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 Electronics Design Automation	
	Semester Spring 2020	Date of Submission

#	Student ID	Grade (PASS/FAIL)
1	17p6000	
2	176009	
3	17p8070	
4	178110	
5	17P3015	

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	
1	**17p6000**	**Ahmed Wael**
2	**17p6009**	**Mohamed Ashraf**
3	**17p8070**	**Mohamed Akram**
4	**17p8110**	**Mahmoud Mohamed Mustafa**
5	**17p3015**	**Marc Nader**

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

Table of Contents

1.0 INTRODUCTION

Our objective is to design and implement Networks-on-Chip (NoC) Router using VHDL. In this report we should cover the router different stages, their functions and the data flow among them. Generally, routers task is to buffer and forward data across a network in form of packets.

Not only used in computer network applications routers are integrated in in system-on-Chip (SoC) based designs to form what is known as Networks-on-Chip (NoC) applications. NoC has shorter communication delay providing more efficient inter-module communication. We could state that routers consist of four main modules input buffer, output buffer, controller and switch fabric. Briefly the controller gets the packets from the input buffer and uses headers and forwarding tables to determine the best path for forwarding the packets by configuring the switch fabric to direct the packets to the appropriate output buffer. There are many further classification for routers based on how they handle the routing process issues (deadlock, livelock, congestion and faults) also the communication performance which is directly related to both algorism and routing protocol used. So, in order to design and implement a simple Noc router we should break the main stages into simpler modules and for each module we should set the specifications required so it can functional with complete synchronization with other modules to insure their correct integration and functionality. The modules should be described in an (input output) form so the designer task is to achieve the desired change throw the module. The designer could choose any CAD tool that supports VHDL for creating and testing the modules and integrating them to complete the simple NoC router. In the following section a detailed description for each subsystem including their digital design and verification flows and the steps that must be performed to complete the logic and physical synthesis.

2. DESIGN FLOW

Our simple NoC router could be divided into 5 main stages (input buffer, outer buffer, output queue, switch fabric (sf) and controller).

2.1 Subsystems in the NoC Router

In the next sections we should cover the digital design and verification flows and the steps that must be performed to complete the logic and physical synthesis for each subsystem.

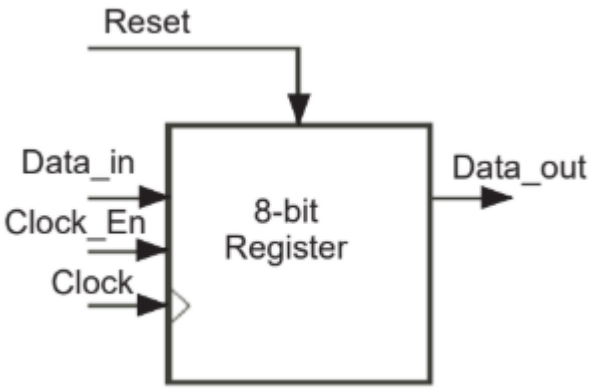
2.1.1 Input Buffer

Objective: Storing packets upon their arrival

Implementation: 8-bit register with Positive-Edge Clock, Asynchronous Reset, and Clock Enable.

Also, its synchronization with other modules is for the designer to decide

2.1.2 Output Buffer



Objective: Delivering the packets from the output queue to the output ports

Implementation: same as the input buffer 8-bit register with Positive-Edge Clock, Asynchronous Reset, and Clock Enable.

The delivery from the queue to the port is completed by the chosen algorism in the schedular

Also, the algorism is to be decided by the designer (we are using a round robin for the schedular)

2.1.3 Output queue

Objective: taking inputs and according to the state it directs the packets to the suitable port

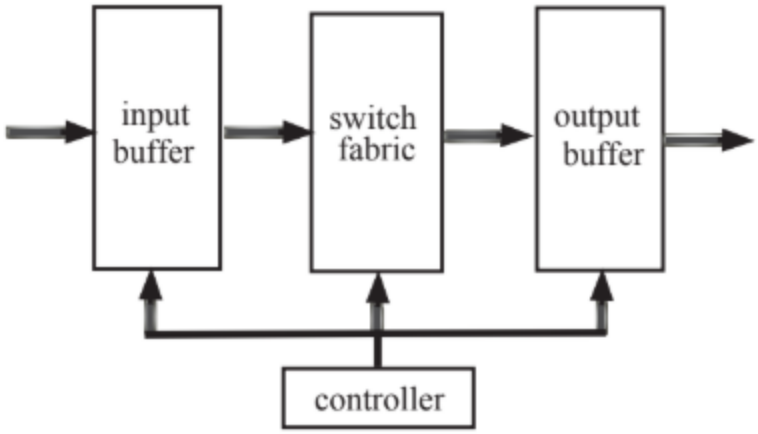
Implementation: Round robin module

2.1.4 switch fabric

Objective: Choosing according to the select line from the input

Implementation: demux

2.1.5 Controller



Objective: Synchronizing the inner modules with the incoming packets, configuring switch fabric and reading packets heads and controlling data flow and applying the schedular algorism

Implementation: Gray counter and converter for reading and writing, also requesting read and write with the rising edge.

These are the main subsystems for a NoC router before comparing this configuration with others the next list contains some tools that designers could use in implementation and integration processes.

2.2 CAD tools recommended

The list contains three CAD tools which are simple and could be quite helpful for the designers

2.2.1 Quartus prime

Powered by Intel® and having three editions (one free) could always be great choice for developers

The free edition could be downloaded from here: https://fpgasoftware.intel.com/?edition=lite

2.2.2 Modelsim

One of the most popular tools in the field powered by Mentor Graphics® also could be downloaded for free

From here: https://www.mentor.com/company/higher_ed/modelsim-student-edition

2.2.3 EDA online playground

https://www.edaplayground.com/ the site provide their users with many popular compilers in addition to the ability to simulate and test online without downloading

3. LITERATURE REVIEW

As NoC enhance the scalability and efficiency of some complex SoC, many researches have been done to consider its design and implementation features at many levels of abstraction, Some approaches like in [1] shows NoC with a mesh based topology and packet switching as communication mechanism and its effectiveness.

A large part of modern NoC architecture is a network monitoring service that makes it possible to show the status of the network and adapt routing based on it, distributed network monitoring methods, router status based monitoring and FIFO status based monitoring are presented [2] by analyzing theoretical features and problems on routing.

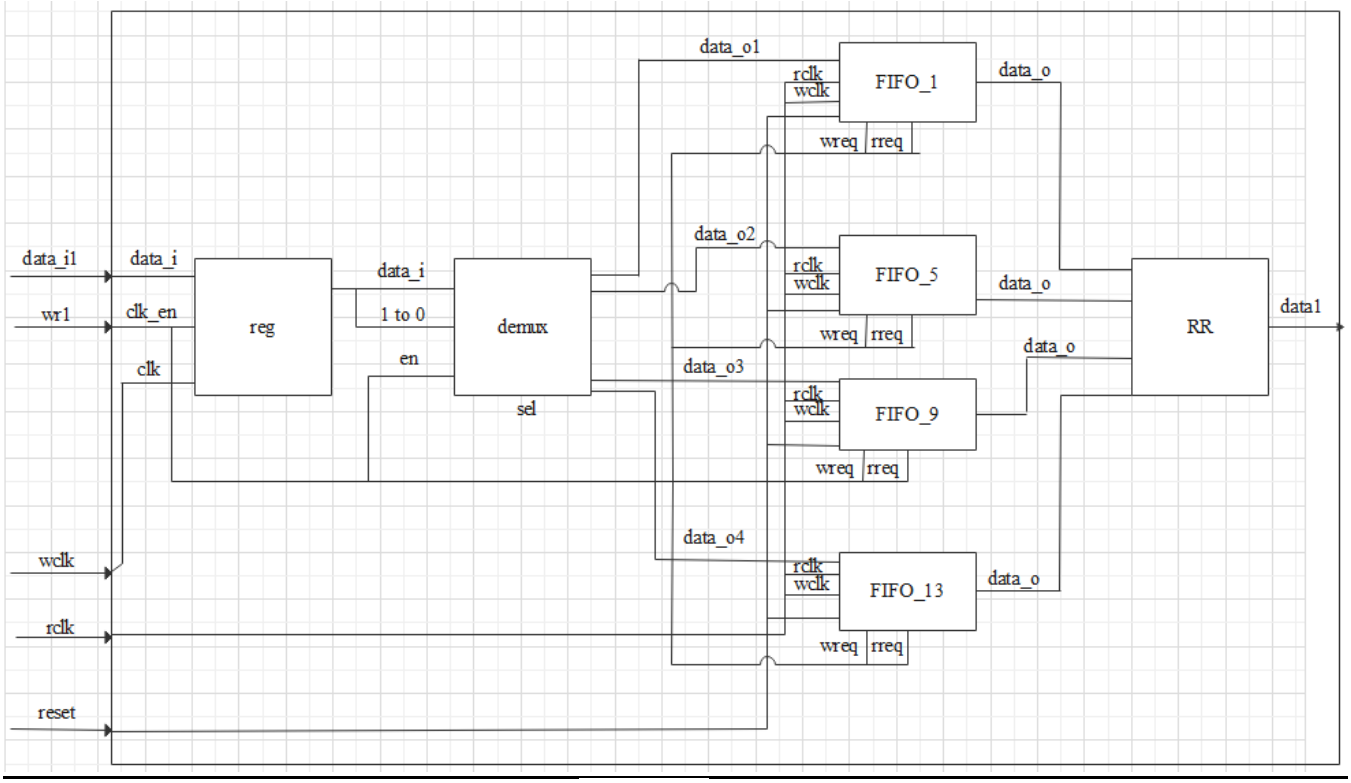
There are multiple different techniques in NoC that are used for forwarding information through the network, each of these different techniques have a large effect on the router architecture design. Nowadays Noc designs are based on packet switching [3].

[4] Consider analyzing NoC properties like bandwidth and latency, some problems of SoC designing as non-scalable wire delays that carry signals across a chip and shows that most of huge designs solve this problem by using FIFO buffers to synchronously propagate data over long distances, there exist some techniques that are proposed to improve NoC performance in terms of latency while others are proposed to improve power consumption and area utilization.

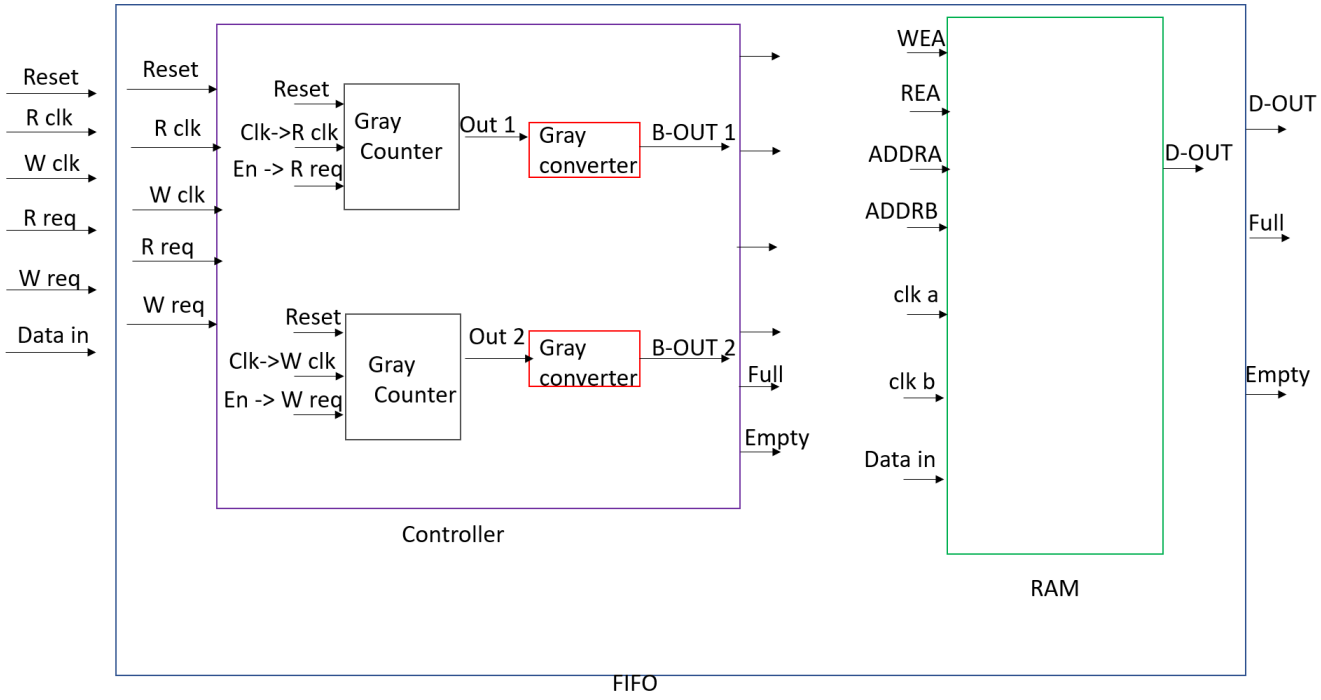
Paper [5] introduces a router architecture which improves the performance of the network by using the same amount of buffers like other architectures but in more efficient manner. It improves the network by providing a way to manage the requests to a busy buffer by the other buffers found in the router, this router is called flexible router.

Designing and implementation of a NoC router based on handshaking communication mechanism between routers showing interaction and how data is exchanged, structure of information packets, router function and different states of the router are analyzed [6].

4. Design Implementation:



Router



Module Functionality

Module (1) register

This module is an 8-bit register which takes an 8 bit input and at the positive clock edge if the enable is equal to one this input will be presented at output. This register is used to synchronize the input time of the 4 data inputs of the router. This module is implemented in VHDL without any difficult challenges, just check if there is a positive edge and the enable equals one the input is transmitted to the output. Looking to synthesis the check on clock infers a flip in addition to gate to check the enable

Module (2) demux

This is a module which presents a demultiplexer which gets an input data, which comes from the register, enable which approves the passing of data from input to output, a select line which determines on which output does the input go to because this demultiplexer has 4 output. When select is 00 data is transferred to first data when 01 the data is transferred to second output and so on. There were really no design challenges in this module. In synthesis a latch is inferred because there is

nothing specified when the select is other than 00 01 10 11. In addition to a multiplexer to choose the correct output port according to the select line

Module (3) RAM

This module represents a dual port ram. The input is two clocks, two addresses, two clocks, two enables and an input. Every input is entered two times other than the input because one is for reading and one is for writing. The main functionality of the dual port ram is to store the address of the write input when write is enabled and clock is positive edge and keeps adding to the array when the input specifies that it must continue writing in the specified address. While when reading is enabled and the reading clock is at its positive edge the value at the requested address is transferred at the output port. The design challenge was to store the input from the array and reading from it. The if of both clocks infers two flip flops, and the check on both write enable and read enable adds gates.

Module (4) gray counter

A simple gray counter is designed in this module. This module has a reset input when it is enabled the gray number is rest to 0000. Gray number representation is a different representation than binary representation as the increment of the number by only one single decimal value changes only a bit to prevent any invalid states from being present, and the amount of switching is minimized. This module also has 2 inputs enable and clock. At the positive edge of the clock if the enable is equated to 1 the current number is incremented by 1 decimal value which changes only 1 bit. Presence of check on clock infers a flip flop. While the check on enable and reset infers a couple of gates.

Module (5) gray converter

This module presents a gray converter which converts the input number from the gray value to the binary value. Without any checks on clock or enables the input is converted using combinational logic to a binary value. There were no design challenges in this module. Looking into synthesis no latches or flip flops are inferred. There is only combinational logic present in this module which is synthesized into a couple of xor gates which changes each bit of the gray number input to a binary number represented as output.

Module (6) FIFO controller

This is the module which controls another module which will come later which is the FIFO. This module has several inputs two clocks, two requests. It contains another two modules which is the gray counter and converter. The main functionality of this module is to check whether read and write are valid and whether the ram is empty or full or neither and when read and write are valid it uses the 2 modules in it to get an address which the data should be written in. The design challenge of this module was to make sure that the pointers don't exceed their limits and that the outputs are synchronized with the change in values. Synthesis in this module there are a lot of gates which keep track of the combinational logic occurred in this module, but no latches will be inferred because in all conditions all outputs are given a certain value.

Module (7) FIFO

In this module two other modules are used which are the FIFO controller and the RAM. There were no difficulties designing this module just mapping the input of this module to both modules and transferring the data between them using signals.

Module (8) Round Robin

This module is represented using a FSM. All the details about the FSM will come later in the FSM section. The main functionality of this module was taking 4 data inputs and at every positive edge on of the 4 values is propagated to the output. In synthesis a multiplexer appears to allow choosing the appropriate state. No latch is inferred because all cases are covered. A flip flop is inferred to check on the rising edge of the clock

Module (9) 4 port Router

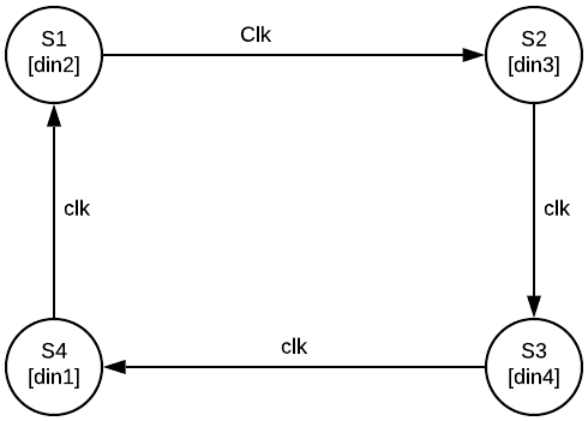
This is the module which links all the modules with each other. The design challenges were to determine which ports of each module are linked to which ports in other modules.

Module (10) router test bench

The main function of this module is to verify that the 4 port router in module 9 is working fine and is functioning according to the specifications and to analyze its performance. The details of testing will be discussed later on in another part.

5. Schedular Design and FSM Implementation

5.1 FSM



5.2 Difference between Mealy and Moore

In Mealy the output depends on the input and the state as knowing the state isn't enough, but as for the Moore the output depends on the state each state always gives a specific output no matter what the output is.

In our design it could have gone both ways, but we found that choosing the **Moore** state module was more convenient. As each state always gave one specific each rising edge so the output basically depended only on the state.

5.3 Difference between 1-process||2-process||3-process VHDL implementation

While using the 1-process style, all the work is imply done in one process (state update and assigning as well as input and output) which will result in the code being a bit confusing and not a nice encoding style, as everything is executed sequentially and this unorganized code increases percentage of error occurring. It uses only one signal (current state) to control all transitions.

Therefore using 2-process style, we simply divide the work into two processes instead of 1 process containing everything. one process handling the update and reset of state each rising edge and one handling the assignment of this state (what the next state will be, as well as the input and output). This style uses 2 signals, one for the current state and one for the next state.

The final style which is the 3-process style, is the clearest and best of them all as the work is divided between 3 processes. one process controlling the reset and change of state each clock cycle, one controlling what the next state will be, and one for the output. This style also uses 2 signals, one for the current state and one for the next state.

Finally, in our design we used the 2 process style because as we said the one process is a mess and completely unfavorable to use, and the code wasn't big so there was no need for the 3-process method so the 2-process was so convenient.

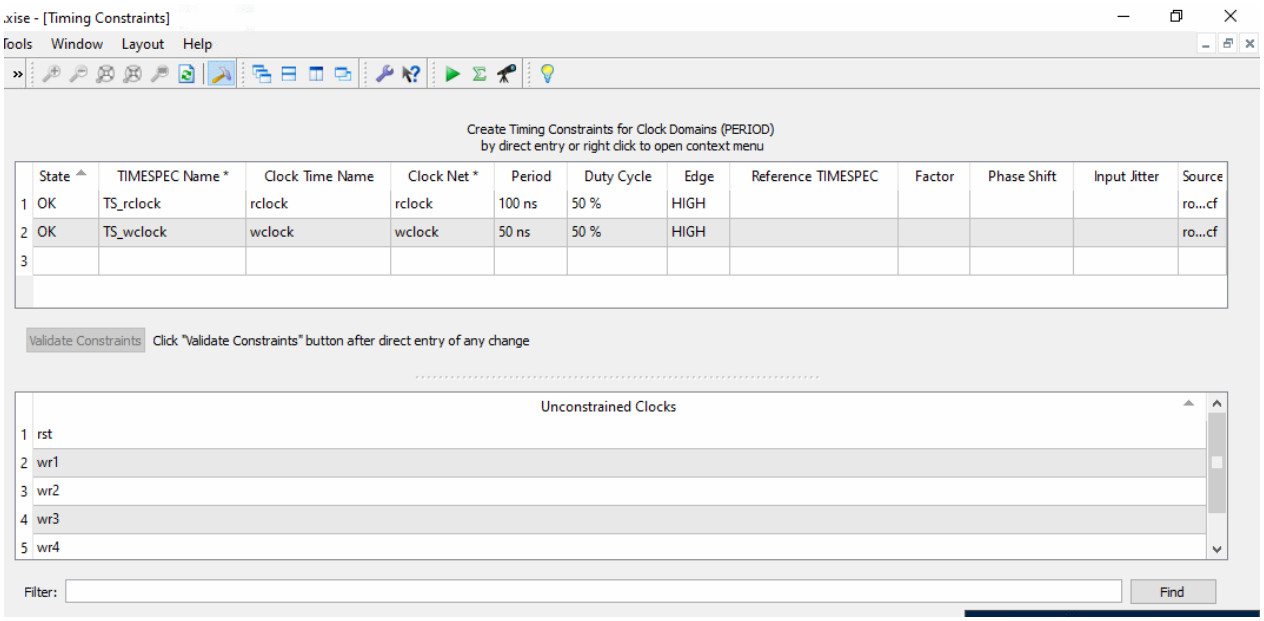
5.4 Timing Constrain

router Project Status (06/02/2020 - 20:14:25)			
Project File:	router.xise	**Parser Errors:**	No Errors
Module Name:	router	**Implementation State:**	Placed and Routed
Target Device:	xc7a100t-3csg324	**•Errors:**	No Errors
Product Version:	ISE 14.5	**•Warnings:**	107 Warnings (0 new)
Design Goal:	Balanced	**•Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	**•Timing Constraints:**	All Constraints Met
Environment:	System Settings	**•Final Timing Score:**	0 (Timing Report)

Device Utilization Summary [-]				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	304	126,800	1%	
Number used as Flip Flops	168			
Number used as Latches	136			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	250	63,400	1%	
Number used as logic	154	63,400	1%	
Number using O6 output only	80			
Number using O5 output only	0			
Number using O5 and O6	74			

Timing Constraints

	Met	Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
1	Yes	TS_wclock = PERIOD TIMEGRP "wclock" 50 ns HIGH 50%	SETUP HOLD	47.353ns 0.230ns	2.647ns	0 0	0 0
2	Yes	TS_rclock = PERIOD TIMEGRP "rclock" 100 ns HIGH 50%	SETUP HOLD MINPERIOD	98.817ns 0.282ns 98.408ns	1.183ns 1.592ns	0 0 0	0 0 0



6. Test and Simulation Results

Module_1 test bench

Test strategy

Tested feature	input			output	delay
	reset	Data in	Clk_en	Data out	(ns)
Resetting the register	1	xxxxxxxx	x	00000000	20
Insert data and check output at rising edge	0	00000001	1	00000001	20
Insert another data	0	00111010	1	00111010	20
Enable check	0	00111010	0	00111010	20

Modle_2 test bench

Test strategy

Tested feature	input			output				delay
	en	Data in	sel	Data out 1	Data out 2	Data out 3	Data out 4	(ns)
Check on output when select is 00	1	01100110	00	01100110	00000000	00000000	00000000	10
Check on output when select is 01	1	01100110	01	00000000	01100110	00000000	00000000	10
Check on output when select is 10	1	00001111	10	00000000	00000000	00001111	00000000	10
Check on output when select is 11	1	11111111	11	00000000	00000000	00000000	11111111	10
Enable check	0	11111111	00	00000000	00000000	00000000	11111111	10

Module_3 test bench

Test strategy

Tested feature	input					output	delay
	wea	rea	addra	addrb	Data in	Data out	(ns)
Add write address	1	0	000	000	00001111	XXXXXXXX	20
Get read address	1	1	000	000	00001111	00001111	20
Add write address	1	1	001	011	00001001	XXXXXXXX	20
Check on read address	1	1	001	001	00001001	00001001	20
Write enable off check	0	1	111	111	11111001	XXXXXXXX	20
Read enable off check	1	0	111	111	11111001	XXXXXXXX	20

Module_4 test bench

Test strategy

Tested feature	input		output	delay
	en	reset	Count out	(ns)
Reset check	1	1	0000	20
First clock rising edge increment	1	0	0001	20
Second clock rising edge increment	1	0	0010	20
sixth clock rising edge increment	1	0	0111	20
enable off check	0	1	0111	20

Module_5 test bench

Test strategy

Tested feature	input	output	delay
	gray_in	b_out	(ns)
first input	0000	0000	20
second input	1000	1111	20
third input	0100	0111	20
fourth input	0111	0101	20

Module_6 test bench

Test strategy

Tested feature	input			output						delay
	reset	rreq	wreq	write valid	read valid	empty	full	read pointer	write pointer	(ns)
reset	1	1	0	1	0	1	0	UUU	UUU	20
read before write	0	1	0	1	0	1	0	UUU	UUU	20
Add write address	0	1	1	1	1	0	0	UUU	000	20
second write address and first read address	0	1	1	1	1	0	0	000	001	20
last write done and last read	0	1	0	0	1	0	1	111	111	20
stable output	0	0	0	0	0	0	1	111	111	20

Module_7 test bench

Test strategy

Tested feature	input				output			delay
	reset	rreq	wreq	data in	empty	full	data out	(ns)
reset check	1	1	1	01010101	1	0	UUUUUUUU	20
first input	0	1	1	01010101	0	0	01010101	20
second input	0	1	1	10101010	0	0	10101010	60
third input	0	1	1	00000000	0	0	00000000	60
sixth input	0	1	1	10111110	0	0	10111110	60
request check	0	0	0	11001100	0	0	10111110	60

Module_8 test bench

Test strategy

Tested feature	input				output	delay
	data in1	data in 2	data in3	data in 4	data out	(ns)
third state	00000001	00000011	00001111	11110000	00001111	21
fourth state	00000001	00000011	00001111	11110000	11110000	21
first state	00000001	00000011	00001111	11110000	00000001	21
second state	00000001	00000011	00001111	11110000	00000011	21
third state second check	00000001	00000011	00001111	11110000	00001111	21

Module_10 test bench

Test strategy

	tested feature	input								output				delay
		reset	data in 1	data in 2	data in 3	data in 4	wr 1	wr 2	wr 3	wr 4	data out 1	data out 2	data out 3	data out 4	
test case 1	reset check	1					0	0	0	0	UUUUUUUU	UUUUUUUU	UUUUUUUU	UUUUUUUU	20 ns
test case 2	request check	0	10000000	1010110	10101010	11001100	0	0	0	0	XXXXXXXX	XXXXXXXX	XXXXXXXX	XXXXXXXX	20 ns
test case 3	random input														
	in all input ports	0	10000000	10100110	10101010	11001100'	1	1	1	1	10000000	00000000'	00000000'	00000000'	150 ns
		0	10000000	10100110	10101010	11001100'	1	1	1	1	00000000'	00000000'	10100110	00000000'	160 ns
		0	10000000	10100110	10101010	11001100'	1	1	1	1	00000000'	00000000'	10101010	00000000'	170 ns
		0	10000000	10100110	10101010	11001100'	1	1	1	1	11001100'	00000000'	00000000'	00000000'	180 ns
test case 4	all input enter	0	01010100'	00110001'	10101010	00011111'	1	1	1	1	01010100'	00000000'	00000000'	00000000'	150 ns
	different scheduler	0	01010100'	00110001'	10101010	00011111'	1	1	1	1	00000000'	00110001'	00000000'	00000000'	160 ns
		0	01010100'	00110001'	10101010	00011111'	1	1	1	1	00000000'	00000000'	10101010	00000000'	170 ns
		0	01010100'	00110001'	10101010	00011111'	1	1	1	1	00000000'	00000000'	00000000'	00011111'	180 ns
test case 5	all input enter	0	00011110'	01011110'	10011110	11011110	1	1	1	1	00000000'	00000000'	00011110'	00000000'	150 ns
	same scheduler	0	00011110'	01011110'	10011110	11011110	1	1	1	1	00000000'	00000000'	01011110'	00000000'	160 ns
		0	00011110'	01011110'	10011110	11011110	1	1	1	1	00000000'	00000000'	10011110	00000000'	170 ns
		0	00011110'	01011110'	10011110	11011110	1	1	1	1	00000000'	00000000'	11011110	00000000'	180 ns

There are five test cases covered in this test bench. The first one is checking that the reset is working properly. Second test case disables the checks and enters data and determine whether the data will propagate to the output or no. Third test case 4 random inputs are inserted and the output is evaluated at the corresponding time and at the corresponding ports. 4th test case all input enter different scheduler so input must exit at different ports. Last test case all input enter the same scheduler so they all must exit from the same port

7. Conclusion

To design and implement Networks-on-Chip (NoC) Router, we have implemented ten different modules to cover the main four modules of the router, Input and output buffer by implementing 8-bit register where its main function is to store the packet when it's arrived and present output, Switch fabric by implementing a demultiplexer module which gets input from the register and pass it

as output by select line, these demultiplexers are controlled from the controller module which we designed to synchronize router incoming packets with internal modules and configure the switch fabric, also it controls the data that flows inside the router and apply the scheduler algorithm round robin at the output ports, also we implemented round robin and presented it as FSM, this module takes four inputs and propagates them at positive edge where in synthesis a multiplexer appears to allow choosing the appropriate state, no latch is inferred because all cases are covered, a flip flop is inferred to check on the rising edge of the clock, Output queues are implemented as FIFO buffers, we implemented it by three modules; FIFO, FIFO controller and RAM, also gray counter and gray converted modules implemented that changes each bit of the gray number input to a binary number, finally we implemented a 4 port router which links all the modules implemented with each other and we tested its functionality by implementing a router test bench, simulation and synthesis is verified through VHDL using ModelSim.
As a future work we intend to assess the performance of the router under routing algorithms and traffic patterns, find a solution to control the lagging distance of the unordered packets.

8. Task Distribution List

The project workload is distributed equally among all team members

9. References

[1] G. Brebner and D. Levi. Networking on chip with platform fpgas. In Proc. FPT, 2003

[2] Rantala, Ville & Lehtonen, Teijo & Plosila, Juha., ”Network on Chip Routing Algorithms,” 2008

[3] A. Mello, L. Tedesco, N. Calazans and F. Moraes, “Virtual channels in networks on chip: implementation and evaluation on hermes NoC,” in Proceedings of the 18th annual symposium on Integrated circuits and system design, ACM: Florianolpolis, Brazil. 2005

[4] P.P. Pande et al. “Performance evaluation and design trade-offs for network-on-chip interconnect architectures,” IEEE Transactions on Computers, Vol. 54, Issue 8, Aug. 2005, pp. 1025 - 1040.

[5] H. El-Sayed, M. Ragab, M. S. Sayed and V. Goulart, "Hardware implementation and evaluation of the Flexible router architecture for NoCs," 2013 IEEE 20th International Conference on Electronics, Circuits, and Systems (ICECS), 2013, pp. 621-624

[6] Seyyed Amir Asghari, Hossein Pedram, Mohammad Khademi, and Pooria Yaghini, “Designing and Implementation of a Network Chip Router Handshaking Applied Seiences on Based on Communication Mechanism,” World Journal, vol.6, no. 1, 2009, pp.88-93

Appendix A VHDL Model Source Code

Module 1

```
library ieee;
use ieee.std_logic_1164.all;
entity reg is
 port( Clock,Clock_En,Reset: in std_logic ;
 Data_in:in std_logic_vector (7 downto 0);
 Data_out: out std_logic_vector (7 downto 0));
end entity reg;
architecture behave of reg is
 BEGIN
 process (Clock,Clock_En,Reset)
 begin
 if(Reset='1') then
 Data_out <="00000000";
 elsif ( rising_edge(Clock)) then
 if(Clock_En='1') then
 Data_out <= Data_in;
 end if;
 end if;
 end process;
end architecture;
```

Module 2

```
library ieee;
use ieee.std_logic_1164.all;
entity demux is
 generic (n: integer:=8);
 port(d_in:in std_logic_vector (n-1 downto 0);
 En : in std_logic;
 sel : in std_logic_vector(1 downto 0);
 d_out1:out std_logic_vector (n-1 downto 0);
 d_out2:out std_logic_vector (n-1 downto 0);
 d_out3:out std_logic_vector (n-1 downto 0);
```

```
 d_out4:out std_logic_vector (n-1 downto 0));
 end entity;
architecture behav of demux is
begin
p1: process(sel,d_in) 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_out1<="00000000";
 d_out3<="00000000";
 d_out4<="00000000";
when "10" =>
 d_out3<=d_in;
 d_out2<="00000000";
 d_out1<="00000000";
 d_out4<="00000000";
when "11" =>
 d_out4<=d_in;
 d_out2<="00000000";
 d_out3<="00000000";
 d_out1<="00000000";
when others => null;
end case;
end if;
end process;
end architecture;
```

Module 3

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
entity ram is
 port(wea,rea,clka,clkb: in std_logic ;
 addra,addrb: in std_logic_vector (2 downto 0);
 d_in: in std_logic_vector (7 downto 0);
 d_out: out std_logic_vector (7 downto 0));
end entity;
```

```
Architecture behave_ram of ram is
type ram is array (0 to 7) of std_logic_vector (7 downto 0);
signal bram : ram :=(others =>"XXXXXXXX") ;
BEGIN
proc1:process(wea,clka)
 begin
 if( rising_edge(clka)) then
 if(wea='1')then
 bram(to_integer( unsigned(addra))) <= d_in;
 end if;
 end if;
 end process;
proc2: process(rea,clkb)
 begin
 if( rising_edge(clkb) ) then
 if(rea<='1') then
 d_out <= bram( to_integer(unsigned(addrb)));
 end if;
 end if;
 end process;
end architecture;
```

Module 4

```
LIBRARY ieee;
 USE ieee.std_logic_1164.ALL;
 USE ieee.numeric_std.ALL;
 ENTITY GrayCounter IS
 GENERIC (N: integer := 4);
 PORT (clock, reset, En: IN std_logic;
 count_out: OUT std_logic_vector (N-1 DOWNTO 0));
 END GrayCounter;

 ARCHITECTURE beh OF GrayCounter IS
 SIGNAL current_number, next_number, hold, next_hold: std_logic_vector (N-1 DOWNTO 0);
 BEGIN

 p1: PROCESS (clock)
 BEGIN
 IF (clock = '1' AND clock'EVENT) THEN
 IF (reset = '1') THEN
 current_number <= ("0000");
 ELSIF (En = '1') THEN
```

```
 current_number <= next_number;
 END IF;
 END IF;
 END PROCESS;

 hold <= current_number XOR ('0' & hold(N-1 DOWNTO 1));
 next_hold <= std_logic_vector(unsigned(hold) + 1);
 next_number <= next_hold XOR ('0' & next_hold(N-1 DOWNTO 1));
 count_out <= current_number;

 END beh;
```

Module 5

```
library ieee;
use ieee.std_logic_1164.all;
entity GrayToBinary is
 port(gray_in:in std_logic_vector(3 downto 0);
 b_out:out std_logic_vector(3 downto 0)
 );
 end entity;
architecture arch of GrayToBinary is
 begin
 b_out(0)<=gray_in(3) xor gray_in(2) xor gray_in(1) xor gray_in(0);
 b_out(1)<=gray_in(3) xor gray_in(2) xor gray_in(1);
 b_out(2)<=gray_in(3) xor gray_in(2);
 b_out(3)<=gray_in(3);
end arch;
```

Module 6

```
library ieee;
use IEEE.std_logic_1164.all;
Entity FIFO_Controller is
 port ( reset: in std_logic;
 rdclk: in std_logic;
 wrclk: in std_logic;
 rreq: in std_logic;
 wreq: in std_logic;
 write_valid: out std_logic;
 read_valid: out std_logic;
 wr_ptr: out std_logic_vector ( 2 downto 0 );
```

```
 rd_ptr: out std_logic_vector ( 2 downto 0);
 empty: out std_logic; --flag
 full: out std_logic --flag
 );
 end FIFO_Controller;

Architecture behave of FIFO_Controller is
 Component GrayCounter IS
 GENERIC (N: integer := 4);
 PORT (clock, reset, En: IN std_logic;
 count_out: OUT std_logic_vector (N-1 DOWNTO 0));
 END component;
 Component GrayToBinary is
 port(gray_in:in std_logic_vector(3 downto 0);
 b_out:out std_logic_vector(3 downto 0)
 );
 end Component;
 signal read_counter_output: std_logic_vector (3 DOWNTO 0);
 signal write_counter_output: std_logic_vector (3 DOWNTO 0);
 signal read_rst: std_logic;
 signal read_en: std_logic;
 signal write_rst: std_logic;
 signal write_en: std_logic;
 signal read_converter_output: std_logic_vector (3 downto 0);
 signal write_converter_output: std_logic_vector (3 downto 0);
 signal write_v :std_logic;
 signal read_v :std_logic;
 signal s_empty: std_logic;
 signal s_full : std_logic;

 begin
 Countv1: GrayCounter Port Map (rdclk,read_rst,read_en, read_counter_output); --read
 Countv2: GrayCounter Port Map (wrclk,write_rst,write_en , write_counter_output);
-- write
 Conv1: GrayToBinary Port Map (read_counter_output, read_converter_output);
 Conv2: GrayToBinary Port Map (write_counter_output, write_converter_output);
 p: process (rreq, wreq, reset,rdclk,wrclk)
 begin
 if(reset='1') then
 write_v<='1';
 read_rst<='1';
 write_rst<='1';
 read_en<='0';
 write_en<='1';
 s_empty<='1';
 s_full<='0';
 wr_ptr<="000";---
```

```
 rd_ptr<="000";---
 else
 read_rst<='0';
 write_rst<='0';

 if(write_converter_output="UUUU") then---
 s_empty<='1';
else
 s_empty<='0';
end if;
 if(rreq='1' and read_v='1') then
 read_en<='1';
 rd_ptr<=read_converter_output(2 downto 0);
 else
 read_en<='0';
 end if;
 if( wreq='1' and write_v='1') then
 write_en<='1';
 wr_ptr<=write_converter_output( 2 downto 0);
 else
 write_en<='0';
 end if;
 if(s_empty='1') then
 read_v<='0';
 elsif(write_converter_output=read_converter_output) then
 read_v<='0';
 read_en<='0';
elsif(read_converter_output="1000") then---
 read_v<='0';
 read_en<='0';
 else
 read_v<='1';
end if;
 if(write_converter_output="1000") then----
 s_full<='1';
 write_v<='0';
 write_en<='0';
 else
 write_v<='1';
 s_full<='0';
 end if;
 end if;
 end process;
 write_valid<=write_v;
 read_valid<=read_v;
 empty<=s_empty;
 full<=s_full;
```

```
 end architecture;
```

Module 7

```
library ieee;
use IEEE.std_logic_1164.all;
USE ieee.numeric_std.ALL;
Entity FIFO is
 port ( reset: in std_logic;
 rdclk: in std_logic;
 wrclk: in std_logic;
 rreq: in std_logic;
 wreq: in std_logic;
 data_in : in std_logic_vector(7 downto 0);
 data_out: out std_logic_vector(7 downto 0);
 empty: out std_logic; --flag
 full: out std_logic --flag
 );
 end FIFO;
 architecture struct of FIFO is
 component FIFO_Controller is
 port ( reset: in std_logic;
 rdclk: in std_logic;
 wrclk: in std_logic;
 rreq: in std_logic;
 wreq: in std_logic;
 write_valid: out std_logic;
 read_valid: out std_logic;
 wr_ptr: out std_logic_vector ( 2 downto 0 );
 rd_ptr: out std_logic_vector ( 2 downto 0);
 empty: out std_logic; --flag
 full: out std_logic --flag
 );
 end component;
---------------------
 component ram is
port(wea,rea,clka,clkb: in std_logic ;
addra,addrb: in std_logic_vector (2 downto 0);
d_in: in std_logic_vector (7 downto 0);
d_out: out std_logic_vector (7 downto 0));
end component;
----------------------------
signal read_valid,write_valid : std_logic;
signal wr_ptr,rd_ptr : std_logic_vector (2 downto 0);
---------------------------
begin
```

```
 controller : FIFO_controller port map(reset,rdclk,wrclk,rreq,wreq,write_valid,
 read_valid,wr_ptr,rd_ptr,empty,full);
 memory : ram port map(write_valid,read_valid,wrclk,rdclk,wr_ptr,rd_ptr,
 data_in,data_out);
 end architecture;
```

Module 8

```
library ieee;
use ieee.std_logic_1164.all;
entity rr is
 port( clock:in std_logic;
 din1,din2,din3,din4:in std_logic_vector(7 downto 0);
 dout:out std_logic_vector(7 downto 0));
end entity;
architecture behave of rr is
Type state is (s1,s2,s3,s4);
signal current_state:state :=s1;
signal next_state:state;
BEGIN
 process(clock)
 begin
 if rising_edge(clock) then
 current_state <= next_state;
 end if;
 end process;

 process(current_state)
 begin
 case current_state is
 when s1 =>
 next_state<=s2;
 dout<=din2;
 when s2 =>
 next_state<=s3;
 dout<=din3;
 when s3 =>
 next_state<=s4;
 dout<=din4;
 when s4 =>
 next_state<=s1;
 dout<=din1;
 end case;
 end process;
end architecture;
```

Module 9

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
entity router is
 port( rst, wclock, rclock: in std_logic;
 wr1,wr2,wr3,wr4:in std_logic;
 datai1,datai2,datai3,datai4:in std_logic_vector (7 downto 0);
 datao1,datao2,datao3,datao4:out std_logic_vector (7 downto 0));
end entity;
ARCHITECTURE behave of router is
 component reg is
 port( Clock,Clock_En,Reset: in std_logic ;
 Data_in:in std_logic_vector (7 downto 0);
 Data_out: out std_logic_vector (7 downto 0));
 end component;
----------------------------
 component demux is
 generic (n: integer:=8);
 port(d_in:in std_logic_vector (n-1 downto 0);
 En : in std_logic;
 sel : in std_logic_vector(1 downto 0);
 d_out1:out std_logic_vector (n-1 downto 0);
 d_out2:out std_logic_vector (n-1 downto 0);
 d_out3:out std_logic_vector (n-1 downto 0);
 d_out4:out std_logic_vector (n-1 downto 0));
 end component;
----------------------------
component FIFO is
 port ( reset: in std_logic;
 rdclk: in std_logic;
 wrclk: in std_logic;
 rreq: in std_logic;
 wreq: in std_logic;
 data_in : in std_logic_vector(7 downto 0);
 data_out: out std_logic_vector(7 downto 0);
 empty: out std_logic; --flag
 full: out std_logic --flag
 );
 end component;
-------------------------------------
component rr is
 port( clock:in std_logic;
 din1,din2,din3,din4:in std_logic_vector(7 downto 0);
 dout:out std_logic_vector(7 downto 0));
```

```
end component;
-----------------------------------------------
signal reg_out1,reg_out2,reg_out3,reg_out4:std_logic_vector (7 downto 0);
signal demux_out1,demux_out2,demux_out3,demux_out4:std_logic_vector (7 downto 0);
signal demux_out5,demux_out6,demux_out7,demux_out8:std_logic_vector (7 downto 0);
signal demux_out9,demux_out10,demux_out11,demux_out12:std_logic_vector (7 downto 0);
signal demux_out13,demux_out14,demux_out15,demux_out16:std_logic_vector (7 downto 0);
signal fifo_out1,fifo_out2,fifo_out3,fifo_out4,
 fifo_out5,fifo_out6,fifo_out7,fifo_out8,
 fifo_out9,fifo_out10,fifo_out11,fifo_out12,
 fifo_out13,fifo_out14,fifo_out15,fifo_out16: std_logic_vector(7 downto 0);
signal empty1,empty2,empty3,empty4,empty5,empty6,empty7,empty8,
 empty9,empty10,empty11,empty12,empty13,empty14,empty15,empty16:std_logic;
signal full1,full2,full3,full4,full5,full6,full7,full8,
 full9,full10,full11,full12,full13,full14,full15,full16:std_logic;
BEGIN
r1:reg port map(wclock,wr1,rst,datai1,reg_out1);
r2:reg port map(wclock,wr2,rst,datai2,reg_out2);
r3:reg port map(wclock,wr3,rst,datai3,reg_out3);
r4:reg port map(wclock,wr4,rst,datai4,reg_out4);
d1:demux port map(reg_out1,wr1,reg_out1(1 downto 0),demux_out1,demux_out2,demux_out3,de
mux_out4);
d2:demux port map(reg_out2,wr2,reg_out2(1 downto 0),demux_out5,demux_out6,demux_out7,de
mux_out8);
d3:demux port map(reg_out3,wr3,reg_out3(1 downto 0),demux_out9,demux_out10,demux_out11,
demux_out12);
d4:demux port map(reg_out4,wr4,reg_out4(1 downto 0),demux_out13,demux_out14,demux_out15
,demux_out16);
f1: fifo port map(rst,wclock,wclock,'1',wr1,demux_out1,fifo_out1,empty1,full1);
f2: fifo port map(rst,wclock,wclock,'1',wr1,demux_out2,fifo_out2,empty2,full2);
f3: fifo port map(rst,wclock,wclock,'1',wr1,demux_out3,fifo_out3,empty3,full3);
f4: fifo port map(rst,wclock,wclock,'1',wr1,demux_out4,fifo_out4,empty4,full4);
f5: fifo port map(rst,wclock,wclock,'1',wr2,demux_out5,fifo_out5,empty5,full5);
f6: fifo port map(rst,wclock,wclock,'1',wr2,demux_out6,fifo_out6,empty6,full6);
f7: fifo port map(rst,wclock,wclock,'1',wr2,demux_out7,fifo_out7,empty7,full7);
f8: fifo port map(rst,wclock,wclock,'1',wr2,demux_out8,fifo_out8,empty8,full8);
f9: fifo port map(rst,wclock,wclock,'1',wr3,demux_out9,fifo_out9,empty9,full9);
f10: fifo port map(rst,wclock,wclock,'1',wr3,demux_out10,fifo_out10,empty10,full10);
f11: fifo port map(rst,wclock,wclock,'1',wr3,demux_out11,fifo_out11,empty11,full11);
f12: fifo port map(rst,wclock,wclock,'1',wr3,demux_out12,fifo_out12,empty12,full12);
f13: fifo port map(rst,wclock,wclock,'1',wr4,demux_out13,fifo_out13,empty13,full13);
f14: fifo port map(rst,wclock,wclock,'1',wr4,demux_out14,fifo_out14,empty14,full14);
f15: fifo port map(rst,wclock,wclock,'1',wr4,demux_out15,fifo_out15,empty15,full15);
f16: fifo port map(rst,wclock,wclock,'1',wr4,demux_out16,fifo_out16,empty16,full16);
rr1: rr port map(rclock,fifo_out1,fifo_out5,fifo_out9,fifo_out13,datao1);
rr2: rr port map(rclock,fifo_out2,fifo_out6,fifo_out10,fifo_out14,datao2);
rr3: rr port map(rclock,fifo_out3,fifo_out7,fifo_out11,fifo_out15,datao3);
```

```
rr4: rr port map(rclock,fifo_out4,fifo_out8,fifo_out12,fifo_out16,datao4);
end ARCHITECTURE;
```

Appendix B VHDL Test Bench Source Code

Testbench 1

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

entity reg_tb is
end entity;

Architecture test of reg_tb is

component reg is
     port( Clock,Clock_En,Reset: in std_logic ;
        Data_in:in std_logic_vector (7 downto 0);
        Data_out: out std_logic_vector (7 downto 0));
 end component;

signal Clock,Clock_En,Reset:  std_logic;
signal data_out,Data_in: std_logic_vector (7 downto 0);

Begin

dut: reg port map(Clock,Clock_En,Reset,data_in,data_out);

process
begin
    Clock <='0';
    wait for 10 ns;
    Clock <='1';
    wait for 10 ns;
end process;

process
begin

reset <= '1';
wait for 20 ns;
assert ( data_out ="00000000") report "error in reset" severity warning;
```

```
reset <='0';
data_in<="00000001";
clock_en<='1';
wait for 20 ns;
assert ( data_out ="00000001") report "error in  data 1" severity warning;

data_in<="00000010";
wait for 20 ns;
assert ( data_out ="00000010") report "error in data 2" severity warning;

data_in<="00000011";
wait for 20 ns;
assert ( data_out ="00000011") report "error in data 3" severity warning;

data_in<="00000100";
wait for 20 ns;
assert ( data_out ="00000100") report "error in data 4" severity warning;

data_in<="10000010";
wait for 20 ns;
assert ( data_out ="10000010") report "error in data 130" severity warning;

data_in<="00111010";
wait for 20 ns;
assert ( data_out ="00111010") report "error in data 58" severity warning;

clock_en<='0';
wait for 20 ns;
assert ( data_out ="00111010") report "error in preserving data 58" severity warning;

data_in<="11111001";
clock_en<='1';
wait for 20 ns;
assert ( data_out ="11111001") report "error in data 249" severity warning;

reset<='1';
wait for 20 ns;
assert ( data_out ="00000000") report "error in reset" severity warning;

end process;
end architecture ;
```

Testbench2

```
library ieee;
use ieee.std_logic_1164.all;
entity demux_test is
end entity;
architecture test_demux of demux_test is
component demux is
  generic (n: integer:=8);
 port(d_in:in std_logic_vector (n-1 downto 0);
 En : in std_logic;
 sel : in std_logic_vector(1 downto 0);
 d_out1:out std_logic_vector (n-1 downto 0);
 d_out2:out std_logic_vector (n-1 downto 0);
 d_out3:out std_logic_vector (n-1 downto 0);
 d_out4:out std_logic_vector (n-1 downto 0));
     end component;
     signal d_in,d_out1,d_out2,d_out3,d_out4: std_logic_vector(7 downto 0);
     signal en: std_logic;
     signal sel :std_logic_vector(1 downto 0);
     begin
     test : demux port map(d_in,en,sel,d_out1,d_out2,d_out3,d_out4);
     process is begin
     en<='1';
     d_in<="01100110";
     sel<="00";
     wait for 10 ns;
     assert(d_out1)<="01100110" report " sel 00 test" severity warning;
     assert(d_out2)<="00000000" report " sel 00 test" severity warning;
     assert(d_out3)<="00000000" report " sel 00 test" severity warning;
     assert(d_out4)<="00000000" report " sel 00 test" severity warning;
     sel<="01";
     wait for 10 ns;
     assert(d_out1)<="00000000" report "sel 01 test" severity warning;
     assert(d_out2)<="01100110" report " sel 01 test" severity warning;
     assert(d_out3)<="00000000" report " sel 01 test" severity warning;
     assert(d_out4)<="00000000" report " sel 01 test" severity warning;
         sel<="10";
         d_in<="00001111";
     wait for 10 ns;
     assert(d_out1)<="00000000" report " sel 10 test" severity warning;
     assert(d_out2)<="00000000" report " sel 10 test" severity warning;
     assert(d_out3)<="00001111" report " sel 10 test" severity warning;
```

```
    assert(d_out4)<="00000000" report " sel 10 test" severity warning;
        sel<="11";
        d_in<="11111111";
    wait for 10 ns;
    assert(d_out1)<="00000000" report " sel 11 test" severity warning;
    assert(d_out2)<="00000000" report " sel 11 test" severity warning;
    assert(d_out3)<="00000000" report " sel 11 test" severity warning;
    assert(d_out4)<="11111111" report " sel 11 test" severity warning;
         sel<="00";
        d_in<="11111111";
        en<='0';
    wait for 10 ns;
    assert(d_out1)<="00000000" report "enable test" severity warning;
    assert(d_out2)<="00000000" report "enable test" severity warning;
    assert(d_out3)<="00000000" report "enable test" severity warning;
    assert(d_out4)<="11111111" report "enable test" severity warning;
  end process;
end architecture;
```

Testbench3

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

entity ram_tb is
end entity;

Architecture behave of ram_tb is
component ram is
    port(wea,rea,clka,clkb: in std_logic ;
      addra,addrb: in std_logic_vector (2 downto 0);
        d_in: in std_logic_vector (7 downto 0);
        d_out: out std_logic_vector (7 downto 0));
end component;

signal wea,rea,clka,clkb: std_logic;
signal addra,addrb:  std_logic_vector (2 downto 0);
signal d_in,d_out:  std_logic_vector (7 downto 0);

Begin
```

```
br:  ram port map (wea,rea,clka,clkb,addra,addrb,d_in,d_out);

process
begin
    clka <='0';
    wait for 5 ns;
    clka <='1';
    wait for 15 ns;
end process;

process
begin
    clkb <='0';
    wait for 5 ns;
    clkb <='1';
    wait for 10 ns;
end process;

process
begin

d_in<="00001111";
addrb<="000";wea<='1';  rea<='0';
addra<="000";
wait for 20 ns;
 assert ( d_out ="XXXXXXXX") report "error 0" severity warning;

rea<='1';

wait for 20 ns;
 assert ( d_out ="00001111") report "error 1" severity warning;

d_in<="00001001";
addra<="001";
addrb<="011";
wait for 20 ns;

 assert ( d_out ="XXXXXXXX") report "error 2" severity warning;

addrb<="001";
wait for 20 ns;
 assert ( d_out ="00001001") report "error 3" severity warning;

d_in<="10001001";
addra<="101";
addrb<="101";
wait for 20 ns;
```

```
 assert ( d_out ="10001001") report "error 4" severity warning;

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

wea<='0';
d_in<="11111001";
addra<="111";
addrb<="111";
wait for 20 ns;
 assert ( d_out ="XXXXXXXX") report "error Write enable supposed to be off " severity w
arning;


wea<='1';
rea<='0';
d_in<="11111001";
addra<="111";
addrb<="111";
wait for 20 ns;
 assert ( d_out ="XXXXXXXX") report "error read enable supposed to be off " severity wa
rning;

rea<='1';
wait for 40 ns;
 assert ( d_out ="11111001") report "error read enable supposed to be on " severity war
ning;

wait;
end process;
end architecture;
```

Testbench4

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

ENTITY gc_tb IS
END entity;

ARCHITECTURE behave OF gc_tb IS

component GrayCounter IS
      GENERIC (N: integer := 4);
      PORT (clock, reset, En: IN std_logic;
            count_out: OUT std_logic_vector (N-1 DOWNTO 0));
END component;

  SIGNAL clock, reset, En:  std_logic;
  SIGNAL count_out :  std_logic_vector(3 DOWNTO 0);

BEGIN
  Comp: GrayCounter  PORT MAP (clock, reset, En, count_out);

 process
  BEGIN
    clock <= '1'; WAIT FOR 10 ns;
    clock <= '0'; WAIT FOR 10 ns;
  END process ;

 process
  BEGIN

reset<='1'; en<='1';
wait for 20 ns;
assert(count_out ="0000") report" error in reset" severity warning;

reset<='0';
wait for 20 ns;
assert(count_out ="0001") report" error in counting 0001" severity warning;

wait for 20 ns;
assert(count_out ="0011") report" error in counting 0011" severity warning;

wait for 20 ns;
assert(count_out ="0010") report" error in counting 0010" severity warning;

wait for 20 ns;
assert(count_out ="0110") report" error in counting 0110" severity warning;
```

```
wait for 20 ns;
assert(count_out ="0111") report" error in counting 0111" severity warning;

en<='0';
wait for 20 ns;
assert(count_out ="0111") report" error in turning enable off" severity warning;

reset<='1'; en<='1';
wait for 20 ns;
assert(count_out ="0000") report" error in reset v2" severity warning;

  WAIT;
 END process ;

end architecture;
```

Testbench5

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

entity gconv_tb is
end entity;

architecture arch of gconv_tb is

component GrayToBinary is
    port(gray_in:in std_logic_vector(3 downto 0);
             b_out:out std_logic_vector(3 downto 0)
    );
end component;

signal gray_in,b_out:std_logic_vector(3 downto 0);

begin
gtb:  GrayToBinary port map(gray_in,b_out);

process
begin
```

```
gray_in<="0000";
wait for 20 ns;
assert (b_out="0000") report"error converting to binary" severity warning;

gray_in<="1000";
wait for 20 ns;
assert (b_out="1111") report"error converting to binary" severity warning;

gray_in<="0100";
wait for 20 ns;
assert (b_out="0111") report"error converting to binary" severity warning;

gray_in<="0111";
wait for 20 ns;
assert (b_out="0101") report"error converting to binary" severity warning;

gray_in<="1100";
wait for 20 ns;
assert (b_out="1000") report"error converting to binary" severity warning;

gray_in<="1010";
wait for 20 ns;
assert (b_out="1100") report"error converting to binary" severity warning;

gray_in<="0001";
wait for 20 ns;
assert (b_out="0001") report"error converting to binary" severity warning;

gray_in<="0011";
wait for 20 ns;
assert (b_out="0010") report"error converting to binary" severity warning;

gray_in<="1001";
wait for 20 ns;
assert (b_out="1110") report"error converting to binary" severity warning;


wait;
end process;
end architecture;
```

Testbench6

```
library ieee;
```

```
use ieee.std_logic_1164.all;
entity fifo_c_test is
end entity;
architecture test of fifo_c_test is
  component FIFO_Controller is
  port  ( reset: in std_logic;
    rdclk: in std_logic;
    wrclk: in std_logic;
    rreq: in std_logic;
    wreq: in std_logic;
    write_valid: out std_logic;
    read_valid: out std_logic;
    wr_ptr: out std_logic_vector ( 2 downto 0 );
    rd_ptr: out std_logic_vector ( 2 downto 0);
    empty: out std_logic; --flag
    full: out std_logic --flag
  );
  end component;
  signal reset,rdclk,wrclk,rreq,wreq,write_valid,read_valid,empty,full : std_logic;
  signal wr_ptr,rd_ptr : std_logic_vector(2 downto 0);
  begin
    controller : FIFO_controller port map(reset,rdclk,wrclk,rreq,wreq,write_valid,read_
valid,wr_ptr,rd_ptr,empty,full);
   rclk: process is begin
   rdclk<='0','1' after 10 ns;
   wait for 20 ns;
   end process;
   wclk : process is begin
   wrclk<='0','1' after 10 ns;
   wait for 20 ns;
   end process;
   x: process is begin
      reset<='1';
      wait for 10 ns;-- after 10ns
      assert write_valid='1' report "write valid reset check" severity warning;
      assert read_valid='0' report "read valid reset check" severity warning;
      assert empty='1' report "empty reset check" severity warning;
      assert full='0' report "full reset check" severity warning;
      wait for 10 ns;
      rreq<='1';
      reset<='0';
      wait for 10 ns;
      assert read_valid='0' report "cant read before writing" severity warning;
      wait for 10 ns;
      reset<='0';
      wreq<='1';
      rreq<='1';
```

```
wait for 10 ns;-- after 30ns
assert write_valid='1' report "first write valid check" severity warning;
assert empty='0' report "first empty check" severity warning;
assert full='0' report "first full check" severity warning;
assert wr_ptr="000" report "first write pointer check" severity warning;
wait for 20 ns;-- after 50ns
assert write_valid='1' report "second write valid check" severity warning;
assert read_valid='1' report "second read valid check" severity warning;
assert empty='0' report "second empty check" severity warning;
assert full='0' report "second full check" severity warning;
assert wr_ptr="001" report "second write pointer check" severity warning;
assert rd_ptr="000" report "first read pointer check" severity warning;
wait for 20 ns;
assert write_valid='1' report "second write valid check" severity warning;
assert read_valid='1' report "second read valid check" severity warning;
assert empty='0' report "second empty check" severity warning;
assert full='0' report "second full check" severity warning;
assert wr_ptr="010" report "second write pointer check" severity warning;
assert rd_ptr="000" report "first read pointer check" severity warning;
wait for 20 ns;-- after 70ns
assert write_valid='1' report "third write valid check" severity warning;
assert read_valid='1' report "third read valid check" severity warning;
assert empty='0' report "third empty check" severity warning;
assert full='0' report "third full check" severity warning;
assert wr_ptr="011" report "third write pointer check" severity warning;
assert rd_ptr="001" report "third read pointer check" severity warning;
wait for 20 ns;-- after 90ns
assert write_valid='1' report "fourth write valid check" severity warning;
assert read_valid='1' report "fourth read valid check" severity warning;
assert empty='0' report "fourth empty check" severity warning;
assert full='0' report "fourth full check" severity warning;
assert wr_ptr="100" report "fourth write pointer check" severity warning;
assert rd_ptr="010" report "fourth read pointer check" severity warning;
wait for 20 ns;-- after 110ns
assert write_valid='1' report "fifth write valid check" severity warning;
assert read_valid='1' report "fifth read valid check" severity warning;
assert empty='0' report "fifth empty check" severity warning;
assert full='0' report "fifth full check" severity warning;
assert wr_ptr="101" report "fifth write pointer check" severity warning;
assert rd_ptr="011" report "fifth read pointer check" severity warning;
wait for 20 ns;-- after 130ns
assert write_valid='1' report "sixth write valid check" severity warning;
assert read_valid='1' report "sixth read valid check" severity warning;
assert empty='0' report "sixth empty check" severity warning;
assert full='0' report "sixth full check" severity warning;
assert wr_ptr="110" report "sixth write pointer check" severity warning;
assert rd_ptr="100" report "sixth read pointer check" severity warning;
```

```
      wait for 20 ns;-- after 150ns
      assert write_valid='1' report "seventh write valid check" severity warning;
      assert read_valid='1' report "seventh read valid check" severity warning;
      assert empty='0' report "seventh empty check" severity warning;
      assert full='0' report "seventh full check" severity warning;
      assert wr_ptr="111" report "seventh write pointer check" severity warning;
      assert rd_ptr="101" report "seventh read pointer check" severity warning;
      wreq<='0';
      wait for 40 ns;
      rreq<='0';
      wait for 20 ns;-- after 150ns
      assert write_valid='0' report "8th write valid check" severity warning;
      assert read_valid='0' report "8th read valid check" severity warning;
      assert empty='0' report "8th empty check" severity warning;
      assert full='1' report "8th full check" severity warning;
      assert wr_ptr="111" report "8th write pointer check" severity warning;
      assert rd_ptr="111" report "8th read pointer check" severity warning;
      wait for 10000 ns;
end process;
end architecture;
```

Testbench7

```
library ieee;
use ieee.std_logic_1164.all;
entity module_7_test is
end entity;
architecture test of module_7_test is
component  FIFO is
  port  ( reset: in std_logic;
    rdclk: in std_logic;
    wrclk: in std_logic;
    rreq: in std_logic;
    wreq: in std_logic;
    data_in : in std_logic_vector(7 downto 0);
    data_out: out std_logic_vector(7 downto 0);
    empty: out std_logic; --flag
    full: out std_logic --flag
  );
  end component;
  signal reset,rdclk,wrclk,rreq,wreq,empty,full: std_logic;
  signal data_in,data_out:std_logic_vector(7 downto 0);

  begin
```

```
    i1 : FIFO port map(reset,rdclk,wrclk,rreq,wreq,data_in,data_out,empty,full);
  rclk: process is begin
   rdclk<='0','1' after 10 ns;
   wait for 20 ns;
   end process;
   wclk : process is begin
   wrclk<='0','1' after 10 ns;
   wait for 20 ns;
   end process;

   process is begin
     reset<='1';
     rreq<='1';
     wreq<='1';
     data_in<="01010101";
     assert data_out<="UUUUUUUU" report "reset check" severity warning;
     wait for 20 ns;
     reset<='0';
     wait for 20 ns;
     assert data_out<="01010101" report "first data" severity warning;
     data_in<="10101010";
     wait for 20 ns;
     data_in<="00000000";
     wait for 20 ns;
     data_in<="11111110";
     wait for 20 ns;
     assert data_out<="10101010" report "second data" severity warning;
     data_in<="11111110";
     wait for 20 ns;
     assert data_out<="00000000" report "third data" severity warning;
     data_in<="11110110";
     wait for 20 ns;
     assert data_out<="11111110" report "fourth data" severity warning;
     data_in<="10111110";
     wait for 20 ns;
     wait for 20 ns;
     assert data_out<="11110110" report "fifth data" severity warning;
     wait for 20 ns;
     assert data_out<="10111110" report "sixth data" severity warning;
     wreq<='0';rreq<='0';
     data_in<="11001100";
     wait for 60 ns;
     assert data_out<="11110010" report "request check" severity warning;

wait;
   end process;
 end architecture;
```

Testbench8

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

entity rr_test is
end entity;

Architecture behave of rr_test is

component rr is
   port( clock:in std_logic;
    din1,din2,din3,din4:in std_logic_vector(7 downto 0);
    dout:out std_logic_vector(7 downto 0));
end component;

signal clock:std_logic;
signal din1,din2,din3,din4,dout: std_logic_vector(7 downto 0);

BEGIN

robin: rr port map(clock,din1,din2,din3,din4,dout);

process
 begin
  clock<='0';
  wait for 10 ns;
  clock<='1';
  wait for 10 ns;
end process;

process
begin

din1<= "00000001"; din2 <= "00000011"; din3<= "00001111"; din4 <="11110000";
wait for 21 ns;
assert(dout="00001111") report "error in s3" severity warning;

wait for 21 ns;
assert(dout="11110000") report "error in s4" severity warning;

wait for 21 ns;
assert(dout="00000001") report "error in s1" severity warning;
```

```
wait for 21 ns;
assert(dout="00000011") report "error in s2" severity warning;

wait for 21 ns;
assert(dout="00001111") report "error in s3 second time" severity warning;

wait;

end process;

end architecture;
```

Testbench10

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.ALL;
entity routertb is
end entity;
architecture behave of routertb is
component router is
port( rst, wclock, rclock: in std_logic;
wr1,wr2,wr3,wr4:in std_logic;
datai1,datai2,datai3,datai4:in std_logic_vector (7 downto 0);
datao1,datao2,datao3,datao4:out std_logic_vector (7 downto 0));
end component;
signal rst, wclock, rclock,wr1,wr2,wr3,wr4: std_logic;
signal datai1,datai2,datai3,datai4,datao1,datao2,datao3,datao4:std_logic_vector (7 down
to 0);
BEGIN
rout: router port map(rst, wclock, rclock,wr1,wr2,wr3,wr4,datai1,datai2,datai3,
datai4,datao1,datao2,datao3,datao4);
process
begin
rclock<='0'; wait for 5 ns;
rclock<='1'; wait for 5 ns;
end process;
process
begin ---- write clock cycle should be 4* read clock cycle
wclock <='0';
wait for 20 ns;
wclock <='1';
wait for 20 ns;
end process;
process
```

```
begin
rst<='1';
wait for 20 ns;
--reset check
assert datao1="UUUUUUUU" report "reset error" severity warning;
assert datao2="UUUUUUUU" report "reset error" severity warning;
assert datao3="UUUUUUUU" report "reset error" severity warning;
assert datao4="UUUUUUUU" report "reset error" severity warning;
wait for 20 ns;
rst<='0';
wr1<='0';wr2<='0';wr3<='0';wr4<='0';
datai1<="10000000";
datai2<="01010110";
datai3<="10101010";
datai4<="11001100";
wait for 40 ns;
--random input
wr1<='1';wr2<='1';wr3<='1';wr4<='1';
wait for 40 ns;
--random input
datai1<="00000011";
datai2<="01010101";
datai3<="10101011";
datai4<="11001101";
wait for 40 ns;
--test case for all inputs enter different scheduler
datai1<="01010100";
datai2<="00110001";
datai3<="10101010";
datai4<="00011111";
wait for 40 ns;
--test case for all inputs enter a single scheduler
datai1<="00011110";
datai2<="01011110";
datai3<="10011110";
datai4<="11011110";
wait for 30 ns;
--this assertion turning into true validates the wr test case
--the output is propagated after one extra cycle of write clock
assert datao1="10000000" report "1st error" severity warning;
assert datao2="00000000" report "1st error" severity warning;
assert datao3="00000000" report "1st error" severity warning;
assert datao4="00000000" report "1st error" severity warning;
wait for 10 ns;
assert datao3="01010110" report "2nd error" severity warning;
assert datao2="00000000" report "2nd error" severity warning;
assert datao1="00000000" report "2nd error" severity warning;
```

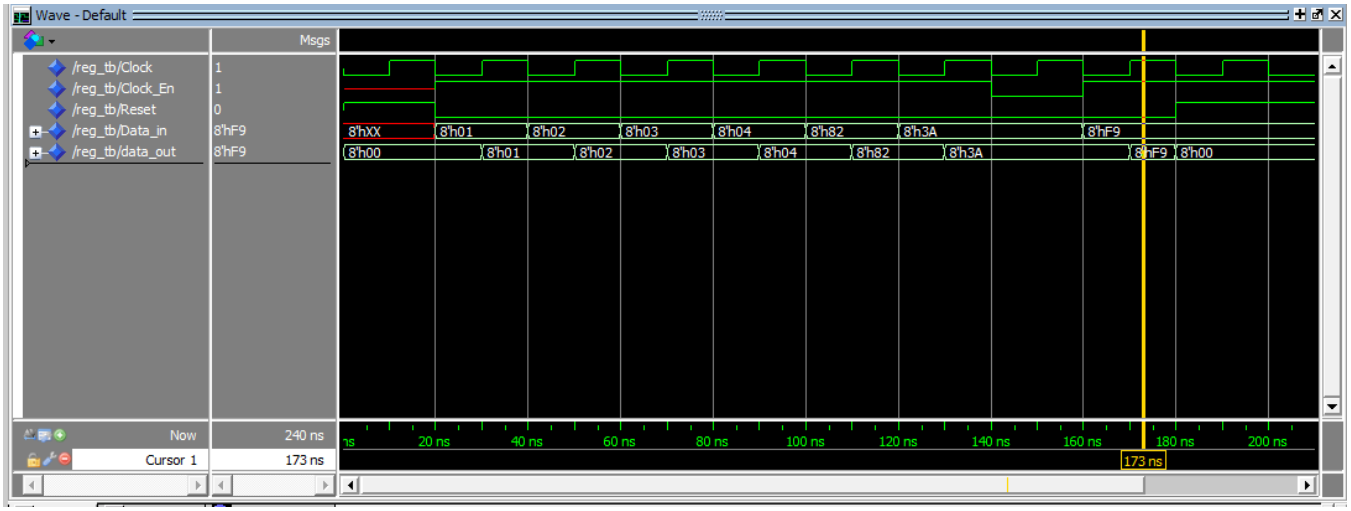
```
assert datao4="00000000" report "2nd error" severity warning;
wait for 10 ns;
assert datao3="10101010" report "3rd error" severity warning;
assert datao2="00000000" report "3rd error" severity warning;
assert datao1="00000000" report "3rd error" severity warning;
assert datao4="00000000" report "3rd error" severity warning;
wait for 10 ns;
assert datao1="11001100" report "4th error" severity warning;
assert datao2="00000000" report "4th error" severity warning;
assert datao3="00000000" report "4th error" severity warning;
assert datao4="00000000" report "4th error" severity warning;
wait for 10 ns;
assert datao4="00000011" report "5th error" severity warning;
assert datao2="00000000" report "5th error" severity warning;
assert datao1="00000000" report "5th error" severity warning;
assert datao3="00000000" report "5th error" severity warning;
wait for 10 ns;
assert datao2="01010101" report "6th error" severity warning;
assert datao3="00000000" report "6th error" severity warning;
assert datao1="00000000" report "6th error" severity warning;
assert datao4="00000000" report "6th error" severity warning;
wait for 10 ns;
assert datao4="10101011" report "7th error" severity warning;
assert datao2="00000000" report "7th error" severity warning;
assert datao1="00000000" report "7th error" severity warning;
assert datao3="00000000" report "7th error" severity warning;
wait for 10 ns;
assert datao2="11001101" report "8th error" severity warning;
assert datao3="00000000" report "8th error" severity warning;
assert datao1="00000000" report "8th error" severity warning;
assert datao4="00000000" report "8th error" severity warning;
wait for 10 ns;
assert datao1="01010100" report "9th error" severity warning;
assert datao2="00000000" report "9th error" severity warning;
assert datao3="00000000" report "9th error" severity warning;
assert datao4="00000000" report "9th error" severity warning;
wait for 10 ns;
assert datao2="00110001" report "10th error" severity warning;
assert datao3="00000000" report "10th error" severity warning;
assert datao1="00000000" report "10th error" severity warning;
assert datao4="00000000" report "10th error" severity warning;
wait for 10 ns;
assert datao3="10101010" report "11th error" severity warning;
assert datao2="00000000" report "11th error" severity warning;
assert datao1="00000000" report "11th error" severity warning;
assert datao4="00000000" report "11th error" severity warning;
wait for 10 ns;
```

```
assert datao4="00011111" report "12th error" severity warning;
assert datao2="00000000" report "12th error" severity warning;
assert datao1="00000000" report "12th error" severity warning;
assert datao3="00000000" report "12th error" severity warning;
wait for 10 ns;
assert datao3="00011110" report "13th error" severity warning;
assert datao2="00000000" report "13th error" severity warning;
assert datao1="00000000" report "13th error" severity warning;
assert datao4="00000000" report "13th error" severity warning;
wait for 10 ns;
assert datao3="01011110" report "14th error" severity warning;
assert datao2="00000000" report "14th error" severity warning;
assert datao1="00000000" report "14th error" severity warning;
assert datao4="00000000" report "14th error" severity warning;
wait for 10 ns;
assert datao3="10011110" report "15th error" severity warning;
assert datao2="00000000" report "15th error" severity warning;
assert datao1="00000000" report "15th error" severity warning;
assert datao4="00000000" report "15th error" severity warning;
wait for 10 ns;
assert datao3="11011110" report "16th error" severity warning;
assert datao2="00000000" report "16th error" severity warning;
assert datao1="00000000" report "16th error" severity warning;
assert datao4="00000000" report "16th error" severity warning;
wait;
end process;
end architecture;
```

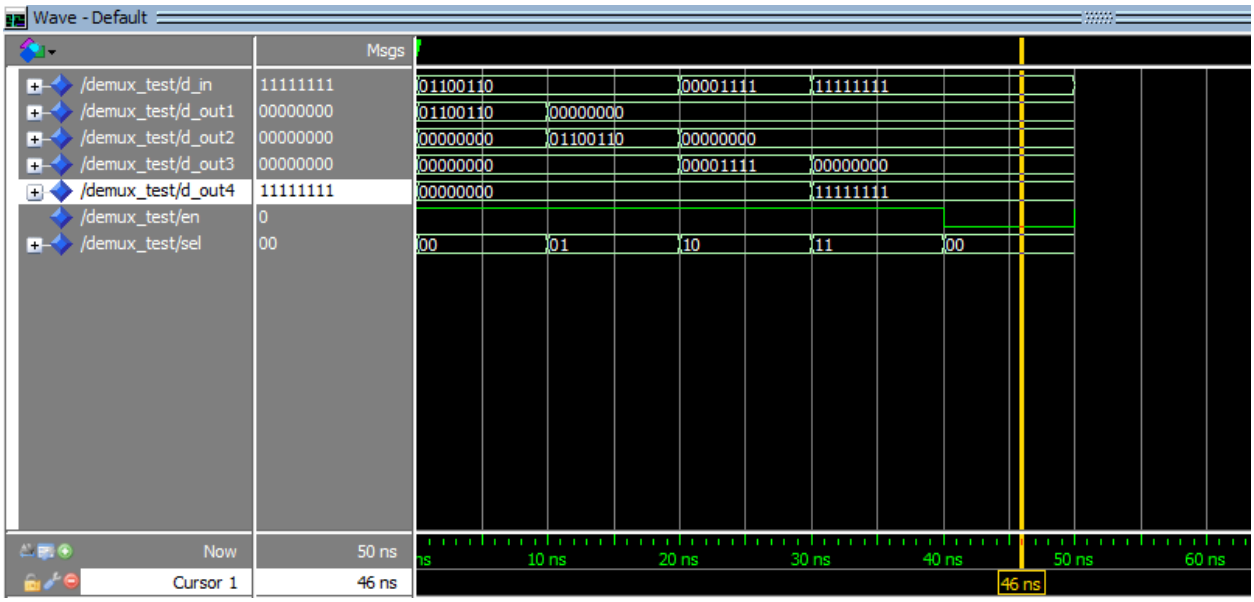
Appendix C Simulation Waveform Output

10 wave forms

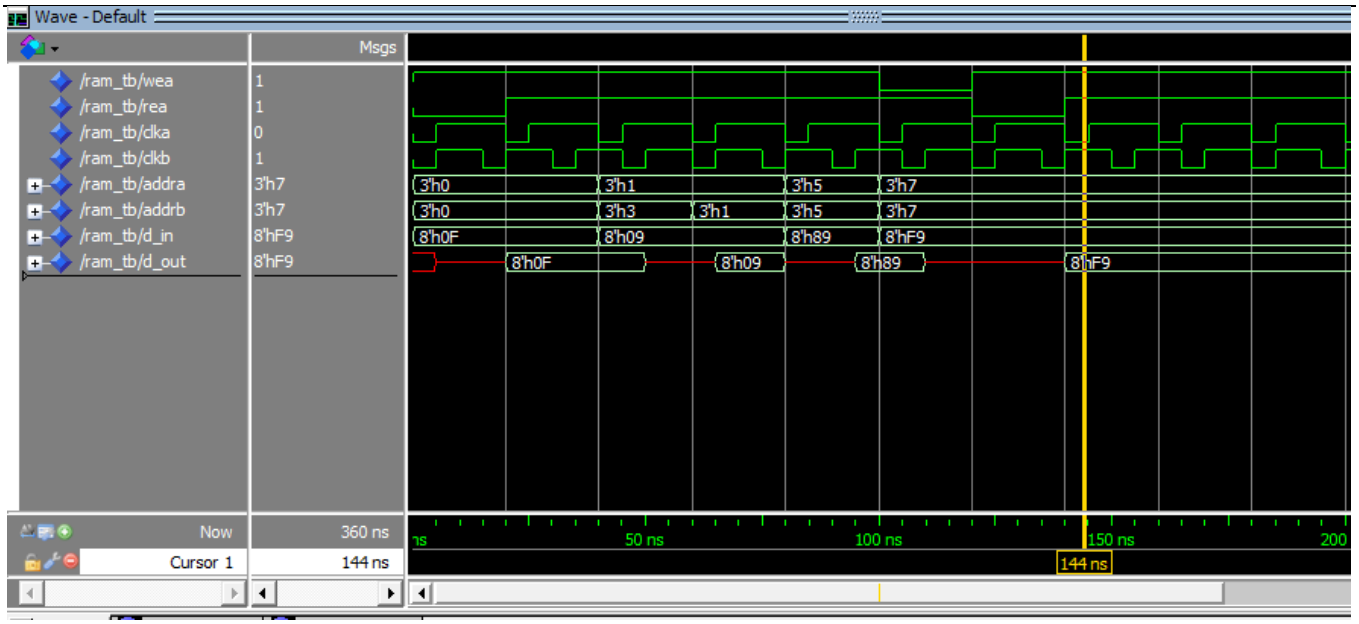
Module M-ROU-01



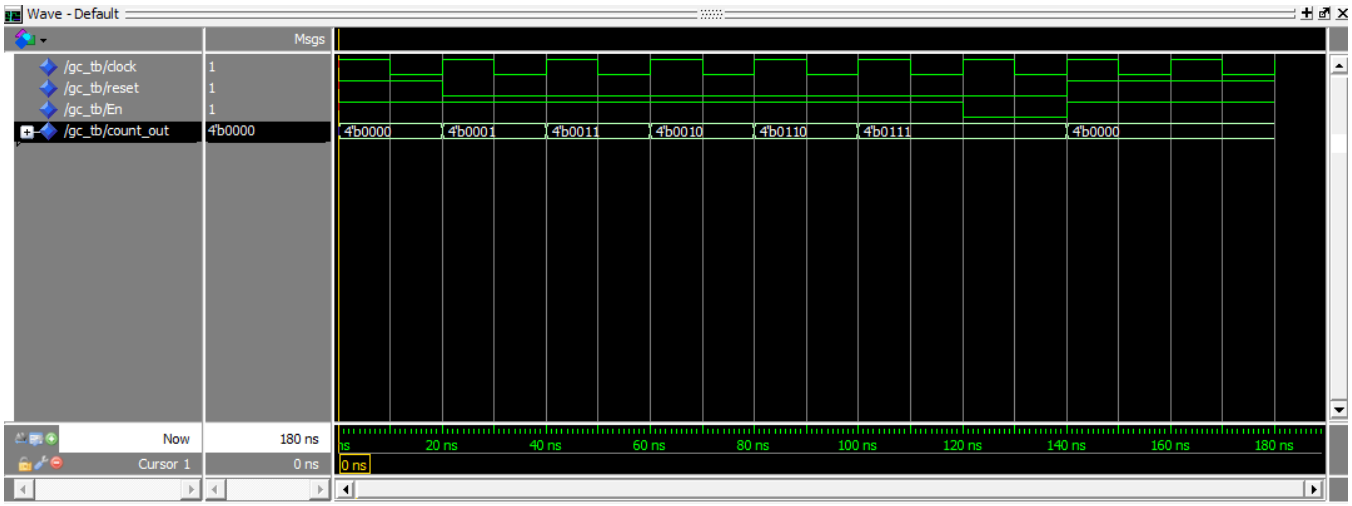
Module M-ROU-02



Module M-ROU-03

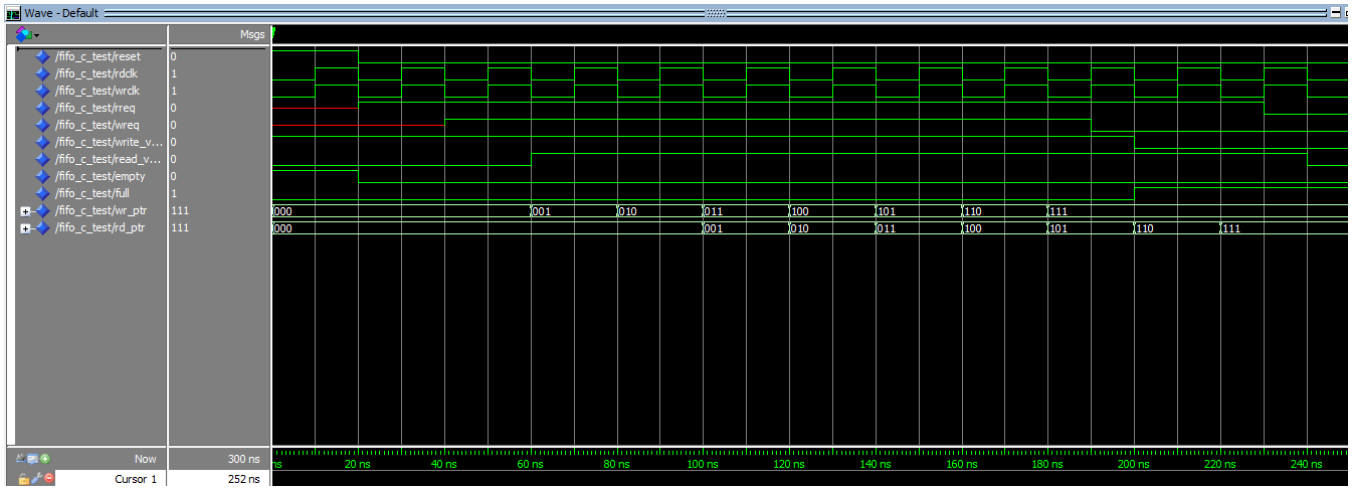


Module M-ROU-04

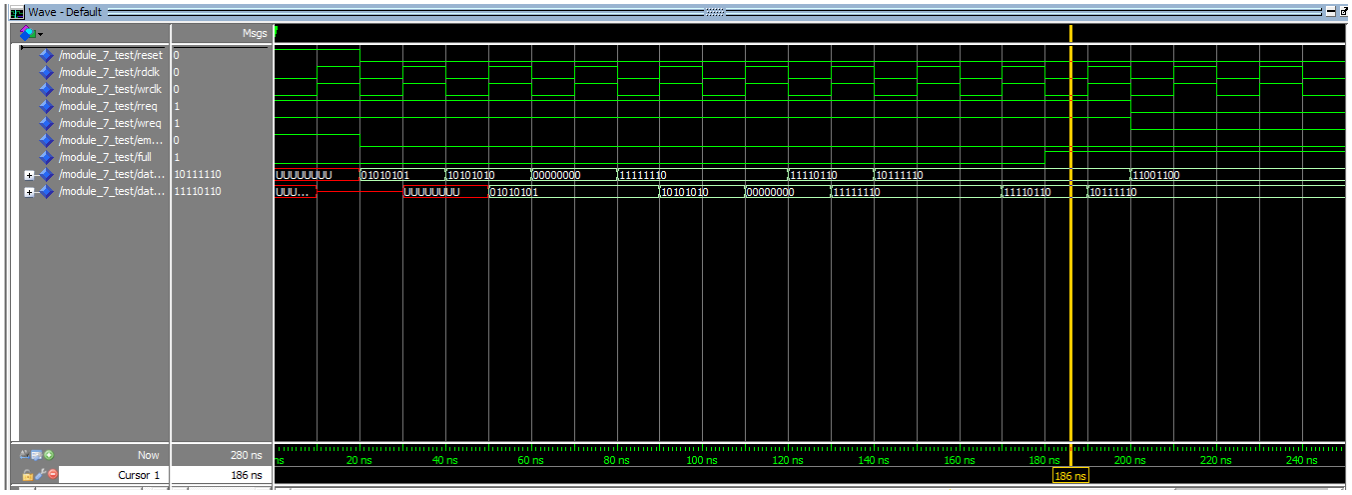


Module M-ROU-05

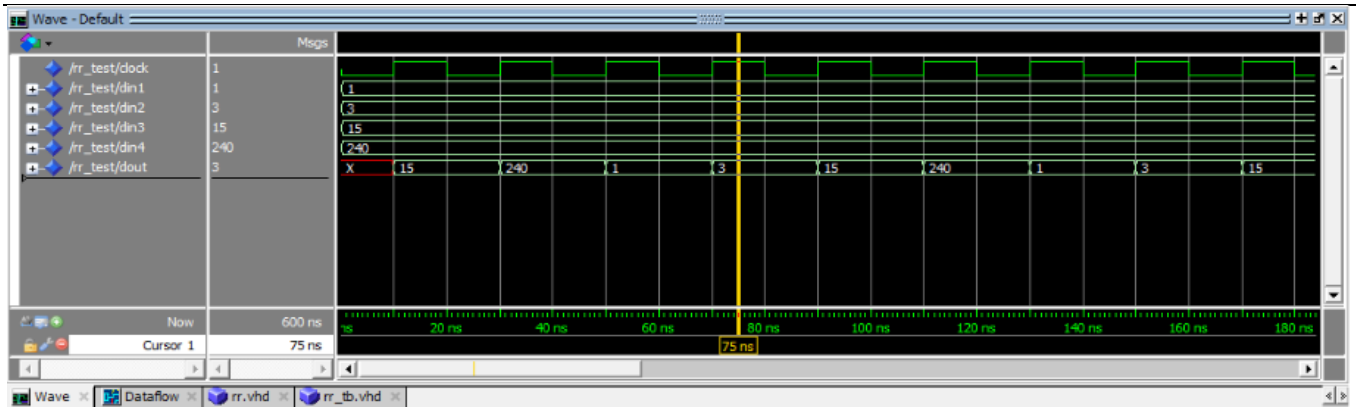
Module M-ROU-06



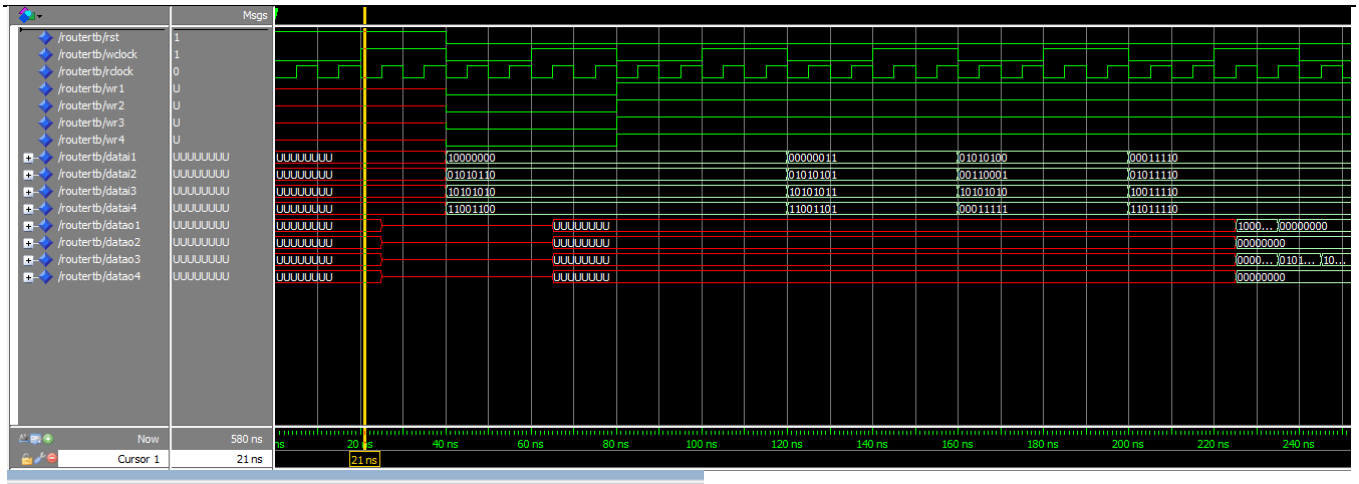
Module M-ROU-07

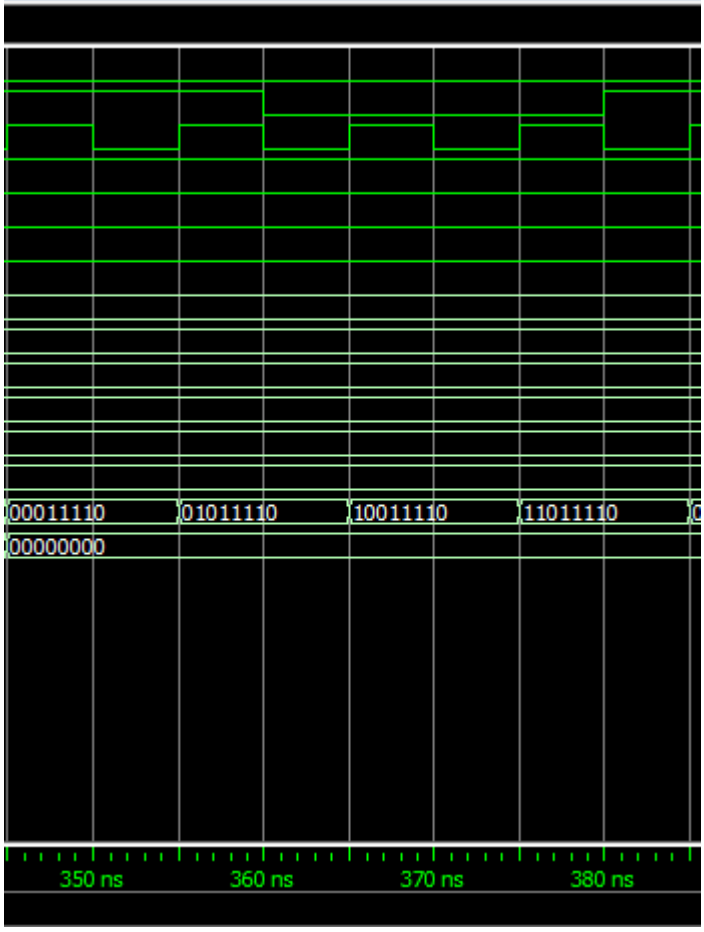


Module M-ROU-08



Module M-ROU-010


Msgs
/routertb/frst
/routertb/wclock
/routertb/rclock
/routertb/wr1
/routertb/wr2
/routertb/wr3
/routertb/wr4
/routertb/datai1
/routertb/datai2
/routertb/datai3
/routertb/datai4
/routertb/datao1
/routertb/datao2
/routertb/datao3
/routertb/datao4
10000000
00000011
01010100
00011110
01010110
01010101
00110001
01011110
10101010
10101011
10101010
10011110
11001100
11001101
00011111
11011110
00000000
Now
580 ns
Cursor 1
21 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns
160 ns
180 ns
200 ns
220 ns
240 ns


00011110
01011110
10011110
11011110
00000000
350 ns
360 ns
370 ns
380 ns