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)

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 submitted for assessment for another course. I certify that I have not copied in part or whole or otherwise plagiarized the work of other students and / or persons."

STUDENTS MUST SIGN THIS PAGE. ELECTRONIC SIGNATURE IS ACCEPTED.

#	Student ID	Student Signature
1	17p6027	Hla Ahmed Mohamed
2	17p6059	Yomna Mohamed Ahmed
3	17p6011	Mayar sherif
4	17p6002	Dina Amr
5	17p6050	Menna Hesham

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

Table of Contents

Contents

1. Introduction:

A router is an extremely important networking device which is responsible for forwarding and receiving data packets from one computer network to another. A router can direct a packet from the current computer network to the following network by using headers (in data packets) and routing tables which contain a list of the addresses needed to reach the next destination and the router chooses the best path to direct this packet. In this project we are implementing a simple router using VHDL language and creating a test bench to check the correctness and how accurate it is working. Moreover, network on a chip (NoC) routers which is more reliable is very good to use between modules in a system as it has higher performance and efficiency and less delay of messages (reducing congestion in routing) which is suitable in real time applications and it is also cheaper in cost.

Routers are mainly classified into many types, first one is broadband router (wired) which allows a computer to send and receive data packets using the internet, another one is the wireless router which is the same as broadband but allows communication between computers wirelessly. Finally, there are core routers which allow connection to the internet but not between multiple networks.

The design of our router consists of 3 main parts input buffer, switch fabric, output buffer, and controller. The input buffer receives the incoming data packets and stores it to prevent loss of any data, then the controller uses the switch fabric to direct these packets to the suitable output buffer according to the header.

Finally, building a simple chip requires preparing specifications from requirements given very well, then use these requirements to decide the architecture and basic structure of the chip (how it looks like, etc.), then build each module we need in the implementation each with its needed functionality and for building these modules from scratch, we had to use computer aided design (CAD) tool to allow our design to go through a machine readable format and in our router module in this project we used Xilinx ISE. After implementation, we should test these modules to make sure it works properly and it matches the expected output, then create the design of the whole on how each module will interact with the other, at last we reach the implementation phase at which we write the code and test benches for testing the system as whole to finally reach the whole layout.

2. Design Flow:

2.1. Design

We first decided to use FPGA design flow to implement our router module. We implemented all the modules we need for designing the router from scratch, which are: 8-bit Register, 4-1 8-bit DeMUX, BLOCK RAM, Gray counter, Gray to Binary converter, FIFO controller, FIFO and Round Robin Scheduler. So, we needed to input our design into a machine-readable format so we used a Computer Aided Design (CAD) tool which is Xilinx ISE. It is a typical CAD tool that provides many design entry means, such as a schematic capture, HDL entry (VHDL/Verilog) or component netlist. Here, all the designs are captured using VHDL.

2.1.1. Design entry:

We created an ISE project as follows:

1. Create a new project
2. Create a source file (VHDL module) which contains the VHDL design code which describes the behavior of the module and defines the input and output ports.
3. Create VHDL testbench or waveform files that drives the stimulus to test the design file.

2.2. Simulation

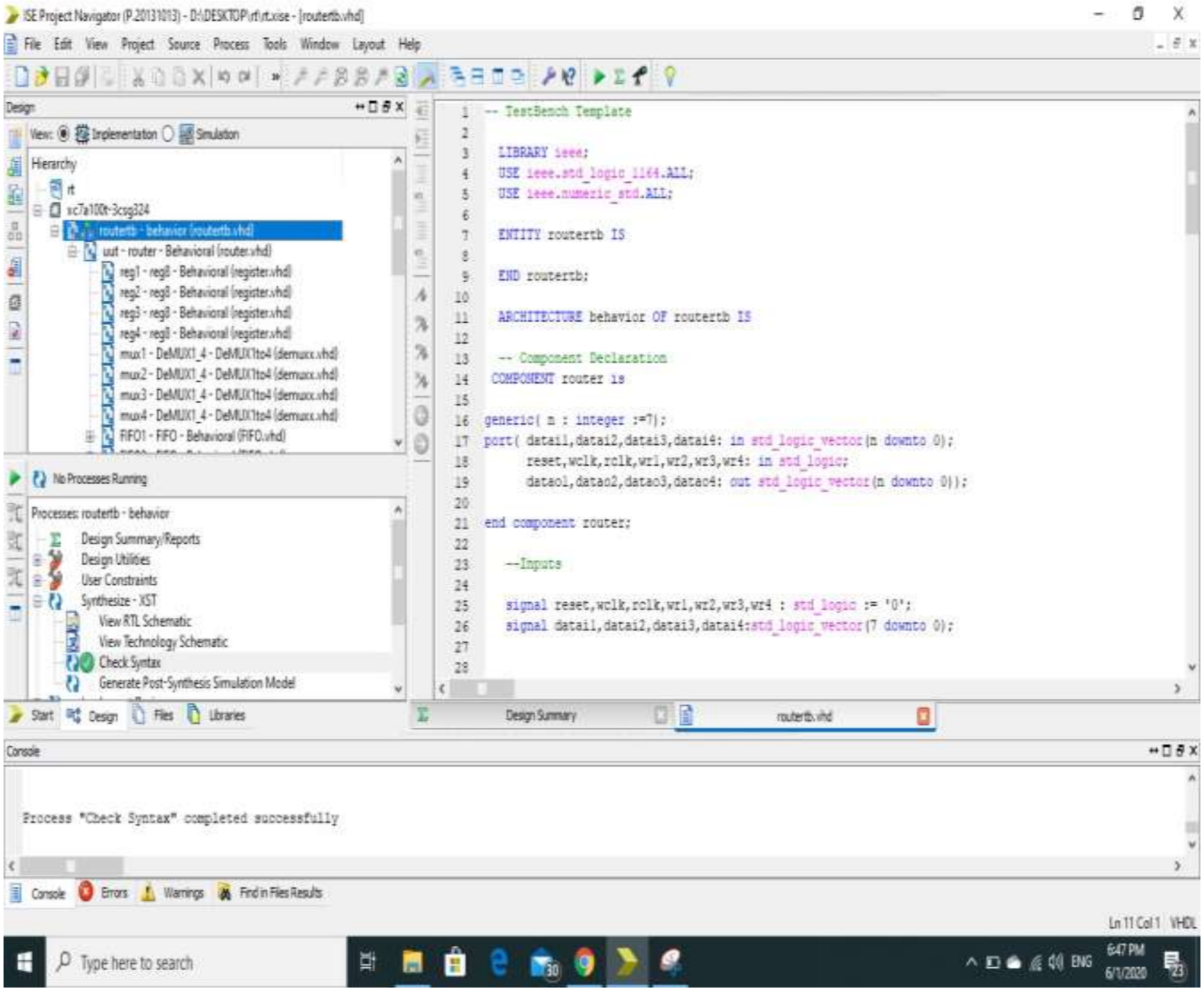
2.2.1. Behavioral simulation (RTL simulation)

1. From the process tab, open Xilinx ISim
2. Double click on simulate behavioral model

The results were displayed in our simulator as shown after the simulation is performed, in which the files that are passed to the simulation are the testbench and the design VHDL

2.2.2. Functional simulation

We also applied functional simulation to verify the functionality of the design

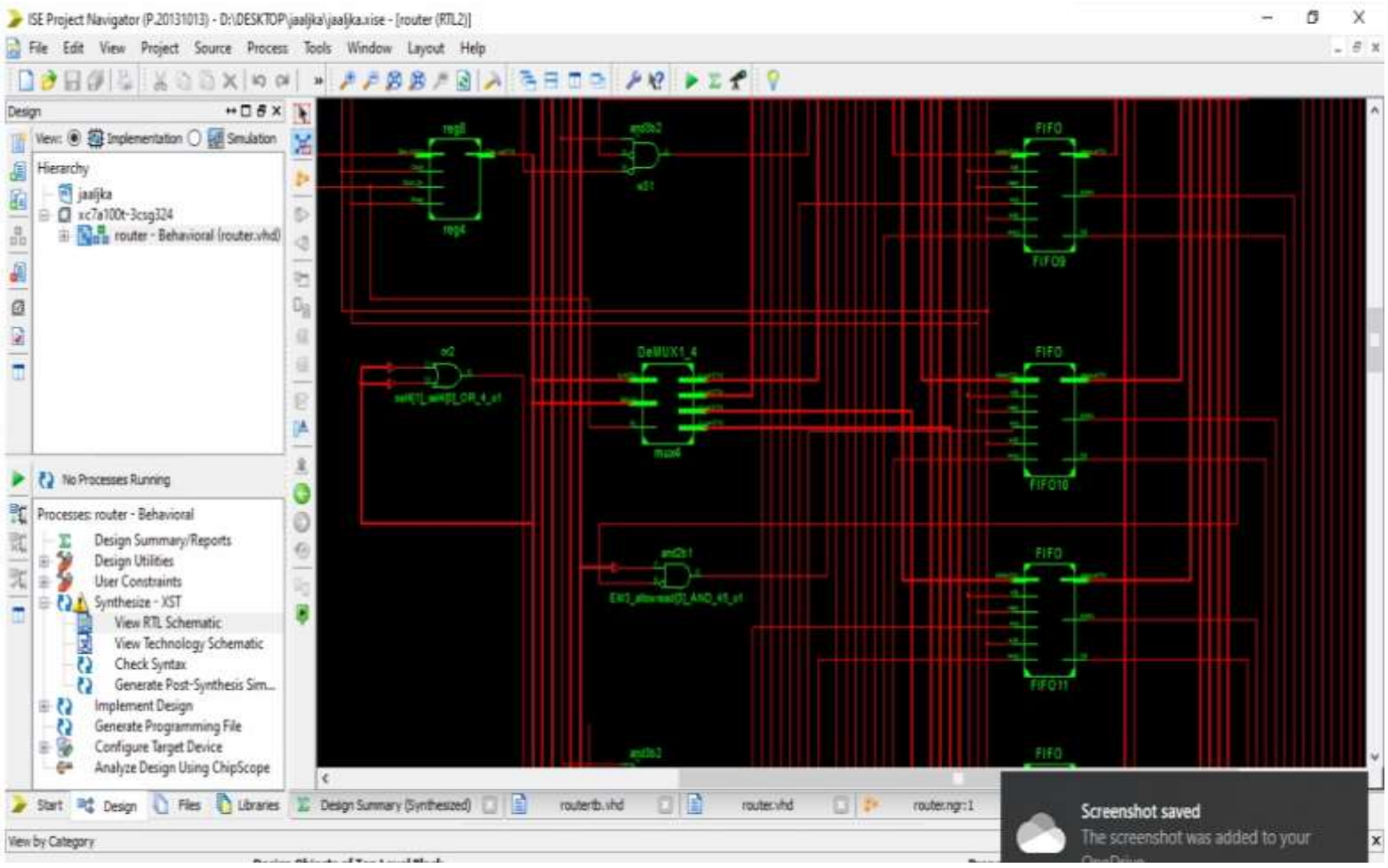

ISE Project Navigator (P.20131013) - D:\DESKTOP\rt\rt.xise - [routertb.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
rt
xc7a100t-3csg324
routertb - behavior (routertb.vhd)
uut - router - Behavioral (router.vhd)
reg1 - reg8 - Behavioral (register.vhd)
reg2 - reg8 - Behavioral (register.vhd)
reg3 - reg8 - Behavioral (register.vhd)
reg4 - reg8 - Behavioral (register.vhd)
mux1 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
mux2 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
mux3 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
mux4 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
FIFO1 - FIFO - Behavioral (FIFO.vhd)
No Processes Running
Processes: routertb - behavior
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Start
Design
Files
Libraries
Design Summary
routertb.vhd
Console
Errors
Warnings
Find in Files Results
Ln 11 Col 1 VHDL
Type here to search
6:47 PM
6/1/2020

```
-- TestBench Template

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

 ENTITY routertb IS

 END routertb;

 ARCHITECTURE behavior OF routertb IS

 -- Component Declaration
COMPONENT router is

generic( n : integer :=7);
port( datai1,datai2,datai3,datai4: in std_logic_vector(n downto 0);
      reset,wclk,rclk,wr1,wr2,wr3,wr4: in std_logic;
      datao1,datao2,datao3,datao4: out std_logic_vector(n downto 0));

end component router;

  --Inputs

  signal reset,wclk,rclk,wr1,wr2,wr3,wr4 : std_logic := '0';
  signal datai1,datai2,datai3,datai4:std_logic_vector(7 downto 0);
```

```
Process "Check Syntax" completed successfully
```

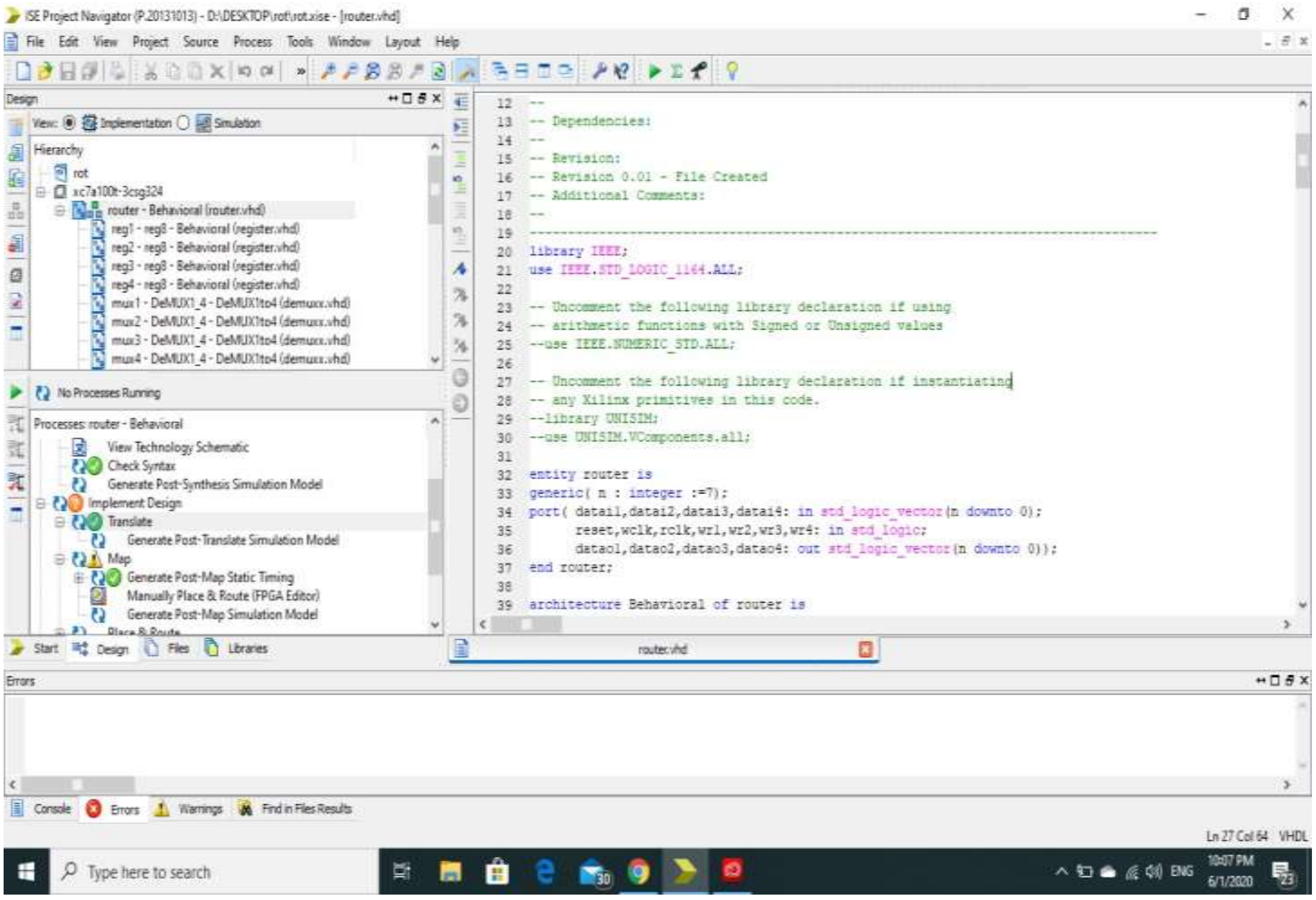
2.3. Synthesis, analysis and verification

In this step we used synthesizing to convert the high-level description of the design into an optimized gate-level representation, In other words, translate our design behavior into an implementation consisting of logic gates as shown.


ISE Project Navigator (P.20131013) - D:\DESKTOP\jaaljka\jaaljka.xise - [router (RTL2)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
jaaljka
xc7a100t-3csg324
router - Behavioral (router.vhd)
No Processes Running
Processes: router - Behavioral
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Sim...
Implement Design
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
reg8
reg4
and2b2
FIFO
FIFO9
or2
DeMUX1_4
mux4
FIFO10
and2b1
FIFO11
Start Design Files Libraries
Design Summary (Synthesized)
routertb.vhd
router.vhd
router.ngr:1
View by Category
Screenshot saved
The screenshot was added to your OneDrive

2.4. Design implementation

In this step we implemented our design by translate, map and place and route as shown.


ISE Project Navigator (P.20131013) - D:\DESKTOP\rot\rot.xise - [router.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Start Design Files Libraries
router.vhd
Errors
Console Errors Warnings Find in Files Results
Ln 27 Col 64 VHDL
Type here to search
ENG
10:07 PM
6/1/2020

```
Hierarchy
  rot
  xc7a100t-3csg324
    router - Behavioral (router.vhd)
      reg1 - reg8 - Behavioral (register.vhd)
      reg2 - reg8 - Behavioral (register.vhd)
      reg3 - reg8 - Behavioral (register.vhd)
      reg4 - reg8 - Behavioral (register.vhd)
      mux1 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
      mux2 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
      mux3 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)
      mux4 - DeMUX1_4 - DeMUX1to4 (demuxx.vhd)

No Processes Running

Processes: router - Behavioral
    View Technology Schematic
    Check Syntax
    Generate Post-Synthesis Simulation Model
  Implement Design
    Translate
      Generate Post-Translate Simulation Model
    Map
      Generate Post-Map Static Timing
      Manually Place & Route (FPGA Editor)
      Generate Post-Map Simulation Model
    Place & Route
```

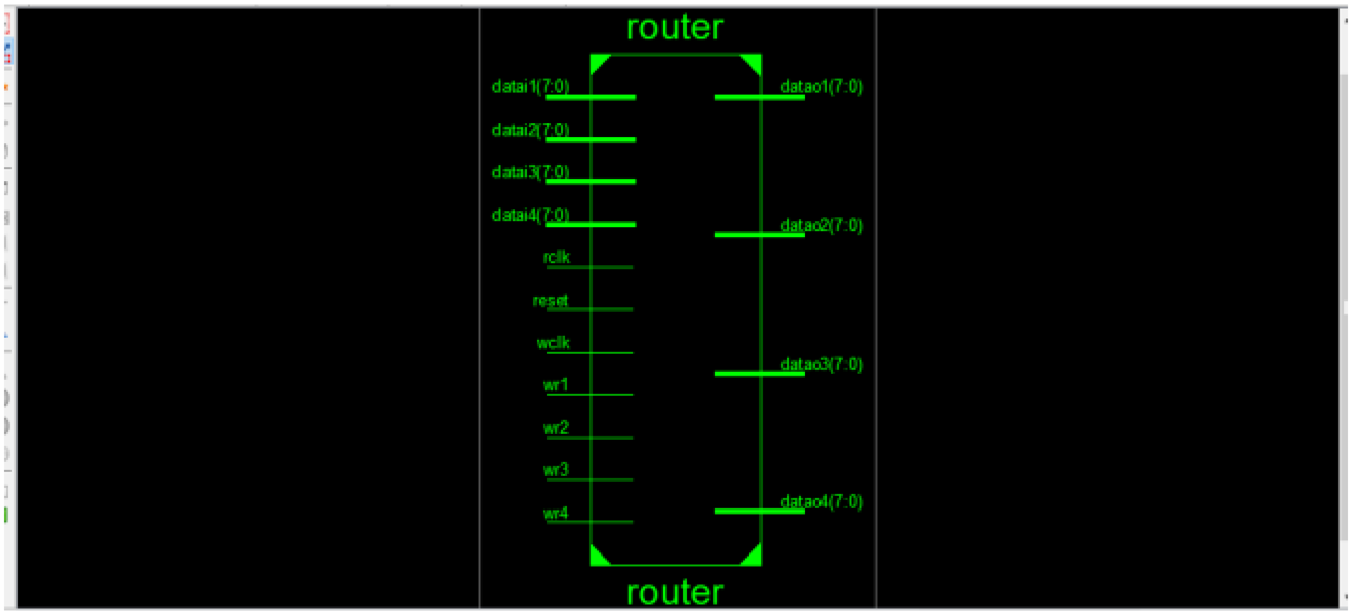
```vhdl
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity router is
generic( n : integer :=7);
port( datai1,datai2,datai3,datai4: in std_logic_vector(n downto 0);
      reset,wclk,rclk,wr1,wr2,wr3,wr4: in std_logic;
      datao1,datao2,datao3,datao4: out std_logic_vector(n downto 0));
end router;

architecture Behavioral of router is
```

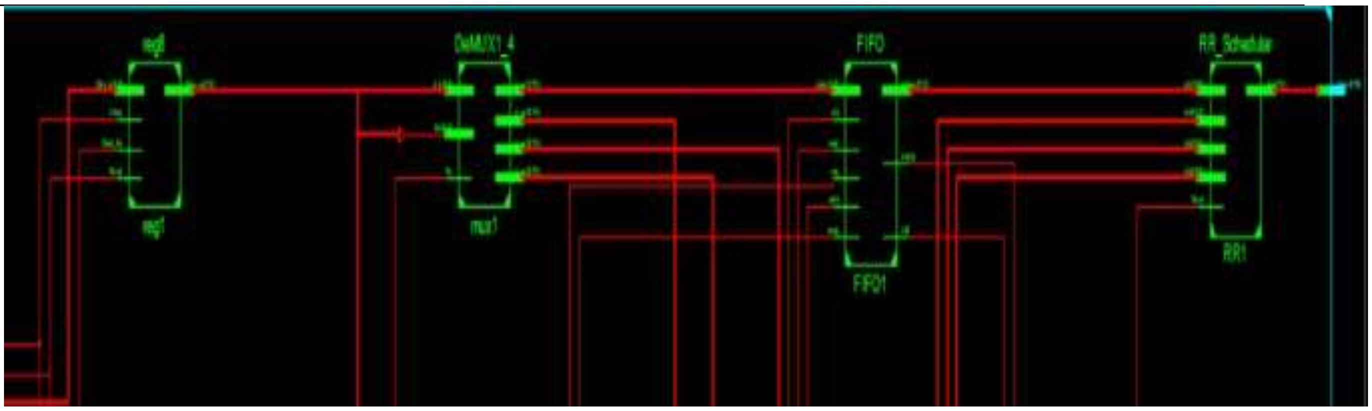
3. Literature Review:

NOC structures depend on bundle exchanged networks. This has prompted new and effective standards for design of routers for NOC, the router follows the work topology that has four data sources and four outputs from/to different routers, and another input and output from/to the Network Interface (NI). Routers can actualize different functionalities - from basic changing to smart steering. Since implanted frameworks are compelled in region and force utilization, yet at the same time need high information rates, routers must be structured in view of equipment use. For circuit-exchanged systems, switches might be structured with no lining (buffering). For packet switched networks, some measure of buffering is required, to help burst information moves. Such information starts in sight and sound applications, for example, video gushing. buffers can be given at the contribution, at the output, or at both input and output

Different structures and usage of routers designs dependent on various directing procedures have been proposed, overall NoC implementation is given by adjusting the workload between equal parallel buffers and router logic. ViChaR: a unique virtual regulator for arrange on-chip switches by Nicopoulos et al (2006), presents a novel bound unified buffer structure, called the dynamic virtual channel controller (ViChaR), which progressively dispenses virtual channels (VC) and buffer resources as per organize traffic conditions. ViChaR amplifies throughput by administering a variable number of VCs on request. route packets, not wires on chip interconnection systems (Dally and Towles 2001), contains framework modules like processors, memories, peripherals which convey by sending packets to each other over the system. The organized network wiring gives all around controlled electrical parameters that take out planning cycles and empower the utilization of superior circuits to lessen inertness and increment data transmission

4. Design Implementation:





Modules	Function
Register	Takes the Data_in and when the clock is rising edge and the enable equals to one outputs the Data_out
DeMultiplexer	the Data_in enters and at clock rising edge based on the selection port is delivered on a certain output port
FIFO	implemented from FIFO controller and memory and the output data is queued as the first in first out
FIFO Controller implemented from grey counter and grey to binary	it gives the permission to memory whether to read or write, and sets the empty and full flags
Round robin Scheduler	Every clock cycle reads data
Memory	Saves the data of the router
Grey Counter	it counts every time the router reads or writes so the address is known from the counting
Gray to binary	it converts the output of Gray counter to binary representation and is given to memory as to read or write from that position

4.1. Implementation:

4.1.1. Register:

The modelling style here is behavioral.

The 8-bit Register module is implemented as to take data_in and output data_out at rising edge and when enable is equal to one and reset is zero, when reset is one the register outputs zeroes. So, we used the clock and reset in the sensitivity list of the register so the process is evaluated whenever the clock or reset change, as the reset is asynchronous

4.1.2. DeMux:

The modelling style here is behavioral.

Our demux has a 8-bit input and a 2 bit selection signal, and 1 bit enable signal, when enable is equal to 1, we output the data on one of the four output ports based on the selection signal, where “00” makes us output data on the first port and the three other ports output zeros, ”01” is for the second port and the three other ports output zeros, the same goes for the rest as “10” indicates port 3 and “11” means we should output data on port 4, the sensitivity list contains Enable and selection so the process is suspended until one of them changes, if enable is equal to zero the output ports are latched and keep outputting their last value.

4.1.3. FIFO:

The architecture of the FIFO is structural and not behavioural as it is built by connecting the controller with the memory using port mapping.
The input reset signal is mapped to the controller reset, write clock and read clock are mapped to both the controller and the memory, the wreq and rreq are mapped to the controller’s wreq and rreq, input data in signal mapped to the dual port, data-out of the memory is mapped to the FIFO output, empty and write flags are outputted from the controller and port mapped to the output of the FIFO.

4.1.4. FIFO Controller:

The modelling style here is a mix between dataflow and behavioral styles.

In the controller, we use a gray counter to keep track of the memory position that we are either reading from or writing to, we have two gray counters, one to keep track of the current write address, and another to keep track of the current read address, the reason for choosing gray counter is that by each increment only one bit is altered and that reduces the possibility of disturbances.
The output of the two gray counters is also used to determine the empty and full flags, where if the output of the read and write counters is the same then we have an empty memory so we rise the flag and disable read valid signal that is mapped to the memory read enable, if the two MSBS are different and the remaining bits are equal then we rise the full flag and disable write valid that is mapped to the memory write enable, this logic was implemented using XOR gates that indicated whether each two bits of the write and read addresses are equal or not.

The output of each gray counter is mapped to a G2B converter and the MSB is disposed using slicing, as it is only used for determining flags, the binary output is mapped to memory read and write addresses.

4.1.5. Round robin scheduler:

The modelling style here is behavioral.

The round robin has four 8 bits data inputs, at each rising edge one of the input ports is read starting from port 1 to port 4, and then the same process is repeated, it is implemented using a FSM whose details are discussed in section 5.5

In our router, four round robins are instantiated, each one of them reads from four consecutive FIFOs, the reading from memory is backlogged and synchronized with the round robin scheduler using a FSM where there is a 4 bit vector, and using one hot encoding, only one bit is equal to one and this bits is shifted at each rising clock edge, the rreq of the first FIFO of each port is anded with the LSB of the vector, the rreq of the second FIFO of each port is anded with the second bit of the vector and so on.

4.1.6. Memory:

The modelling style here is behavioral.

We have a dual port memory, where we have a clock for write and another clock from read, the memory is implemented using two processes.
The first process is responsible for the write operation, its sensitivity list contains the write clock, at each rising edge of the write clock, we check on the write enable and if it is high then we write into a 2D array where the index to the array is the write address that we convert to integer number and then we store the 8 bits data.
The second process is responsible for the read operation, its sensitivity list contains the read clock, at each rising edge of the read clock, we check on the read enable and if it is high then we read from a 2D array where the index to the array is the read address that we convert to integer number and then we output the 8 bit data, if the read enable is low the output is high impedance.

4.1.7. Gray Counter:

The modelling style here is behavioral.

It is implemented using one process whose sensitivity list contains reset and clk, the reset is asynchronous and if it is high then the current count is zeros, if reset is low and we have a rising edge, we check on the enable, if it is high we update the count, if it is low the count is latched and it keeps its current value.

4.1.8. Gray to Binary:

The modelling style here is dataflow.

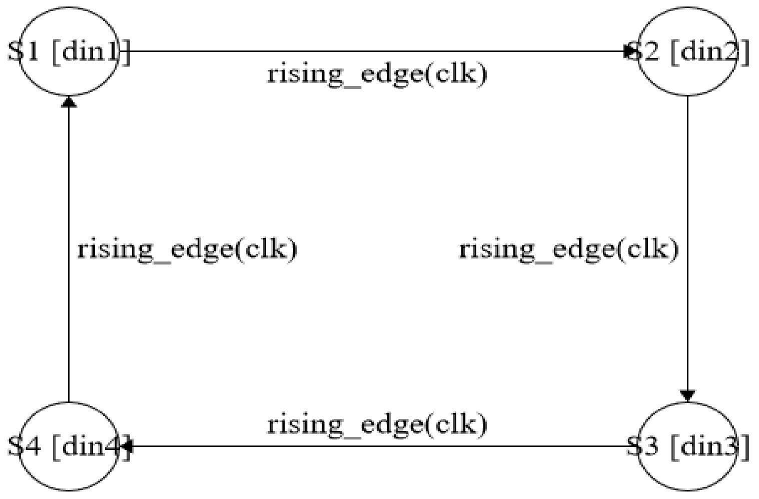
Here, we convert the input that comes in gray representation to binary representation and output it, we didn’t use processes, we used gates to get each bit of the four output bits based on the relation between binary and gray representation, where the MSB is the same, the second bit from the left in binary representation is the result of the Xor between the MSB and the second MSB in the gray representation, the third bit from the left is Xor between the three MSB bits, and finally bit number zero is an Xor between all the bits in gray representation.

5. Scheduler Design and FSM Implementation:

The round robin scheduler in our design has four inputs and one output. For every clock cycle, rising edge, it outputs one of its inputs.

The finite state machine implementing the discussed design works as follows:

1. It has four states, one for each input.
2. It increments its state at every rising edge for the clock.
3. At each state, the round robin outputs the corresponding input.



Current State	**Next State**	**Output**
S1	S2	Din1
S2	S3	Din2
S3	S4	Din3

S4	S1	Din4

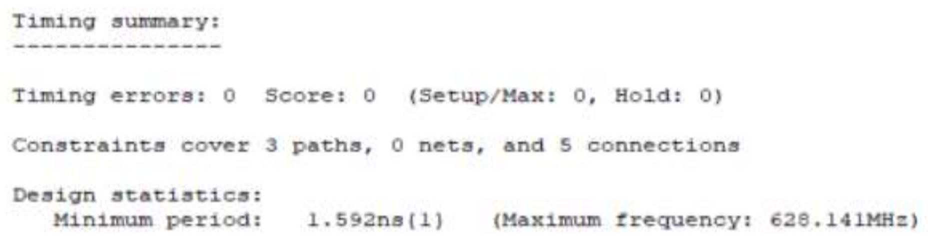
Since the output is based on the state, the finite state machine is implemented as moore type

Modelling the architecture of the finite state machine is done through one, two, or three processes:

	One process	Two processes		Three processes		
Process number	P1	P1	P2	P1	P2	P3
Tasks	RS CS NS OP	RS CS	NS OP	RS CS	NS	OP
Sensitivities in Mealy	Clock Reset CS Input	Clock Reset	CS Input	Clock Reset	CS Input	CS Input
Sensitivities in Moore	Clock Reset CS	Clock Reset	CS Input	Clock Reset	CS Input	CS

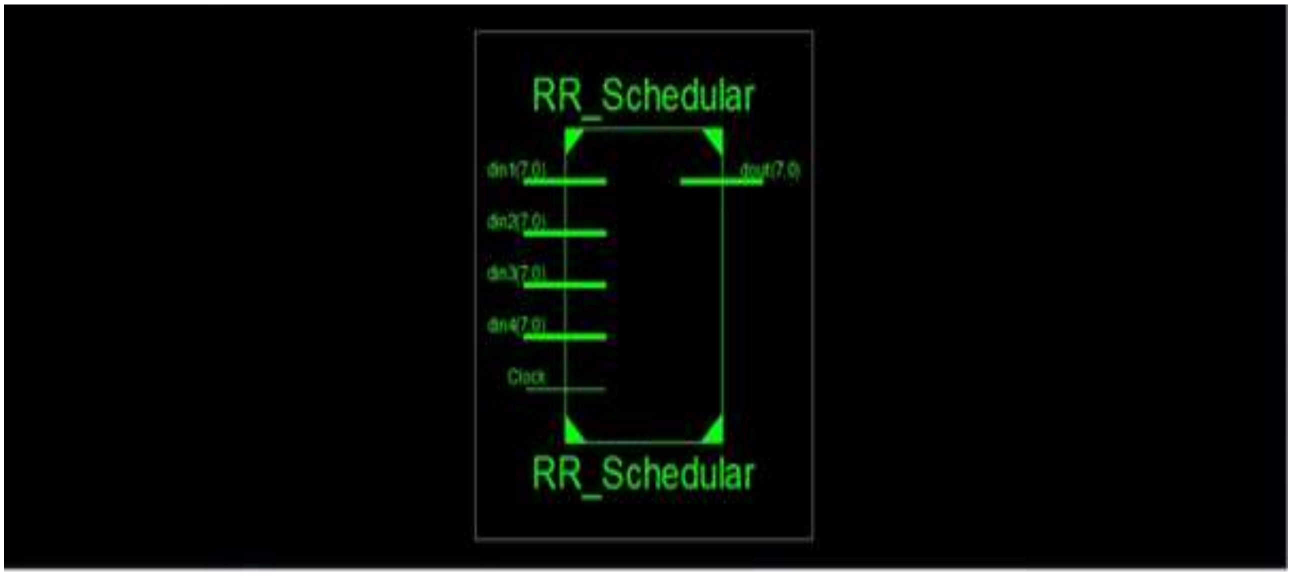
We used the two processes moore style in our implementation.

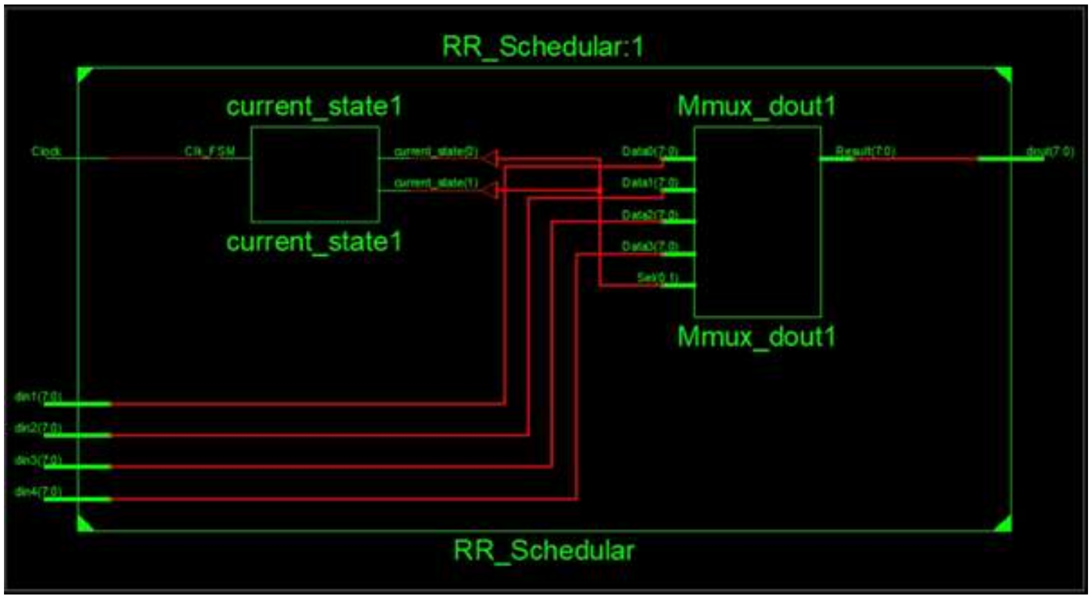
Timing Analysis:



```
Timing summary:
---------------

Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 3 paths, 0 nets, and 5 connections

Design statistics:
   Minimum period:   1.592ns(1)   (Maximum frequency: 628.141MHz)
```

RTL Synthesis:





6. Test and Simulation Results:

The main function of the router is to deliver the input data to its appropriate destination, based on the last 2 bits in the data packet, and the router is expected to deliver all data without any losses. This is the main functionality that our testbench code is testing, if the router can perform it well then, all its modules are working appropriately.

In the testbench process “tb”, it starts by setting the reset by 1 for one clock period, The round robin starts its scheduling at the first rising edge of the read clock and the iteration takes four rising edges in the read clock ,after the first clock cycle the packet available “wr” is set to 1, then data is entered where data inputs have different addresses to check that the routing to the correct ports functions correctly, then the packet available is set to 0 once again. The four schedulers iterate over an empty FIFO for the first three rising edges, thus they output high impedance as in the memory module when read valid is low then we output high impedance, and read valid is set to zero when the empty flag is high.

After the first three read rising edges, data starts to be stored in the appropriate FIFO allowing the scheduler to read and output it appropriately.

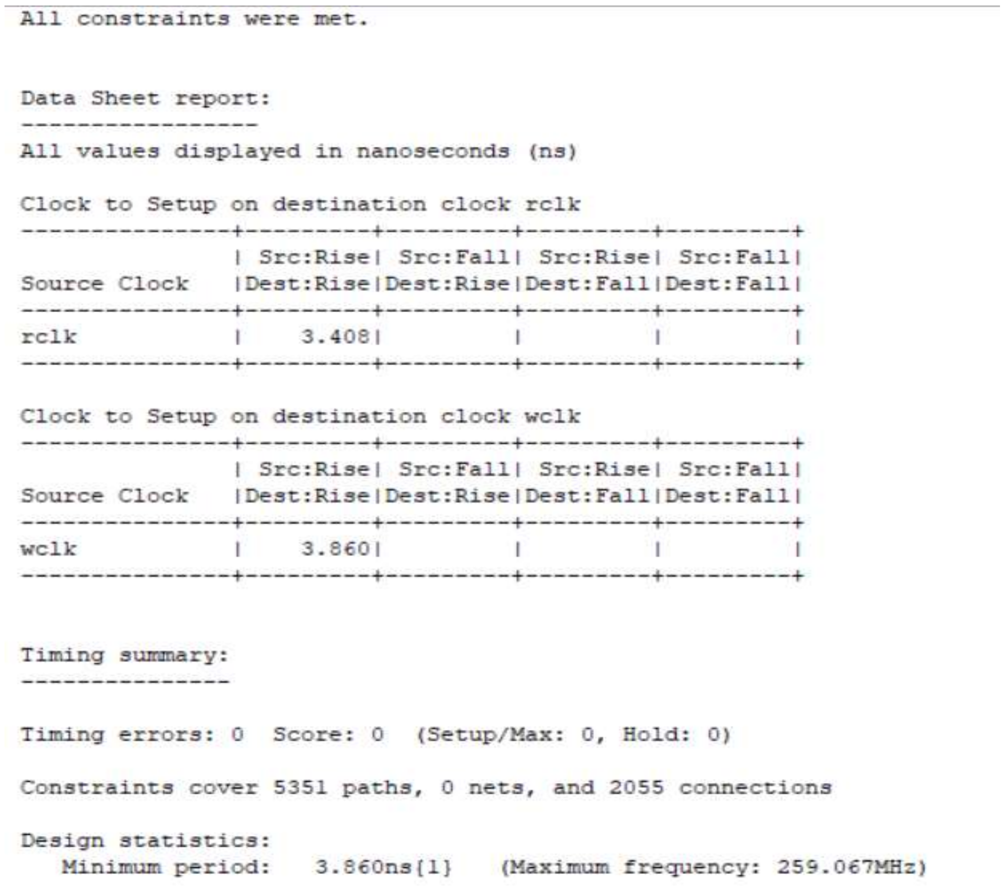
Read and write clocks are different to stress test the router.

we created equivalence classes to avoid wasting time and trying too many combinations, logically if an input port is given data of different addresses and the router managed to route them to the correct output ports based on their addresses then there is no need to test the same functionality over and over again by entering different input data with all the possible addresses to each one of the input ports, as the logic is implemented using the same strategy, and it is always preferred to reduce the number of test cases in a test suite, and at the same time make sure that the functionality is fully tested.

Data input (Datai1, Datai2, Datai3, Datai4)	Output Port
**** **00	Datao1
**** **01	Datao2
**** **10	Datao3
**** **11	Datao4

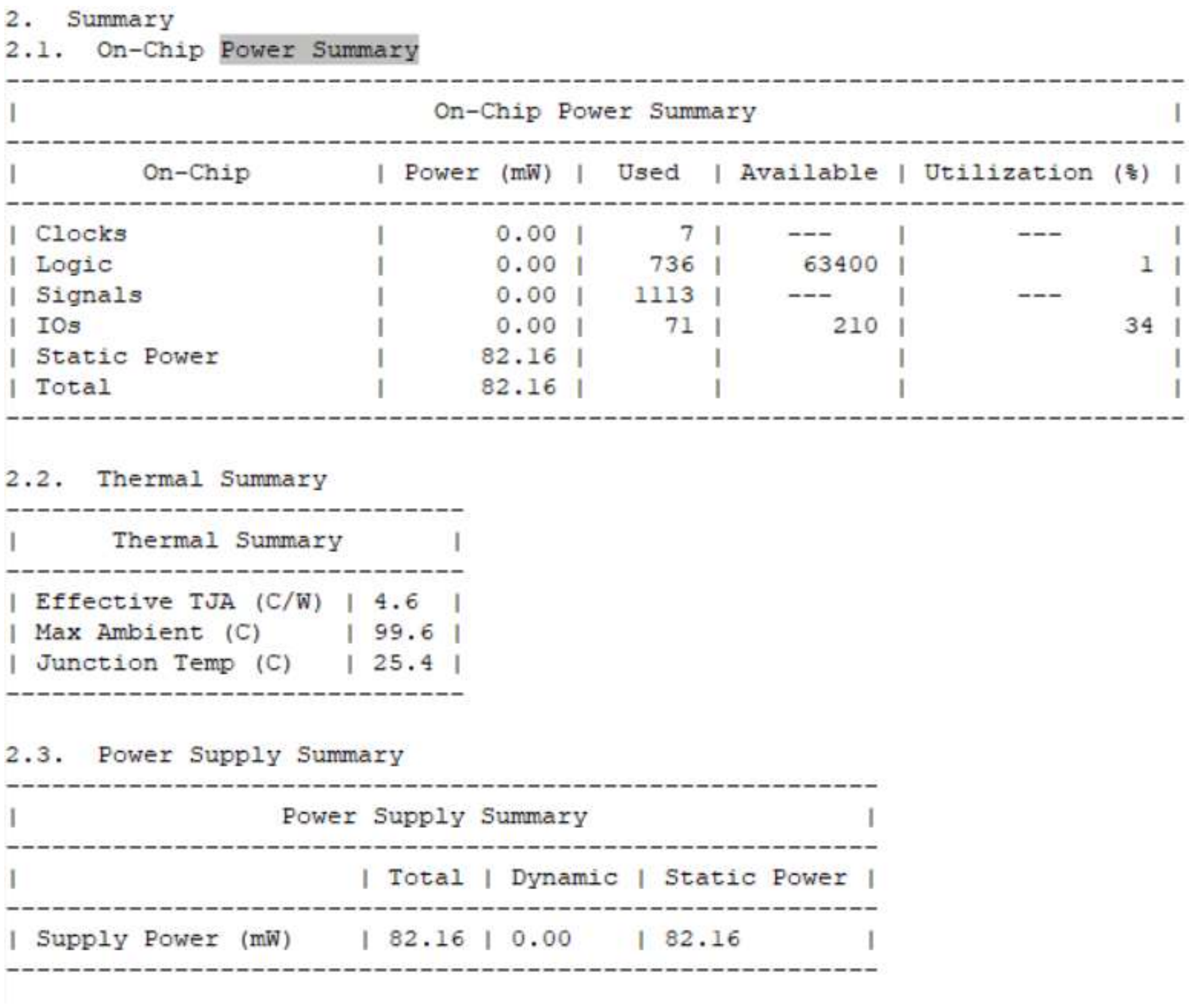
The router performed well with the test-cases without any errors in the delivering or storing as shown in appendix C.

router time analysis report:
we added use constraints, in which we chose the wclk and rclk to be 10 ns, we checked slack values and they were all positive.



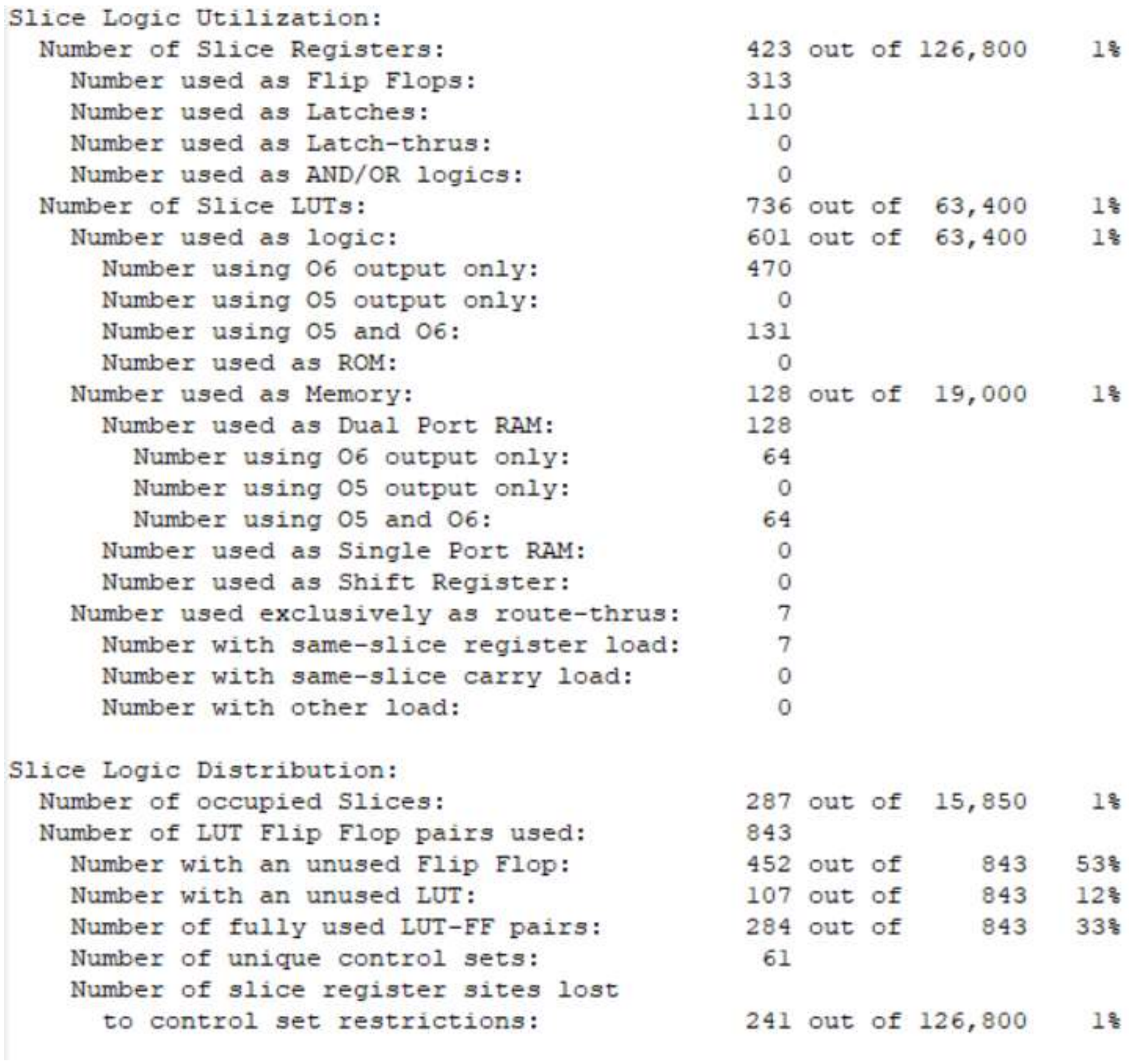
```
All constraints were met.


Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Clock to Setup on destination clock rclk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
rclk           |    3.408|         |         |         |
---------------+---------+---------+---------+---------+

Clock to Setup on destination clock wclk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
wclk           |    3.860|         |         |         |
---------------+---------+---------+---------+---------+


Timing summary:
---------------

Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 5351 paths, 0 nets, and 2055 connections

Design statistics:
   Minimum period:   3.860ns{1}   (Maximum frequency: 259.067MHz)
```

router power analysis report:



```
2.  Summary
2.1.  On-Chip Power Summary
-----------------------------------------------------------------------------
|                           On-Chip Power Summary                           |
-----------------------------------------------------------------------------
|        On-Chip        | Power (mW) |  Used  | Available | Utilization (%) |
-----------------------------------------------------------------------------
| Clocks                |       0.00 |      7 |    ---    |       ---       |
| Logic                 |       0.00 |    736 |     63400 |               1 |
| Signals               |       0.00 |   1113 |    ---    |       ---       |
| IOs                   |       0.00 |     71 |       210 |              34 |
| Static Power          |      82.16 |        |           |                 |
| Total                 |      82.16 |        |           |                 |
-----------------------------------------------------------------------------

2.2.  Thermal Summary
------------------------------
|      Thermal Summary       |
------------------------------
| Effective TJA (C/W) | 4.6  |
| Max Ambient (C)     | 99.6 |
| Junction Temp (C)   | 25.4 |
------------------------------

2.3.  Power Supply Summary
---------------------------------------------------------
|                 Power Supply Summary                  |
---------------------------------------------------------
|                       | Total | Dynamic | Static Power |
---------------------------------------------------------
| Supply Power (mW)     | 82.16 | 0.00    | 82.16        |
---------------------------------------------------------
```

router area analysis report:



```
Slice Logic Utilization:
  Number of Slice Registers:                   423 out of 126,800    1%
    Number used as Flip Flops:                 313
    Number used as Latches:                    110
    Number used as Latch-thrus:                  0
    Number used as AND/OR logics:                0
  Number of Slice LUTs:                        736 out of  63,400    1%
    Number used as logic:                      601 out of  63,400    1%
      Number using O6 output only:             470
      Number using O5 output only:               0
      Number using O5 and O6:                  131
      Number used as ROM:                        0
    Number used as Memory:                     128 out of  19,000    1%
      Number used as Dual Port RAM:            128
        Number using O6 output only:            64
        Number using O5 output only:             0
        Number using O5 and O6:                 64
      Number used as Single Port RAM:            0
      Number used as Shift Register:             0
    Number used exclusively as route-thrus:      7
      Number with same-slice register load:      7
      Number with same-slice carry load:         0
      Number with other load:                    0

Slice Logic Distribution:
  Number of occupied Slices:                   287 out of  15,850    1%
  Number of LUT Flip Flop pairs used:          843
    Number with an unused Flip Flop:           452 out of     843   53%
    Number with an unused LUT:                 107 out of     843   12%
    Number of fully used LUT-FF pairs:         284 out of     843   33%
    Number of unique control sets:              61
    Number of slice register sites lost
      to control set restrictions:             241 out of 126,800    1%
```

7. Conclusion:

Our designing process faced some challenges from the very beginning. The design flow works as follows: design specification, coding, synthesis, optimization and at the end is implementation. We chose the method of bottom-up design to design our system, we firstly start with individual blocks then combine them together and then the system is complete. In designing, each block is created alone and goes through the design flow process (designed, implemented, tested,..etc.) and after that all of the blocks are combined together and tested together as a whole system. But the problem here was that the number of blocks took us a long time to simulate and test after combining the blocks together and it was somehow challenging. Another issue was that having to finish the work sequentially which took more time and also was not so easy to modify.

As a result of the implementation process of the system (router module), the system had the ability to deliver the input data that comes at the input buffer and outputs them to the correct destination without losses, and this is done by the register module. That helps in storing the data until the next clock edge to avoid any losses while waiting to be delivered to its destination. This delivery process happens at each clock edge when enable is equal to one, so at a clock edge the first input data is output at its destination and with the next clock edge the next input data is output and so on.

We have 3 different types of implementation modelling styles, they are dataflow style, behavioral style, and structural style. The behavioral style is when a module consists of processes, in the process the lines are simulated one by one like any program written in any other programming language. In our project there are more than one module that used this style like the demux, register, RR scheduler, gray counter, and router. The dataflow style is mainly used for modules that have concurrent signal assignment statements, assigning computed variables to signals which use logic gates (eg.AND , XOR gates). It represents the data flowing through a system. In our project the one used this style in implementation is the FIFO controller as the empty flag and full flag were created using logic gates, and the gray to binary converter which also consists of XOR gates. The structural style is the one that mainly uses port mapping connecting between components. And what used this style in implementation in our project is the FIFO.

In our implementation we used a round robin scheduler to arrange outputting each of the input data at each rising edge. To implement this design, we created a finite state machine FSM, this FSM goes through several states to finally allow the round robin to output the corresponding input. So as a result of the dependence of the output on the state, the FSM is usually implemented using the behavioral style (moore type) as it may be done through one or more processes, in our code we used the 2 processes moore style.

The test strategy used in while creating this router module is as follows, for each individual block implemented it had its own test bench to test its functionality (register, demux, FIFO, FIFIO controller, RR scheduler, gray to binary converter, etc), we used exhaustive testing to test all available combinations of inputs and outputs, after testing each one alone we combine them and make other test benches to ensure the functionality of the whole new module, so in the router module we made equivalence classes and tested each one to avoid wasting time trying each input output combination. So we built 10 test cases to try to test the system, we entered input data with different addresses in different input ports, and then checked if the data reached its correct destination and its expected output port or not.And as for the timing in these modules, we used the clock as for each clock edge a module works. Between inputting the data and outputting it we have the modules in between which delays reaching the data to the output buffer all together to avoid any losses can happen. So at each clock edge we find a new output at a port.

Finally, the flow we should use in any future project is to decide the design first and what blocks we need from specifications, then implementing all small modules and test each alone to insure its functionality, then start to combine them one by one to finally get the required system and test it by comparing its results to the calculated expected results.

8. References:

- Adaptive Backpressure: Efficient Buffer Management for On-ChipNetworks, (Daniel Becker,2012)
- Application-Specific Buffer Space Allocation for network on chip router design, (Jincao Hu, 2004)
- Flow-Aware Allocation for On-Chip Networks,(Bannerjee 2009)
- 2-level FIFO architecture design for switch fabrics in network-on-chip, (Po-TsangHuang & Wei Hwang 2006)
- Dynamically-allocated multi queue buffers for VLSI Communication switches, (Tamir & Frazier,1992)
- Adaptive Buffer size routing for Wireless Sensor Networks, (Kalyani & Narasimha 2013)
- A Dynamic Virtual Channel Regulator for Network-on-Chip Routers, (Nicopouloset al 2006)
- Design of Energy-Efficient Channel Buffers with Router Bypassing for Network-on-Chips, (Kodi2009)
- Enhanced Reliability Aware NoC Router (Neishaburi & Zilic 2011)
- Dynamic packet fragmentation for increased virtual channel utilization in on-chip routers, (Kang etal 2009)

9. Task Distribution List

#	Student ID	Tasks
1	**17p6027**	**Module 3, module 6, module 7, module 9, module 10**
2	**17p6059**	**Module 4, module 8, module 9 Report sections: 5, 6**
3	**17p6011**	**Module 2, module 9 Report sections: 2,4**
4	**17p6002**	**Module 1, module 9 Report sections: 3,4**
5	**17p6050**	**Module 5, module 9 Report sections: 1,7**

Appendix A – <<VHDL Model Source Code>>

Register:

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

entity reg8 is
    Port ( Data_in : in  STD_LOGIC_vector(7 downto 0);
           Clock : in  STD_LOGIC;
           Clock_En : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Data_out : out STD_LOGIC_vector(7 downto 0));
end reg8;

architecture Behavioral of reg8 is
--Signal temp: STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
begin
process(Clock,Reset)
begin
if Reset = '1' then
Data_out <= (others => '0');
  elsif (Clock'event and Clock ='1') then
    if(Clock_En='1') then
         Data_out <= Data_in;
         end if;
end if;
end process;

end Behavioral;
```

DeMux:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DeMUX1_4 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 DeMUX1_4;

architecture DeMUX1to4 of DeMUX1_4 is

begin
 process(En,Sel)
```

```
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_out1 <="00000000";
          d_out2 <="00000000";
          d_out4 <="00000000";
     when"11" =>
          d_out4 <= d_in;
          d_out1 <="00000000";
          d_out2 <="00000000";
          d_out3 <="00000000";
     when others =>
          d_out1 <="00000000";
          d_out2 <="00000000";
          d_out3 <="00000000";
     d_out4 <="00000000";
          end case;
  end if;
end process;
end DeMUX1to4;
```

FIFO:

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



entity FIFO is
generic(
    dataw : integer := 7);
port( reset: in std_logic :='1';
        rclk,wclk: in std_logic;
                 rreq,wreq: in std_logic;
                 datain: in std_logic_vector(dataw downto 0);
                 dataout: out std_logic_vector(dataw downto 0);
                 empty,full: out std_logic);

end FIFO;

```

```
architecture struct of FIFO is
COMPONENT dualport IS
                  generic(
     addrw : integer := 2;
     dataw : integer := 7);
         port
         (
                  D_in    : in std_logic_vector(dataw downto 0);
                  ADDRA   : in std_logic_vector(addrw downto 0);
                  ADDRB   : in std_logic_vector(addrw downto 0);
                  WEA             : in std_logic := '1';
                  REA             : in std_logic := '1';
                  CLKA    : in std_logic;
                  CLKB    : in std_logic;
                  D_out           : out std_logic_vector(dataw downto 0)
         );

         END COMPONENT dualport;
         FOR Mem: dualport USE ENTITY work.dualport(behave);

COMPONENT FIFOcontroller is
     generic(
     addrw : integer := 2);
     port( reset: in std_logic :='1';
            rdclk,wrclk: in std_logic;
                           rreq,wreq: in std_logic;
                      write_valid,read_valid: out std_logic;
                      wr_ptr,rd_ptr: out std_logic_vector( addrw downto 0);
                      empty,full: out std_logic);
      end component FIFOcontroller;
         FOR controller : FIFOcontroller USE ENTITY
 work.FIFOcontroller(Behavioral);

signal wr_addr,r_addr: std_logic_vector( 2 downto 0);
signal write_valid, read_valid, f,e : std_logic;
signal data_out: std_logic_vector(dataw downto 0);
BEGIN
         Mem: dualport
         PORT
 MAP(datain,wr_addr,r_addr,write_valid,read_valid,wclk,rclk,data_out);
dataout<=data_out;
         controller : FIFOcontroller
         PORT
 MAP(reset,rclk,wclk,rreq,wreq,write_valid,read_valid,wr_addr,r_addr,e,f);

empty <= e;
full <= f;
end struct;
```

FIFO Controller:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity FIFOcontroller is
generic(
    addrw : integer := 2);
port( reset: in std_logic :='1';
      rdclk,wrclk: in std_logic;
                rreq,wreq: in std_logic;
                write_valid,read_valid: out std_logic;
                wr_ptr,rd_ptr: out std_logic_vector( addrw downto 0);
                empty,full: out std_logic);
end FIFOcontroller;

architecture Behavioral of FIFOcontroller 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 writecount: Gray_Counter USE ENTITY work.Gray_Counter(Counter_Beh);
FOR readcount: Gray_Counter USE ENTITY work.Gray_Counter(Counter_Beh);
component GrayToBin is
port
 (
       G: in std_logic_vector (3 downto 0); -- G is the gray_in
       B: out std_logic_vector (3 downto 0) -- B is the bin_out
);
end component GrayToBin;
FOR writeG2B: GrayToBin USE ENTITY work.GrayToBin(behav);
FOR readG2B : GrayToBin USE ENTITY work.GrayToBin(behav);
signal WEN,REN: std_logic;
signal current_wr_ptr,current_rd_ptr: std_logic_vector(3 downto 0);
signal current_full,current_empty: std_logic;
signal wr,rd: std_logic_vector(3 downto 0);
begin
writecount: Gray_Counter
PORT MAP(wrclk,reset,wreq and not(current_full),current_wr_ptr);
readcount: Gray_Counter
PORT MAP(rdclk,reset,rreq and not(current_empty),current_rd_ptr);
writeG2B: GrayToBin
PORT MAP(current_wr_ptr,wr);
readG2B: GrayToBin
PORT MAP(current_rd_ptr,rd);
wr_ptr<=wr(addrw downto 0);
rd_ptr<=rd(addrw downto 0);
current_empty <= not((current_wr_ptr(3) xor
current_rd_ptr(3))or(current_wr_ptr(2) xor
current_rd_ptr(2))or(current_wr_ptr(1) xor
current_rd_ptr(1))or(current_wr_ptr(0) xor current_rd_ptr(0))) ;
```

```
current_full <= (current_wr_ptr(3)xor current_rd_ptr(3)) and
(current_wr_ptr(2)xor current_rd_ptr(2)) and not((current_wr_ptr(1) xor
current_rd_ptr(1))or(current_wr_ptr(0) xor current_rd_ptr(0))) ;
full <=current_full;
empty <= current_empty;
write_valid<= wreq and not(current_full);
read_valid <= rreq and not(current_empty);
end Behavioral;
```

Round Robin Scheduler:

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

Entity RR_Schedular 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 RR_Schedular;

ARCHITECTURE RR_Fsm OF RR_Schedular IS
  TYPE state IS (s1, s2, s3, s4);
  SIGNAL current_state: state:= s4;
  SIGNAL next_state: state:= s1;
BEGIN
  cs: PROCESS (Clock)
  BEGIN
    IF (rising_edge(Clock)) THEN
      current_state <= next_state;
    END IF;
  END PROCESS cs;

  ns: PROCESS (current_state)
  BEGIN
  CASE current_state IS
    WHEN s1 =>
      next_state <= s2;
      dout <= din1;
    WHEN s2 =>
      next_state <= s3;
      dout <= din2;
    WHEN s3 =>
      next_state <= s4;
      dout <= din3;
    WHEN s4 =>
      next_state <= s1;
      dout <= din4;
  END CASE;
  END PROCESS ns;
END ARCHITECTURE RR_Fsm;
```

Memory:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity dualport is
  generic(
    addrw : integer := 2;
    dataw : integer := 7);
 port
  (
                D_in    : in std_logic_vector(dataw downto 0);
                ADDRA   : in std_logic_vector(addrw downto 0);
                ADDRB   : in std_logic_vector(addrw downto 0);
                WEA             : in std_logic := '1';
                REA             : in std_logic := '1';
                CLKA    : in std_logic;
                CLKB    : in std_logic;
                D_out           : out std_logic_vector(dataw downto 0)
        );

 end dualport;
 architecture behave of dualport is

        -- 2-D array
        type dblock is array(addrw downto 0) of std_logic_vector(dataw downto 0);

        -- Declare the RAM signal.
        signal temp : dblock;

 begin

        process(CLKA)
        begin
                if(rising_edge(CLKA)) then
                        if(WEA = '1') then
                                temp(conv_integer(ADDRA)) <= D_in;
                        end if;
                end if;
        end process;

        process(CLKB)
        begin
                if(rising_edge(CLKB)) then
                  if(REA = '1') then
                        D_out <= temp(conv_integer(ADDRB));
                        else
                        D_out <= (others=>'Z');
                end if;
                end if;
        end process;

end behave;
```

Gray Counter:

```
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 Counter_Beh of Gray_Counter is
  SIGNAL CS, NS, tmp, tmp_next: std_logic_vector (3 DOWNTO 0);
  Begin
  process(Reset,Clock) is
    Begin
    If(Reset ='1') then       --Asynchorouns Reset
        CS <= "0000";
    Elsif(rising_edge(Clock)) then
        If(En = '1') then
        CS <= NS;
      End if;
    End If;
  End process;

  tmp <= CS XOR ('0' & tmp(3 DOWNTO 1));
  tmp_next <= std_logic_vector(unsigned(tmp) + 1);
  NS <= tmp_next XOR ('0' & tmp_next(3 DOWNTO 1));
  Count_out <= CS;
End Counter_Beh;
```

Gray to Binary:

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

entity GrayToBin is
port
(
 G: in std_logic_vector (3 downto 0); -- G is the gray_in
 B: out std_logic_vector (3 downto 0) -- B is the bin_out
);
end GrayToBin;

architecture behav of GrayToBin is

begin
```

```

B(3) <= G(3);
B(2) <= G(3) xor G(2);
B(1) <= G(3) xor G(2) xor G(1);
B(0) <= G(3) xor G(2) xor G(1) xor G(0);

end behav;
```

Router:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity router is
generic( n : integer :=7);
port( datai1,datai2,datai3,datai4: in std_logic_vector(n downto 0);
      reset,wclk,rclk,wr1,wr2,wr3,wr4: in std_logic;
         datao1,datao2,datao3,datao4: out std_logic_vector(n downto 0));
end router;

architecture Behavioral of router is
component reg8 is
    Port ( Data_in : in  STD_LOGIC_vector(7 downto 0);
           Clock : in  STD_LOGIC;
           Clock_En : in  STD_LOGIC;
           Reset : in  STD_LOGIC;
           Data_out : out STD_LOGIC_vector(7 downto 0));
end component reg8;
FOR reg1: reg8 USE ENTITY work.reg8(Behavioral);
FOR reg2: reg8 USE ENTITY work.reg8(Behavioral);
FOR reg3: reg8 USE ENTITY work.reg8(Behavioral);
FOR reg4: reg8 USE ENTITY work.reg8(Behavioral);
component DeMUX1_4 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 DeMUX1_4;
FOR mux1: DeMUX1_4 USE ENTITY work.DeMUX1_4(DeMUX1to4);
FOR mux2: DeMUX1_4 USE ENTITY work.DeMUX1_4(DeMUX1to4);
FOR mux3: DeMUX1_4 USE ENTITY work.DeMUX1_4(DeMUX1to4);
FOR mux4: DeMUX1_4 USE ENTITY work.DeMUX1_4(DeMUX1to4);
component FIFO is
generic(
    dataw : integer := 7);
port( reset: in std_logic :='1';
      rclk,wclk: in std_logic;
                 rreq,wreq: in std_logic;
                 datain: in std_logic_vector(dataw downto 0);
                 dataout: out std_logic_vector(dataw downto 0);
```

```
                  empty,full: out std_logic);

end component FIFO;
FOR     FIFO1: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO2: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO3: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO4: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO5: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO6: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO7: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO8: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO9: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO10: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO11: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO12: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO13: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO14: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO15: FIFO USE ENTITY work.FIFO(struct);
FOR     FIFO_16: FIFO USE ENTITY work.FIFO(struct);
component RR_Schedular 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 RR_Schedular;
FOR     RR1: RR_Schedular USE ENTITY work.RR_Schedular(RR_Fsm);
FOR     RR2: RR_Schedular USE ENTITY work.RR_Schedular(RR_Fsm);
FOR     RR3: RR_Schedular USE ENTITY work.RR_Schedular(RR_Fsm);
FOR     RR4: RR_Schedular USE ENTITY work.RR_Schedular(RR_Fsm);

signal Data_out1,Data_out2,Data_out3,Data_out4: STD_LOGIC_vector(7 downto 0);--reg outputs & demux inputs
signal Demux_out1,Demux_out2,Demux_out3,Demux_out4,Demux_out11,Demux_out12,Demux_out13,Demux_out14,Demux_out21,Demux_out22,Demux_out23,Demux_out24,Demux_out31,Demux_out32,Demux_out33,Demux_out34: STD_LOGIC_vector(7 downto 0);--demux outputs
signal EM,FULL,EM1,FULL1,EM2,FULL2,EM3,FULL3,EM4,FULL4,EM5,FULL5,EM6,FULL6,EM7,FULL7,EM8,FULL8,EM9,FULL9,EM10,FULL10,EM11,FULL11,EM12,FULL12,EM13,FULL13,EM14,FULL14,EM15,FULL15,EM16,FULL16: std_logic;
signal FIFOout1,FIFOout2,FIFOout3,FIFOout4,FIFOout5,FIFOout6,FIFOout7,FIFOout8,FIFOout9,FIFOout10,FIFOout11,FIFOout12,FIFOout13,FIFOout14,FIFOout15,FIFOout16:STD_LOGIC_vector(7 downto 0);--FIFO outputs
signal sel1,sel2,sel3,sel4:  std_logic_vector(1 downto 0);
signal allowread: std_logic_vector(3 downto 0);
signal nextt: std_logic_vector(3 downto 0);
signal write1,write2,write3,write4,write5,write6,write7,write8,write9,write10,write11,write12,write13,write14,write15,write16:std_logic;
signal w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16:std_logic;
signal r:std_logic;
signal sync,nsync: std_logic_vector(1 downto 0);
begin
```

```
reg1: reg8
PORT MAP(datai1,wclk,wr1,reset,Data_out1);
reg2: reg8
PORT MAP(datai2,wclk,wr2,reset,Data_out2);
reg3: reg8
PORT MAP(datai3,wclk,wr3,reset,Data_out3);
reg4: reg8
PORT MAP(datai4,wclk,wr4,reset,Data_out4);
--demux
sel1<=Data_out1(1 downto 0);
sel2<=Data_out2(1 downto 0);
sel3<=Data_out3(1 downto 0);
sel4<=Data_out4(1 downto 0);
w1<= not(FULL) and (not(sel1(1) or sel1(0)))  and wr1;
w2<= not(FULL1)  and not(sel2(1) or sel2(0))  and wr2;
w3<= not(FULL2) and not(sel3(1) or sel3(0))  and wr3;
w4<= not(FULL3)  and not(sel4(1) or sel4(0))  and wr4;
w5<=not(FULL4)  and (not(sel1(1)) and sel1(0)) and wr1 ;
w6<=not(FULL5)  and (not(sel2(1)) and sel2(0)) and wr2 ;
w7<=not(FULL6)  and (not(sel3(1)) and sel3(0)) and wr3 ;
w8<=not(FULL7)  and (not(sel4(1)) and sel4(0)) and wr4 ;
w9<=not(FULL8)  and (not(sel1(0)) and sel1(1)) and wr1 ;
w10<=not(FULL9) and (not(sel2(0)) and sel2(1)) and wr2;
w11<=not(FULL10)  and (not(sel3(0)) and sel3(1)) and wr3;
w12<=not(FULL11) and (not(sel4(0)) and sel4(1)) and wr4;
w13<=not(FULL12)  and (sel1(1) and sel1(0)) and wr1 ;
w14<=not(FULL13) and (sel2(1) and sel2(0)) and wr2;
w15<=not(FULL14) and (sel3(1) and sel3(0)) and wr3;
w16<=not(FULL15) and (sel4(1) and sel4(0)) and wr4 ;
mux1: DeMUX1_4
PORT MAP(sel1,wr1,Data_out1,Demux_out1,Demux_out2,Demux_out3,Demux_out4);
mux2: DeMUX1_4
PORT MAP(sel2,wr2,Data_out2,Demux_out11,Demux_out12,Demux_out13,Demux_out14);
mux3: DeMUX1_4
PORT MAP(sel3,wr3,Data_out3,Demux_out21,Demux_out22,Demux_out23,Demux_out24);
mux4: DeMUX1_4
PORT MAP(sel4,wr4,Data_out4,Demux_out31,Demux_out32,Demux_out33,Demux_out34);
--FIFO
FIFO1: FIFO
PORT MAP(reset,rclk,wclk,not(EM) and allowread(0) ,w1 ,Demux_out1,FIFOout1,EM,FULL);
FIFO2: FIFO
PORT MAP(reset,rclk,wclk,not(EM1) and allowread(1),w2,Demux_out11,FIFOout2,EM1,FULL1);
FIFO3: FIFO
PORT MAP(reset,rclk,wclk,not(EM2) and allowread(2),w3,Demux_out21,FIFOout3,EM2,FULL2);
FIFO4: FIFO
PORT MAP(reset,rclk,wclk,not(EM3) and allowread(3),w4,Demux_out31,FIFOout4,EM3,FULL3);
FIFO5: FIFO
```

```
PORT MAP(reset,rclk,wclk,not(EM4)  and
   allowread(0),w5,Demux_out2,FIFOout5,EM4,FULL4);
FIFO6: FIFO
PORT MAP(reset,rclk,wclk,not(EM5)and
   allowread(1),w6,Demux_out12,FIFOout6,EM5,FULL5);
FIFO7: FIFO
PORT MAP(reset,rclk,wclk,not(EM6)and
   allowread(2),w7,Demux_out22,FIFOout7,EM6,FULL6);
FIFO8: FIFO
PORT MAP(reset,rclk,wclk,not(EM7)and
   allowread(3),w8,Demux_out32,FIFOout8,EM7,FULL7);
FIFO9: FIFO
PORT MAP(reset,rclk,wclk,not(EM8)and
   allowread(0),w9,Demux_out3,FIFOout9,EM8,FULL8);
FIFO10: FIFO
PORT MAP(reset,rclk,wclk,not(EM9)and
   allowread(1),w10,Demux_out13,FIFOout10,EM9,FULL9);
FIFO11: FIFO
PORT MAP(reset,rclk,wclk,not(EM10)and
   allowread(2),w11,Demux_out23,FIFOout11,EM10,FULL10);
FIFO12: FIFO
PORT MAP(reset,rclk,wclk,not(EM11)and
   allowread(3),w12,Demux_out33,FIFOout12,EM11,FULL11);
FIFO13: FIFO
PORT MAP(reset,rclk,wclk,not(EM12)and allowread(0),w13
   ,Demux_out4,FIFOout13,EM12,FULL12);
FIFO14: FIFO
PORT MAP(reset,rclk,wclk,not(EM13)and
   allowread(1),w14,Demux_out14,FIFOout14,EM13,FULL13);
FIFO15: FIFO
PORT MAP(reset,rclk,wclk,not(EM14)and
   allowread(2),w15,Demux_out24,FIFOout15,EM14,FULL14);
FIFO_16: FIFO
PORT MAP(reset,rclk,wclk,not(EM15)and
   allowread(3),w16,Demux_out34,FIFOout16,EM15,FULL15);
--RR
RR1: RR_Schedular
port map(rclk,FIFOout1,FIFOout2,FIFOout3,FIFOout4,datao1);
RR2: RR_Schedular
port map(rclk,FIFOout5,FIFOout6,FIFOout7,FIFOout8,datao2);
RR3: RR_Schedular
port map(rclk,FIFOout9,FIFOout10,FIFOout11,FIFOout12,datao3);
RR4: RR_Schedular
port map(rclk,FIFOout13,FIFOout14,FIFOout15,FIFOout16,datao4);

 p1:PROCESS (reset,rclk)
  BEGIN
  if(reset='1') then
  allowread<="0100";
  else IF (rising_edge(rclk)) THEN
   allowread <= nextt;
        end if;
        end if;
        end process;

 p2:PROCESS (allowread )
 begin
    CASE allowread IS
    WHEN "0001" =>
       nextt<="0010";
    WHEN "0010"=>
      nextt<= "0100";
    WHEN "0100" =>
      nextt<= "1000";
    WHEN "1000" =>
      nextt<= "0001";
         when others =>
         nextt<= "0001";
  END CASE;
  END PROCESS;
end Behavioral;
```

Appendix B – <<VHDL Test Bench Source Code>>

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

  ENTITY routertb is

  END routertb;

  ARCHITECTURE behavior OF routertb IS

    -- Component Declaration
   COMPONENT router is

 generic( n : integer :=7);
 port( datai1,datai2,datai3,datai4: in std_logic_vector(n downto 0);
       reset,wclk,rclk,wr1,wr2,wr3,wr4: in std_logic;
                datao1,datao2,datao3,datao4: out std_logic_vector(n downto
0));

 end component router;

    --Inputs

    signal reset,wclk,rclk,wr1,wr2,wr3,wr4 : std_logic := '0';
    signal datai1,datai2,datai3,datai4:std_logic_vector(7 downto 0);


         --Outputs
    signal datao1,datao2,datao3,datao4:std_logic_vector(7 downto 0);


```

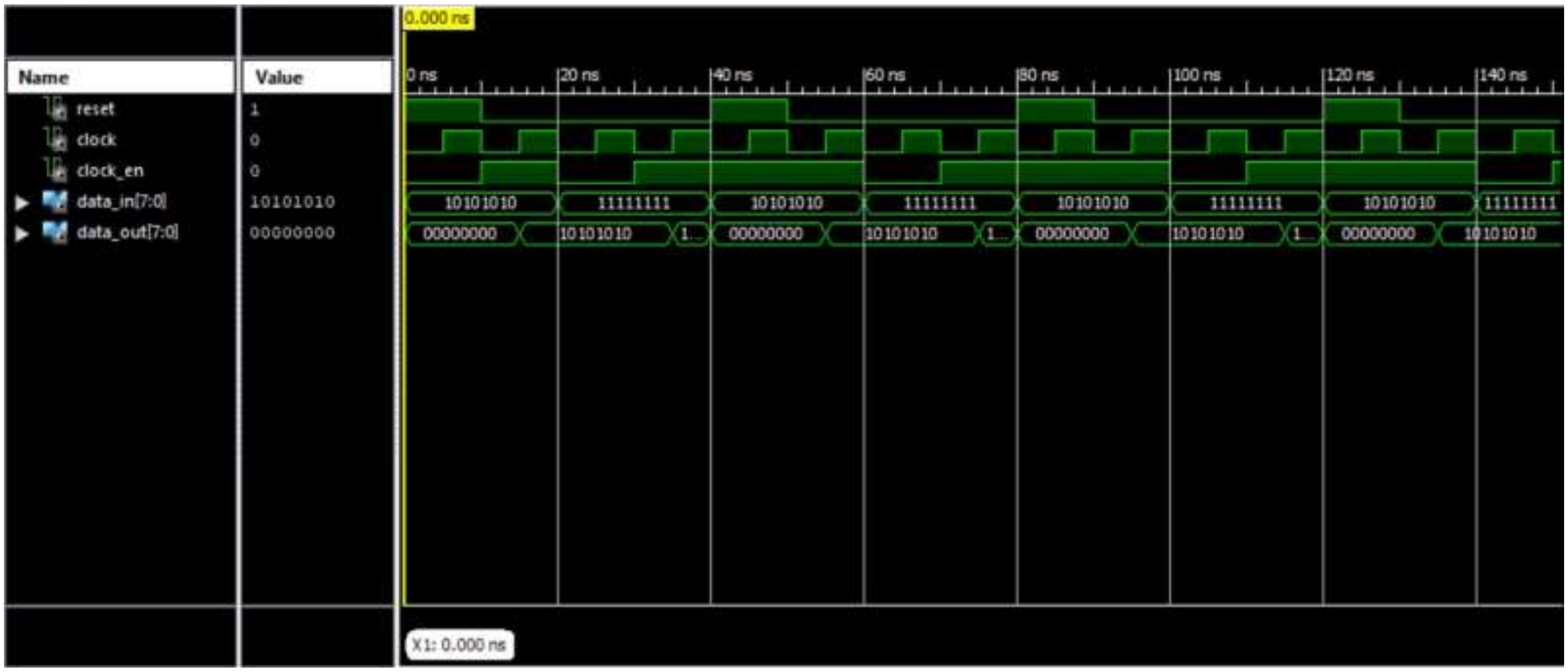
```vhdl
   -- Clock period definitions
   constant rclk_period : time := 10 ns;
   constant wclk_period : time := 10 ns;


  BEGIN

  -- Component Instantiation
         uut: router PORT MAP (
          reset => reset,
          rclk => rclk,
          wclk => wclk,
          wr1=> wr1,
          wr2 => wr2,
                        wr3=>wr3,
                        wr4=>wr4,
                        datai1=>datai1,
                        datai2=>datai2,
                        datai3=>datai3,
                        datai4=>datai4,
                        datao1=>datao1,
                        datao2=>datao2,
                        datao3=>datao3,
                        datao4=>datao4
        );
rclk_process :process
   begin
               rclk <= '0';
               wait for rclk_period/2;
               rclk <= '1';
               wait for rclk_period/2;
   end process;

   wclk_process :process
   begin
               wclk <= '1';
               wait for wclk_period/2;
               wclk <= '0';
               wait for wclk_period/2;
   end process;
  --  Test Bench Statements
     tb : PROCESS
     BEGIN
           reset<='1'; the round robin begins its cycle from the first rising edge of the read clk.
               wait for 10 ns;
--so here each RR has read from the first FIFO in each port
               reset<='0';
          wr1<='1';
               wr2<='1';
               wr3<='1';
               wr4<='1';
         datai1<="00011010";
               datai2<="11100001";
               datai3<="10000110";
               datai4<="01111111";
          wait for 10 ns;--so here each RR has read from the second FIFO in each port
          wr1<='1';
               wr2<='1';
               wr3<='1';
               wr4<='1';
               datai1<="11000100";
               datai2<="10111100";
               datai3<="11000100";
               datai4<="10111100";
               wait for 10 ns;
--so here each RR has read from the third FIFO in each port
          datai1<="11011011";
               datai2<="11000001";
               datai3<="11000110";
               datai4<="11111101";
                wr1<='0';
               wr2<='0';
               wr3<='0';
               wr4<='0';
     wait for 10 ns;
--so here each RR has read from the fourth FIFO in each port
          assert(datao4 = "01111111")
          Report("error reading from FIFO 16");
           wr1<='1';
               wr2<='1';
               wr3<='1';
               wr4<='1';
      wait for 10 ns;
--so here each RR has read from the first FIFO in each port, and in this rotation data were already stored
               assert(datao1 = "11000100")
               Report("error reading from FIFO 1");
               assert(datao3 = "00011010")
               Report("error reading from FIFO 9");
               datai1<="10000001";
               datai2<="10001011";
               datai3<="10000111";
               datai4<="10001111";
               wr1<='1';
               wr2<='1';
               wr3<='1';
               wr4<='1';
               wait for 10 ns;
--so here each RR has read from the second FIFO in each port, and in this rotation data were already stored
               assert(datao1 = "10111100")
               Report("error reading from FIFO 2");
               assert(datao2 = "11100001")
               Report("error reading from FIFO 6");
               reset<='0';
           wr1<='0';
```

```
                    wr2<='0';
                    wr3<='0';
                    wr4<='0';
              wait for 10 ns;--so here each RR has read from the third FIFO in each port, and in this rotation data were already stored
                    assert(datao1 = "11000100")
                    Report("error reading from FIFO 3");
                    assert(datao3 = "10000110")
                    Report("error reading from FIFO 11");
               datai1<="11011001";
                    datai2<="11000011";
                    datai3<="11000111";
                    datai4<="11111111";
                     wr1<='1';
                    wr2<='1';
                    wr3<='1';
                    wr4<='1';
         wait for 10 ns;
--so here each RR has read from the fourth FIFO in each port, and in this rotation data were already stored
                    assert(datao1 = "10111100")
                    Report("error reading from FIFO 4");
                    assert(datao2 = "11111101")
                    Report("error reading from FIFO 8");
                    assert(datao4 = "10001111")
                    Report("error reading from FIFO 16");
                    reset<='0';
               wr1<='0';
                    wr2<='0';
                    wr3<='0';
                    wr4<='0';
              wait for 10 ns;
--so here each RR has read from the first FIFO in each port, and in this rotation data were already stored
               datai1<="10000001";
                    datai2<="10001011";
                    datai3<="10000111";
                    datai4<="10001111";
                    wr1<='1';
                    wr2<='1';
                    wr3<='1';
                    wr4<='1';
                    wait for 10 ns;
--so here each RR has read from the second FIFO in each port, and in this rotation data were already stored


      END PROCESS tb;
   --  End Test Bench

   END;

```

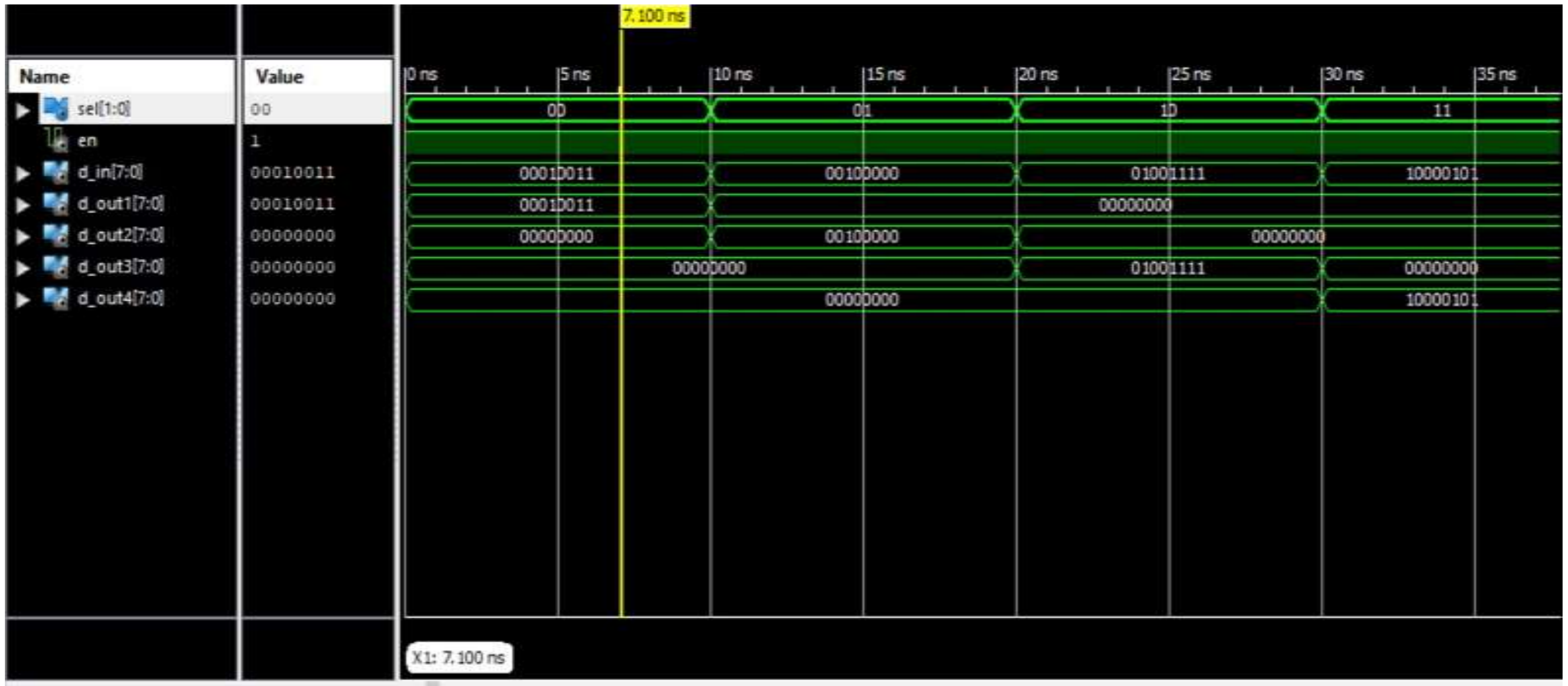
1.

Appendix C – <<Simulation Waveform Output>>

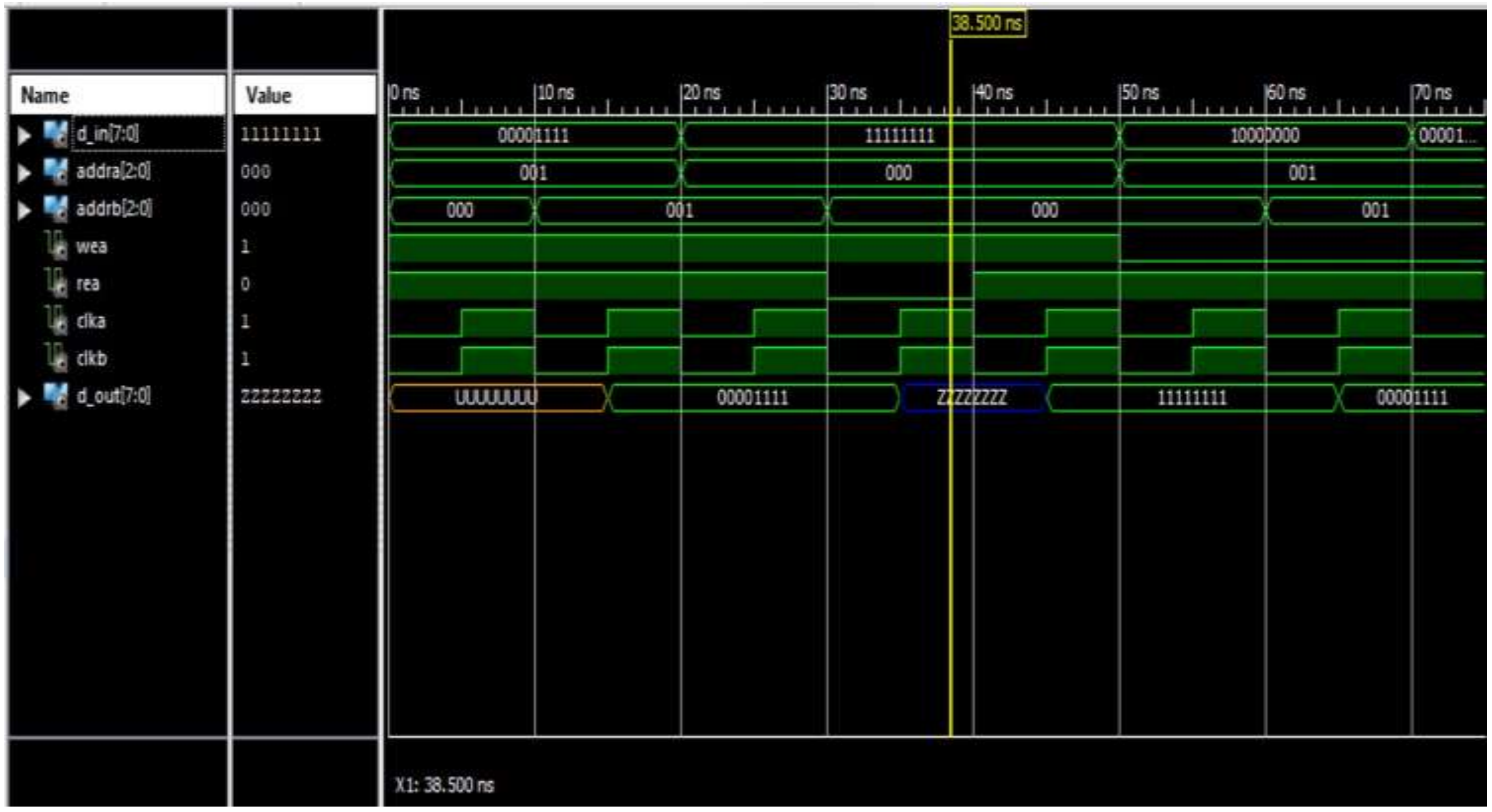
1. Register



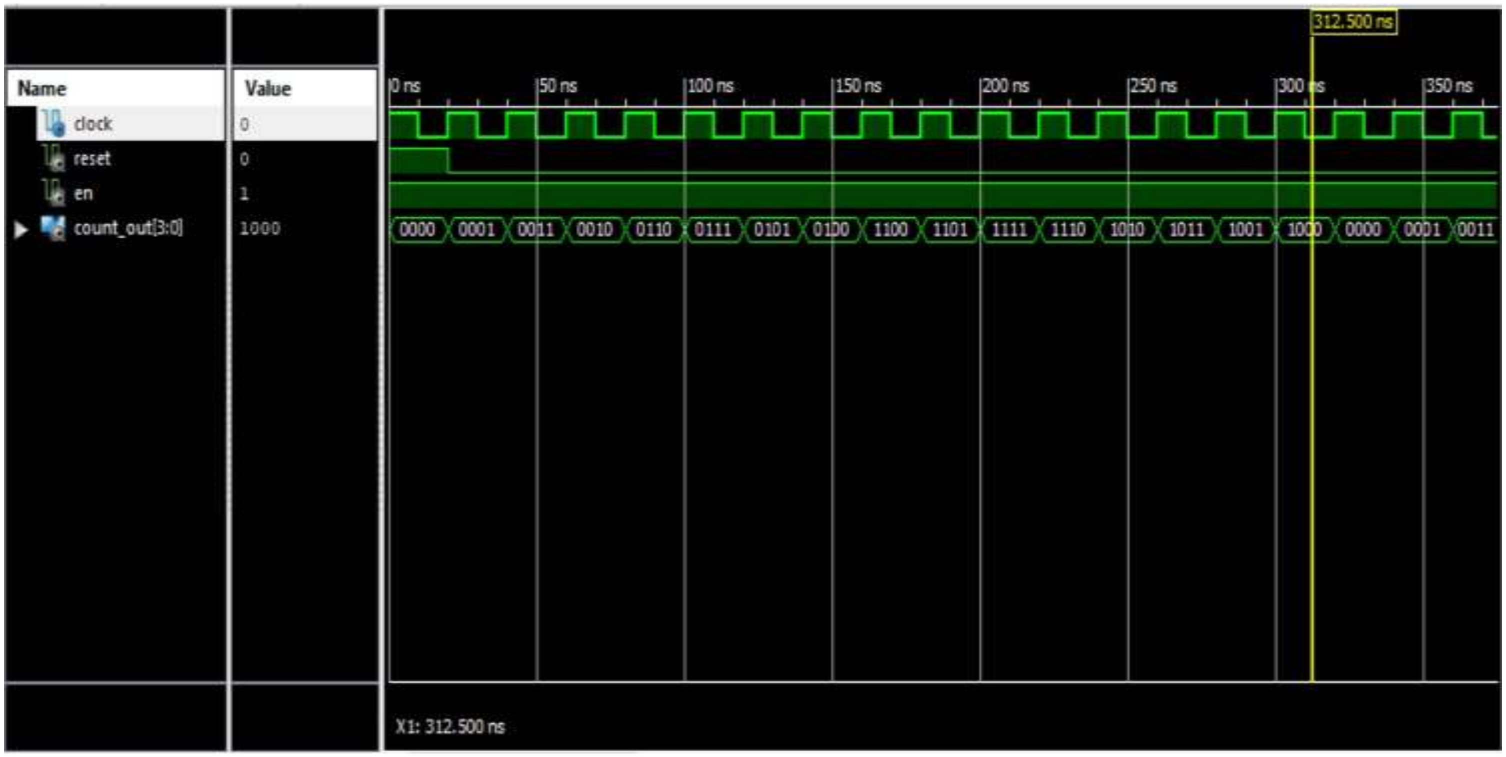
2. DeMux



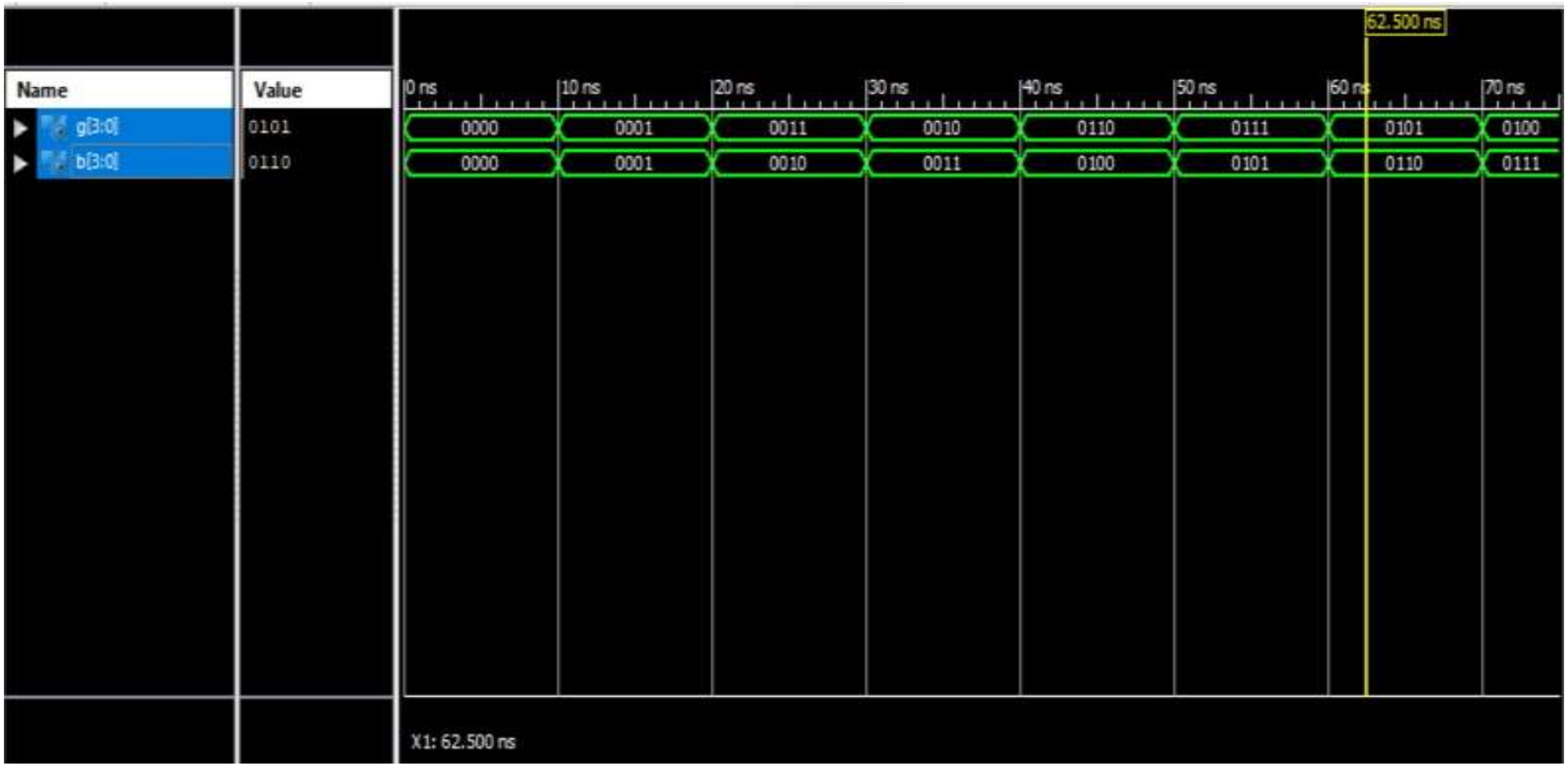
3. Block RAM



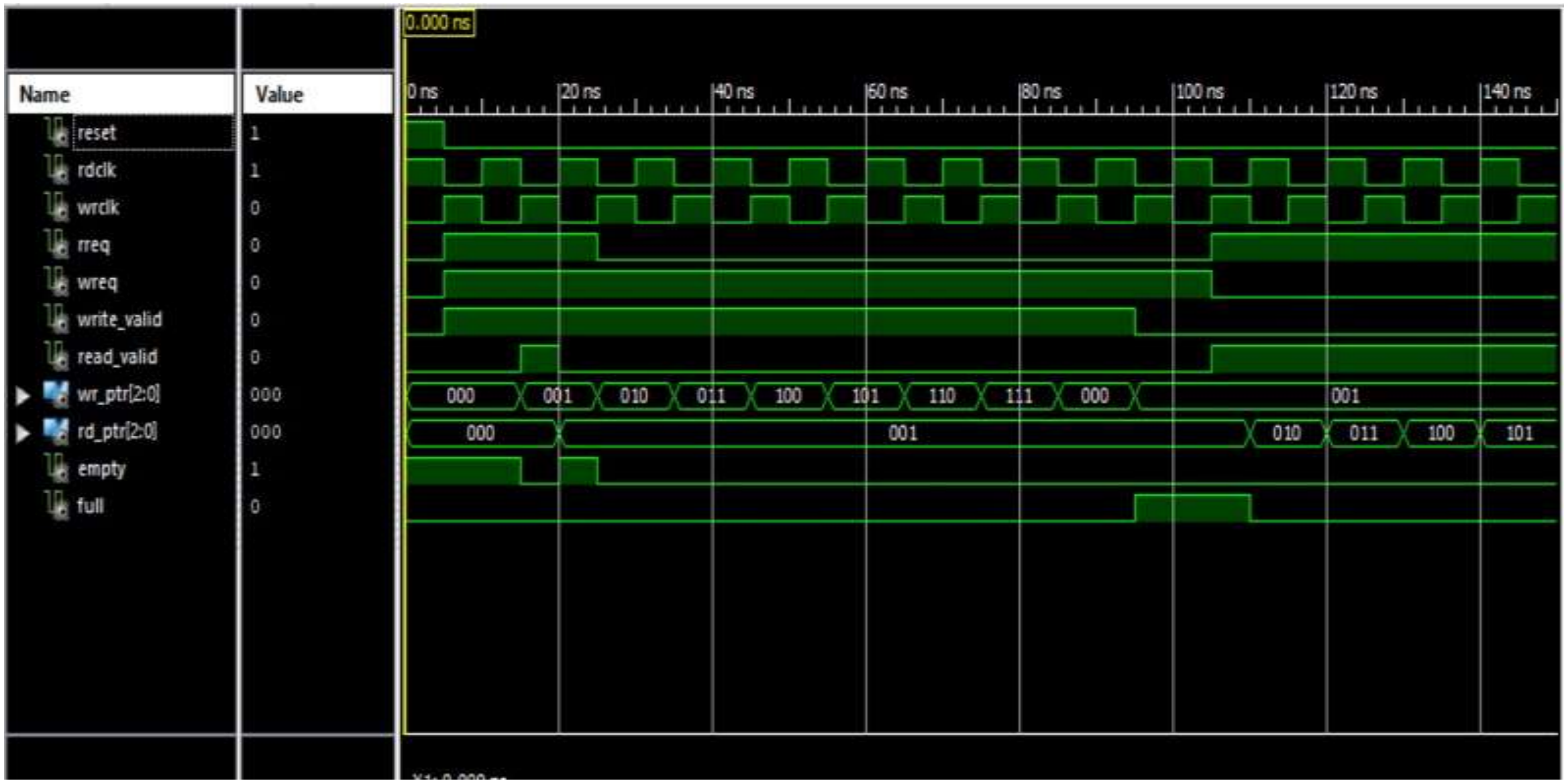
4. Gray Counter



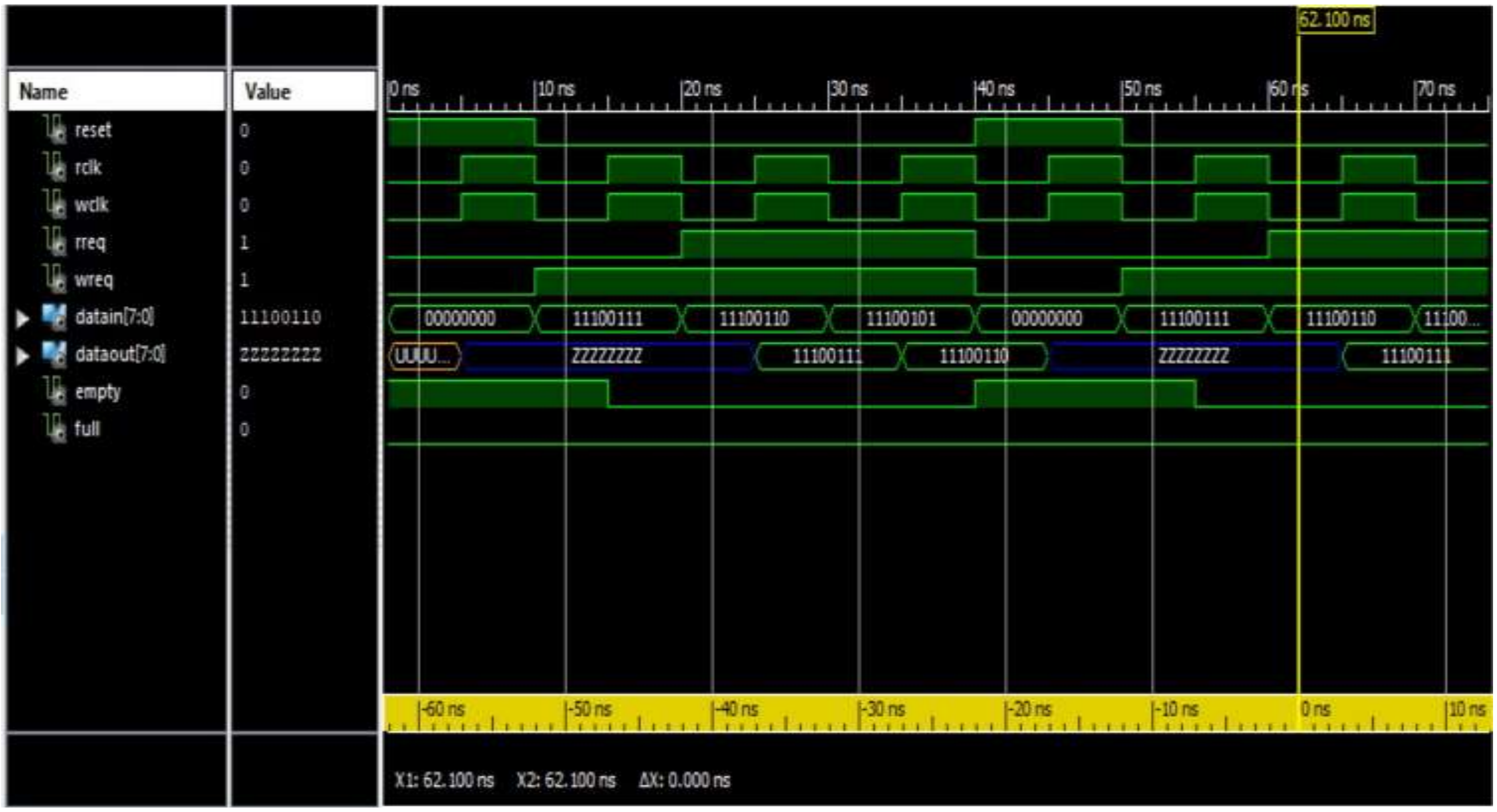
5. Gray to Binary Converter



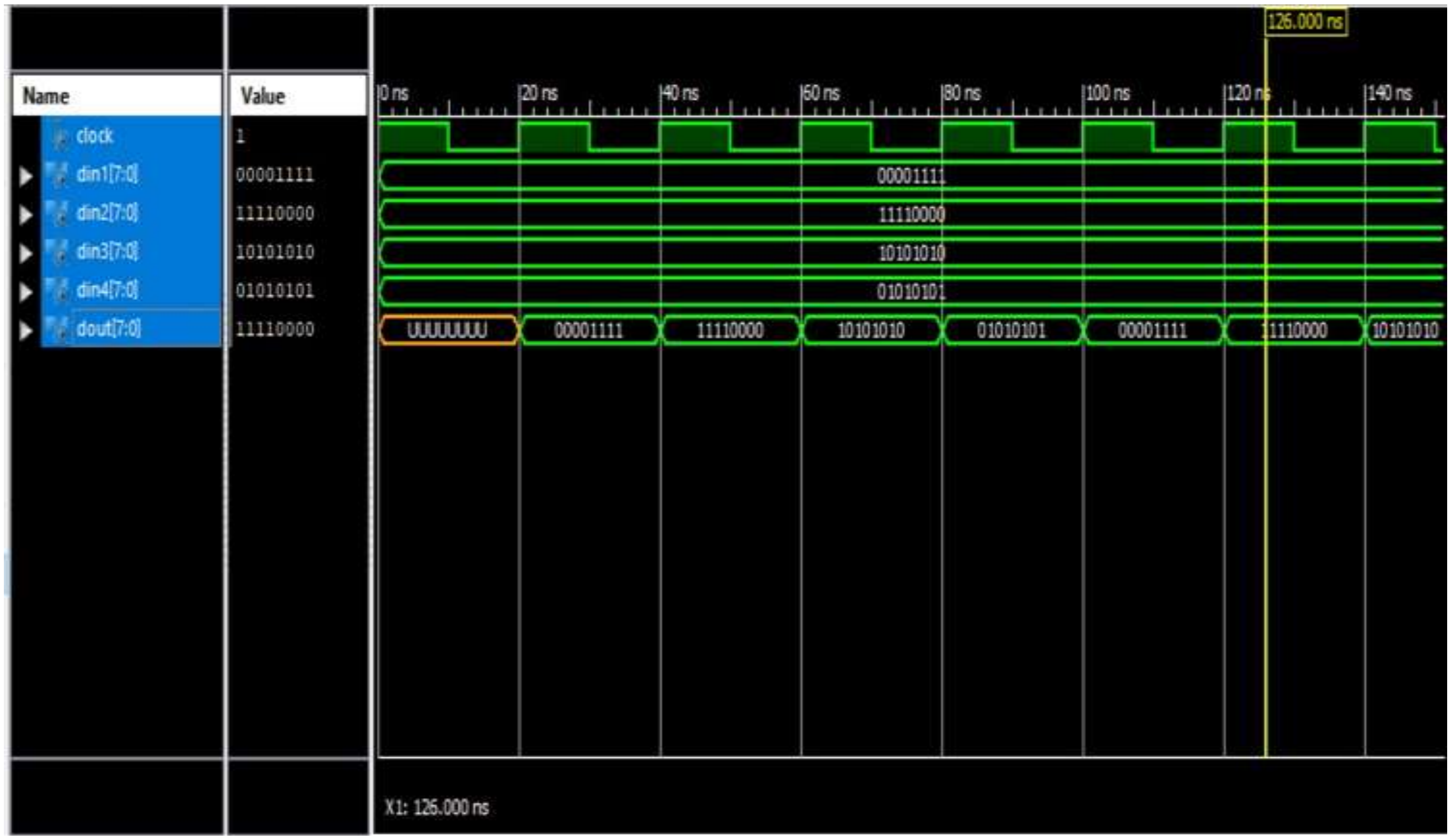
6. FIFO Controller



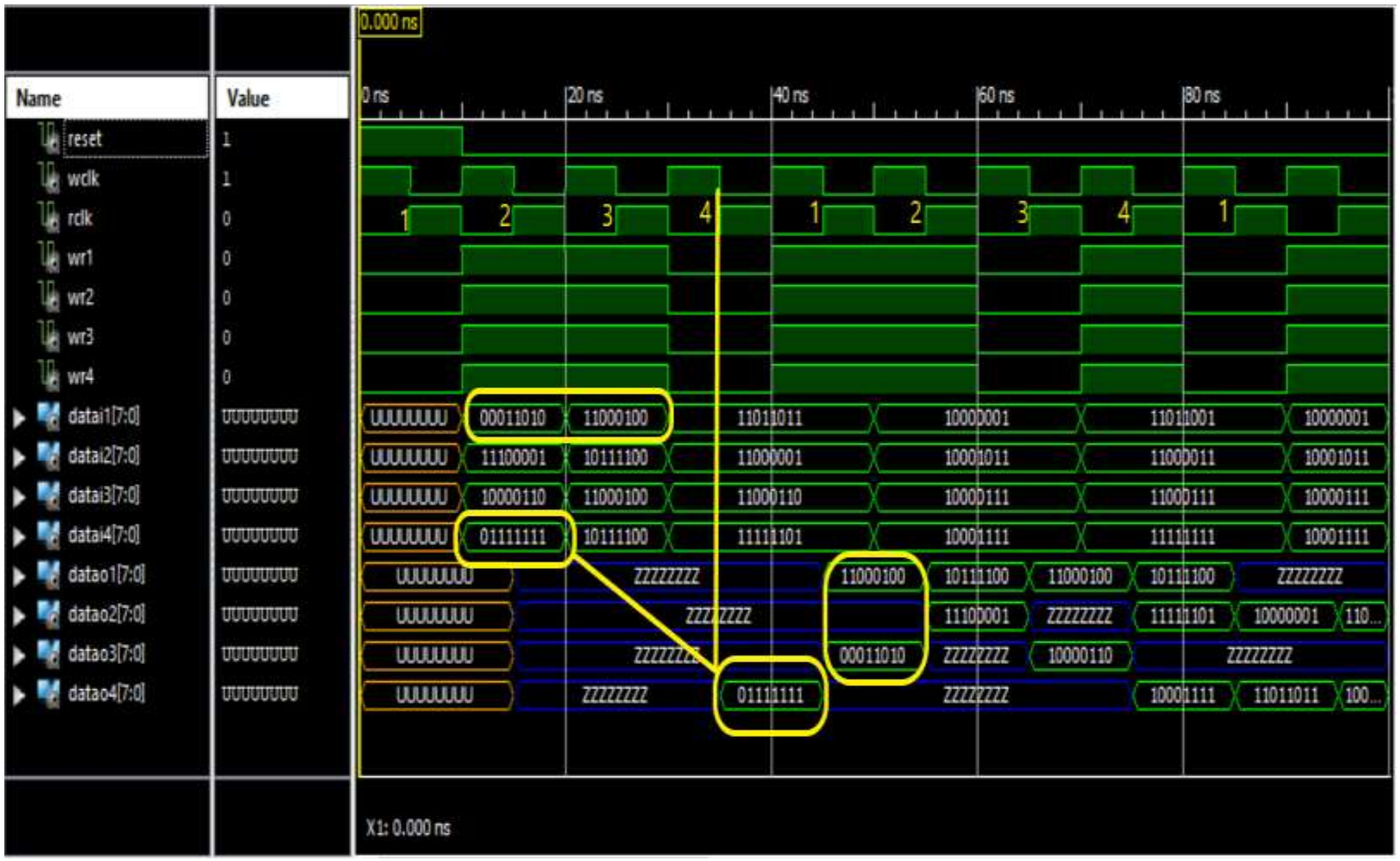
7. FIFO



8. Round Robin scheduler



9. Router


0.000 ns
Name
Value
0 ns
20 ns
40 ns
60 ns
80 ns
reset 1
wclk 1
rclk 0
wr1 0
wr2 0
wr3 0
wr4 0
datai1[7:0] UUUUUUUU
datai2[7:0] UUUUUUUU
datai3[7:0] UUUUUUUU
datai4[7:0] UUUUUUUU
datao1[7:0] UUUUUUUU
datao2[7:0] UUUUUUUU
datao3[7:0] UUUUUUUU
datao4[7:0] UUUUUUUU
1 2 3 4 1 2 3 4 1
UUUUUUUU 00011010 11000100 11011011 10000001 11011001 10000001
UUUUUUUU 11100001 10111100 11000001 10001011 11000011 10001011
UUUUUUUU 10000110 11000100 11000110 10000111 11000111 10000111
UUUUUUUU 01111111 10111100 11111101 10001111 11111111 10001111
UUUUUUUU ZZZZZZZZ 11000100 10111100 11000100 10111100 ZZZZZZZZ
UUUUUUUU ZZZZZZZZ 11100001 ZZZZZZZZ 11111101 10000001 110...
UUUUUUUU ZZZZZZZZ 00011010 ZZZZZZZZ 10000110 ZZZZZZZZ
UUUUUUUU ZZZZZZZZ 01111111 ZZZZZZZZ 10001111 11011011 100...
X1: 0.000 ns