CSE 215 Electronic Design Automation

Research Project

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

Supervised by:

Dr. Haytham Azmi

Submitted by:

1. Ahmed Adel Mounir Abd El Latif :: 19P9647
2. Mostafa Lotfy Mostafa :: 18P6000
3. Mohamed Sameh Ahmed :: 17p3033
4. Shehab Mohamed Ibrahim :: 18P7213
5. Yusuf Sameh Fawzi :: 18P1399

Table of Contents

Introduction

1. Network on a chip (NoC)

A network on a chip (NOC) is the communication system used on a microchip and it is a network-based subsystem. The network on the microchip is router-based network that sends information between two or more SoC modules by packet switching method.

The network on a chip is a very efficient way of communication as it uses the methods and theories of computer network which made a notable improvement for communication on chip over the traditional crossbar and bus communication architectures. The improvement in the power efficiency and scalability of systems on chip that the NoCs made compared to the many other designs of communication subsystems made the NoC used widely such as in the computer's GUPs that is commonly used in video gaming and computer graphics and accelerating artificial intelligence.

2. Router

The router is the virtual or physical device that forwards information between computer networks in the form of data packets.

A router is connected to two or more lines of data from different internet protocol networks. The network address information in the packet header is read by the router and then the router uses the information in the routing policy to direct the packet to its next destination.

There are five types of routers:

1. Wired router
2. Edge router
3. Core router
4. Virtual router
5. Wireless router

Objectives

The aim of the project is to design, implementation and test a simple router using VHDL.

The project main objective is to design a synthesizable VHDL code for the router and its test bench using the design methodologies studied in EDA course this year.

Design Flow

1. Design

The first step in the design process is to the design the code in a machine-readable format, and to do so we will be going to use a CAD (computer aided design) tool, using the model sim free student version offered by mentor graphics.

Model sim is a CAD tool that supports many design entry methods such as schematic capture component netlist, HDL entry (VHDL, Verilog). In this project all the designs are written using VHDL language.

2. Simulation
   Once we are done with the design, the next step is to simulate this design to make sure that the design meets all the requirements specified for this project.

3. Synthesis
   In this step, our CAD tool will translate the VHDL design we wrote and deduce standard building blocks (such as: adders, latches, registers, ex.) to implement the design. Any delicate change in the VHDL description may result in the inferences in different hardware which may cause different system performance.

4. Implementation
   For implementation we will use another CAD tool called Xilinx which will take the design through three processes which are
   - **Translate:** process which will convert the netlist generated from the previous process (synthesis step) into a form specific to the target device.
   - **Map:** translates the standard building blocks into the specific resources available in the target hardware.
   - **Place and Route:** this process complete the map process by placing the specific resources allocated and interconnecting them as the design

5. Device configuration
   After verifying the design, then come to the last step where a binary hardware configuration file (bitstream) is generated, then this file will be downloaded on a FPGA.

Literature Review

We are in a global of a totally fast paced changing technology, a big zone of gadgets is included into just one chip. With this being made these devices that is integrated in a chip need to communicate between each other so they can send information and date that are essential for a proper working chip. Back in days the conventional communication used was the bus and the crossbar communication methods which was very slow and was not efficient enough in its power consumption which caused a lot of limitations to our designs. Then the NoC (network on chip) technology was introduced for communication between the different modules integrated on the chip, this new way of communication brought a notable improvement over the old traditional ways of communications. NoC improves system-on-chip scalability and power efficiency of complex SoCs compared to other communication subsystem design, NoC also guarantees flexibility, balance and a better bandwidth. This study targets on diverse implementations of Noc and distinctive approaches of enduring faults of communication among the number of the nodes and henceforth growing the velocity of processing and keep throughput.

By checking the previous research we note that the often primary blocks of the router are similar to the proposed module withinside the file of which it consists of specifically 3 parts (Registers and demultiplexers, First In First Out Registers, and Schedulers)

References

1. Neelakrishnan S. Design and implementation of NoC routers and their application to Prdt-based NoC's.
2. 2. Lotlikar S. Design, implementation and evaluation of a configurable NoC for AcENoCS FPGA accelerated emulation platform. [College Station, Tex.]: [Texas A & M University]; 2011.

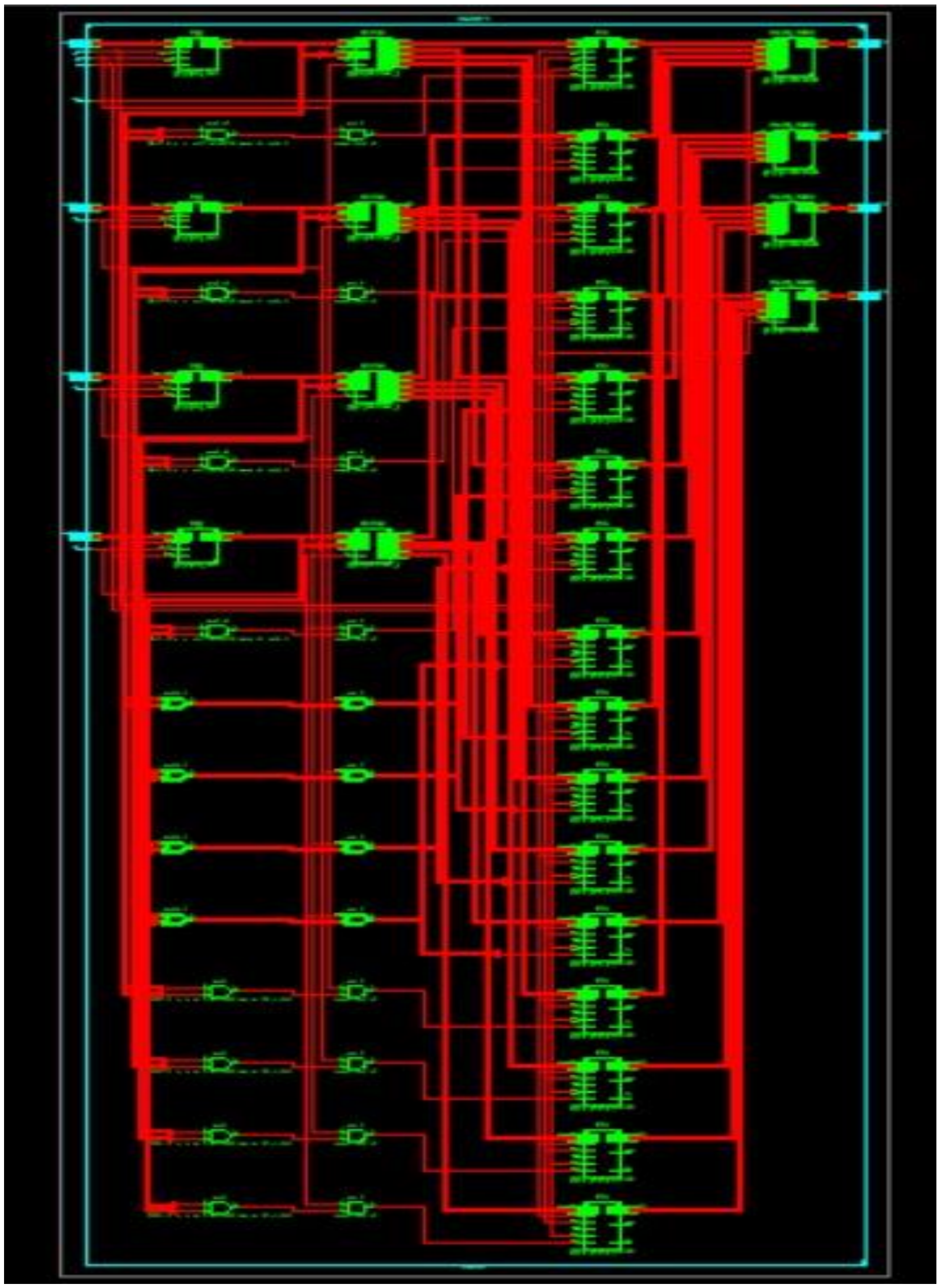
Design Implementation

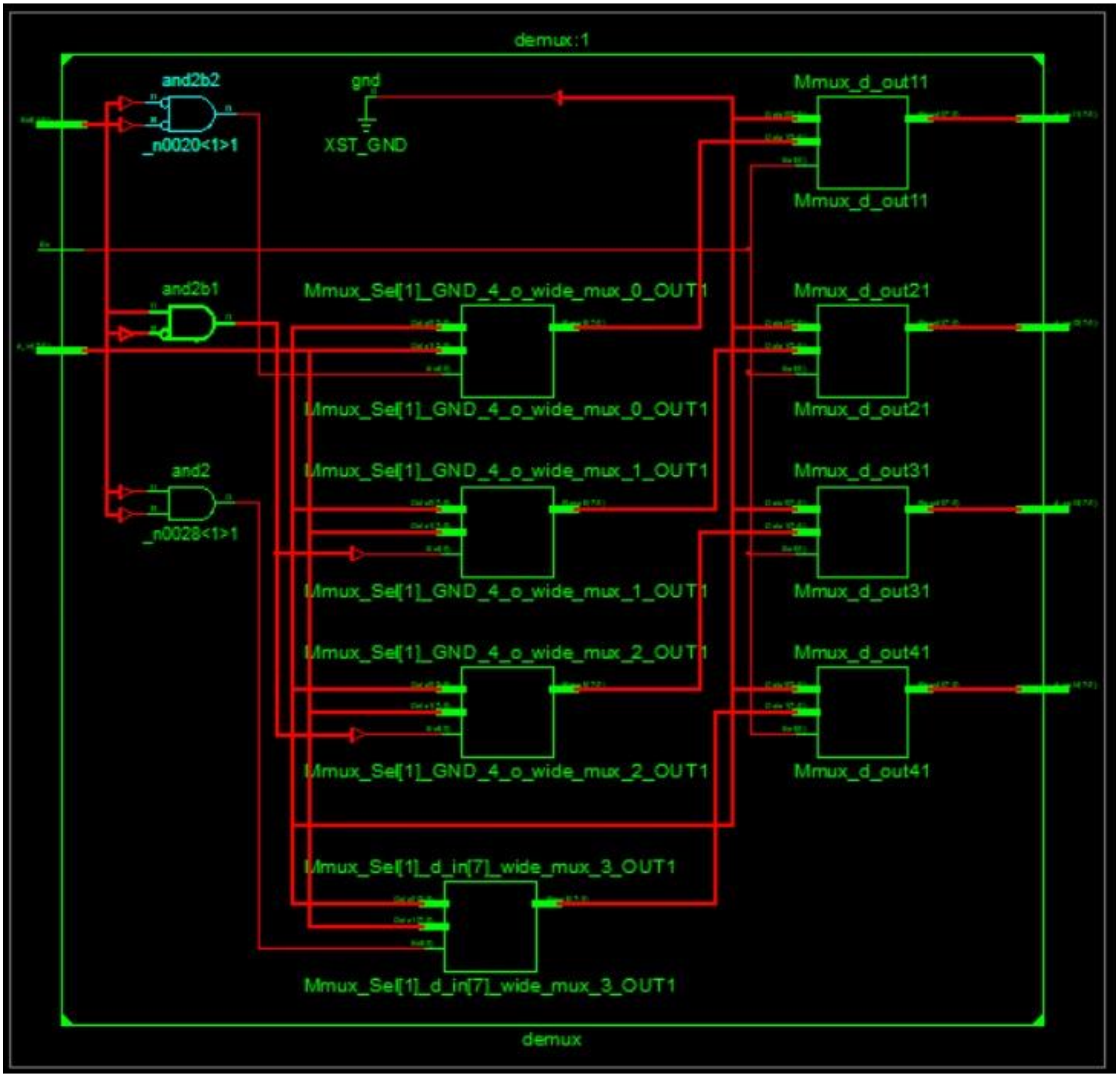
There are 9 modules of NoC router used in this project each of unique functionality and a defined purpose such as:

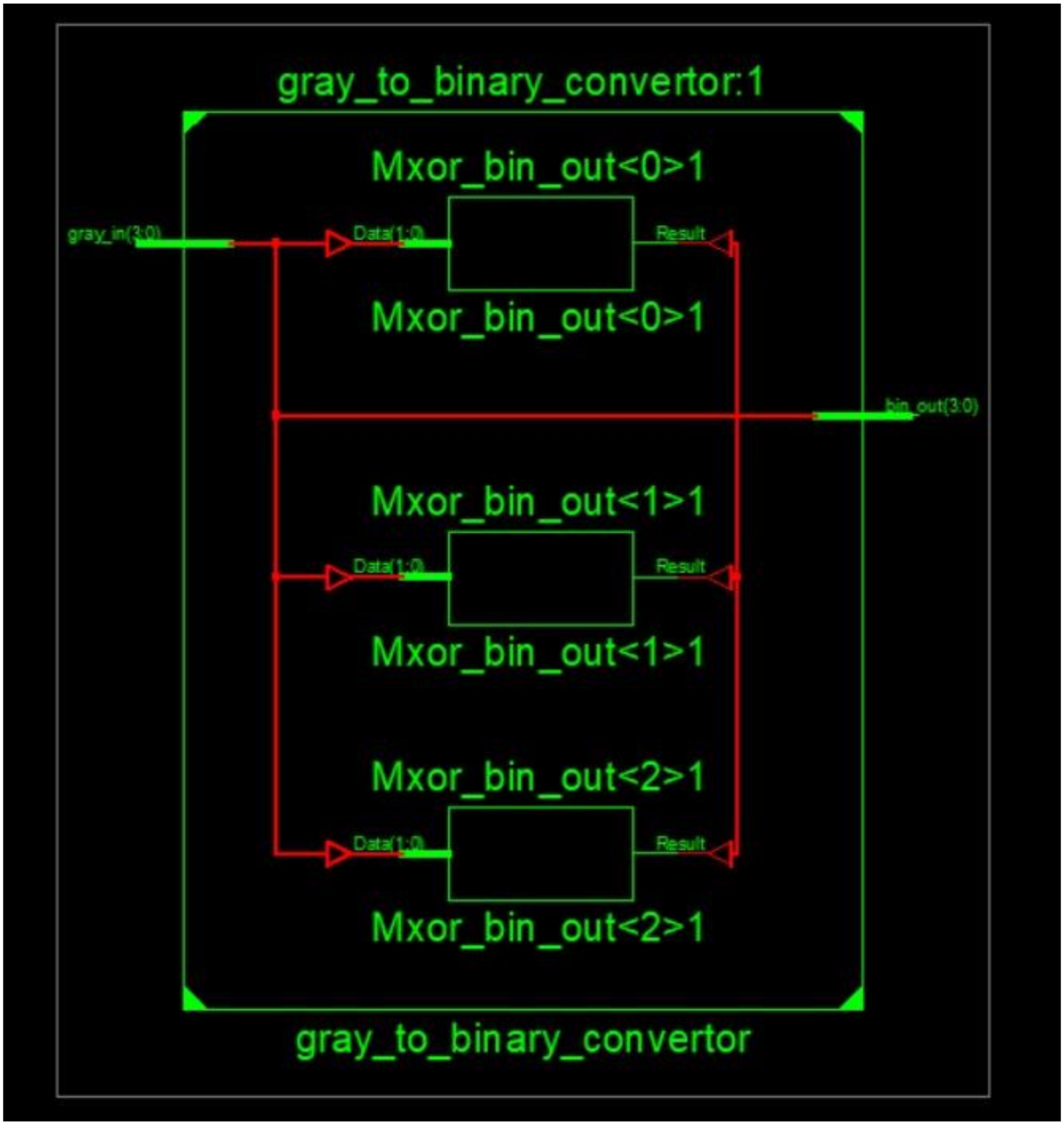
- 8 bit register
- 8 bit demultiplexer
- Block Ram
- Gray to Binary Converter
- FIFO Controller
- FIFO
- Round Robin schedular
- Router

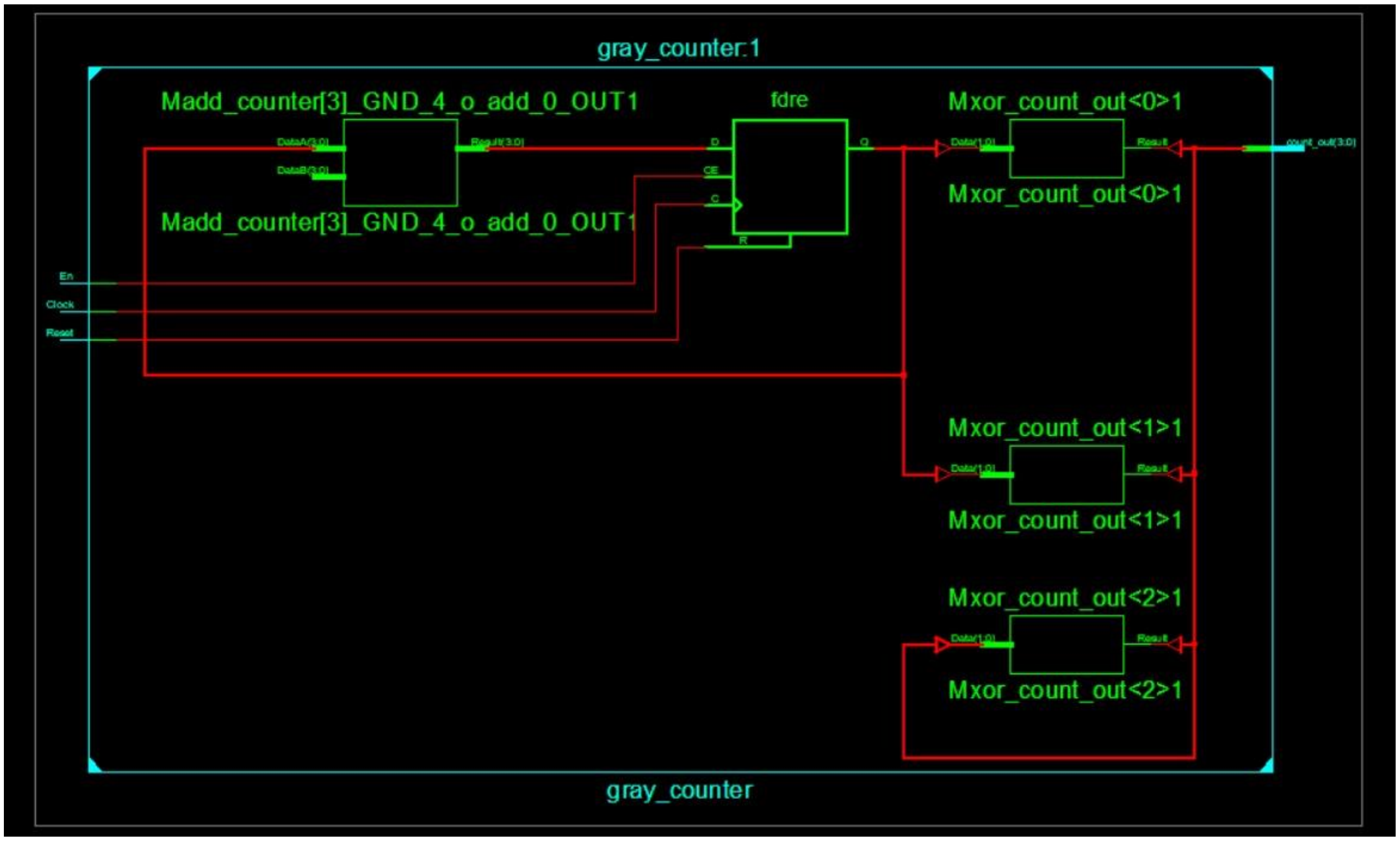
Module	Function
M_ROU_01: 8-bit Register	Store packet once arrived and propagates it
M_ROU_02: 4-1 8-bit DeMux	Switch fabric to provide connectivity between inputs and outputs
M_ROU_03: Block Ram	Save data; it contains an array to store data and controlled by the clock
M_ROU_04: Gray Counter	Counts in gray
M_ROU_05: Gray to Binary Converter	Convert gray code to Binary
M_ROU_06: FIFO controller	Validate read or write and check memory availability, read packet header and configure the switch fabric, synchronize router internal modules, control data flow inside the router, apply round robin schedular
M_ROU_07: FIFO	Work in First in First out sequence, it contains the controller and the memory
M_ROU_08: Round Robin Schedular	Set data in an organized queue, data passes only at clock cycle
M_ROU_09: Router	route the input data to the correct output port according to the address it gets

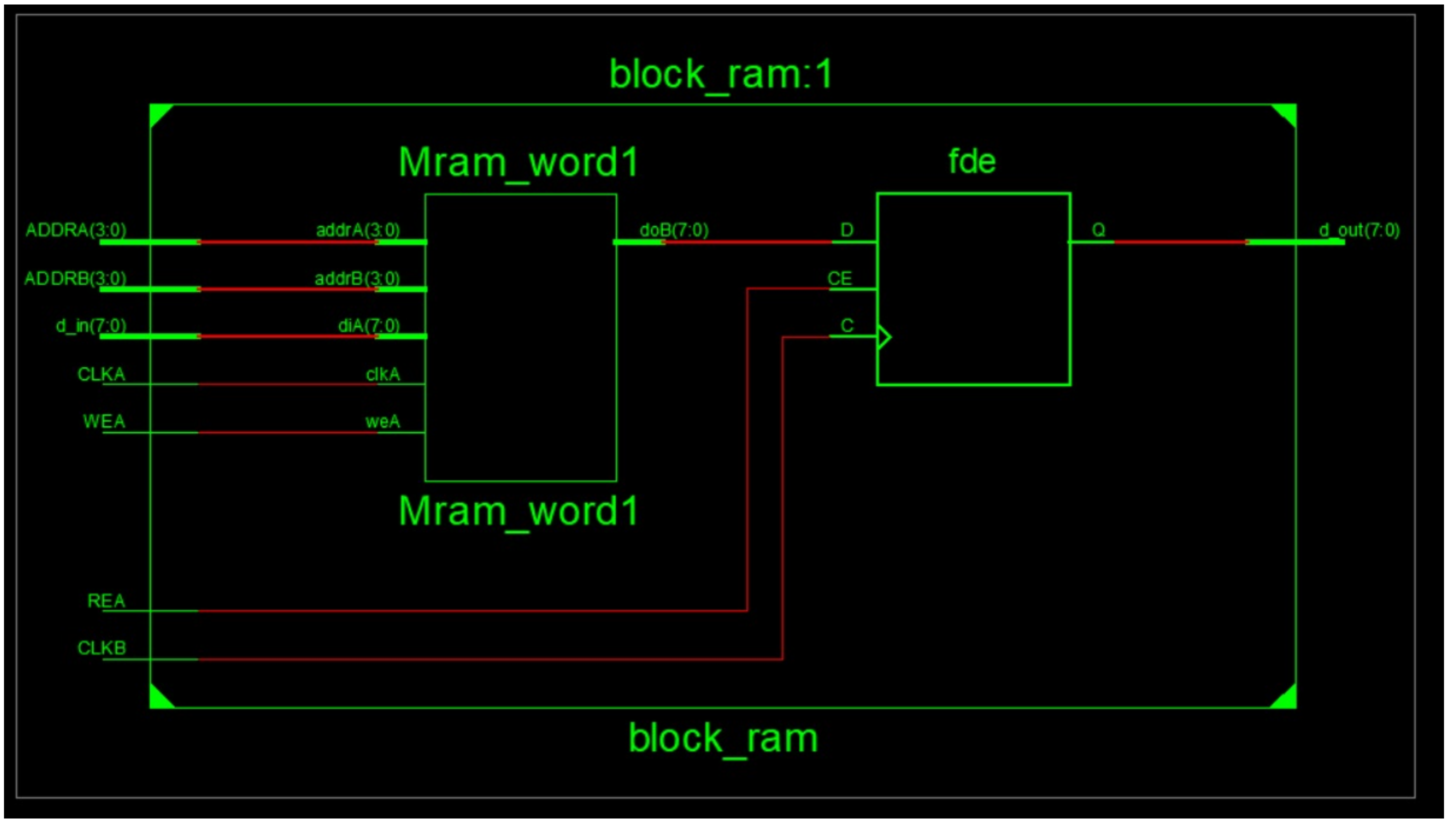
Block Diagram:

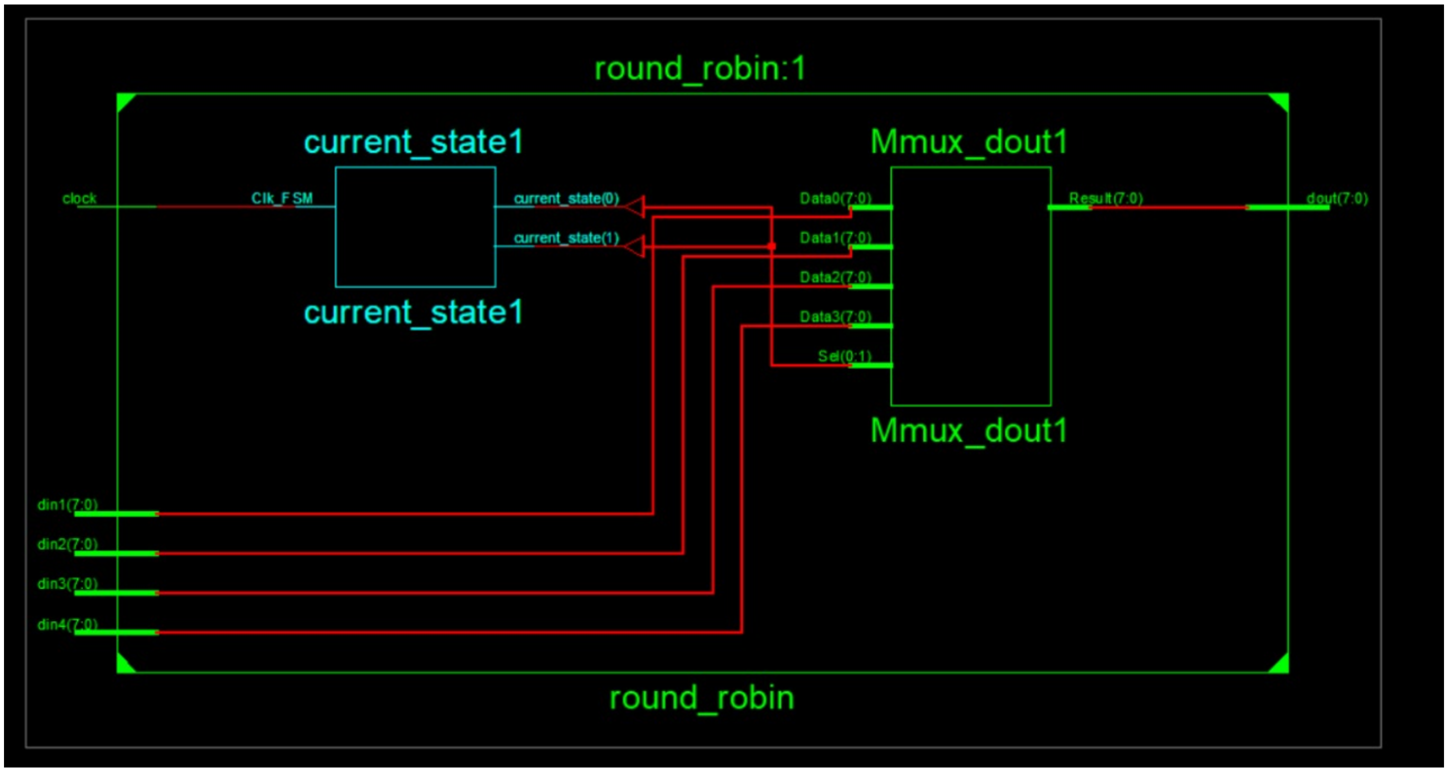


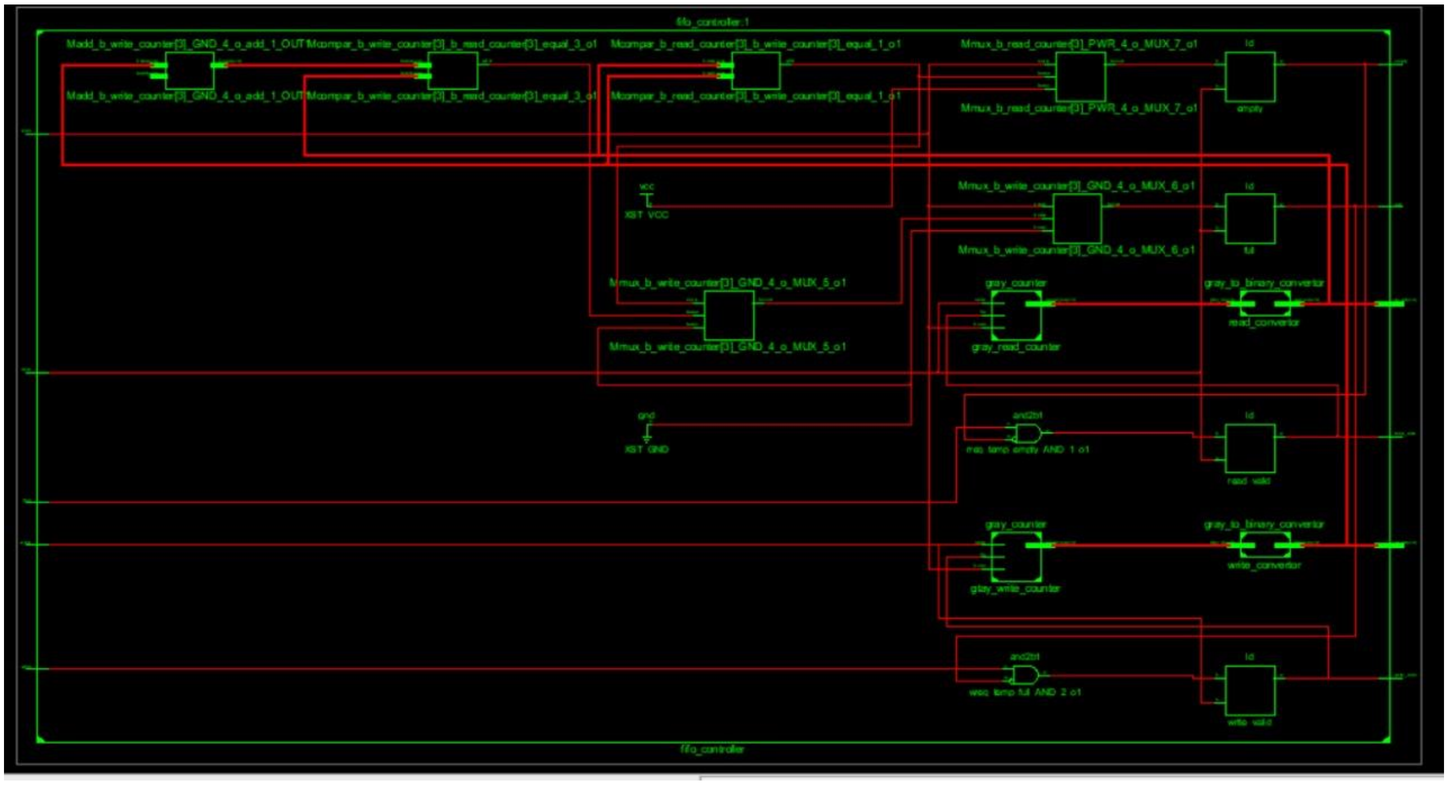

demux:1
and2b2
_n0020<1>1
gnd
XST_GND
Mmux_d_out11
Mmux_d_out11
and2b1
Mmux_Sel[1]_GND_4_o_wide_mux_0_OUT1
Mmux_Sel[1]_GND_4_o_wide_mux_0_OUT1
Mmux_d_out21
Mmux_d_out21
and2
_n0028<1>1
Mmux_Sel[1]_GND_4_o_wide_mux_1_OUT1
Mmux_Sel[1]_GND_4_o_wide_mux_1_OUT1
Mmux_d_out31
Mmux_d_out31
Mmux_Sel[1]_GND_4_o_wide_mux_2_OUT1
Mmux_Sel[1]_GND_4_o_wide_mux_2_OUT1
Mmux_d_out41
Mmux_d_out41
Mmux_Sel[1]_d_in[7]_wide_mux_3_OUT1
Mmux_Sel[1]_d_in[7]_wide_mux_3_OUT1
demux

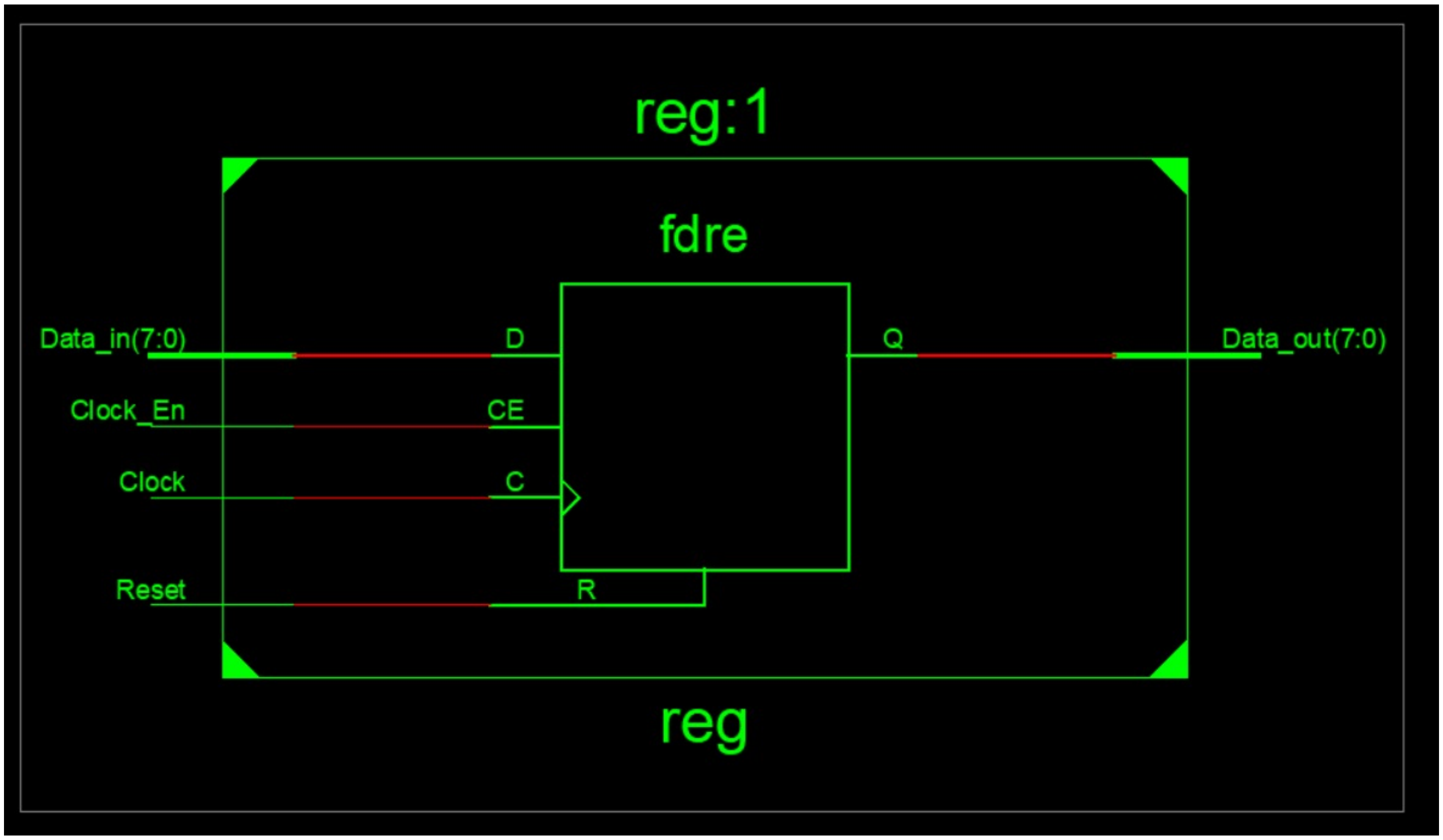

gray_to_binary_convertor:1
Mxor_bin_out<0>1
gray_in(3:0)
Data(1:0)
Result
Mxor_bin_out<0>1
bin_out(3:0)
Mxor_bin_out<1>1
Data(1:0)
Result
Mxor_bin_out<1>1
Mxor_bin_out<2>1
Data(1:0)
Result
Mxor_bin_out<2>1
gray_to_binary_convertor

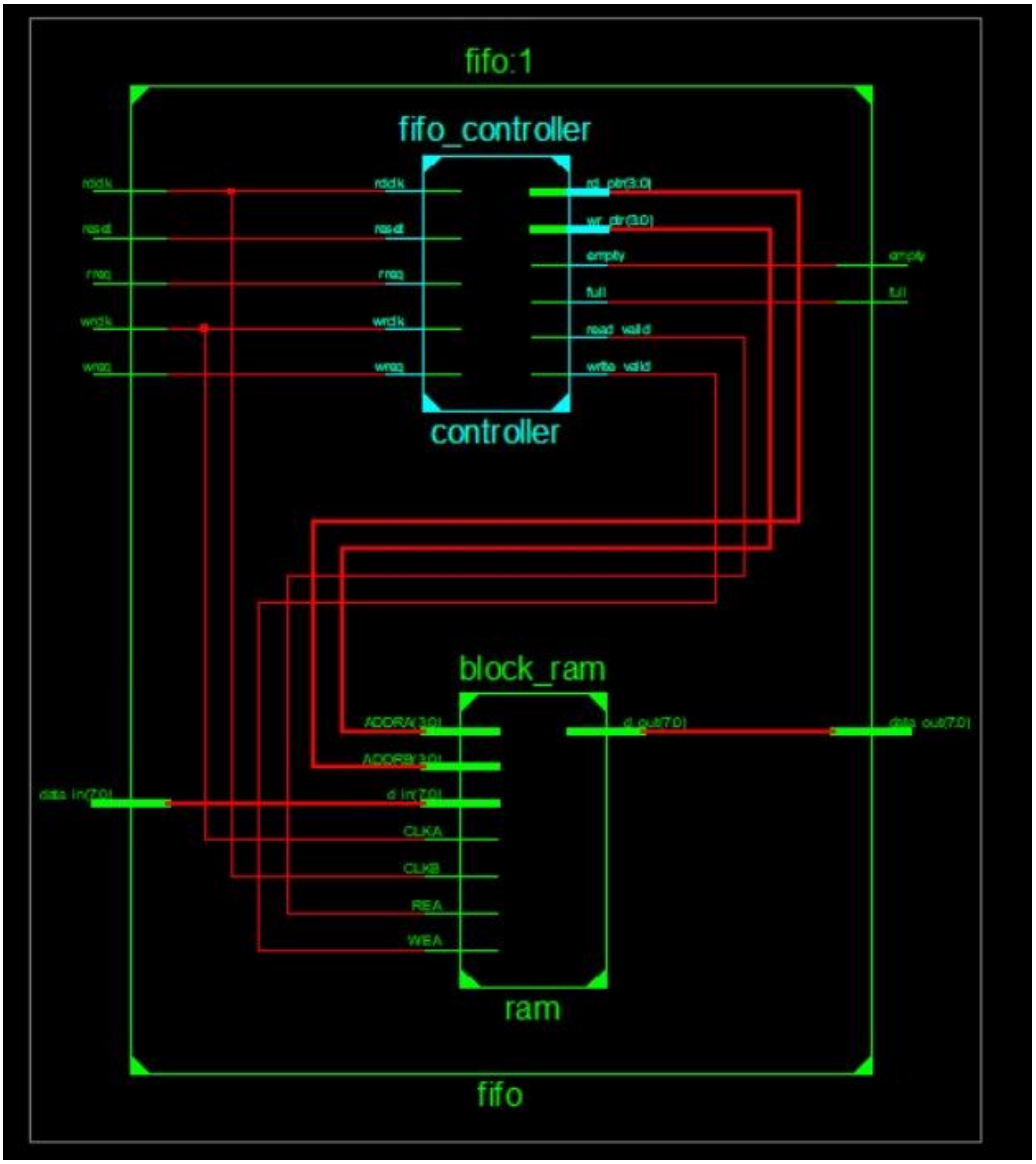

gray_counter:1
Madd_counter[3]_GND_4_o_add_0_OUT1
DataA(3:0)
DataB(3:0)
Result(3:0)
Madd_counter[3]_GND_4_o_add_0_OUT1
fdre
D
CE
C
R
Q
Mxor_count_out<0>1
Data(1:0)
Result
Mxor_count_out<0>1
count_out(3:0)
En
Clock
Reset
Mxor_count_out<1>1
Data(1:0)
Result
Mxor_count_out<1>1
Mxor_count_out<2>1
Data(1:0)
Result
Mxor_count_out<2>1
gray_counter


block_ram:1
Mram_word1
fde
ADDRA(3:0)
ADDRB(3:0)
d_in(7:0)
CLKA
WEA
addrA(3:0)
addrB(3:0)
diA(7:0)
clkA
weA
doB(7:0)
D
CE
C
Q
d_out(7:0)
Mram_word1
REA
CLKB
block_ram

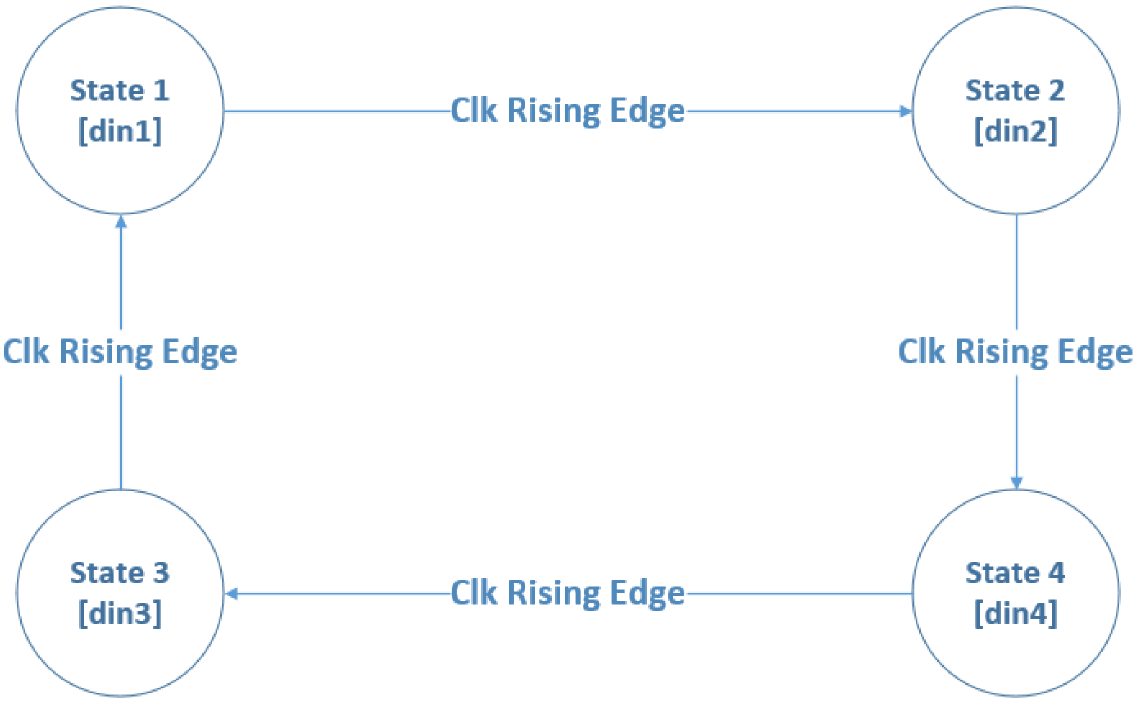

round_robin:1
current_state1
Mmux_dout1
clock
Clk_FSM
current_state(0)
current_state(1)
Data0(7:0)
Data1(7:0)
Data2(7:0)
Data3(7:0)
Sel(0:1)
Result(7:0)
dout(7:0)
current_state1
Mmux_dout1
din1(7:0)
din2(7:0)
din3(7:0)
din4(7:0)
round_robin


fifo_controller:1
Madd_b_write_counter[3]_GND_4_o_add_1_OUT
Mcompar_b_write_counter[3]_b_read_counter[3]_equal_3_o1
Mcompar_b_read_counter[3]_b_write_counter[3]_equal_1_o1
Mmux_b_read_counter[3]_PWR_4_o_MUX_7_o1
ld
empty
vcc
XST VCC
Mmux_b_write_counter[3]_GND_4_o_MUX_6_o1
ld
full
Mmux_b_write_counter[3]_GND_4_o_MUX_5_o1
gray_counter
gray_read_counter
gray_to_binary_convertor
read_convertor
gnd
XST GND
and2b1
rreq_temp_empty_AND_1_o1
ld
read_valid
gray_counter
gray_write_counter
gray_to_binary_convertor
write_convertor
and2b1
wreq_temp_full_AND_2_o1
ld
write_valid
fifo_controller


reg:1
fdre
Data_in(7:0)
Clock_En
Clock
Reset
D
CE
C
R
Q
Data_out(7:0)
reg


fifo:1
fifo_controller
rdclk
reset
rreq
wrclk
wreq
rd_ptr(3:0)
wr_ptr(3:0)
empty
full
read_valid
write_valid
controller
block_ram
ADDRA(3:0)
ADDRB(3:0)
d_in(7:0)
CLKA
CLKB
REA
WEA
d_out(7:0)
data_in(7:0)
data_out(7:0)
ram
fifo

Schedular Design and FSM Implementation



We implemented the FSM of the Round Robin scheduler using the Moore (state-based) coding technique, it consists of 2 processes.

The first process is for updating the current state with each clock cycle.

The second process is for updating the next state and the output based on the current state.

Since the output in the code depends only on the current state, this is a Moore FSM.

Moore (state-based) FSM	Mealy (transition-based) FSM
The output only depends on the current state.	The output depends on the current state in addition to the input.

	1 process	2 processes	3 processes
Mealy	Consists of only 1 process that is sensitive to: • Clock • Reset • Current state • Input	Consists of 2 processes, the first process is sensitive to: • Clock • Reset	Consists of 3 processes, the first process is sensitive to: • Clock • Reset

		The second process is sensitive to: • Current state • Input	The second process is sensitive to: • Current state • Input The third process is sensitive to: • Current state • Input
Moore	Consists of only 1 process that is sensitive to: • Clock • Reset • Current state	Consists of 2 processes, the first process is sensitive to: • Clock • Reset The second process is sensitive to: • Current state • Input	Consists of 3 processes, the first process is sensitive to: • Clock • Reset The second process is sensitive to: • Current state • Input The third process is sensitive to: • Current state

Test and Simulation Results

Tested Feature	Inputs					Expected Output			
	datai1	datai2	datai3	datai4	wr1,wr2, wr3,wr4	datao1	datao2	datao3	datao4
Routing form input port to correct output port	11001000	10010001	01010110	10001011	1	11001000	10010001	01010110	10001011
	11011011	10010000	01110101	10001110	1	10010000	01110101	10001110	11011011
	11011010	10110011	11110100	00001001	1	11110100	00001001	11011010	10110011
	11001101	11001010	01010111	10111100	1	10111100	11001101	11001010	01010111

Timing Report

```
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.


Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
rdclk                              | BUFGP                  | 242   |
wrclk                              | BUFGP                  | 160   |
-----------------------------------+------------------------+-------+


Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design


Timing Summary:
---------------
Speed Grade: -3
```

```
   Minimum period: 2.533ns (Maximum Frequency: 394.726MHz)
   Minimum input arrival time before clock: 1.256ns
   Maximum output required time after clock: 1.538ns
   Maximum combinational path delay: No path found

Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default period analysis for Clock 'rdclk'
  Clock period: 2.533ns (frequency: 394.726MHz)
  Total number of paths / destination ports: 1011 / 434
-------------------------------------------------------------------------
Delay:               1.267ns (Levels of Logic = 2)
  Source:            g2[0].g3[0].queue/controller/gray_read_counter/counter_2 (FF)
  Destination:       g2[0].g3[0].queue/controller/temp_full (LATCH)
  Source Clock:      rdclk rising
  Destination Clock: rdclk falling

  Data Path: g2[0].g3[0].queue/controller/gray_read_counter/counter_2 to g2[0].g3[0].queue/controller/temp_full
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDRE:C->Q             9   0.361   0.416  g2[0].g3[0].queue/controller/gray_read_counter/counter_2 (g2[0].g3[0].queue/controller/gray_read_counter/counter_2)
     LUT5:I3->O            1   0.097   0.295  g2[0].g3[0].queue/controller/Mmux_b_write_counter[3]_GND_11_o_MUX_17_o1_SW1 (N64)
     LUT6:I5->O            1   0.097   0.000  g2[0].g3[0].queue/controller/Mmux_b_write_counter[3]_GND_11_o_MUX_17_o1 (g2[0].g3[0].queue/controller/b_write_counter[3]_GND_11_o_MUX_17_o)
     LD:D                     -0.028          g2[0].g3[0].queue/controller/temp_full
    ----------------------------------------
    Total                      1.267ns (0.555ns logic, 0.712ns route)
```

```
                              (43.8% logic, 56.2% route)


=========================================================================
Timing constraint: Default period analysis for Clock 'wrclk'
  Clock period: 2.375ns (frequency: 421.124MHz)
  Total number of paths / destination ports: 816 / 496
-------------------------------------------------------------------------
Delay:               1.187ns (Levels of Logic = 0)
  Source:            g2[0].g3[0].queue/controller/write_signal (LATCH)
  Destination:       g2[0].g3[0].queue/ram/Mram_word1 (RAM)
  Source Clock:      wrclk falling
  Destination Clock: wrclk rising

  Data Path: g2[0].g3[0].queue/controller/write_signal to g2[0].g3[0].queue/ram/Mram_word1
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     LD:G->Q               7   0.472   0.307  g2[0].g3[0].queue/controller/write_signal
(g2[0].g3[0].queue/controller/write_signal)
     RAM32M:WE                 0.408          g2[0].g3[0].queue/ram/Mram_word1
    ----------------------------------------
    Total                      1.187ns (0.880ns logic, 0.307ns route)
                                       (74.1% logic, 25.9% route)


=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'wrclk'
  Total number of paths / destination ports: 288 / 288
-------------------------------------------------------------------------
Offset:              1.256ns (Levels of Logic = 2)
  Source:            wr1 (PAD)
  Destination:       g2[0].g3[0].queue/ram/Mram_word1 (RAM)
  Destination Clock: wrclk rising
  Data Path: wr1 to g2[0].g3[0].queue/ram/Mram_word1

                          Gate     Net

    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)

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

     IBUF:I->O           39   0.001   0.619  wr1_IBUF (wr1_IBUF)

     LUT4:I1->O           1   0.097   0.279  g1[0].demux_i/Mmux_d_out451 (demux_out<0><3><4>)

     RAM32M:DIC0              0.260          g2[0].g3[3].queue/ram/Mram_word1

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

    Total                      1.256ns (0.358ns logic, 0.898ns route)

                                  (28.5% logic, 71.5% route)


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

Timing constraint: Default OFFSET IN BEFORE for Clock 'rdclk'

  Total number of paths / destination ports: 96 / 96

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

Offset:              0.923ns (Levels of Logic = 2)

  Source:            rst (PAD)

  Destination:       g2[0].g3[0].queue/controller/temp_empty (LATCH)

  Destination Clock: rdclk falling


  Data Path: rst to g2[0].g3[0].queue/controller/temp_empty

                          Gate     Net

    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)

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

     IBUF:I->O          192   0.001   0.825  rst_IBUF (rst_IBUF)

     LUT6:I0->O           1   0.097   0.000  g2[0].g3[0].queue/controller/Mmux_b_read_counter[3]_PWR_11_o_MUX_18_o1
(g2[0].g3[0].queue/controller/b_read_counter[3]_PWR_11_o_MUX_18_o)

     LD:D                    -0.028          g2[0].g3[0].queue/controller/temp_empty

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

    Total                      0.923ns (0.098ns logic, 0.825ns route)

                                  (10.6% logic, 89.4% route)
```

```
=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'rdclk'
  Total number of paths / destination ports: 192 / 32
-------------------------------------------------------------------------
Offset:              1.538ns (Levels of Logic = 2)
  Source:            g1[3].scheduler/current_state_FSM_FFd2 (FF)
  Destination:       datao1<7> (PAD)
  Source Clock:      rdclk rising

  Data Path: g1[3].scheduler/current_state_FSM_FFd2 to datao1<7>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FD:C->Q              34   0.361   0.800  g1[3].scheduler/current_state_FSM_FFd2
(g1[3].scheduler/current_state_FSM_FFd2)
     LUT6:I0->O            1   0.097   0.279  g1[0].scheduler/Mmux_dout11 (datao1_0_OBUF)
     OBUF:I->O                 0.000          datao1_0_OBUF (datao1<0>)
    ----------------------------------------
    Total                      1.538ns (0.458ns logic, 1.080ns route)
                                       (29.8% logic, 70.2% route)

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

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

Clock to Setup on destination clock rdclk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
rdclk          |    1.273|    0.897|    1.267|         |
wrclk          |    1.656|         |    1.818|         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock wrclk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
rdclk          |    0.706|         |    1.125|         |
wrclk          |    1.660|    1.187|    0.944|         |
---------------+---------+---------+---------+---------+

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


Total REAL time to Xst completion: 11.00 secs
Total CPU time to Xst completion: 10.66 secs

-->

Total memory usage is 4617572 kilobytes

Number of errors   :    0 (   0 filtered)
Number of warnings :    4 (   0 filtered)
Number of infos    :   35 (   0 filtered)
```

Conclusion

NoC router (network on chip router) is used for communication and networking between two or more computer networks and data is sent using the packet switching method. The NoC is very fast for communicating two or more different networks together at a very high speed and very low power consumption compared to other communication designs. We used two CAD tools for all the process done in this project (Model sim, Xilinx).

There were a lot of challenges in the implementation process as there were so many components depending on each other and the connection between them was not that easy as well, but we managed our way through to figure it out.

There were 9 modules in our design each single module of them has its own function. Demux was responsible for the data selection that for implemented using switch cases, Registers was implemented using the if conditions and it was responsible for data transferring, Block Ram in our router was the memory element, FIFO was responsible for mapping the controller with the Block Ram for controlling the flow, Gray to Binary converter was responsible for converting gray to binary code using Xor gates. Finally, our router was implemented by directly mapping all the used modules together.

Task Distribution List

The project workload is distributed equally among all the team members

Appendix A

Module No. M-ROU-01

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

ENTITY reg IS
port(Data_in : IN std_logic_vector (7 downto 0);
Clock : IN std_logic;
Data_out : OUT std_logic_vector (7 downto 0);
Clock_En : IN std_logic;
Reset : IN std_logic);
end;


ARCHITECTURE behave OF reg IS
BEGIN
process(Clock,Clock_En,Reset) IS
BEGIN
   IF Clock'event and Clock = '1' then
      IF Reset = '1' THEN
         Data_Out <= (others =>'0') ;
      ELSIF Clock_En = '1' THEN
         Data_out <= Data_in;
      END IF;
   END IF;
 END PROCESS;
END;
```

Module No. M-ROU-02

```
library ieee;

use ieee.std_logic_1164.ALL;

use ieee.numeric_std.ALL;


ENTITY demux IS

PORT (Sel: IN std_logic_vector (1 downto 0);

En : IN std_logic;

d_in : IN std_logic_vector ( 7 downto 0);

d_out1,d_out2,d_out3,d_out4 : OUT std_logic_vector (7 downto 0));

END;


ARCHITECTURE behave OF demux IS

BEGIN

PROCESS (Sel,En, d_in) IS

BEGIN

        IF EN = '1' THEN

                case (sel) is

                        when "00" => d_out1 <= d_in; d_out2 <= (others => '0');d_out3 <= (others => '0');d_out4 <=
(others => '0');

                        when "01" => d_out2 <= d_in; d_out1 <= (others => '0');d_out3 <= (others => '0');d_out4 <=
(others => '0');

                        when "10" => d_out3 <= d_in; d_out2 <= (others => '0');d_out1 <= (others => '0');d_out4 <=
(others => '0');

                        when "11" => d_out4 <= d_in; d_out2 <= (others => '0');d_out3 <= (others => '0');d_out1 <=
(others => '0');

                        when others => d_out1 <= (others => '0'); d_out2 <= (others => '0');d_out3 <= (others =>
'0');d_out4 <= (others => '0');

                end case;

        Else

                d_out1 <= (others => '0');d_out2 <= (others => '0');d_out3 <= (others => '0');d_out4 <= (others => '0');

        END IF;
END PROCESS;
END;
```

Module No. M-ROU-03

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.ALL;

library ieee;
use ieee.std_logic_1164.ALL;
use ieee.STD_LOGIC_unsigned.all;

ENTITY block_ram IS
  PORT(d_in : IN std_logic_vector (7 downto 0);
  ADDRA,ADDRB :IN std_logic_vector (3 downto 0);
  WEA,REA,CLKA,CLKB :IN std_logic;
  d_out : OUT std_logic_vector (7 downto 0));
END ENTITY block_ram;

ARCHITECTURE behave OF block_ram IS
  TYPE wrm IS ARRAY (0 to 31) OF std_logic_vector (7 downto 0);
  SIGNAl word : wrm;
BEGIN
  wm : PROCESS (WEA,ADDRA,CLKA,d_in) IS
  BEGIN
    IF CLKA'event and CLKA = '1' AND WEA = '1' THEN
      word(conv_integer(ADDRA)) <= d_in;
    END IF;
  END PROCESS wm;

  rm : PROCESS (REA,ADDRB,CLKB,d_in) IS
  BEGIN
    IF CLKB'event and CLKB = '1' Then
```

```
      IF REA = '1' THEN
        d_out <= word(conv_integer(ADDRB));
      end if;
    end if;
  END PROCESS rm;
END;
```

Module No. M-ROU-04

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity gray_counter is
  port
  (
        Clock, Reset, En: in std_logic;
        count_out: out std_logic_vector(3 downto 0)
  );
end entity gray_counter;

architecture behav of gray_counter is
  signal counter: unsigned (3 downto 0);
begin
  clock_tick: process(Reset, clock, En) is
        begin
              if Clock'event and Clock = '1' then
                    if Reset = '1' then
                                counter <= "0000";
                    elsif En = '1' then
                          counter <= counter + 1;
                    end if;
              end if;
```

```
            end process clock_tick;

            count_out(3) <= counter(3);

            count_out(2) <= counter (2) xor counter(3);

            count_out(1) <= counter (1) xor counter(2);

            count_out(0) <= counter (0) xor counter(1);

end architecture;
```

Module No. M-ROU-05

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity gray_to_binary_convertor is

Port ( gray_in : in STD_LOGIC_VECTOR (3 downto 0);

      bin_out : out STD_LOGIC_VECTOR (3 downto 0)

    );

end gray_to_binary_convertor;


architecture behav of gray_to_binary_convertor is

begin

   p1:process (gray_in) 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 process p1;

end behav;
```

Module No. M-ROU-06

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity fifo_controller is
   port (
            reset, rdclk, wrclk, rreq, wreq: in std_logic ;
            write_valid, read_valid: out std_logic;
            wr_ptr, rd_ptr: out std_logic_vector (3 downto 0);
            empty, full: out std_logic
          );
end entity fifo_controller;
architecture behav of fifo_controller is
   component gray_counter is
         port
         (
            Clock, Reset, En: in std_logic;
            count_out: out std_logic_vector(3 downto 0)
         );
   end component gray_counter;
   for all: gray_counter use entity work.gray_counter(behav);
   component gray_to_binary_convertor is
         Port ( gray_in : in STD_LOGIC_VECTOR (3 downto 0);
                bin_out : out STD_LOGIC_VECTOR (3 downto 0)
              );
   end component gray_to_binary_convertor;
         for all: gray_to_binary_convertor use entity work.gray_to_binary_convertor(behav);
   -- gray counters signals
   signal read_counter, write_counter: std_logic_vector(3 downto 0);
   -- binary counters signals
   signal b_read_counter, b_write_counter: std_logic_vector(3 downto 0);
   -- signals for write_valid and read_valid
   signal read_signal, write_signal: std_logic;
         signal temp_empty, temp_full :std_logic;
begin
```

```
gray_read_counter: gray_counter port map  (rdclk, reset, read_signal, read_counter);
gtay_write_counter: gray_counter port map (wrclk, reset, write_signal, write_counter);


read_convertor: gray_to_binary_convertor port map (read_counter, b_read_counter);
        write_convertor: gray_to_binary_convertor port map (write_counter, b_write_counter);


        empty_full: process (rdclk,wrclk, reset, b_read_counter, b_write_counter) is
        begin
if rdclk = '1' then
                if reset = '1' then
                        temp_full <= '0';
                        temp_empty <= '1';
                        elsif b_write_counter = b_read_counter then
                                temp_full  <= '0';
                                temp_empty <= '1';
                        elsif unsigned(b_write_counter) + 1 = unsigned(b_read_counter) then
                                temp_full <= '1';
                                temp_empty <= '0';
                        else
                        temp_empty <= '0';
                        temp_full  <= '0';
                end if;
        end if;
        end process;


process (rdclk, rreq, temp_empty) is
        begin
                        if rdclk = '1'  then
                                if rreq = '1' and temp_empty = '0' then
                                        read_signal <= '1';
                                else
                                        read_signal <= '0';
                                        end if;
                                end if;
                end process;

                process (wrclk, wreq, temp_full) is
                begin
                                if wrclk = '1' then
                                        if wreq = '1' and temp_full = '0' then
                                                write_signal <= '1';
                                        else
                                                write_signal <= '0';
                                        end if;
                                end if;
                end process;
                read_valid <= read_signal;
                write_valid <= write_signal;
                empty <= temp_empty;
                full <= temp_full;
                wr_ptr <= b_write_counter;
                rd_ptr <= b_read_counter;
end architecture behav;
```

Module No. M-ROU-07

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

entity fifo is
   port (
            reset, rdclk, wrclk, rreq, wreq: in std_logic ;
            data_in : in std_logic_vector(7 downto 0);
```

```
                data_out : out std_logic_vector(7 downto 0);
                empty, full: out std_logic
              );
end entity fifo ;

architecture behav of fifo is

component fifo_controller is
port (
                reset, rdclk, wrclk, rreq, wreq: in std_logic ;
                write_valid, read_valid: out std_logic;
                wr_ptr, rd_ptr: out std_logic_vector (3 downto 0);
                empty, full: out std_logic
              );
 end component fifo_controller ;
 for all: fifo_controller use entity work.fifo_controller(behav);
component block_ram IS
            PORT(
                        d_in : IN std_logic_vector (7 downto 0);
                        ADDRA,ADDRB :IN std_logic_vector (3 downto 0);
                        WEA,REA,CLKA,CLKB :IN std_logic;
                        d_out : OUT std_logic_vector (7 downto 0)
                );
END component block_ram;
for all : block_ram use entity work.block_ram(behave);
signal write_valid, read_valid:std_logic;
signal wr_ptr, rd_ptr:std_logic_vector(3 downto 0);
begin

 fc: fifo_controller port map  (reset,rdclk, wrclk, rreq, wreq, write_valid, read_valid, wr_ptr, rd_ptr, empty, full);
 rm: block_ram port map (data_in, wr_ptr, rd_ptr, write_valid, read_valid, wrclk, rdclk, data_out);
```

```
end architecture behav;
```

Module No. M-ROU-08

```
library ieee;

use ieee.std_logic_1164.ALL;

use ieee.numeric_std.ALL;


ENTITY round_robin 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;


ARCHITECTURE behave OF round_robin IS

TYPE state_type IS (d1,d2,d3,d4);

SIGNAL current_state: state_type := d1;

SIGNAL next_state: state_type;

BEGIN

  clk:PROCESS (clock) IS

  BEGIN

    IF rising_edge(clock) THEN current_state <= next_state;

    END IF;

  END PROCESS clk;

    ns: PROCESS (current_state, din1, din2, din3, din4) IS

    BEGIN

      CASE(current_state, din1, din2, din3, din4) IS

        WHEN d1 =>

          dout<=din1;

          next_state <= d2;

        WHEN d2 =>

          dout<=din2;

          next_state <= d3;
```

```
            WHEN d3 =>
               dout<=din3;
               next_state <= d4;
            WHEN d4 =>
               dout<=din4;
               next_state <= d1;
        END CASE;
    END PROCESS ns;
END;
```

Module No. M-ROU-09

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
entity router is
    port(
            datai1, datai2, datai3, datai4: in std_logic_vector(7 downto 0);
            wr1, wr2, wr3, wr4: in std_logic;
            datao1, datao2, datao3, datao4: out std_logic_vector(7 downto 0);
            wrclk, rdclk: in std_logic;
            rst: in std_logic
            );
end entity router;

architecture behav of router is

    component reg IS
            port(Data_in : IN std_logic_vector (7 downto 0);
            Clock : IN std_logic;
            Data_out : OUT std_logic_vector (7 downto 0);
            Clock_En : IN std_logic;
```

```
        Reset : IN std_logic);
end component reg;
for all :reg use entity work.reg(behave);
component demux IS
PORT (
            Sel: IN std_logic_vector (1 downto 0);
            En : IN std_logic;
            d_in : IN std_logic_vector ( 7 downto 0);
            d_out1,d_out2,d_out3,d_out4 : OUT std_logic_vector (7 downto 0)
        );
END component demux;
for all :demux use entity work.demux(behave);

component round_robin 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 round_robin;
for all: round_robin use entity work.round_robin(behave);
component fifo is
        port (
                reset, rdclk, wrclk, rreq, wreq: in std_logic ;
                data_in : in std_logic_vector(7 downto 0);
                data_out : out std_logic_vector(7 downto 0);
                empty, full: out std_logic
            );
end component fifo;
        for all: fifo use entity work.fifo(behav);



type vector_array is array (0 to 3) of std_logic_vector(7 downto 0);
type vector_2d_array is array (0 to 3, 0 to 3) of std_logic_vector(7 downto 0);
```

```
   type logic_2d_array is array (0 to 3, 0 to 3) of std_logic;


   signal datai_arr, datao_arr, in_buff_out: vector_array;
   signal demux_out, fifo_out: vector_2d_array;
   signal wreq, empty, full: logic_2d_array;
          signal wr_arr: std_logic_vector(0 to 3);
begin
   datai_arr <= (datai1, datai2, datai3, datai4);
   wr_arr <= (wr1, wr2, wr3, wr4);
          g1: for i in 0 to 3 generate
          in_buf: reg port map (datai_arr(i),wrclk, in_buff_out(i), wr_arr(i), rst);
          demux_i: demux port map(in_buff_out(i)(1 downto 0), wr_arr(i), in_buff_out(i),
                                                  demux_out(i, 0), demux_out(i, 1), demux_out(i, 2),
demux_out(i, 3));
          scheduler: round_robin port map(rdclk, fifo_out(i,0), fifo_out(i, 1),
                                                              fifo_out(i, 2), fifo_out(i, 3), datao_arr(i));
   end generate g1;


   g2: for i in 0 to 3 generate
                g3: for j in 0 to 3 generate
                                  queue: fifo port map (rst, rdclk, wrclk,'1', wreq(j, i), demux_out(i, j),
                                                                    fifo_out(j, i) ,empty(j, i), full(j, i));
                end generate g3;


          end generate g2;


          process(wrclk, rdclk, in_buff_out, datao_arr, wr_arr) is
          begin
                l1:for i in 0 to 3 loop
                      l2: for j in 0 to 3 loop
                            if (conv_integer(in_buff_out(j)(1 DOWNTO 0))) = i and wr_arr(j) = '1' THEN
                                  wreq(i, j) <= '1';
```

```
                                        else
                                                wreq(i, j) <= '0';
                                        end if;
                                end loop l2;
                        end loop l1;
                end process;
                datao1 <= datao_arr(0);
                datao2 <= datao_arr(1);
                datao3 <= datao_arr(2);
                datao4 <= datao_arr(3);

end  architecture behav;
```

Module No. M-ROU-10

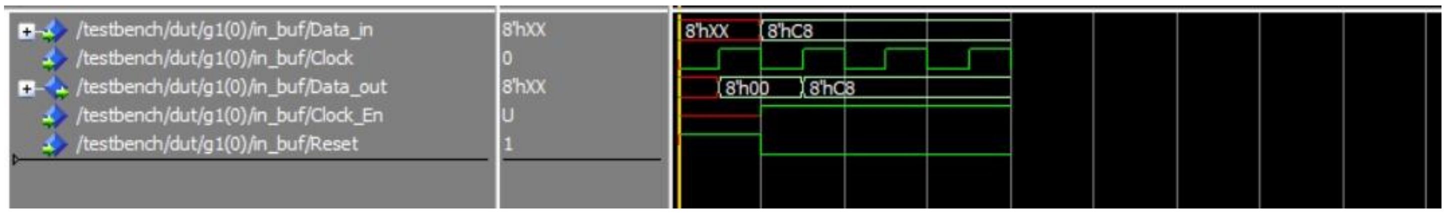
```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric_std.ALL;

ENTITY testbench IS
END ENTITY;

ARCHITECTURE behave OF testbench IS
SIGNAL datai1,datai2,datai3,datai4,datao1,datao2,datao3,datao4 : std_logic_vector(7 DOWNTO 0);
SIGNAL wr1,wr2,wr3,wr4,wclock,rclock,rst : std_logic;

component Router is
   port(
                datai1, datai2, datai3, datai4: in std_logic_vector(7 downto 0);
                wr1, wr2, wr3, wr4: in std_logic;
                datao1, datao2, datao3, datao4: out std_logic_vector(7 downto 0);
                wrclk, rdclk: in std_logic;
                rst: in std_logic
                );
```

```
end component Router;

for dut: Router use entity work.Router(behav);

constant clock_period : time := 20 ns;

BEGIN

dut: Router port map( datai1, datai2, datai3, datai4,

                                                wr1, wr2, wr3, wr4,

                                                datao1, datao2, datao3, datao4,

                                                wclock, rclock, rst);

wclk: PROCESS IS

BEGIN

        wclock <= '0';

        wait for clock_period / 2 ;

        wclock <= '1';

        wait for clock_period / 2 ;

END PROCESS wclk;


rclk: PROCESS IS

BEGIN

        rclock <= '0';

        wait for clock_period / 2 ;

        rclock <= '1';

        wait for clock_period / 2 ;

END PROCESS rclk;


p1: PROCESS IS

BEGIN

        rst <= '1';

        wait for clock_period;

        rst <= '0';

        wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';

        datai1 <= "11001000";

        datai2 <= "10010001";
        datai3 <= "01010110";

        datai4 <= "10001011";

        wait for clock_period;

        datai1 <= "11011011";

        datai2 <= "10010000";

        datai3 <= "01110101";

        datai4 <= "10001110";

        wait for clock_period;

        datai1 <= "11011010";

        datai2 <= "10110011";

        datai3 <= "11110100";

        datai4 <= "00001001";

        wait for clock_period;

        datai1 <= "11001101";

        datai2 <= "11001010";

        datai3 <= "01010111";

        datai4 <= "10111100";

        wait;

END PROCESS p1;


assert_proc: PROCESS(datao1, datao2, datao3, datao4) IS

        BEGIN

                if datao1'event then

                        assert datao1(1 downto 0) = "00" report "wrong output from port1 the address should be 00"
severity error;

                end if;

                if datao2'event then

                        assert datao2(1 downto 0) = "01" report "wrong output from port2 the address should be 01"
severity error;

                end if;

                if datao3'event then

                        assert datao3(1 downto 0) = "10" report "wrong output from port3 the address should be 10"
severity error;
```

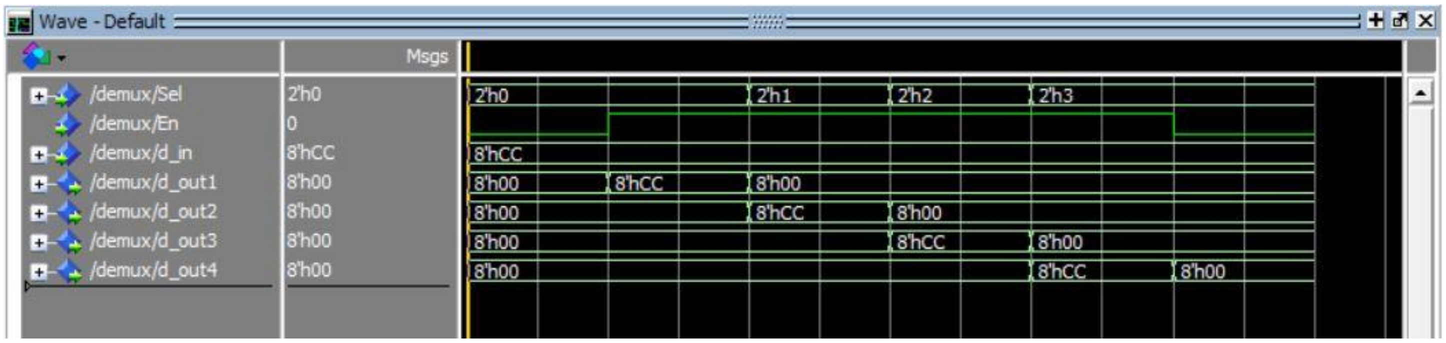
```
                end if;

                if datao4'event then

                        assert datao4(1 downto 0) = "11" report "wrong output from port4 the address should be 11"
severity error;

                end if;

        END PROCESS;

END ARCHITECTURE;
```

Appendix B

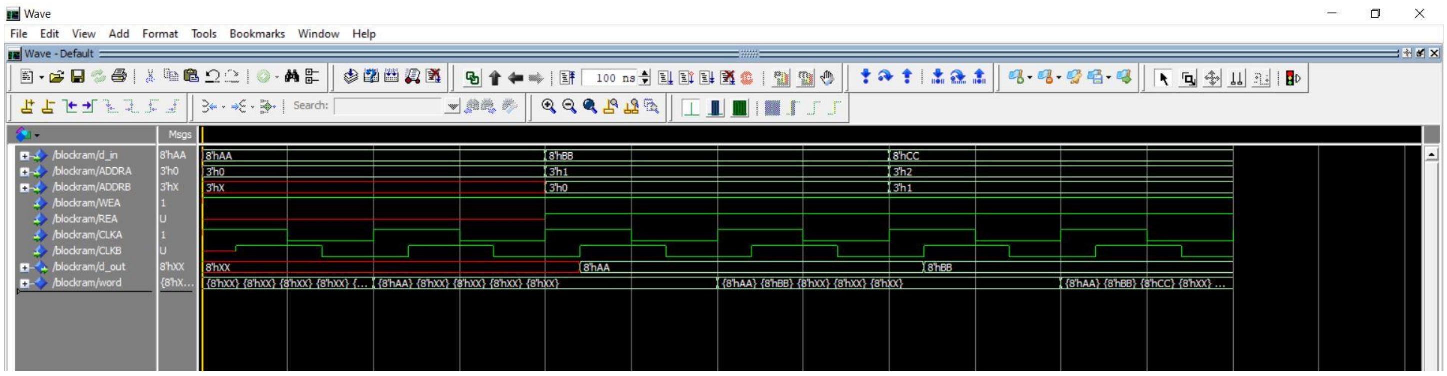
Module No. M-ROU-01 waveform


/testbench/dut/g1(0)/in_buf/Data_in 8'hXX
/testbench/dut/g1(0)/in_buf/Clock 0
/testbench/dut/g1(0)/in_buf/Data_out 8'hXX
/testbench/dut/g1(0)/in_buf/Clock_En U
/testbench/dut/g1(0)/in_buf/Reset 1
8'hXX
8'hC8
8'h00
8'hC8

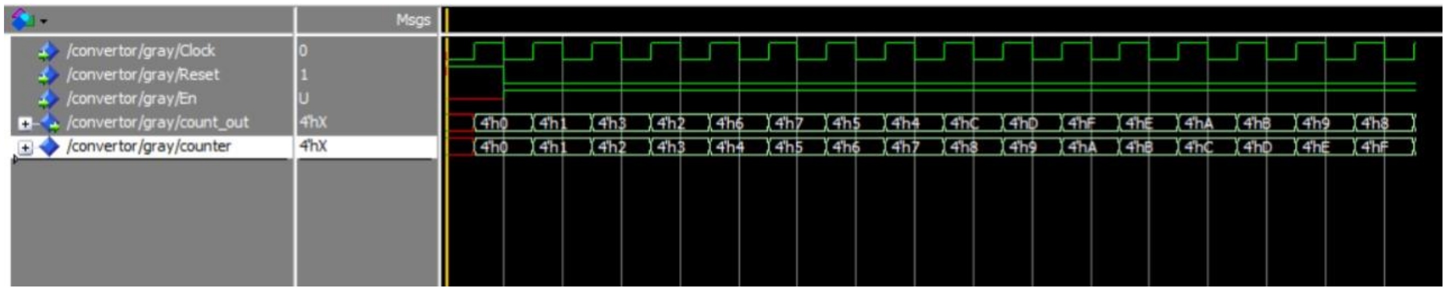
Module No. M-ROU-02 waveform



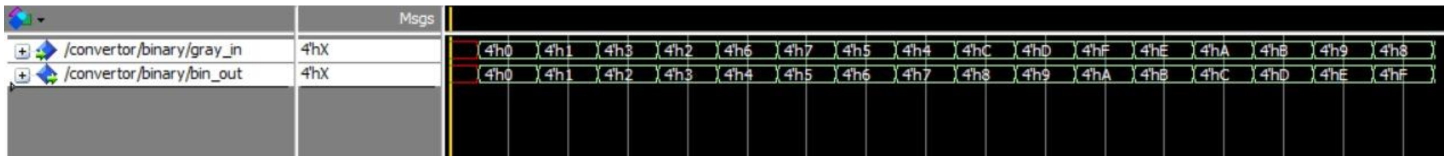
Module No. M-ROU-03 waveform



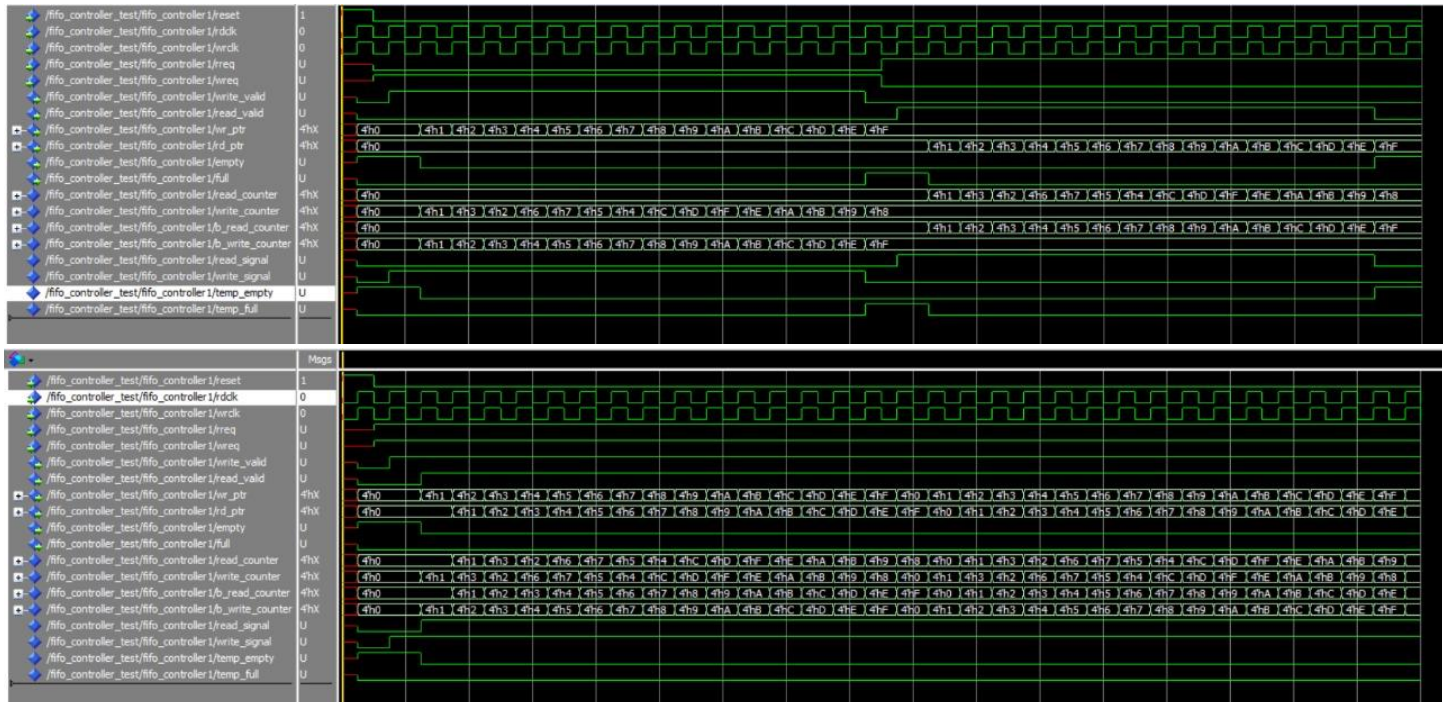
Module No. M-ROU-04 waveform



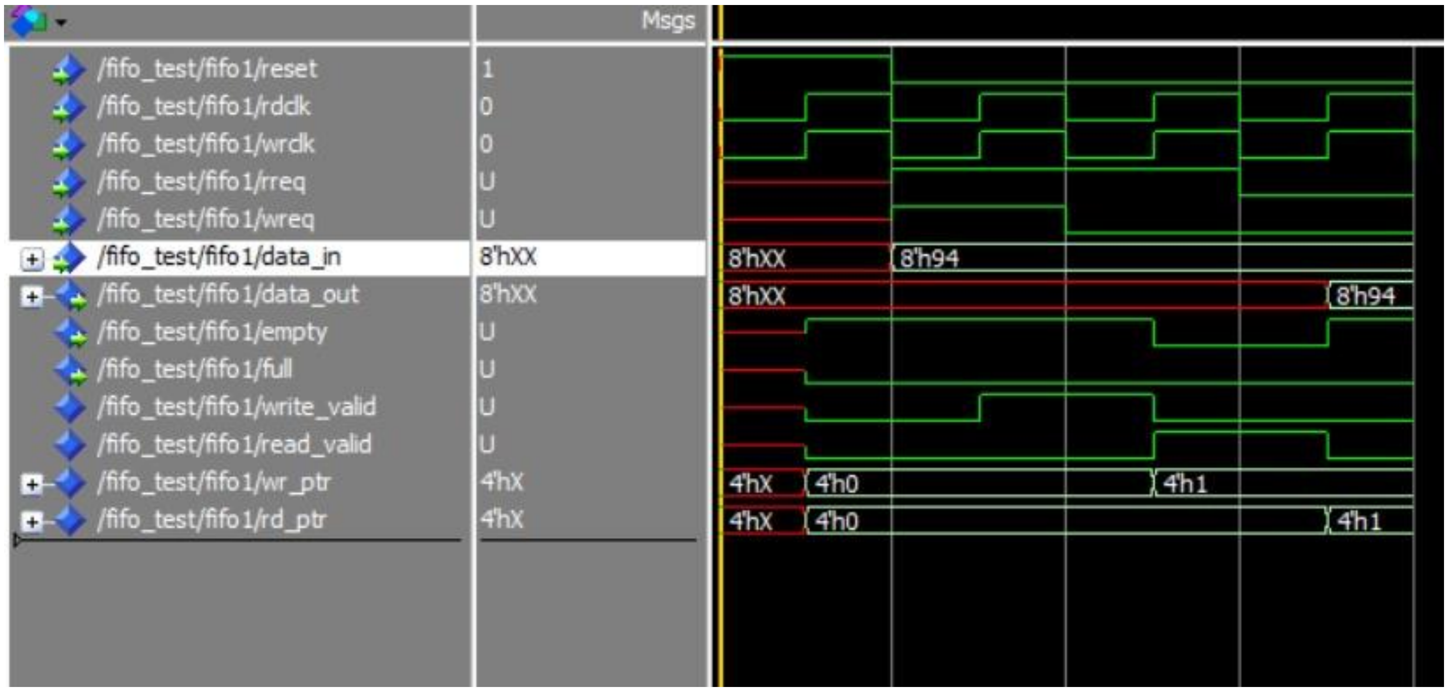
Module No. M-ROU-05 waveform


Msgs
/convertor/binary/gray_in 4'hX
4'h0 4'h1 4'h3 4'h2 4'h6 4'h7 4'h5 4'h4 4'hC 4'hD 4'hF 4'hE 4'hA 4'hB 4'h9 4'h8
/convertor/binary/bin_out 4'hX
4'h0 4'h1 4'h2 4'h3 4'h4 4'h5 4'h6 4'h7 4'h8 4'h9 4'hA 4'hB 4'hC 4'hD 4'hE 4'hF

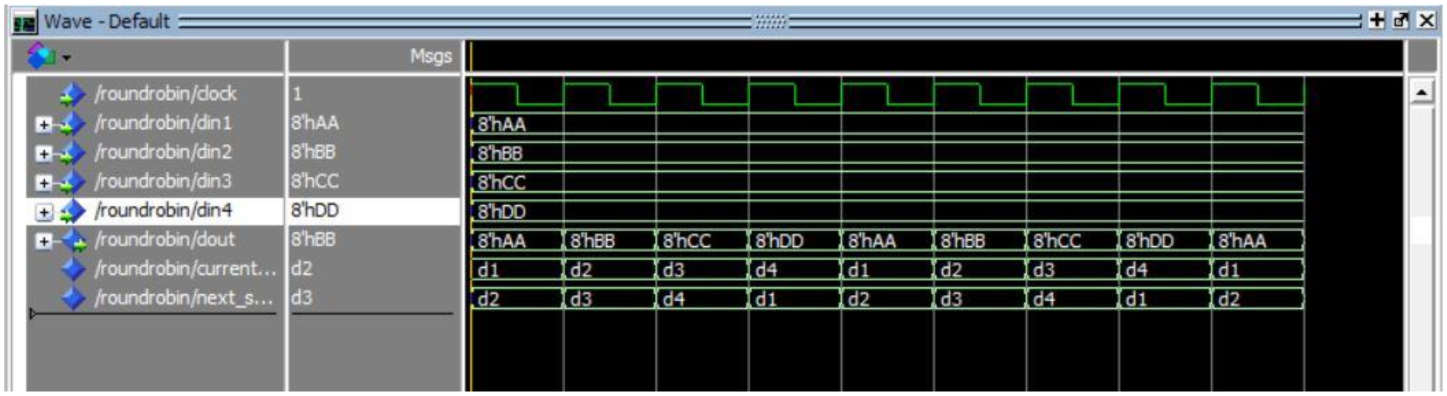
Module No. M-ROU-06 waveform


/fifo_controller_test/fifo_controller1/reset 1
/fifo_controller_test/fifo_controller1/rdclk 0
/fifo_controller_test/fifo_controller1/wrclk 0
/fifo_controller_test/fifo_controller1/rreq U
/fifo_controller_test/fifo_controller1/wreq U
/fifo_controller_test/fifo_controller1/write_valid U
/fifo_controller_test/fifo_controller1/read_valid U
/fifo_controller_test/fifo_controller1/wr_ptr 4'hX
/fifo_controller_test/fifo_controller1/rd_ptr 4'hX
/fifo_controller_test/fifo_controller1/empty U
/fifo_controller_test/fifo_controller1/full U
/fifo_controller_test/fifo_controller1/read_counter 4'hX
/fifo_controller_test/fifo_controller1/write_counter 4'hX
/fifo_controller_test/fifo_controller1/b_read_counter 4'hX
/fifo_controller_test/fifo_controller1/b_write_counter 4'hX
/fifo_controller_test/fifo_controller1/read_signal U
/fifo_controller_test/fifo_controller1/write_signal U
/fifo_controller_test/fifo_controller1/temp_empty U
/fifo_controller_test/fifo_controller1/temp_full U
Msgs

Module No. M-ROU-07 waveform

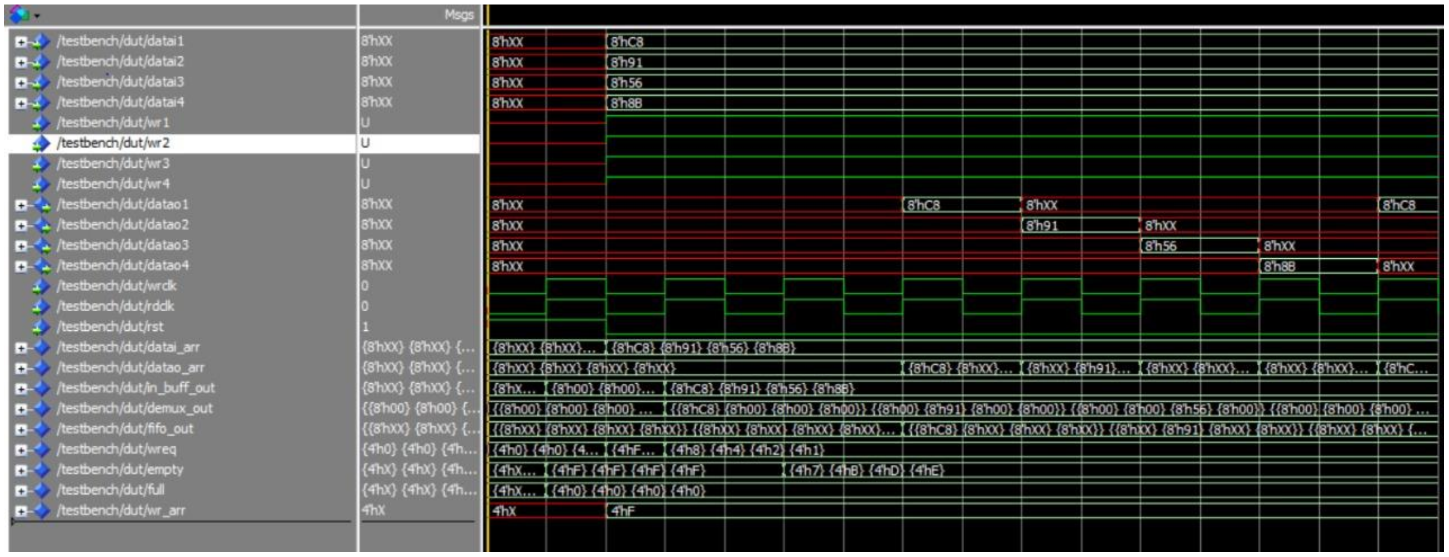


Module No. M-ROU-08 waveform



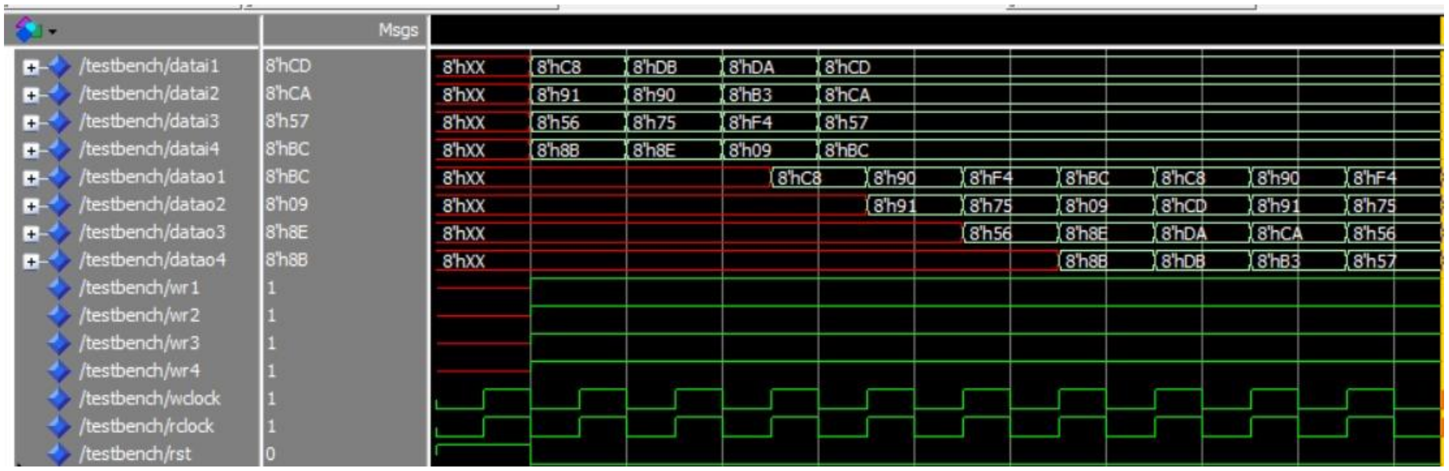
```
Wave - Default
                                Msgs
/roundrobin/clock               1
/roundrobin/din1                8'hAA
/roundrobin/din2                8'hBB
/roundrobin/din3                8'hCC
/roundrobin/din4                8'hDD
/roundrobin/dout                8'hBB
/roundrobin/current...          d2
/roundrobin/next_s...           d3
```

Module No. M-ROU-09 waveform



```
                                Msgs
/testbench/dut/datai1           8'hXX
/testbench/dut/datai2           8'hXX
/testbench/dut/datai3           8'hXX
/testbench/dut/datai4           8'hXX
/testbench/dut/wr1              U
/testbench/dut/wr2              U
/testbench/dut/wr3              U
/testbench/dut/wr4              U
/testbench/dut/datao1           8'hXX
/testbench/dut/datao2           8'hXX
/testbench/dut/datao3           8'hXX
/testbench/dut/datao4           8'hXX
/testbench/dut/wrclk            0
/testbench/dut/rdclk            0
/testbench/dut/rst              1
/testbench/dut/datai_arr        {8'hXX} {8'hXX} {...
/testbench/dut/datao_arr        {8'hXX} {8'hXX} {...
/testbench/dut/in_buff_out      {8'hXX} {8'hXX} {...
/testbench/dut/demux_out        {{8'h00} {8'h00} {...
/testbench/dut/fifo_out         {{8'hXX} {8'hXX} {...
/testbench/dut/wreq             {4'h0} {4'h0} {4'h...
/testbench/dut/empty            {4'hX} {4'hX} {4'h...
/testbench/dut/full             {4'hX} {4'hX} {4'h...
/testbench/dut/wr_arr           4'hX
```

Module No. M-ROU-10 waveform



```
                                Msgs
/testbench/datai1               8'hCD
/testbench/datai2               8'hCA
/testbench/datai3               8'h57
/testbench/datai4               8'hBC
/testbench/datao1               8'hBC
/testbench/datao2               8'h09
/testbench/datao3               8'h8E
/testbench/datao4               8'h8B
/testbench/wr1                  1
/testbench/wr2                  1
/testbench/wr3                  1
/testbench/wr4                  1
/testbench/wclock               1
/testbench/rclock               1
/testbench/rst                  0
```