES AND APPLICATIONS

ELSEVIER

Certificate of Participation

This is to certify that **Dr./Mr./Ms.** LOPAMUDRA BARUAH *has participated in* **ICACTA-2015** *organized by* **Dwarkadas J. Sanghvi College of Engineering**, *Mumbai held on 26th & 27th of March, 2015. He has Authored/Co-Authored a paper titled* " ROTATOR ON CHIP (RoC) DESIGN BASED ON RING TOPOLOGICAL NETWORK ON CHIP (NoC). "

Dr. N. M Shekokar	Dr. A. R. Joshi	Dr. Hari Vasudevan
Program Co-Chair	Program Chair	General Chair
Prof. & Head, Dept. of Comp. Engg.	Vice Principal (Acad.), Prof. & Head, Dept. of IT	Principal