AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
Computer and Systems Engineering
International Credit Hours Engineering Programs (i.CHEP)

Research Project Report

Design, Implementation and Test of a Networks-on-Chip (NoC) Router using VHDL

Course Code **CSE215**	*Course Name* **Electronic Design Automation**	
	Semester Spring 2020	**Date of Submission**

#	Student ID	Grade (PASS/FAIL)
1	**17P6006**	
2	**17P3067**	
3	**17P6018**	
4	**17P6024**	
5	**17P6019**	

In case of group research; list all students IDs.
DO NOT WRITE STUDENTS NAMES

"I certify that this report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they are books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this report has not been previously been submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons."

STUDENTS MUST SIGN THIS PAGE. ELECTRONIC SIGNATURE IS ACCEPTED.

#	Student ID	Student Signature
1	**17P6006**	
2	**17P3067**	
3	**17P6018**	
4	**17P6019**	
5	**17P6024**	

In case of group research; list all students IDs.
DO NOT WRITE STUDENTS NAMES

Table of Contents

1 INTRODUCTION

A router is a physical or virtual device that buffers and sends information between two or more packet-switched computer networks. The router inspects the destination Internet Protocol (IP) address of the data packet and calculates the best way to reach its destination and then forwards it accordingly. High throughput routers will allow the network-on-chip to meet the communication needs of multi-core applications, or higher attainable throughput that can be traded for power savings with less resources being used to achieve the aimed bandwidth. Also, providing high throughput is also crucial for minimizing the delay time for applications which have high communication workloads as the more the network approaches saturation, the faster the queueing delays will grow.

1.1 Objectives

This study aims to focus on the implementation and the verification of a four-port router. The building blocks of the router are buffering registers, demultiplexer, FIFO registers, and schedulers. The scheduler uses the round robin algorithm. The proposed architecture of four-port router. The source code is written in VHDL.

1.2 Benefits of Noc Router

Network-on-Chip (Noc) architectures are setting off to be - in fact – the foundation for both application-specific systems-on-chips (SoCs) and for general-purpose chip multi-processors (CMPs) .In the design of NoCs, low latency and high throughput are equally important design parameters and the router microarchitecture has a crucial function in achieving meeting these performance aims. The role of the router lies in the efficient multiplexing of packets into the network links. Router buffering is used to house arriving flits1 that can’t instantly be forwarded to the output links because of contention. Such buffering will be done either at the router's inputs or outputs, which refers to an input buffered router (IBR) or an output buffered router (OBR).

Types of routers

Routers can be classified according to:

- Switch the fabric (SF)
- the location of buffers and queues in the router.

There is 5 types of Routers:

- Wireless
- Wired Router
- Edge Router
- Core Router
- Virtual Router

1. Wireless

It is present in office, home or railway station, etc. It creates a wireless signal. Suppose you are in office; we can connect to the internet using wireless signals because your laptop is within the range. We can provide security to routers by entering user id and password. When we try to connect to the router, it will ask for a password and User ID. User ID and password comes along with the device. Due to security, no information about the user is harmed. When we visit public places, we can observe that on our phone Wi-Fi window will prop up to use the internet and you can observe that it is secure with a password. Wireless routers are publicly available. N number of users can connect to it.

2. **Wired Router**

Name itself defines its meaning. Wire is available to connect to the network. If we visit a bank or small college or office, we can observe that PC or Laptop is connected to the internet using Ethernet cable and that is the wired router. It has a separate Wi-Fi access point. If a user wants to connect to the phone, then they can use VIOP (voice-over IP technology). There is an ADSL (modem) that has two jacks to connect to ethernet and mobile phones.

3. **Edge Router**

It seats at the edge of the backbone of the network and can connect to the core routers. It can be wired or wireless and will distribute internet data packets between one or more networks. But it will not distribute internet data packets within networks.

4. **Core Router**

It is designed to operate in the internet backbone or core. It supports multiple telecommunication interfaces of the highest speed and usage in the core internet. It can forward IP packets at full speed on all of them. It supports the routing protocol that is used in the core. It will distribute internet data packets within the network. But core will not distribute internet data packets between networks.

5. **Virtual Router**

It is default for a computer sharing network. It functions as per the virtual router redundancy protocol (VRRP), it becomes active when the main or primary router fails or becomes disabled. It takes multiple routers in a group so that they can share a virtual IP address. It has a master for each group that handles IP packets. If the master fails while forwarding packets then other routers will take a position

2 Design Flow:

An integrated circuit, or IC, is small chip that can function as an amplifier, oscillator, timer, microprocessor, or even computer memory. An IC is a small wafer, usually made of silicon, that can hold anywhere from hundreds to millions of transistors, resistors, and capacitors. These extremely small electronics can perform calculations and store data using either digital or analog technology.

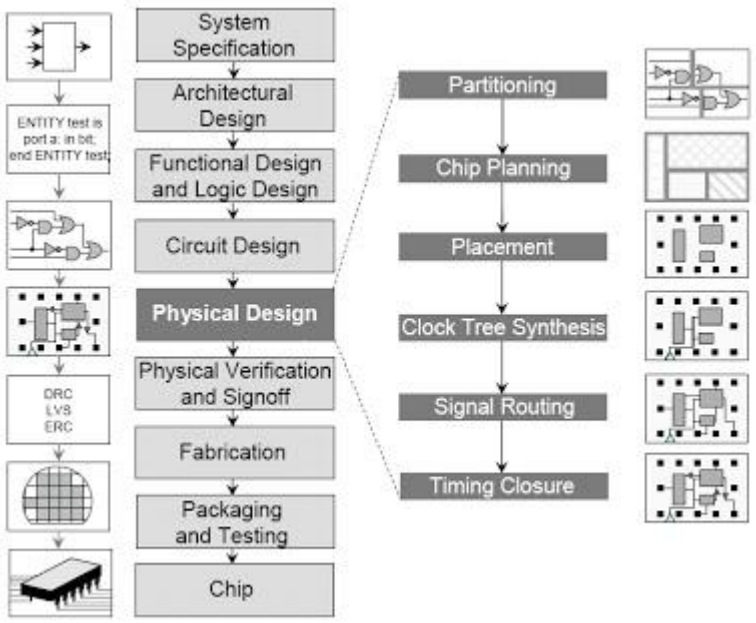
Digital ICs use logic gates, which work only with values of ones and zeros. A low signal sent to a component on a digital IC will result in a value of 0, while a high signal creates a value of 1. Digital ICs are the kind you will usually find in computers, networking equipment, and most consumer electronics.

In the network on Chip (NOC) router, it consists of subsystem like register, ram, counter, gray to binary converter in which all components together make up the router as a whole functioning chip.

The four basic steps of design flow are

- Design
- Simulation
- Synthesis, Analysis and verification
- Layout, Manufacturing and Preparing

The digital design and verification flows and the steps that must be performed to complete the logic and physical synthesis are as below, from system specification till chip component.



System Specification

- Feasibility study and size estimate
- Function analysis

Architectural Design:

- Structure of the system.

Functional and Logic Design

- Digital Design, Simulation, verification and Layout

Circuit Design

- Digital design synthesis
- Design for test
- Design for manufacturability (IC)

Physical Design

- Floor planning
- Place and Route

Physical Verification

- Timing and critical paths verification

For the network on chip Router, the steps below were done.

- **Requirement analysis**
- **Design entry:** Hardware description language HDL was chosen as a design entry. And project files were created.
- **Implementation:** Implementing each module as specified in the project documentation.
- **Testbench**
- **Simulation**
- **Block diagram generation**
- **Timing analysis and verification**

CAD tools that could be used to complete the chip design flow

- Cadance - electronic design automation software
- Verilog or VHDL- hardware description language (HDL) used to model electronic systems, which was used.
- LASI 7- LASI is shortening for Layout System for Individuals. LASI is a PC-based CAD program used for design of the physical layout of integrated circuits.
- Xlinix

3 Literature Review:

Introduction

In a world of a very fast-paced changing of technology, a vast sector of devices is integrated into a single chip. With this being made the persistent need of communication between devices was essential; as it's important for different devices to send and receive different date with each other. Considering how slow the communication between devices via the common bus architecture, The network on chip (NoC) technology is used for such communication. It promises flexibility, stability and a higher bandwidth. This study aims to focus on various implementations of Noc and different ways of tolerating faults of communication among the nodes and henceforth increasing the speed of processing and preserve throughput. Going through previous studies It's been noted that the mostly the main blocks of the router are the same as the proposed module in the report of which it contains mainly three parts (Registers and demultiplexers, First In First Out Registers, and Schedulers) as found in [1] [2] [3] so in this review we will divaricate into studies about different switching techniques and how routers tolerate faults.

Switching techniques

There are many switching techniques that are used for sending data via network. A switched network is a network that is made up of a series of interconnected nodes called switches. Literature on related topics suggests that packet switching is the most efficient type of switching that is used in NoC designs *(Meenakshi M. Wangari, R. V. Kshirsagar)* [4]. There are more inner categories of packet switching Store and Forward (SAF), Wormhole (WH) and Virtual Cut Through (VCT). However, these techniques have to face Head-on-Line (HoL) blocking problem. *J. Kim. Dally* (2015) has suggested the idea of a virtual channel for deadlock-free routing for networks. Virtual channel router is used to solve network deadlock by accepting adaptive routing and provide guaranteed service and best-effort service, as Network-on-Chip is a strict resource constrained, so virtual-channel router should make a better adjustment between performance and implementation

cost. *J. K. Dally* illustrated the basic architecture of a virtual channel router and showed how the virtual channel router works in a pipeline in order to decrease the time delay [5].

Fault tolerance

According to (*Avizienis, A.; Laprie, J.C.; Randell, B.; Landwehr, C.)* [6] there is a relation between fault, error and failure. A fault might manifest an error which in turn may result in creating an abnormal behavior which is the called a failure. Faults and errors can easily be masked to prevent failure occurrence. Faults and errors can be classified according to their duration and can be transient, intermittent, or permanent. Mainly, former studies addressed both transient and permanent faults in NoC. Such as, the works in [7] studied Single Event Upset (SEU) in NoC components and dealt with transient faults that occurs due to crosstalk. The authors *(Tosun, S.; Ajabshir, V.B.; Mercanoglu, O.; Ozturk, O)* [8] investigated the faults that happens due to short and open circuit in the linking of an NoC.

(Pallavi B V, Dr. Baswaraj Gadgay, Veeresh Pujari) stated that fault does not occur only because of the change in the estimation of the sent message but also happens because of the wrong steering way. They added that it's very complicated To differentiate between the bypass of a busy element from accurate routing flaw. There will be the possibility of packets being lost, If this routing flaw is not identified. Thus, they claimed that in order to identify this error made in routing depends on the slanting state indications. They proposed a method to recognize faults which is routing the date during runtime included in a switch. the routing algorithm can also be operated along with this method. The main idea of this method is that the received data should be checked by each router whether the routing made by the previous mode is correct or not. This is done after data error correction is done [9]. According to *(Radetzki, M.; Feng, C.; Zhao, X.; Jantsch,)* (2013) Redundancy is the basic mechanism for providing fault tolerance in a system [10]. This

mechanism aims to detect and, in some cases, to fix errors in the components. Redundancy techniques can be classified as Spatial, Temporal, and Information and are applied in NoCs as follows

- Spatial redundancy involves the addition of circuits, with the replication of modules and compare their outputs by a voter. Which is is often performed using Dual Modular Redundancy (DNR)

- Temporal redundancy involves the re-execution of an operation resulting in comparison and validation. It is typically implemented by running an algorithm n times on the same hardware.

- Information redundancy relies on additional bits for error detection and correction. Error-Detecting Code (EDC) techniques can detect an error incidence, while Error-Correcting Code (ECC) not only detect, but also correct an erroneous data word [10].

There are two main types of error recovery techniques. The first one is Forward Error Recovery (FER). It enables operational continuity in the presence of errors, without having to return to a preceding state [11] [12].

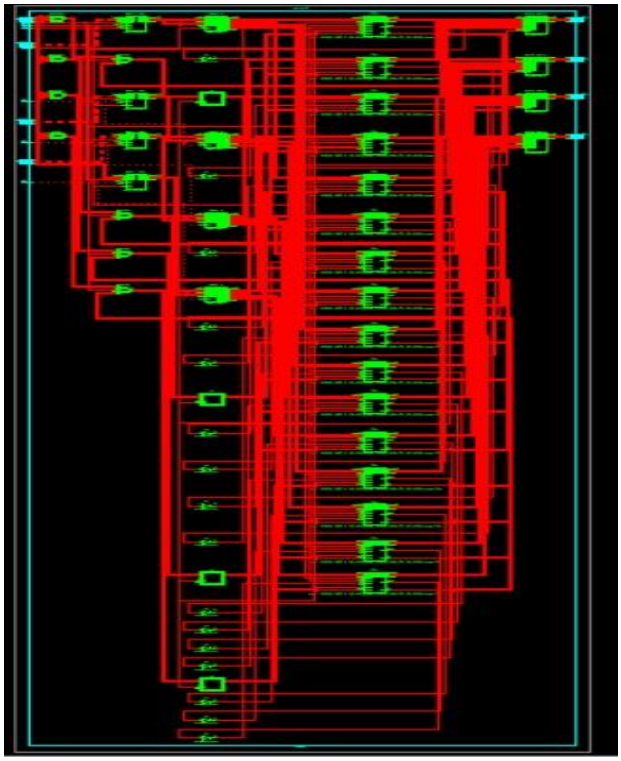
The second one is Backward Error Recovery (BER) which ensure the system the ability to return to its previous state when it is healthy. They use checkpoints or logs and mainly require additional memory elements to preserve these states. according to *(Kohler, A.; Schley, G.; Radetzki*) (2010) FER is more widely used than BER because the BER requires more memory elements, which makes the system more expensive and more susceptible to SEU faults [13].

4 Design Implementation:

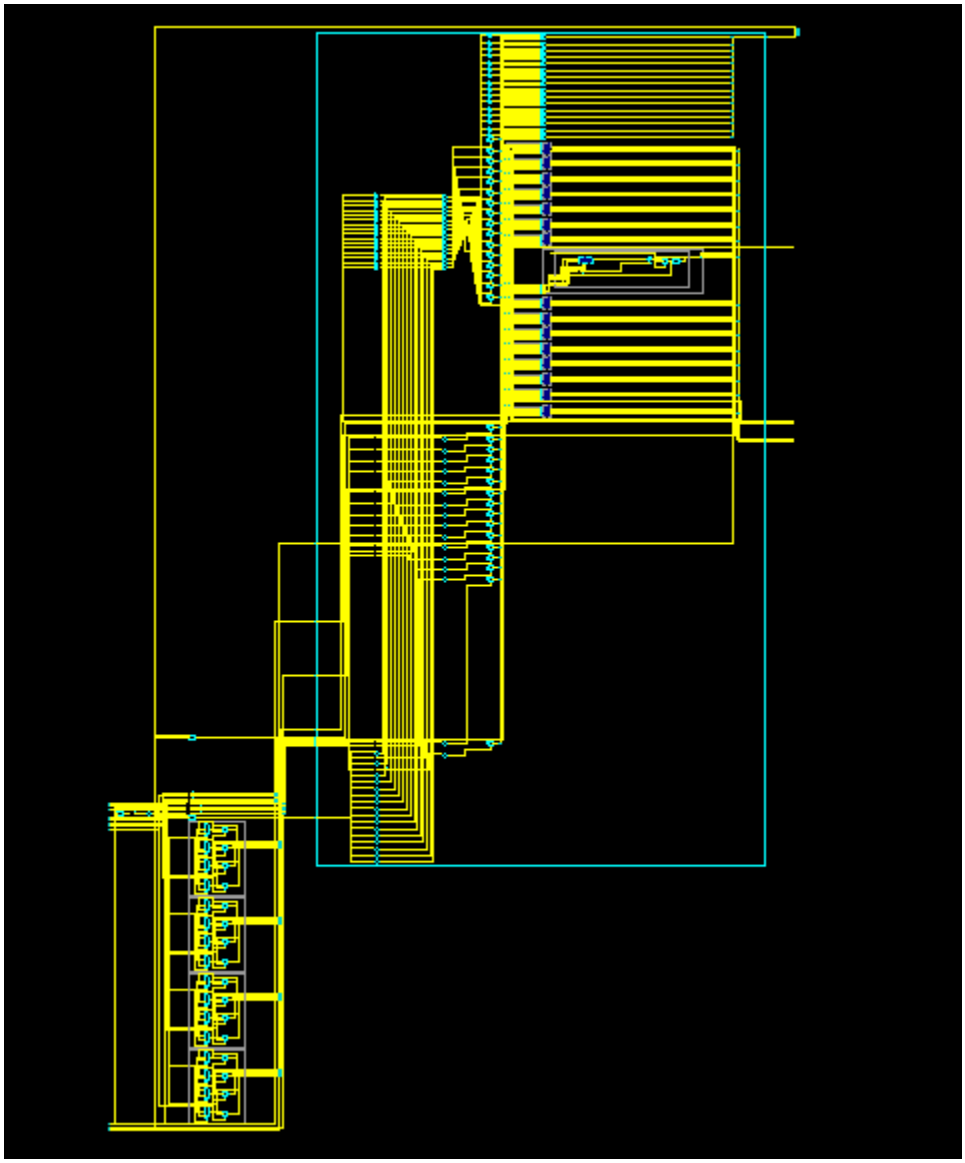
In this section we will discuss each Modules of NoC-router and its function and Challenges faced its implementation.

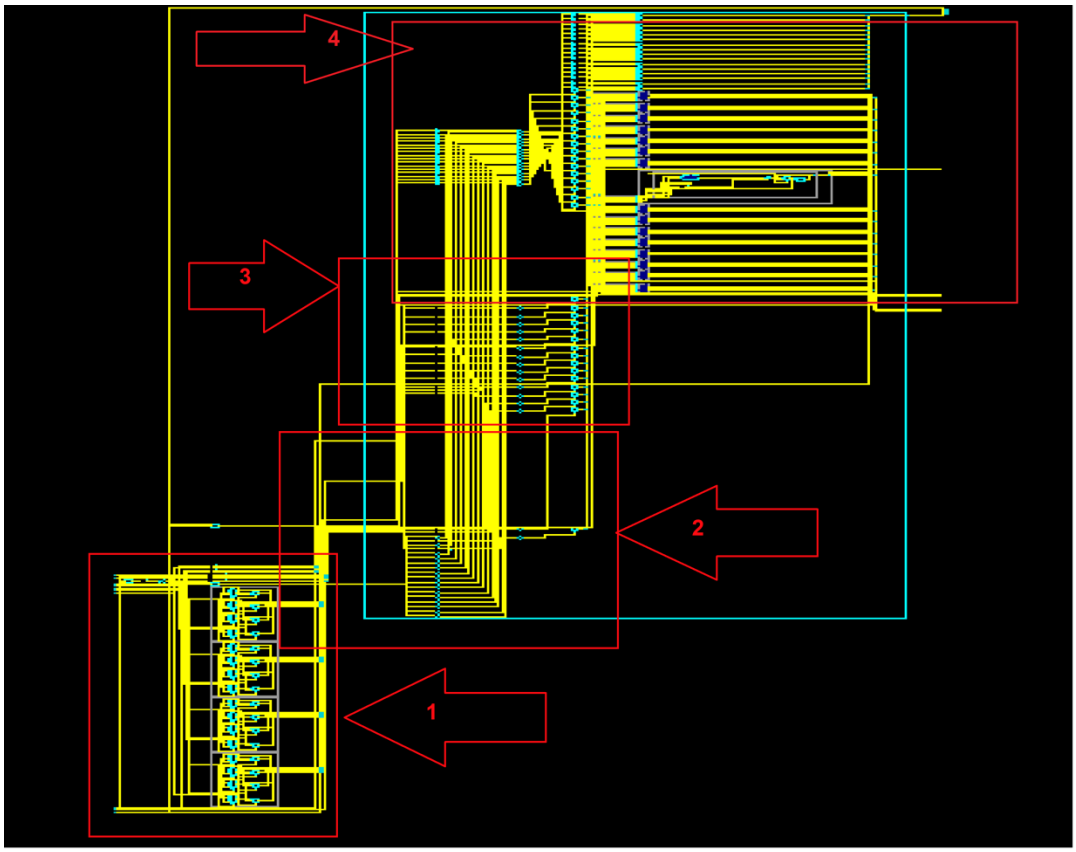
4.1 Block Diagram:

Xilinx Screenshot:

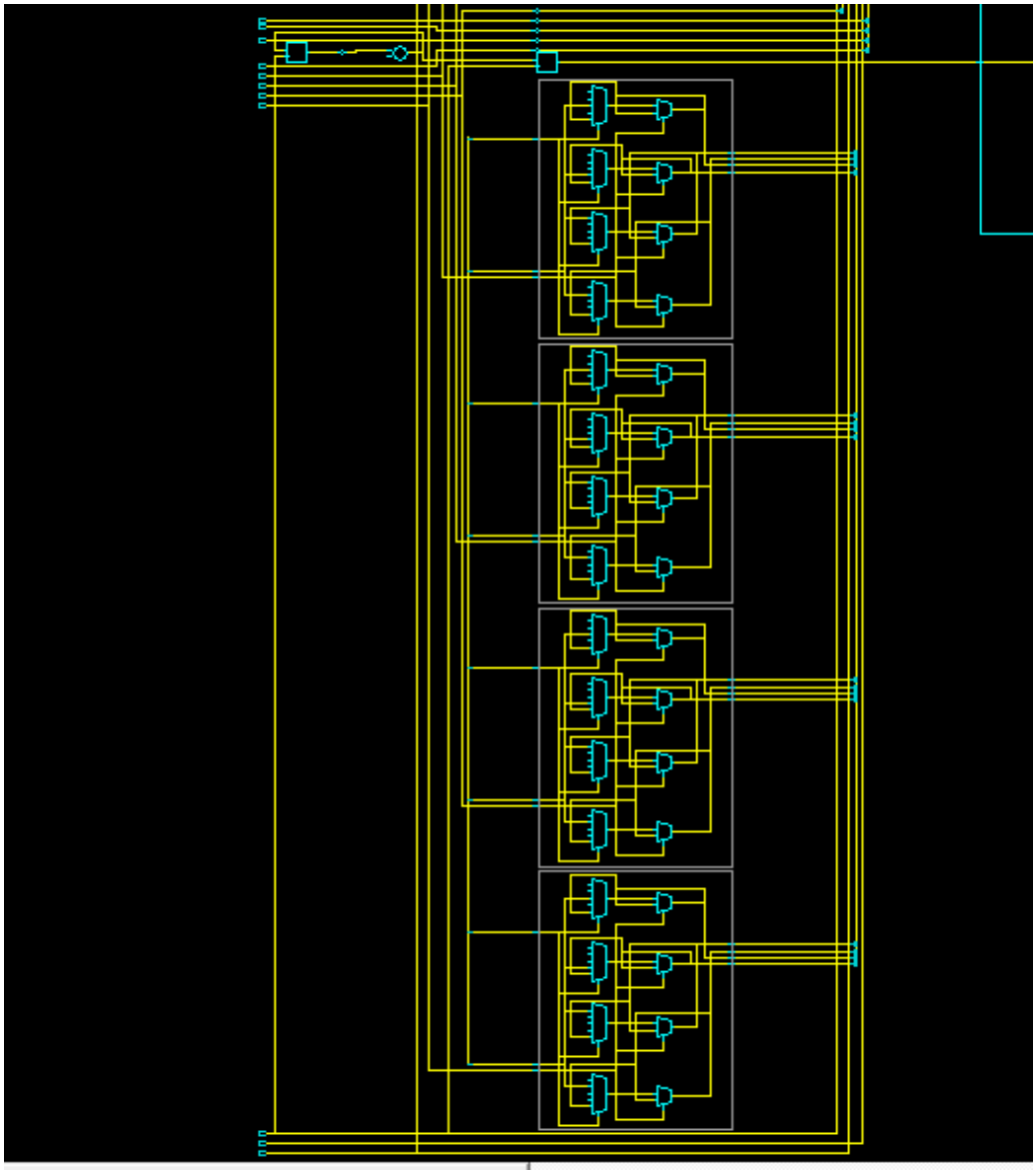


Modelsim Screenshots:

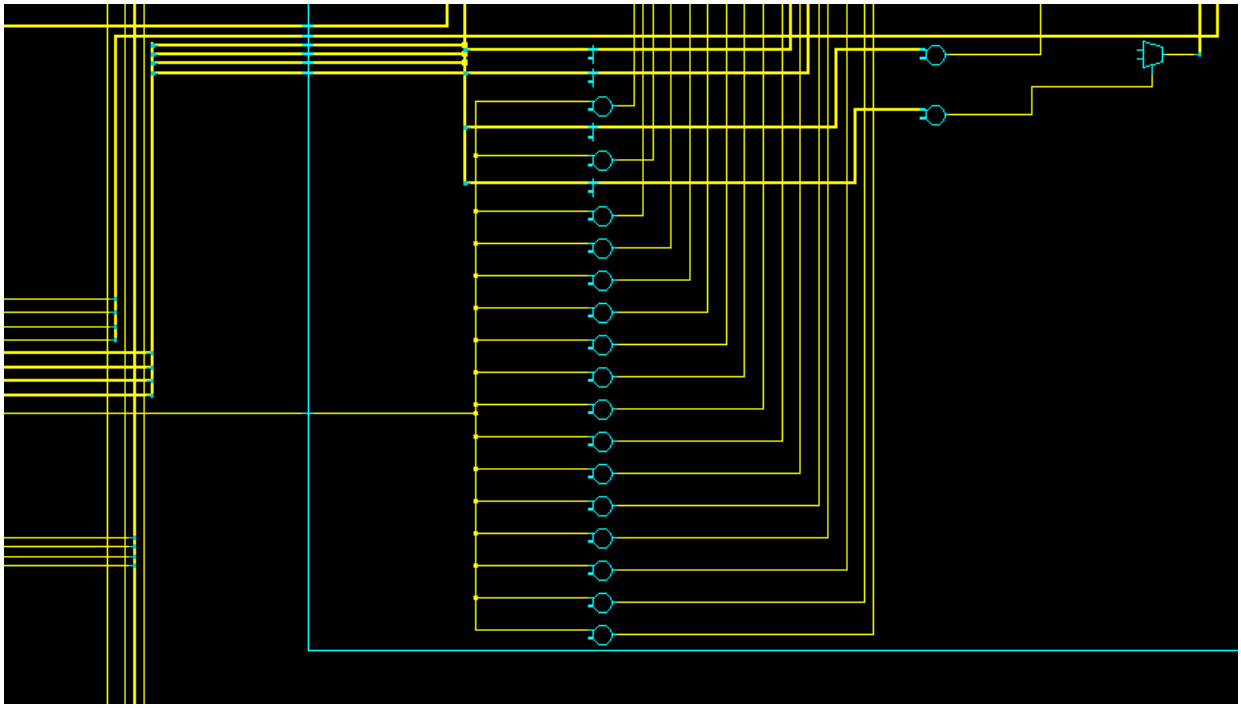



4
3
2
1

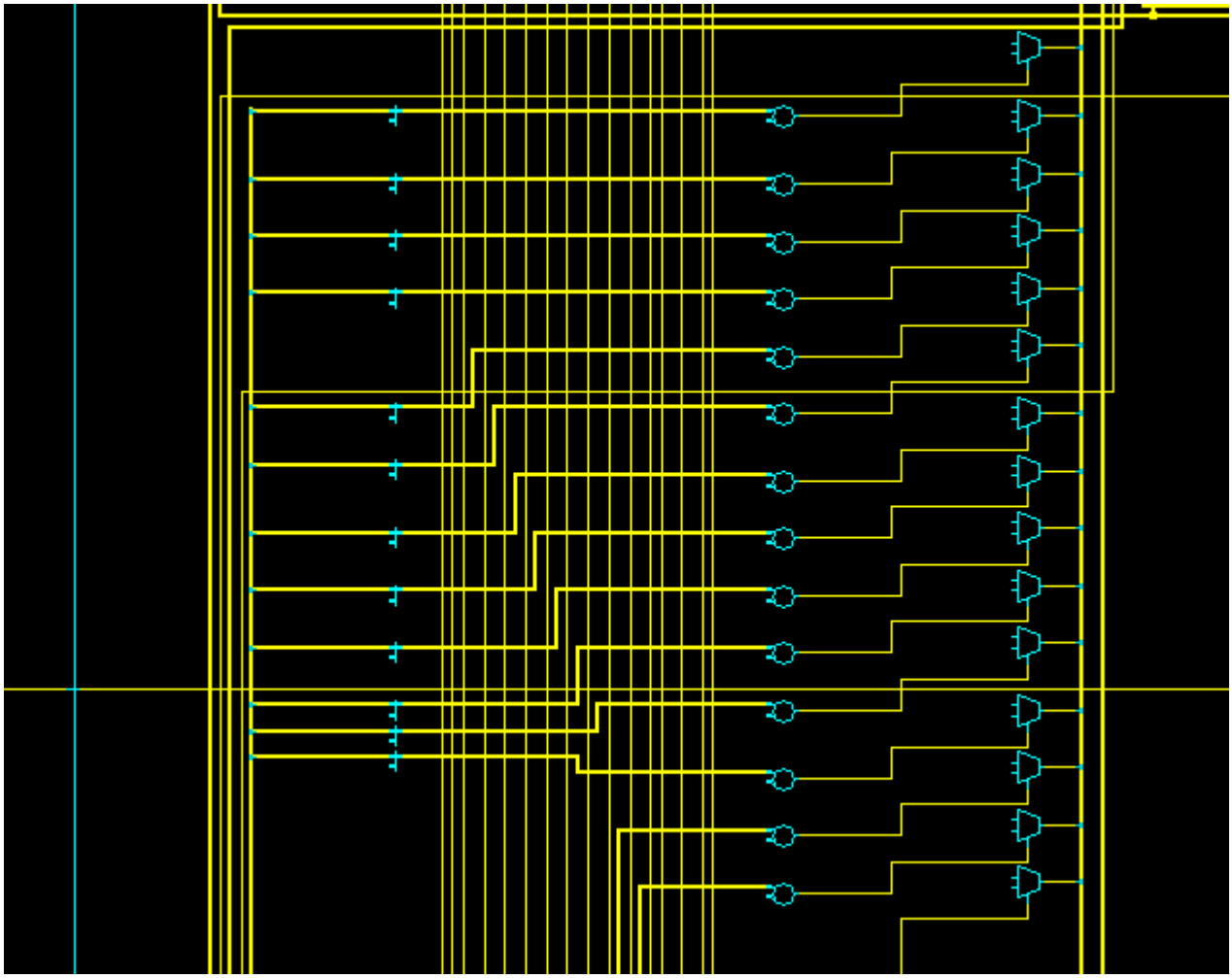
Block 1:



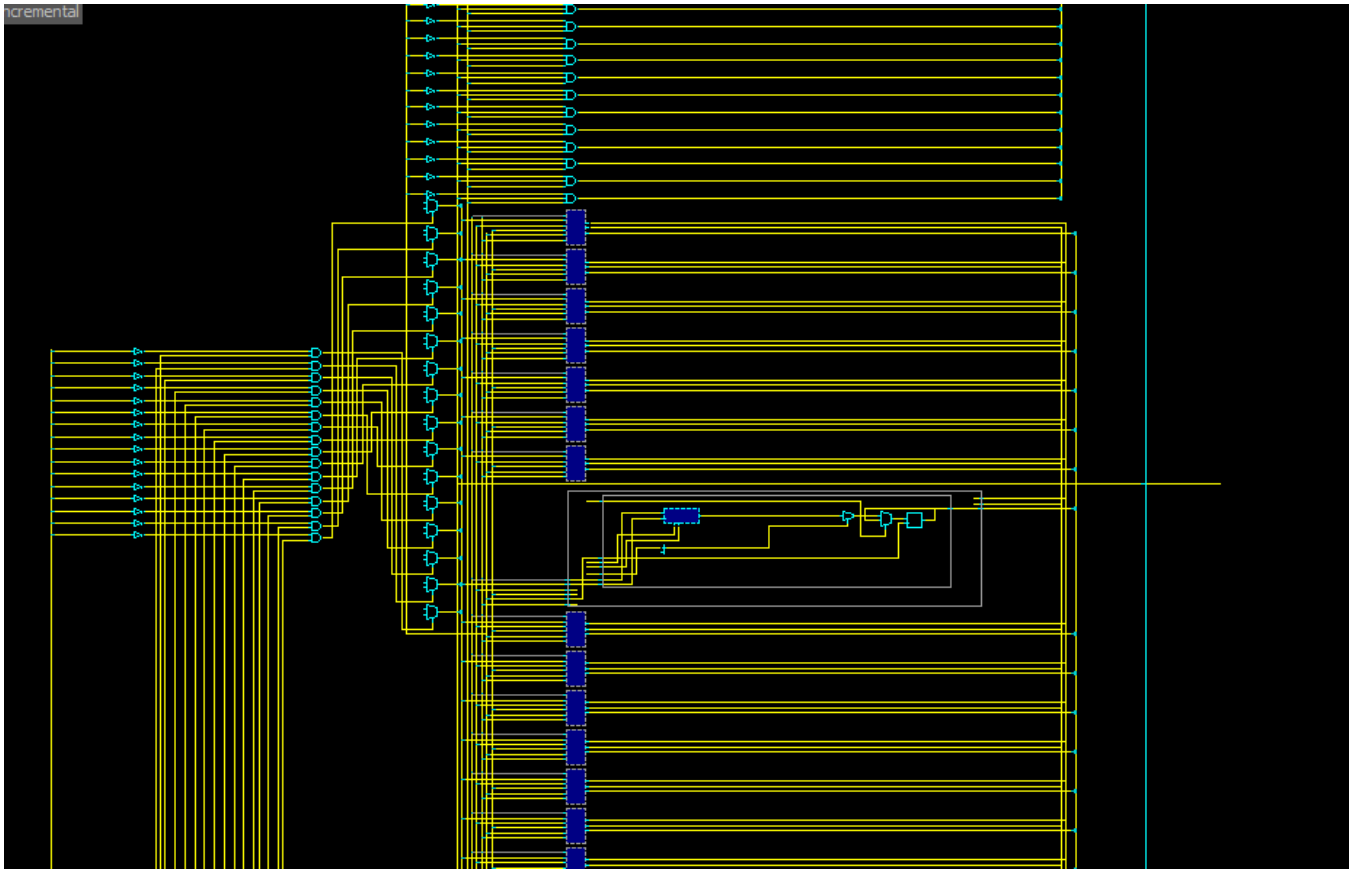
Block 2:



Block 3:



Block 4:



4.2 Modules and each Function:

Modules Content	
M_ROU_01: 8-bit Register	Achieving data and sent it as output for other, The main function of the input buffer is to store the packet once it arrived and propagates it.
M_ROU_02: 4-1 8-bit DeMux	The switch fabric which consist of demuxes which responsible of providing full connectivity between the inputs and outputs of the switch with reasonable hardware and delay, it's also a selector to propagate data to appropriate ports according to sel value.
M_ROU_03: Block Ram	Used to save data, as it contains array that hold the data and two pointer which help in read and write data which is controlled and organized by their clock

M_ROU_04: Gray Counter	Counts in Gray code.
M_ROU_05: Gray to Binary Converter	Used to convert gray code to Binary by changing 1 single bit, mostly used in application in which Hardware may produce error or ambiguity.
M_ROU_06: FIFO controller	Organize the validity of the read or write and check memory availability, synchronize router internal modules with the incoming packets, read packet header and configure the switch fabric. Control data flow inside the router and apply round-robin schedular at the output ports.
M_ROU_07: FIFO	The whole memory unit that contain the Controller and Block ram and organize their function to work on the First in First out sequence.
M_ROU_08: Round Robin Schedular	Setting data in organized queue and allow data passing only each clock cycle.
M_ROU_09: Router	Alternative for System-on-chip (SoC) due to the complexity, on other hand (NoC) is fast and reliable with low Latency, router transfer the input packets to suitable output ports.

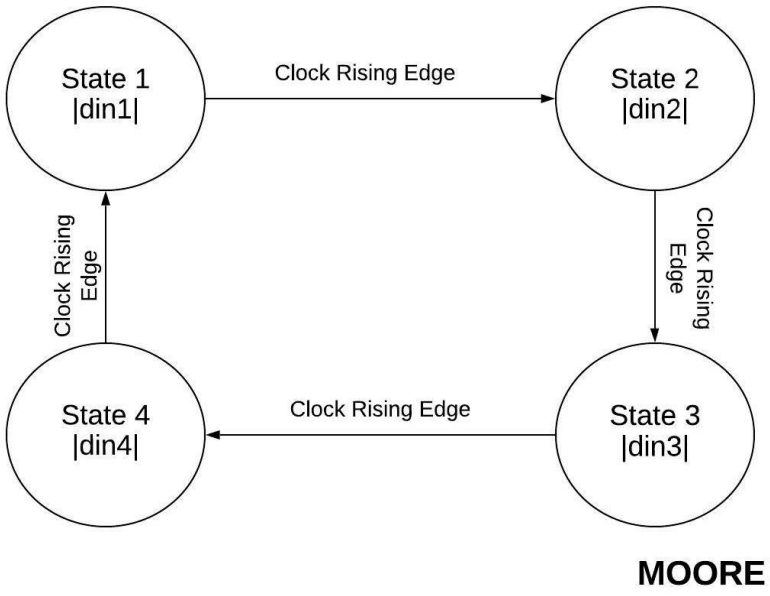
4.3 Implementation Discuss:

Modules Discuss	
M_ROU_01: 8-bit Register:	in this module, data is transferred from **data in** to **data out** if there rising edge and Reset = 0 and clock enable =1, in case the reset =1 the data out = 0.
M_ROU_02: 4-1 8-bit DeMux:	in this module, data is transferred from input to output according to “sel” value if En = ‘1’, else if enable = ‘0’, all output ports keep their current values. Input ports was made as declared in the module documentation. Case enable 1 and sel 00, transfer input to d_out1 and other outputs set to zero Case enable 1 and sel 01, transfer input to d_out2 and other outputs set to zero Case enable 1 and sel 10, transfer input to d_out3 and other outputs set to zero Case enable 1 and sel 11, transfer input to d_out4 and other outputs set to zero Case enable 1 and sel other, set to null

	Case en = 0 keep current output ports, inferred latch.
M_ROU_03: Block Ram:	block ram is the real memory element in the router as it identify array can store logic vector and two processes Process1 is to write as the write clock edge up happen and the write enable is 1 then write in the array at addra Process2 is to read as the read clock edge up happen and the read enable is 1 then read from the array at addrb
M_ROU_04: Gray Counter:	After reset the current state value is 0000. We then xor this current state with the 5 least significant bits of current state while concatenating the 0 at most significant bit of current state. & is concatenating operator. The result is placed in hold variable hold <= Currstate XOR ('0' & hold(N-1 DOWNTO 1));. In the next statement we increment the hold by 1. The result is placed in next_hold variable next_hold <= std_logic_vector(unsigned(hold) + 1); . In the third statement we perform the same concatenation and logical xor but this time on next_hold variable. The result is placed in Nextstate variable Nextstate <= next_hold XOR ('0' & next_hold(N-1 DOWNTO 1)); . The current state is assigned to the output and next state is calculated. This next state is transferred to current state on next rising edge of clock.
M_ROU_05: Gray to Binary Converter:	In first block: Declared input array where gray code is written gray_in and output array bin_out where the converted to binary code is outputted In second block: the logic of converting take place where saving the most

	significant bit from gray_in into bin_out then we Xor this gray[3] with the next gray[2] and saving in in bin[2] then we taking g[3]&[2] Xor with [1] and saving it in bin [1] and so on.
M_ROU_06: FIFO Controller:	this block is implemented depending on if conditions to put values to empty and full and requests validity of read and write depending on empty and full gotten values
M_ROU_07: FIFO:	this module maps the controller with the block ram in one element to control the flow (from or in it) in first in first out strategy.by mapping similar port in the two modules
M_ROU_08: Round Robin Schedular:	This module implemented by using three process style, first process is for checking the clock and the second process used to identify the next state by using switch case and the third process is for identifying the output by using switch case
M_ROU_09: Router:	this is the main module in our project we implemented it by identifying and instancing all used old modules and mapping them by direct mapping or “for generator” and add to process to put values to requests as it is approved or no

5 Schedular Design and FSM Implementation:



In our design we choose **Moore** FSM because in our design the output depends only on the present state and doesn't depend on inputs. As the output **dout** depend on what state it is in, it then takes the value of the specific **din**. In our design we also choose three process **Moore** coding style in the implementation of our architecture.

FSM can be implemented in three different coding style according to four tasks:

1. Asynchronous reset (RS)
2. Update current state register (CS)
3. Determine next state (NS)

4. Determine outputs (OP)

One Process		
	Mealy	**Moore**
Sensitive To	Clock Reset CSs Inputs	Clock Reset CSs

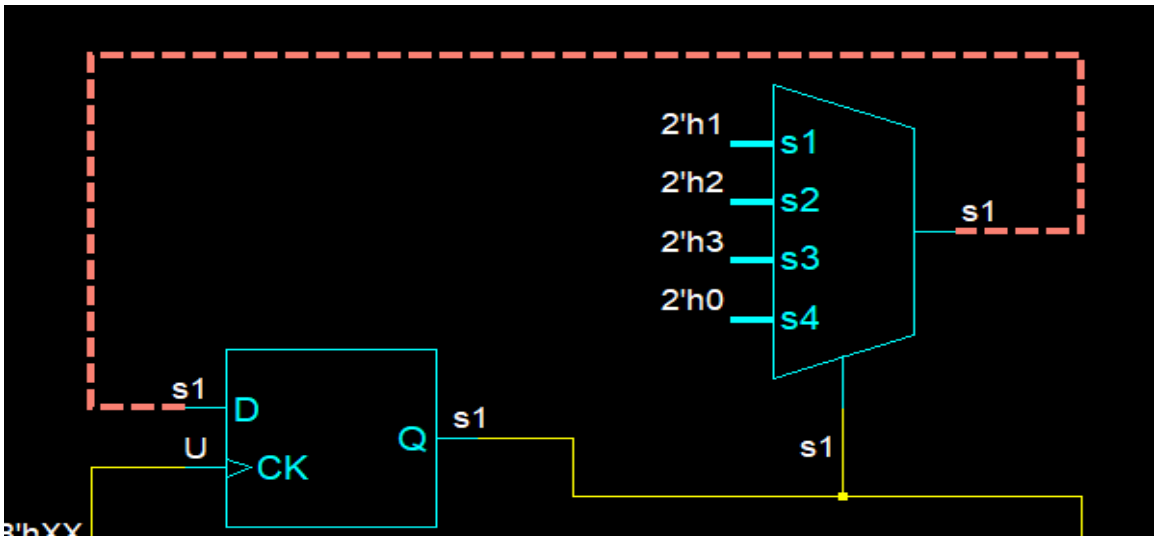
Two Process				
	Mealy		**Moore**	
	P1	P2	P1	P2
Sensitive To	Clock Reset	CSs Inputs	Clock Reset	CSs Inputs

Three Process						
	Mealy			**Moore**		
	P1	P2	P3	P1	P2	P3
Sensitive To	Clock Reset	CSs Inputs	CSs Inputs	Clock Reset	CSs Inputs	CSs

Timing Analysis:

The setup time of flip flop (**Tsetup**) plus the clock to output propagation delay (**Tpd**) plus the propagation delay of the combinational circuit (**Tpcq**) must be less than or equal the clock time (**Tc**).

$$Tsetup + Tpd + Tpcq \leq Tc$$



Critical path:

Combinational logic which is MUX, the critical path is between the **MUX** of next state and the **Flip Flop,** the minimum clock cycle (**Tc-min**) equal the propagation delay of the combinational circuit (**Tpcq**) plus the clock to output propagation delay (**Tpd**) plus the setup time of flip flop (**Tsetup**)

$$Tc\text{-}min = Tsetup + Tpd + Tpcq$$

Minimum period: 1.082ns, Maximum Frequency: 924.300MHz

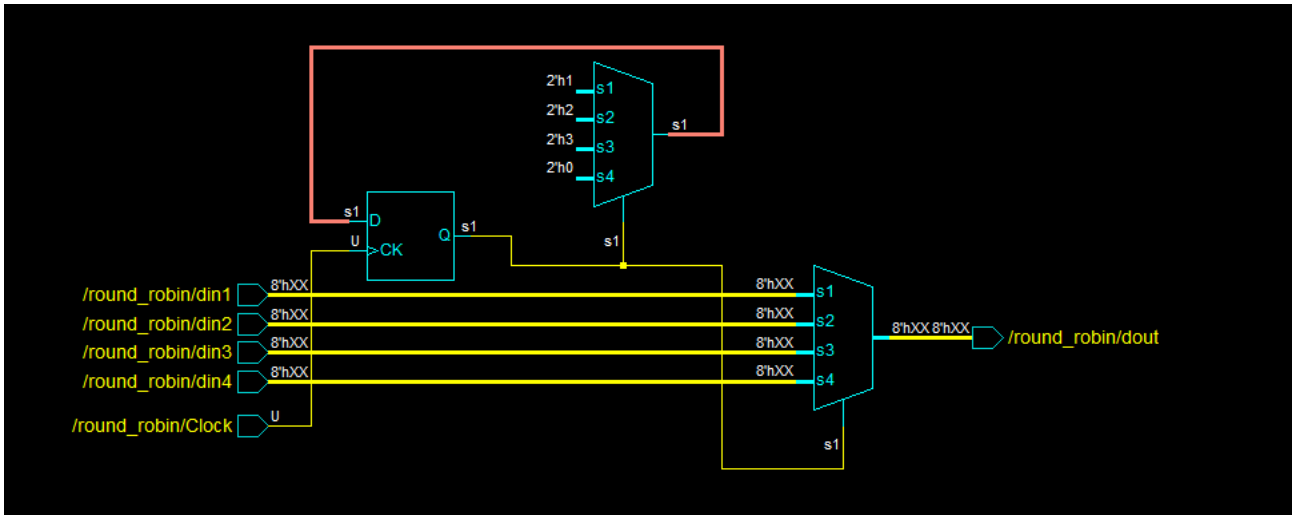
Generated from Timing Report from Xilinx tool

Also, Contamination delay (**Tcd**) plus Clock to output contamination delay (**Tccq**) must be bigger than or equal hold time (**Thold**)

$$Thold \leq Tcd + Tccq$$

$$Slack = Tc - Tsetup - Tpcq + Tpd$$

Synthesis Analysis:



As in our design there are two switch case statements, it synthesizes them into two **MUX**s. The upper **MUX** indicating the next state and the other **MUX** indicating the output(**dout**). Also, in our design a clock edge detection condition is used in IF statement and the first process sensitivity list include the clock and nothing else so **Flip Flop** is inferred.

Timing Report:

Timing report generated by Xilinx

Clock Information:

```
------------------
-----------------------------------+-----------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+-----------------------+-------+
Clock                              | BUFGP                 | 2     |
-----------------------------------+-----------------------+-------+
```

Asynchronous Control Signals Information:

```
----------------------------------------
No asynchronous control signals found in this design
```

Timing Summary:

```
---------------
Speed Grade: -3

   Minimum period: 1.082ns (Maximum Frequency: 924.300MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 1.472ns
   Maximum combinational path delay: 1.061ns
```

Timing Details:

```
---------------
All values displayed in nanoseconds (ns)

=========================================================
Timing constraint: Default period analysis for Clock 'Clock'
  Clock period: 1.082ns (frequency: 924.300MHz)
  Total number of paths / destination ports: 3 / 2
--------------------------------------------------------------------------------
Delay:               1.082ns (Levels of Logic = 1)
  Source:            present_state_FSM_FFd2 (FF)
  Destination:       present_state_FSM_FFd2 (FF)
  Source Clock:      Clock rising
  Destination Clock: Clock rising

  Data Path: present_state_FSM_FFd2 to present_state_FSM_FFd2
                            Gate     Net
```

```
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q             10   0.361   0.321  present_state_FSM_FFd2
(present_state_FSM_FFd2)
     INV:I->O             1   0.113   0.279  present_state_FSM_FFd2-In1_INV_0
(present_state_FSM_FFd2-In)
     FD:D                      0.008          present_state_FSM_FFd2
    ----------------------------------------
    Total                      1.082ns (0.482ns logic, 0.600ns route)
                                   (44.6% logic, 55.4% route)

=========================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'Clock'
  Total number of paths / destination ports: 16 / 8
-------------------------------------------------------------------------
Offset:              1.472ns (Levels of Logic = 2)
  Source:            present_state_FSM_FFd2 (FF)
  Destination:       dout<7> (PAD)
  Source Clock:      Clock rising

  Data Path: present_state_FSM_FFd2 to dout<7>
                             Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q             10   0.361   0.735  present_state_FSM_FFd2
(present_state_FSM_FFd2)
     LUT6:I0->O           1   0.097   0.279  Mmux_dout11 (dout_0_OBUF)
     OBUF:I->O                 0.000          dout_0_OBUF (dout<0>)
    ----------------------------------------
    Total                      1.472ns (0.458ns logic, 1.014ns route)
                                   (31.1% logic, 68.9% route)

=========================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 32 / 8
-------------------------------------------------------------------------
Delay:               1.061ns (Levels of Logic = 3)
  Source:            din4<7> (PAD)
  Destination:       dout<7> (PAD)

  Data Path: din4<7> to dout<7>
                             Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
```

```
    IBUF:I->O             1   0.001   0.683  din4_7_IBUF (din4_7_IBUF)
    LUT6:I1->O            1   0.097   0.279  Mmux_dout81 (dout_7_OBUF)
    OBUF:I->O                 0.000          dout_7_OBUF (dout<7>)
    ----------------------------------------
    Total                     1.061ns (0.098ns logic, 0.963ns route)
                                  (9.2% logic, 90.8% route)

=========================================================

Cross Clock Domains Report:
--------------------------

Clock to Setup on destination clock Clock
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
Clock          |    1.082|         |         |         |
---------------+---------+---------+---------+---------+

=========================================================


Total REAL time to Xst completion: 39.00 secs
Total CPU time to Xst completion: 39.21 secs

Total memory usage is 270268 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    4 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

6. Test and Simulation Results:

A testbench was written for the router module, and declared in appendix B and simulation waveform, output is declared in appendix C.

Router test Strategy

Testcases	Input	Output
Test case 1 Sel --> 00 --> port 1 Note: Any input can be assigned first but all in first out port	datai1 <= "11001000" datai2 <= "10010000" datai3 <= "01010100" datai4 <= "10001000" Wclk =clock edge high wr1,wr2,w3w4<=”1”	Rdclk=rising edge high datao1 <= "11001000" After clock cycle datao1<= "10010000" After clock cycle datao1 <= "01010100" After clock cycle datao1 <= "10001000"
Test case 2 Sel --> 01 --> port 2 Note: Any input can be assigned first but all in second output port	datai1 <= "10100001" datai2 <= "01110001" datai3 <= "10100101" datai4 <= "11100001" Wclk =clock edge high wr1,wr2,w3,w4 <=”1”	Rdclk=clockedgehigh Datao2<="01110001" After clock cycle Datao2<="10100101" After clock cycle Datao2<="11100001" After clock cycle Datao2<= "10100001"
Test case 3 Sel --> 10 --> port 3 Note: Any input can be assigned first but all in third output port	datai1 <= "10111010" datai2 <= "10110010" datai3 <= "10110110" datai4 <= "10100010" Wclk =clock edge high wr1,wr2,w3,w4<=”1”	Rdclk=clockedgehigh Datao3<="10110110" After clock cycle Datao3<= "10100010" After clock cycle Datao3<= "10111010" After clock cycle Datao3<= "10110010"
Test case 4 Sel --> 11 --> port 4 Note: Any input can be assigned first but all in fourth output port	datai1 <= "10101111" datai2 <= "01100111" datai3 <= "01110011" datai4 <= "01100111" Wclk =clock edge high wr1,wr2,w3w4<=”1”	Rdclk=clockedgehigh Datao4<="01100111" After clock cycle Datao4<= "10101111" After clock cycle Datao4<= "01100111"

		After clock cycle Datao4<=”01110011"

- Four testcases were chosen
- Assert statement were used
- Report severity error

7 Conclusion

NoC router(Networks-on-Chip router), is used for networking and sends information between two or more packet-switched computer networks. The synthesis and the simulation of the router is done using **Modelsim** , and for generating time report we used **xilinix** tool.

We have many challenges in implementation of router as router consists of many components and of course it depends on all other modules in this project, Also the connection of components with each other in the module router was not easy.

in our design we have 9 modules each module is responsible for function. Register is responsible for transferring data implemented by using **if condition.** demultiplexer is responsible for the selection of data implemented by using **Switch case**. block ram is the real memory element in the router, Gray Counter is responsible for counting every clock cycle in grey code, Gray to Binary Converter used **Xor** gate to convert gray code to Binary. Fifo controller is implemented depending on **if conditions** to put values to empty and full and requests validity of read and write. Fifo maps the controller with the block ram in one element to control the flow by mapping similar port in the two modules. Round robin implemented by three process and **two Switch cases**, one Switch case to determine the next state and the other Switch case to determine the output. Finally, Router implemented by identifying and instancing all used old modules and mapping them by direct mapping.

In our design we choose **Three Process Moore** implementation style because the output depends only on the present state and doesn't depend on inputs. And three process to be more Concise. We used manual test for Testing the Round robin by Force the value of **din’s.**

For generating the Time report we used **Xilinx** tool and we found that Minimum period equal **1.082ns** and Maximum Frequency equal **924.300MHz**.

In future we look forward to improve the design by making more generic code and improve the scheduling algorithm for better performance and change the router to be bigger than 4 bits.

8 Task Distribution List

#	Student ID	Tasks
1	17P6006	the project workload is distributed equally among all team members
2	17P6024	the project workload is distributed equally among all team members
3	17P3067	the project workload is distributed equally among all team members
4	17P6019	the project workload is distributed equally among all team members
5	17P6018	the project workload is distributed equally among all team members

Appendix A

Module M-ROU-01

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

entity eight_reg is
    Port (
        Data_in  : in  STD_LOGIC_VECTOR (7 DOWNTO 0);
           Clock : in  STD_LOGIC;
           Clock_En  : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Data_out    : out  STD_LOGIC_VECTOR (7 DOWNTO 0));

end eight_reg;
architecture Behavioral of eight_reg is
begin
     PROCESS(Clock,Reset)
       BEGIN
       IF(Reset='1') THEN
        Data_out <=(others =>'0');
       ELSIF(Clock'event AND Clock='1')THEN
          IF(Clock_En ='1') THEN
             Data_out <= Data_in;
          END IF;
       END IF;
     END PROCESS;
end Behavioral;
```

Module M-ROU-02

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity demux is
     port(
          d_in : in std_logic_vector(7 downto 0);
          Sel : in std_logic_vector(1 downto 0);
          En: in std_logic;
          ------------------------------------------------
          d_out1 : out std_logic_vector(7 downto 0);
          d_out2 : out std_logic_vector(7 downto 0);
          d_out3 : out std_logic_vector(7 downto 0);
          d_out4 : out std_logic_vector(7 downto 0)
          );
end demux;

----------------------------------------

architecture behav of demux is
begin

    p1: process (d_in,Sel,En) is
    begin
        if En ='1' then
          case Sel is
            when "00" =>
            d_out1 <= d_in;
            d_out2 <= "00000000";
             d_out3 <= "00000000";
             d_out4 <= "00000000";

            when "01" =>
            d_out2 <= d_in;
            d_out3 <= "00000000";
             d_out1 <= "00000000";
             d_out4 <= "00000000";


            when "10" =>
           d_out3 <= d_in;
           d_out1 <= "00000000";
             d_out2 <= "00000000";
             d_out4 <= "00000000";


            when "11" =>
            d_out4 <= d_in;
            d_out1 <= "00000000";
            d_out2 <= "00000000";
            d_out3 <= "00000000";

        when others => null;


          end case;
        end if;
    end process p1;

end behav;
```

Module M-ROU-03

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity  blockram is
port (
    d_in : in std_logic_vector(7 downto 0); --input
    d_out : out std_logic_vector(7 downto 0); --output
    wea : in std_logic ; --enable write
    rea : in std_logic ; --enable read
    addra : in std_logic_vector(2 downto 0); --plsce to write
    addrb : in std_logic_vector(2 downto 0); --place to read
    clka : in std_logic ; --write clock
    clkb : in std_logic  --read clock
        );
end blockram;
Architecture ram of blockram is
    type ram_type is ARRAY(0 to 7) of std_logic_vector(7 downto 0);
    signal my_memory :ram_type;
begin
process (clka) is
begin
    if(rising_edge(clka)) then
        if(wea='1') then
            my_memory(conv_integer(addra)) <= d_in;
        end if;
    end if;
end process;
process (clkb) is
begin
    if(rising_edge(clkb)) then

        if(rea='1') then
            d_out <= my_memory(conv_integer(addrb));
        --else d_out<="ZZZZZZZZ";
        end if;
    end if;
end process;

end ram ;
```

Module M-ROU-04

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;


ENTITY counter IS

  PORT (Clock, Reset, En: IN std_logic;
        Count_output: OUT std_logic_vector (3 DOWNTO 0));
END counter;

ARCHITECTURE GreyCounter_beh OF counter IS
  SIGNAL CurResetate,Nextstate, hold, next_hold: std_logic_vector (3 DOWNTO 0);
BEGIN

  StateReg: PROCESS (Clock)
  BEGIN
    IF (Clock = '1' AND Clock'EVENT) THEN
      IF (Reset = '1') THEN
        CurResetate <= (OTHERS =>'0');
      ELSIF (En = '1') THEN
        CurResetate <= Nextstate;
      END IF;
    END IF;
  END PROCESS;

  hold <= CurResetate XOR ('0' & hold(3 DOWNTO 1));
  next_hold <= std_logic_vector(unsigned(hold) + 1);
  Nextstate <= next_hold XOR ('0' & next_hold(3 DOWNTO 1));
  Count_output <= CurResetate;

END GreyCounter_beh;
```

Module M-ROU-05

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;


entity Converter is
port( gray_in: IN std_logic_vector(3 downto 0);
bin_out: out std_logic_vector (3 downto 0));
end Converter;


architecture Behavioral of Converter is
begin
bin_out(3)<= gray_in(3);
bin_out(2)<= gray_in(3) xor gray_in(2);
bin_out(1)<= gray_in(3) xor gray_in(2) xor gray_in(1);
bin_out(0)<= gray_in(3) xor gray_in(2) xor gray_in(1) xor gray_in(0);
end behavioral;
```

Module M-ROU-06

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fifoController is
    port(
    reset,rdclk,wrclk,r_req,w_req : in  std_logic;
    write_valid, read_valid, empty, full : out std_logic;
    wr_ptr, rd_ptr : out std_logic_vector(2 downto 0)
);
end entity ;


architecture behav of fifoController is
component counter is

  PORT (Clock, Reset, En: IN std_logic;
        Count_output: OUT std_logic_vector (3 DOWNTO 0));

end component;

component Converter is
port( gray_in: IN std_logic_vector(3 downto 0);
bin_out: out std_logic_vector (3 downto 0));
end component;
```

```
SIGNAL readConverter: STD_LOGIC_VECTOR(3 DOWNTO 0);--gray to binary read
SIGNAL writeConverter: STD_LOGIC_VECTOR(3 DOWNTO 0);--gray to binary write
SIGNAL rdBinaryout : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL wrBinaryout : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL readvalid, writevalid : STD_LOGIC;


begin
    rd_counter : counter
    PORT MAP(
        Clock => rdclk,
        En => readvalid,
        Reset => reset,
        Count_output => readConverter
    );
-------------------------------------------

wr_counter : counter
        PORT MAP(
        Clock => wrclk,
        En => writevalid,
        Reset => reset,
        Count_output => writeConverter
    );

-------------------------------------------

    read_Converter : Converter
        PORT MAP(
        gray_in => readConverter,
        bin_out => rdBinaryout
    );


        write_Coverter : Converter
        PORT MAP(
        gray_in => writeConverter,
        bin_out => wrBinaryout
    );
-------------------------------------------
 p1: process(r_req,w_req,rdBinaryout,wrBinaryout,reset)
  variable tmp,tmp0,tmp1 : std_logic;  -- for full and empty checking

   begin
        tmp0 := wrBinaryout(2) XNOR rdBinaryout(3);
        tmp1 := wrBinaryout(3) XOR rdBinaryout(2);
        tmp  := tmp0 AND tmp1; -- check for full


        if(reset = '1') THEN
            full <= '0';
```

```
            empty <= '1';
            readvalid <= '0';
            writevalid <= '1';

        else
            if(tmp='1') then --full
        full <='1';
        writevalid <='0'; -- no access to write
        else
        full <='0';
            if(w_req='1') then -- access to write
           writevalid <='1';

        else
          writevalid <='0';
                end if;

            end if;

            if(wrBinaryout = rdBinaryout) then -- empty --modified
                empty <='1';
        readvalid <= '0';
            else
               empty <= '0';
               if(r_req = '1') then
         readvalid <= '1';
               else
                readvalid <= '0';
               end if;
            end if;
        end if;
end process p1;
-- update read and write
 read_valid <= readvalid;
 write_valid <= writevalid;
------------------------------
-- update ptrs
rd_ptr <= rdBinaryout(2 downto 0);
wr_ptr <= wrBinaryout(2 downto 0);


end  behav;
```

Module M-ROU-07

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
entity fifo is
port (
    reset,rclk,wclk,rreq,wreq : in std_logic;
    empty,full : out std_logic;
    datain :in std_logic_vector (7 downto 0);
    dataout : out std_logic_vector(7 downto 0)
        );
end entity;
```

```
architecture fifobehav of fifo is
    component fifoController is
        port(
        reset,rdclk,wrclk,r_req,w_req : in  std_logic;
        write_valid, read_valid, empty, full : out std_logic;
        wr_ptr, rd_ptr : out std_logic_vector(2 downto 0)
        );
    end component;

    component blockram is
        port (
        d_in : in std_logic_vector(7 downto 0); --input
        d_out : out std_logic_vector(7 downto 0); --output
        wea : in std_logic ; --enable write
        rea : in std_logic ; --enable read
        addra : in std_logic_vector(2 downto 0); --plsce to write
        addrb : in std_logic_vector(2 downto 0); --place to read
        clka : in std_logic ; --write clock
        clkb : in std_logic  --read clock
        );
    end component;


    signal rdvptr,wvptr :std_logic;
    signal addra_ptr,addrb_ptr :std_logic_vector(2 downto 0);
    signal read_intp,write_intp :integer;



    begin


      controller : FifoController
    port map(
        reset => reset,
        rdclk => rclk,
        wrclk => wclk,
        r_req => wreq,
        w_req => wreq,
        write_valid => wvptr,
        read_valid => rdvptr,
        wr_ptr => addra_ptr,
        rd_ptr =>addrb_ptr,
        full => full,
        empty => empty
            );


    memory : blockram
    port map(
        wea =>wvptr,
        rea =>rdvptr,
        addra =>addra_ptr,
        addrb =>addrb_ptr,
        clka =>rclk,
        clkb =>wclk,
        d_in =>datain,
        d_out =>dataout);

end architecture ;
```

Module M-ROU-08

```
library ieee;
use ieee.std_logic_1164.all;

entity roundrobin is

  port (
    din1 : in  std_logic_vector(7 downto 0);
    din2 : in  std_logic_vector(7 downto 0);
    din3 : in  std_logic_vector(7 downto 0);
    din4 : in  std_logic_vector(7 downto 0);
    dout : out  std_logic_vector(7 downto 0);
    Clock :  in  std_logic);

end roundrobin;


architecture arch of roundrobin is

type state is (s1,s2,s3,s4);
signal present_state,next_state : state;
begin

  cs: process (Clock)

  begin

  if(rising_edge(Clock))THEN
      present_state <=next_state;
end if;
end process cs ;
 ns: process (present_state)
begin
case present_state is

when s1 =>
 next_state <=s2;
when s2 =>
 next_state <=s3;
when s3 =>
 next_state <=s4;
when s4 =>
 next_state <=s1;

end case;
end process ns ;

op: process (present_state)
begin
case present_state is

when s1 =>
dout <= din1;

when s2 =>
dout <= din2;

when s3 =>
dout <= din3;

when s4=>
```

```
dout <= din4;

end case;
end process op ;
end architecture arch;
```

Module M-ROU-09

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.ALL;

entity router is
    port(
        datai1,datai2,datai3,datai4 : in std_logic_vector(7 downto 0);
        datao1,datao2,datao3,datao4 :out std_logic_vector(7 downto 0);
        wr1,wr2,wr3,wr4,wclock,rclock,rst : in std_logic);
end entity;
architecture router_behav of router is

component eight_reg is

    Port (
        Data_in  : in  STD_LOGIC_VECTOR (7 DOWNTO 0);
           Clock : in  STD_LOGIC;
           Clock_En  : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Data_out    : out  STD_LOGIC_VECTOR (7 DOWNTO 0));
end component;

component demux is
    port(
         d_in : in std_logic_vector(7 downto 0);
         Sel : in std_logic_vector(1 downto 0);
         En: in std_logic;
         ------------------------------------------------
         d_out1 : out std_logic_vector(7 downto 0);
         d_out2 : out std_logic_vector(7 downto 0);
         d_out3 : out std_logic_vector(7 downto 0);
         d_out4 : out std_logic_vector(7 downto 0)
         );
end component;

component roundrobin is

  port (
    din1 : in  std_logic_vector(7 downto 0);
    din2 : in  std_logic_vector(7 downto 0);
    din3 : in  std_logic_vector(7 downto 0);
    din4 : in  std_logic_vector(7 downto 0);
    dout : out  std_logic_vector(7 downto 0);
```

```
    Clock :  in  std_logic);
end component;

component fifo is
port (
    reset,rclk,wclk,rreq,wreq : in std_logic;
    empty,full : out std_logic;
    datain :in std_logic_vector (7 downto 0);
    dataout : out std_logic_vector(7 downto 0)
        );
end component;

component counter is

PORT(Clock, Reset, En: IN std_logic;
        Count_output: OUT std_logic_vector (3 DOWNTO 0));
end component;
---signals
type arrayofvectors is array(0 to 3) of std_logic_vector (7 downto 0);  --2d array
vector
type vectors2darray is array (0 to 3,0 to 3) of std_logic_vector(7 downto 0);  --2d
array vector
type logic2darray is array(0 to 3,0 to 3) of std_logic; --2d array logic
signal RegisterOut : arrayofvectors;
signal wr_arr : std_logic_vector (3 downto 0);   --array logic
signal demuxcarr : vectors2darray;
signal wreq : logic2darray;
signal rdreq : logic2darray;
signal datain : arrayofvectors;
signal fifoWriteReq : logic2darray;
signal dataout : vectors2darray;
signal empty : logic2darray;
signal full : logic2darray;


begin
--registermapping
--using direct mapping depending on port places

InputBuffer1 : eight_reg
port map(datai1,wclock,wr1,rst,RegisterOut(0));
InputBuffer2 : eight_reg
port map(datai2,wclock,wr2,rst,RegisterOut(1));
InputBuffer3 : eight_reg
port map(datai3,wclock,wr3,rst,RegisterOut(2));
InputBuffer4 : eight_reg
port map(datai4,wclock,wr4,rst,RegisterOut(3));

--Demuxes mapping
--using direct mapping depending on port places
demux1 :demux
port map(RegisterOut(0),(RegisterOut(0)(1 downto
0)),wr1,demuxcarr(0,0),demuxcarr(0,1),
```

```
demuxcarr(0,2),demuxcarr(0,3));
demux2 :demux
port map(RegisterOut(1),(RegisterOut(1)(1 downto
0)),wr2,demuxcarr(1,0),demuxcarr(1,1),
demuxcarr(1,2),demuxcarr(1,3));
demux3 :demux
port map(RegisterOut(2),(RegisterOut(2)(1 downto
0)),wr3,demuxcarr(2,0),demuxcarr(2,1),
demuxcarr(2,2),demuxcarr(2,3));
demux4 :demux
port map(RegisterOut(3),(RegisterOut(3)(1 downto
0)),wr4,demuxcarr(3,0),demuxcarr(3,1),
demuxcarr(3,2),demuxcarr(3,3));



write_process :process(rclock,wr1,wr2,wr3,wr4)
begin
if (rising_edge(wclock))then
fifoWriteReq <=wreq;
end if;
end process;

datain <=(datai1,datai2,datai3,datai4);
wr_arr <=(wr1,wr2,wr3,wr4);

scheduler1 : roundrobin
Port map( dataout(0,0),dataout(0,1),dataout(0,2),dataout(0,3),datao1,rclock);
scheduler2 : roundrobin
Port map( dataout(1,0),dataout(1,1),dataout(1,2),dataout(1,3),datao2,rclock);
scheduler3 : roundrobin
Port map( dataout(2,0),dataout(2,1),dataout(2,2),dataout(2,3),datao3,rclock);
scheduler4 : roundrobin
Port map( dataout(3,0),dataout(3,1),dataout(3,2),dataout(3,3),datao4,rclock);



process (empty,full,wr_arr,datain,wclock,rclock,rst)
begin
for i in 0 to 3 loop
for j in 0 to 3 loop
IF empty(i,j)= '0' THEN
        rdreq(i,j) <= '1';
    ELSE
        rdreq(i,j) <= '0';
    END IF;
    IF full(i,j)='0' AND wr_arr(j) = '1' AND (conv_integer(unsigned(datain(j)(1
DOWNTO 0))) = i) THEN
        wreq(i,j) <= '1';
        ELSE
        wreq(i,j) <= '0';
    END IF;
```

```
end loop;
end loop;
end process;

       outer_GENERATE_FOR : FOR i IN 0 TO 3 GENERATE

          inner_GENERATE_FOR : FOR j IN 0 TO 3 GENERATE
           routerfifo : fifo port
map(rst,rclock,wclock,rdreq(i,j),fifoWriteReq(i,j),empty(i,j),
            full(i,j),demuxcarr(j,i),dataout(i,j));

      end generate;
    end generate;
end architecture;
```

Appendix B

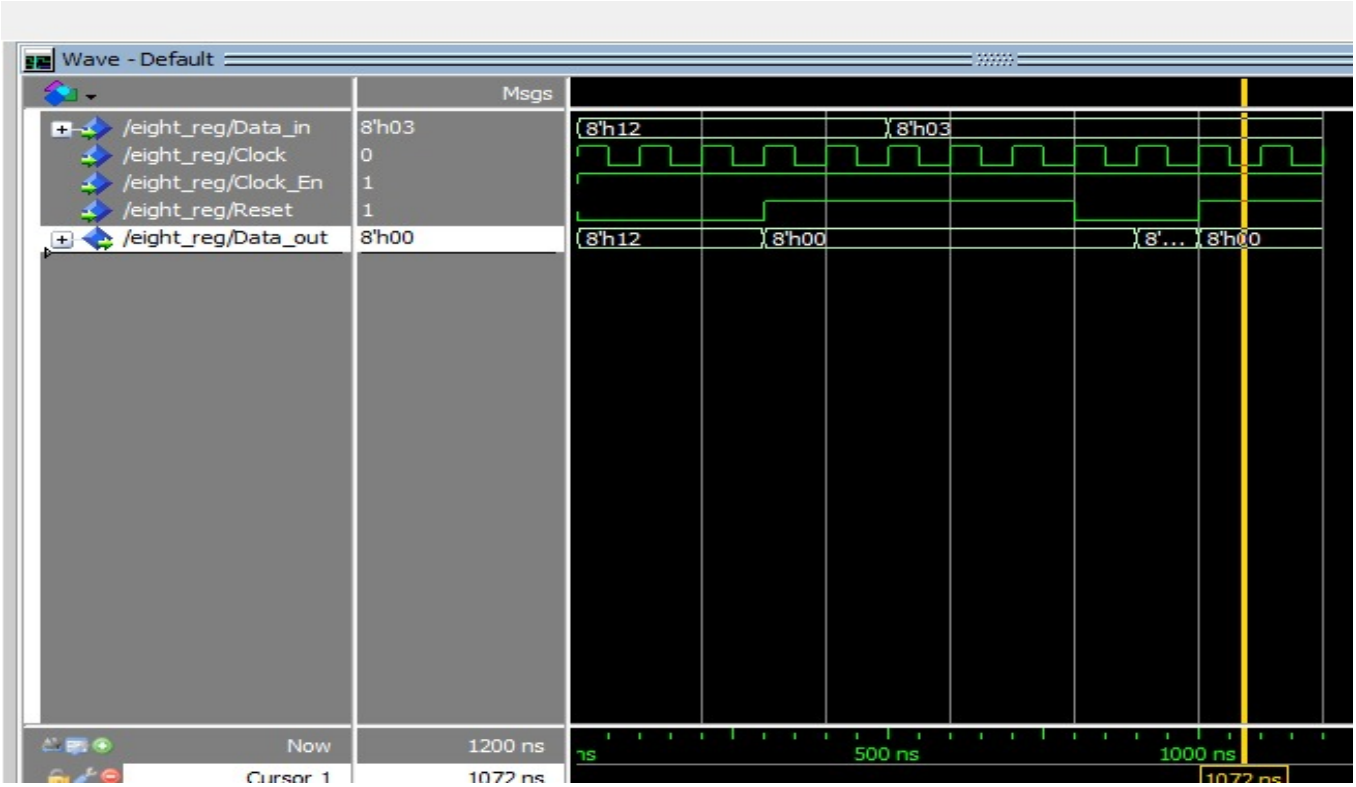
```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.ALL;

entity tb is
end entity;

architecture behav of tb is
SIGNAL datai1 : std_logic_vector(7 DOWNTO 0);
SIGNAL datai2 : std_logic_vector(7 DOWNTO 0);
SIGNAL datai3 : std_logic_vector(7 DOWNTO 0);
SIGNAL datai4 : std_logic_vector(7 DOWNTO 0);
------------

SIGNAL datao1 : std_logic_vector(7 DOWNTO 0);
SIGNAL datao2 : std_logic_vector(7 DOWNTO 0);
SIGNAL datao3 : std_logic_vector(7 DOWNTO 0);
SIGNAL datao4 : std_logic_vector(7 DOWNTO 0);
--------------
SIGNAL wr1 : std_logic;
SIGNAL wr2 : std_logic;
SIGNAL wr3 : std_logic;
SIGNAL wr4 : std_logic;
-----------
SIGNAL wclock : std_logic;
SIGNAL rclock : std_logic;
SIGNAL rst : std_logic;



component router

port(
    datai1,datai2,datai3,datai4 : in std_logic_vector(7 downto 0);
    datao1,datao2,datao3,datao4 :out std_logic_vector(7 downto 0);
    wr1,wr2,wr3,wr4,wclock,rclock,rst : in std_logic);

end component;




begin


--Porting--
 dut: Router port map (datai1,datai2,datai3,datai4,datao1,datao2,datao3,datao4,
```

```
   wr1,wr2,wr3,wr4,wclock,rclock,rst);
------



p1: process is
begin

wclock <= '0', '1' after 50 ns;
wait for 100 ns;

end process;

--------

p2: process is
begin

rclock <= '0', '1' after 50 ns;
wait for 100 ns;
end process;

------------

p3: process is
begin

rst <= '1';
wait for 100 ns;


----Select --> 00 ----
datai1 <= "11001000";datai2 <= "10010000";datai3 <= "01010100";datai4 <=
"10001000";
rst <= '0';wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
wait for 100 ns;
-- assert datao1=datai1 report " diferrent data 1" severity error;
------------------



----Select --> 01 ----
datai1 <= "10100001";datai2 <= "01110001";datai3 <= "10100101";datai4 <=
"11100001";
wait for 100 ns;

-- assert datao2=datai2 report " diferrent data 2" severity error;


----Select --> 10 ----
datai1 <= "10111010";datai2 <= "10110010";datai3 <= "10110110";datai4 <=
"10100010";
wait for 100 ns;
```

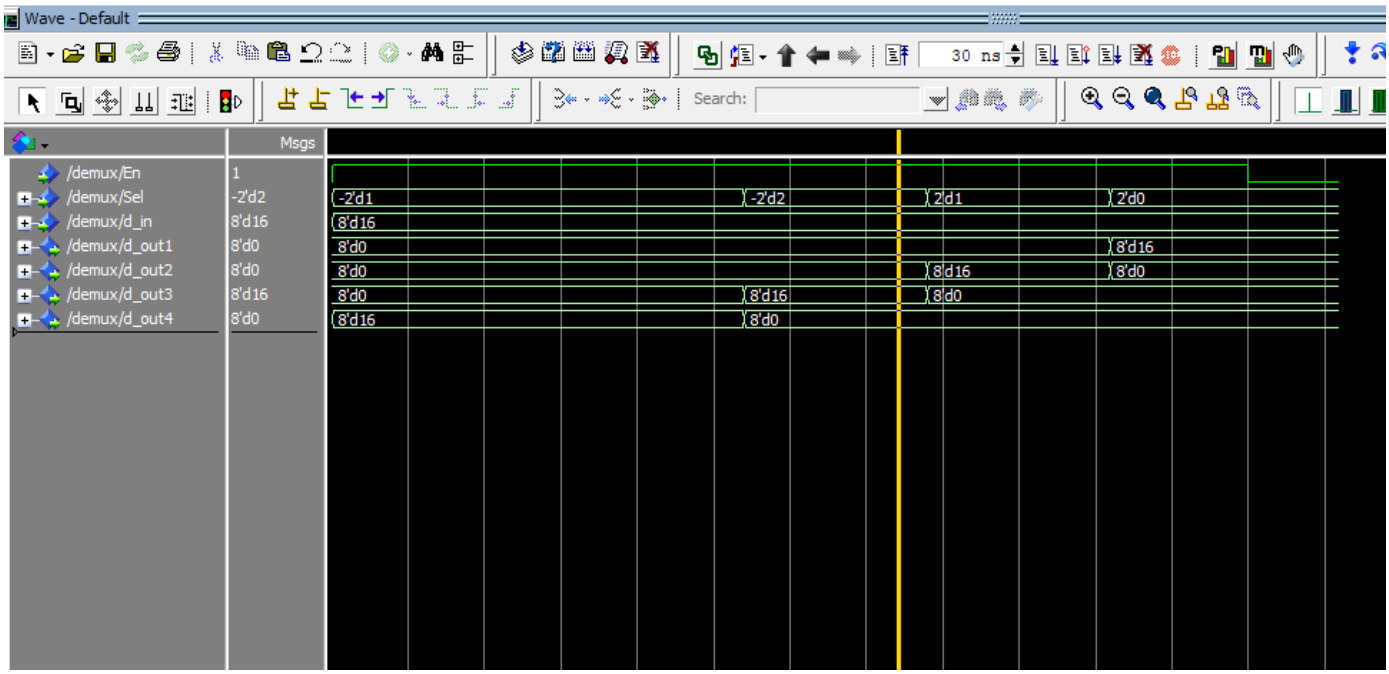
```
----Select --> 11 ----
datai1 <= "10101111";datai2 <= "01100111";datai3 <= "01110011";datai4 <=
"01100111";
wait for 100 ns;
--------------------
--wait for 1100 ns;
--rst <='1';
--wait for 100 ns;

wait;
end process;
end architecture;
```

Appendix C

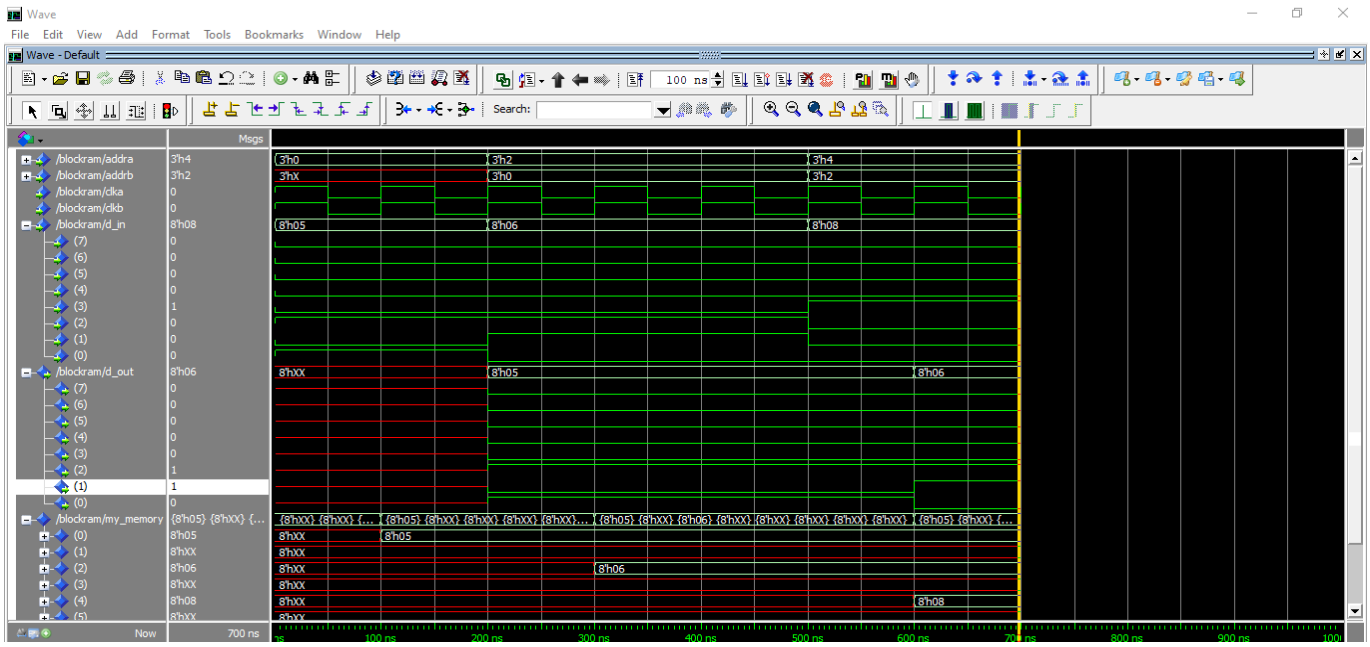
Module M-ROU-01 waveform:



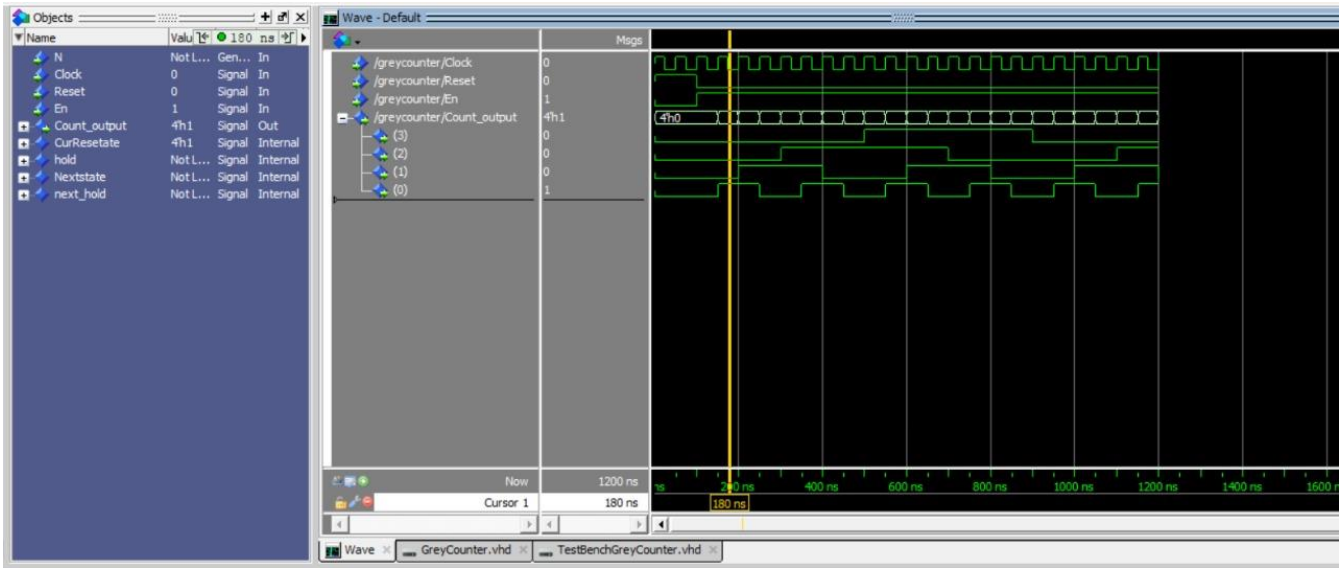
Module M-ROU-02 waveform:



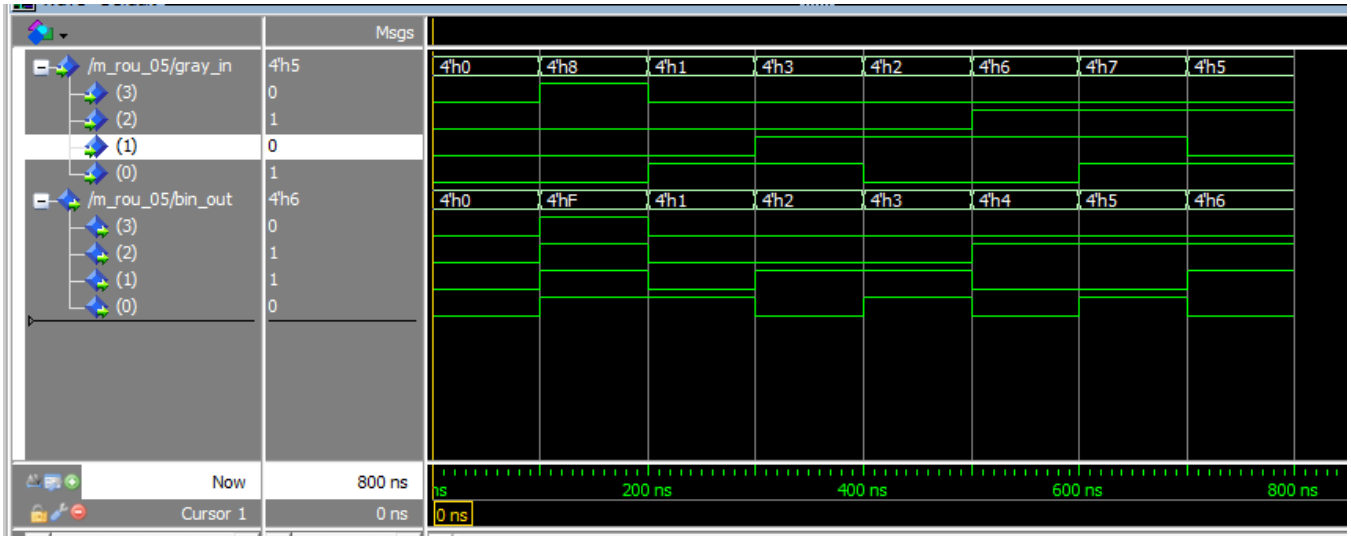
Module M-ROU-03 waveform:



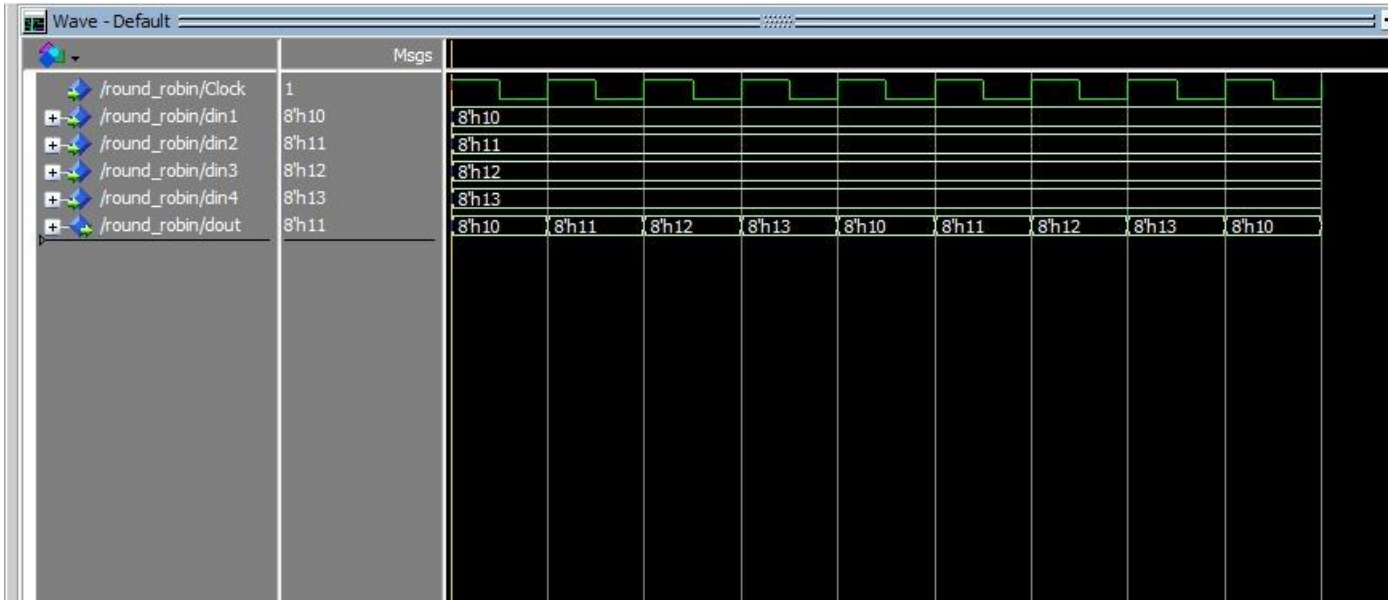
Module M-ROU-04 waveform:



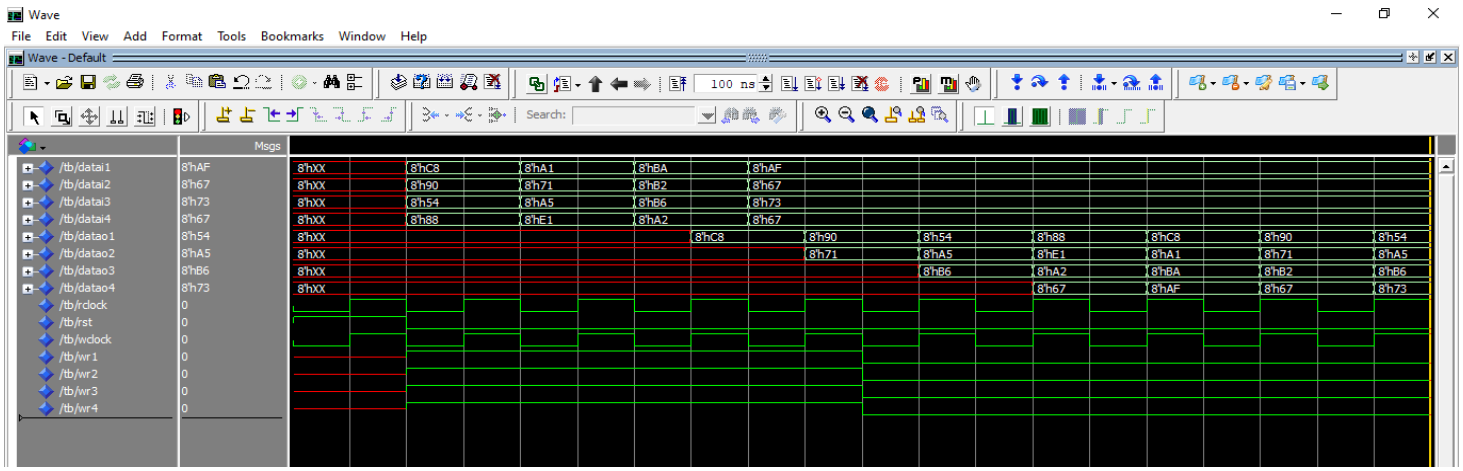
Module M0ROU-05 waveform:



Module M-ROU-08 waveform:



Module M-ROU-10 waveform:



References

[1] Swapna S Ayas Kanta Swain Kamala Kanta Mahapatra Asia Pacific Conference on Postgraduate Research

in Microelectronics & Electronics (PRIMEASIA) 2012

[2] Ruchika Chandravanshi 2Vivek Tiwari INTERNATIONAL JOURNAL OF INNOVATIVE TRENDS IN ENGINEERING (IJITE) ISSN: 239

VOLUME-14, NUMBER-01, 2016

[3] Douglas R. Melo, Cesar A. Zeferino, Luigi Dilillo, and Eduardo A. Bezerra Laboratory of Embedded and Distributed Systems (LEDS), University of Vale do Itajaí, 88302-902, Brazil 2019

[4] International Journal of Computer Applications (0975 – 8887) Volume 115 – No. 4, April 2015 Meenakshi M. Wangari, R. V. Kshirsagar

[5] J. Kim, "Low-cost router micro architecture for on chip networks," in 42nd Annual IEEE/ACM Int. Symp. Micro architecture (MICRO-42), New York, NY, USA, December 2009, pp. 255–266."

[6] Avizienis, A.; Laprie, J.C.; Randell, B.; Landwehr, C. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secur. Comput. 2004, 1, 11–33.

[7] Frantz, A.P.; Kastensmidt, F.L.; Carro, L.; Cota, E. Dependable network-on-chip router able to simultaneously tolerate soft errors and crosstalk. 22 October 2006.

[8] Tosun, S.; Ajabshir, V.B.; Mercanoglu, O.; Ozturk, O. Fault-tolerant topology generation method for application-specific network-on-chips. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2015.

[9] Pallavi B V, Dr. Baswaraj Gadgay, Veeresh Pujari: International Journal for Research in Applied Science & Engineering Technology (IJRASET) , 2016

[10] Radetzki, M.; Feng, C.; Zhao, X.; Jantsch, A. Methods for fault tolerance in networks-on-chip. ACM Comput. Surv. 2013, 46, 8

[11] Fiorin, L.; Sami, M. Fault-tolerant network interfaces for networks-on-Chip. IEEE Trans. Dependable Secur. Comput. 2014, 11, 16–29.

[12] Feng, C.; Lu, Z.; Jantsch, A.; Zhang, M.; Xing, Z. Addressing transient and permanent faults in NoC with efficient fault-tolerant deflection router. IEEE Trans. Very Large Scale Integr. Syst. 2013, 21, 1053–1066.

[13] Kohler, A.; Schley, G.; Radetzki, M. Fault tolerant network on chip switching with graceful performance degradation. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2010, 29, 883–896.