CESS PROGRAM

CSE 312
ELECTRONIC DESIGN AUTOMATION

FINAL PROJECT

NoC Router Project

BY:

Youssef Maher	18P6713
Mazen Mostafa	18P2491
Mohamed Mokhtar	18P7232
Steven Sameh	18P9325
Dona Samir	18P7377
Nardine Yousry	18P6427

Table of Contents

1.0 Introduction 5
2.0 Design Flow 7
3.0 Literature Review 12
4.0 Design Implementation 16
4.1 Register 17
4.2 Demux 18
4.3 Gray counter 19
4.4 Gray to Binary Convertor 19
4.5 Ram 20
4.6 FIFO-Controller 21
4.7 FIFO 22
4.8 Round Robin 23
4.9 Router 24
5.0 Schedular design and FSM implementation 25
5.1 Schedular Design: 25
5.2 FSM Diagram: 25
5.3 FSM Type: 26
5.4 Timing Analysis: 26
5.5 Synthesis: 26
5.6 Critical Path: 28
5.7 FSM Implementation Styles 29
5.7.1 One Process 29
5.7.2 Two Processes 29
5.7.3 Three Processes 29
6.0 Testing and Simulation Results 31
7.0 Conclusion 34
8.0 Task Distribution 34
9.0 Appendix A 35
9.1 Register 35
9.1.1 VHDL source code 35
9.1.2 Total equivalent gate count 35
9.1.3 Timing analysis 36
9.2 Demux 36
9.2.1 VHDL source code 36
9.2.2 Total equivalent gate count 37
9.2.3 Timing analysis 37

9.3 Ram 37
9.3.1 VHDL source code 37
9.3.2 Total equivalate gate count 38
9.3.3 Timing analysis 39
9.4 Gray Counter 39
9.4.1 VHDL source code 39
9.4.2 Total equivalent gate count 40
9.4.3 Timing analysis 40
9.5 Grey to binary convertor 40
9.5.1 VHDL source code 40
9.5.2 Total equivalent gate count 41
9.5.3 Timing analysis 41
9.6 FIFO-Controller 41
9.6.1 VHDL source code 41
9.6.2 Total equivalent gate count 43
9.6.3 Timing analysis 43
9.7 FIFO 43
9.7.1 VHDL source code 43
9.7.2 Total equivalent gate count 45
9.7.3 Timing analysis 45
9.8 Round Robin 45
9.8.1 VHDL source code 45
9.8.2 Total equivalent gate count 47
9.8.3 Timing analysis 47
9.9 Router 47
9.9.1 VHDL source code 47
9.9.2 Total equivalent gate count 51
9.9.3 Timing analysis 51
10.0 Appendix B 52
10.1 Router Testbench 52
10.2 Warning process code 55
10.3 Test case 1 code 56
10.4 Test case 2 code 56
10.5 Test case 3 code 57
10.6 Test case 4 code 57
11.0 Appendix C 58
11.1 Test case 1 waveform 58

11.2 Test case 2 waveform 58
11.3 Test case 3 waveform 58
11.4 Test case 4 waveform 58
11.5 Demux: 59
11.6 Register: 59
11.7 Round Robin: 59
11.8 Ram: 59
11.9 Gray Counter: 59
11.10 Converter: 60
11.11 Controller: 60
11.12 FIFO: 60
12.0 References 61

Table of Figures

Figure 1 shows the main building blocks of the router 5
Figure 2 shows the IC design flow 8
Figure 3 shows the clock after the CTS.. 9
Figure 4 shows ASIC design flow.. 11
Figure 5 shows FPGA design flow 11
Figure 6 shows a bus system and a NoC system 12
Figure 7 shows the block diagram of 8-Bit register 17
Figure 8 shows the block diagram of Demux 18
Figure 9 shows the block diagram of Gray Counter 19
Figure 10 shows the block diagram of the gray to binary converter 20
Figure 11 shows the block diagram for RAM... 20
Figure 12 shows the block diagram of the FIFO controller....................................... 21
Figure 13 shows the block diagram of the FIFO unit.. 22
Figure 14 shows the block diagram of the Round Robin Scheduler......................... 23
Figure 15 shows the block diagram of the router 24
Figure 16 shows the FSM diagram.. 25
Figure 17 shows the scheduler's block diagram.. 27

1.0 Introduction

With the technological developments, a sheer number of devices can be integrated in a single chip. So, the communication between these devices becomes crucial. Because of the common bus Architecture in System-on-Chip (SoC), performance becomes slow to respond which limits processing speed, Network-on-Chip (NoC) is introduced.

NoC is an advanced design technique of communication network into SoC. It offers solution to the complications of traditional bus based SoC. NoC has shorter communication delay, as well as high scalability and flexibility. It achieves efficient inter-module communication as well. A router is the key component and named as the communication backbone in NoC. Moreover, it is being used in another computer networks applications.

Routers are used on a network for directing the traffic from the source to the destination. It coordinates the data flow which is very crucial in communication networks. Routers are intelligent devices that receive incoming data packets, inspect their destination, and figure out the best path for the data to move from source to destination, then delivers these packets to their corresponding output buffers. This process is called Routing.

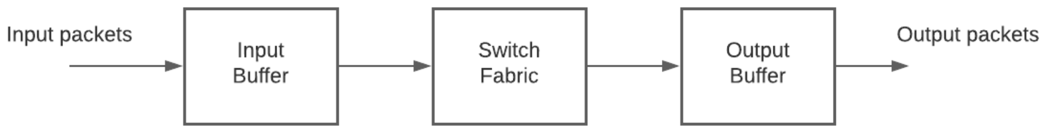


Figure 1 shows the main building blocks of the router

As shown in this diagram, once an input packet arrives at the router, packet is first collected if packet ready signal is high and stored in buffer then the routing decisions are taken and channel arbitration is made by the switch fabric, then the packet is put in an output queue scheduled to be buffered out through an output buffer from the appropriate output port.

Routing algorithms vary depending on headers and forwarding tables according to design requirements. Routers can be classified according to their switching fabric and the place of the location of buffers and queues within the router. Routers can be SoC with either common bus architecture or NoC as stated before. In addition, they can be implemented on a larger scale (wired and wireless routers) which are mostly used in homes and small offices.

To build a simple chip first we should define the functional description which is what is this chip supposed to do, then we define any design constraints, which are performance, power consumption, latency …etc. Then we take decisions on any constraint tradeoffs, what to choose and what to not, according to which is more important. Then we start to suggest possible implementations through Hardware description Language and select the best according to the constraints. A CAD Tool can be very helpful in this stage because it can help showing timing analysis for example. And then we start the actual implementation.

The aim of this report and project is to design, implement and test a simple router using VHDL. The desired outcome is to route data entering the router from four data in ports and inspect them from the data out ports.

2.0 Design Flow

Design flows are generally the graphic blend of EDA devices to perform the configuration of an "Integrated-Circuit". The Moore's Law must drive the whole IC implementation from synthesizable RTL languages like VHDL or Verilog to the layout-mask that uses CIF or GDSII design flow of a unit that uses fundamentally stand-alone placement, synthesis, and routing algorithms to a combined system and interpretation flows for the design closing. The difficulties of building interconnect suspension drove the design to a different method of creating and combining design closing devices. The RTL to GDSII stream experienced notable developments from till 2005. The advanced scaling of CMOS technologies significantly improved the purposes of the different design rounds. The absence of reliable predictors for the suspension has driven notable variations in modern design flows. New computing objections: variability, leakage power, and reliability will remain to expect meaningful adjustments to the design closing method. Several constituents explain what inspired the design flow from a kit of separate design actions to a completely integrated strategy and what additional changes that are coming to rant the most advanced objections.

There are *4* main steps of Design Flow:

1) **Design:** for the designers, they have three alternatives if they are operating hardware:

    a) Use VHDL or Verilog language
    b) Draw a schematic design that defines the project 3D. A variety of alternative design theories are examined to determine the quality of the executed design and an excellent accomplishment of the project.
    c) Use IPs-Integration

2) **Simulation:** There are different simulation phases; the functional simulation which is the step of writing the Test-Bench that is not significantly timing since all the parameters going through simulations do not have a connection with the technology that will run through it, thus the actual timing parameters of the gate is unknown. So, we have another simulation step called Post-Synthesis simulation. After the simulation, a netlist is created, so we apply the Post-Synthesis simulation on the netlist to determine the consequences of the gate and see if there is any violation. After this place and routing step is applied where all the components are put together in a physical place; this step is called post-layout simulation that checks whether the timing is correct or not. There is also a thermal analysis that checks if there are hot spots on the gates and the impact of the current flow with the heat dissipation.

3) **Synthesis, Analysis and Verifications:** its mission is to discover or generates a gate-level netlist that represents the circuit or hardware that needed to be implemented. After synthesis, a set of reports is created which allows the verification of synthesis, as timing violations or errors resulting from the technology or netlist is generated correctly.

4) **Layout, Manufacturing and Preparation**

The geometric description shown in Figure 2 is called an integrated circuit layout. This step is regularly divided into various sub-steps, that involves both the design, verification, and validation of the layout.

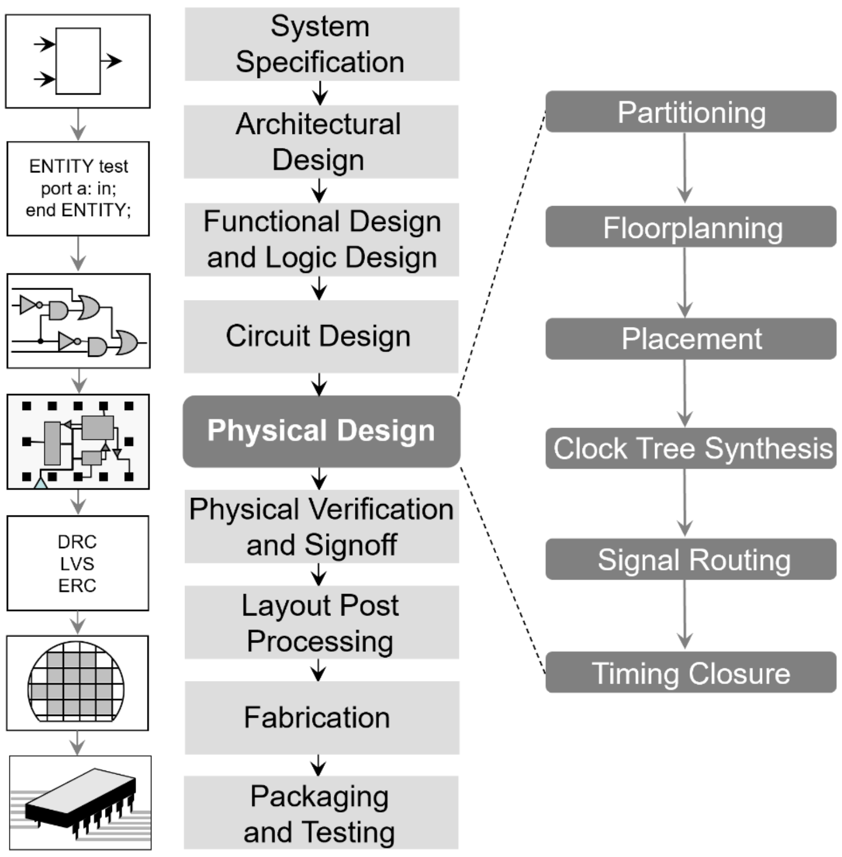


Figure 2 shows the IC design flow

Level one: is the ***System Specification*** that includes the practicability knowledge and measurement are assessment and function examination

Level two: is the ***Architectural Design*** that includes the construction of the system and improving the construction resident to the movement from one place to another.

Level three: is the ***Functional Design and Logic Design*** that includes

i. Digital Design, Simulation, Verification and Layout Circuit Design

Level Four: The ***Circuit Design*** that includes

i. Digital Design Synthesis
ii. Design for testing
iii. Design for Manufacturability (IC)

Level Five: The ***Physical Design*** that includes

i. *Partitioning*: It is to break the circuit such that the amount of bonds within partitions is reduced, so separating the chip into little pieces or blocks. This is arranged essentially to classify various functional blocks as well as to getting placement and routing; more obvious and manageable. Partitioning is done in the RTL design stage at the designer partitions the whole design into sub-blocks and next returns to design every module. Those modules are joined mutually in a central module called the Top-Level module. This type of partitioning is regularly assigned to as Logical-Partitioning.

ii. *Floor-planning* is a process of classifying fabrications that should be located near mutually and designating a place for them to satisfy the seldom contradictory purposes of free area, expected production, and the thirst to have everything stuck together. Based on the operation of the scheme of the hierarchy, a fitting floorplan is determined. Floor-planning demands into the record the macros applied in the memory and IP cores and their placement and routing possibilities, and the region of the whole design. Floor-planning additionally defines the IO construction and perspective proportion of the design. A poor and faulty floorplan will drive to wastage of die space and routing blockage. In various layout methodologies, acceleration and space are the points of trade-offs, due to restricted routing devices. Optimizing for the smallest space provides the design to utilize less resources, and for larger contiguity of the parts of the design. This directs to smaller interconnect objectives, less routing resources used, active end-to-end signal pathways, and yet more durable and more logical area and route events. A common command, data-path regions profit the most from floor-planning, whereas state machines, random logic, and additional non-structured logic can securely be neglected to the placer section of the place and route, models of the structures the design the data-path are Counters, Multiplexers, Adders, and Registers.

iii. *Placement*: Placement manages RC benefits from Virtual Router (VR) to measure timing. VR is the lowest Manhattan length within 2 pins. VR RCs are more precise than wireless lifecycles RCs. Placement is done in 4 optimization stages: Pre-placement optimization, placement optimization, post-Placement optimization before and after clock tree synthesis.

iv. Clock Tree Synthesis: Its purpose is to reduce insert lag as well as the skew lag. The Clock tree cycle is not created before, but behind CTS so the slack must develop as in Figure 3 The clock tree starts through **[.sdc]** which determines the clock source and terminates at pause pins of the flops.

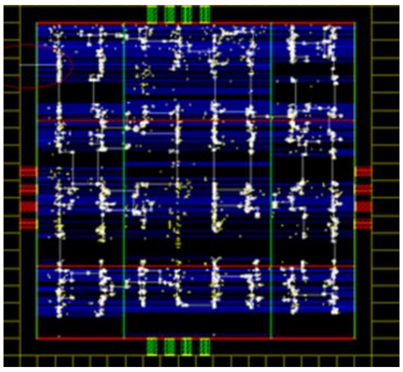


Figure 3 shows the clock after the CTS

v. *Routing:* Various restrictions are necessitated consideration throughout the routing, such as the wire length, DRC, and timing. There are a pair of sorts of routing in the physical design; the detailed routing arranges the original joints and links, while the global routing and the detailed routing. Global routing designates routing supplies that are utilized for joints and links, as well as the track designation for a special net.

Level Six: The ***Physical Verification*** that includes verification of timing and critical paths.

Level Seven: The ***Layout Post Processing*** usually terminates the physical design and verification. It transforms the physical layout into mask data, which comprises: the Chip finishing and generating a net layout.

The NoC Router is achieved in the following levels:

i. Requirement analysis
ii. Design entry
iii. Implementation of modules
iv. Test Bench
v. Simulation
vi. Block diagram generation
vii. Timing analysis and verification

The CAD tools are applied to develop the chip design flow:

1. Xilinx
2. Electronic Design Automation Software
3. VHDL or Verilog
4. LASI is PC-based CAD program used for the design of the dynamic layout of integrated circuits

Here is a small comparison between ASIC flow and FPGA flow:

Developers and designers prefer a further uncomplicated and modest design method. Accordingly, in the primitiveness of design flow, FPGA is less complex than ASIC design flow. FPGA is the abbreviation of Field Programmable Gate Array. It is in straight racing with ASIC chip technology. FPGA is a chip that can be programmed and reprogrammed to operate various purposes. It is adjustable, adaptable, smaller time to business, as well as it is reprogrammable Moreover, a single chip is composed of many parts called logic blocks that are associated together. On the Other side, ASIC is extra concerned with design flow since it cannot be re-programmable and demands expensive EDA devices for the design.

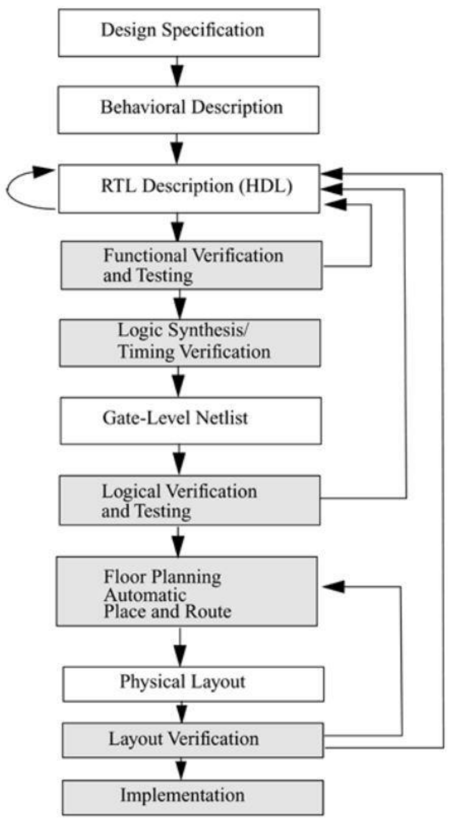

Design Specification
Behavioral Description
RTL Description (HDL)
Functional Verification and Testing
Logic Synthesis/ Timing Verification
Gate-Level Netlist
Logical Verification and Testing
Floor Planning Automatic Place and Route
Physical Layout
Layout Verification
Implementation

Figure 4 shows ASIC design flow

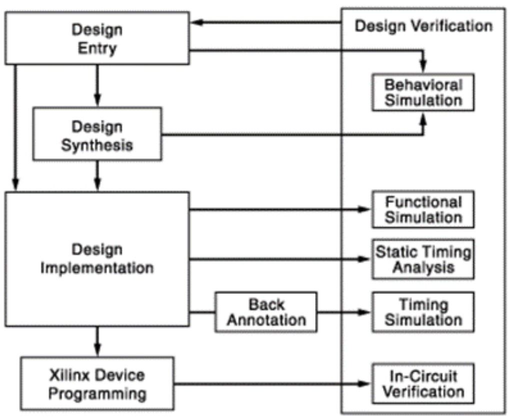


Figure 5 shows FPGA design flow

3.0 Literature Review

As technology advanced, more complex systems have been implemented into singular chips, these systems are known as “System on a Chip” (SoC for short). This increase in complexity within a chip has caused the need for on-chip communication to arise. One solution to this issue is the introduction of “Network on a Chip” (NoC for short), this method allows the communication between processors and memory elements, this is done by the exchange of data packets (a concept previously pioneered in the computer networking field). This method is preferred over establishing a bus system as in a system with multiple nodes this bus can become a bottleneck for the speed of communication. Additionally, in bus systems, each element connected to the bus introduces additional parasitic capacitance. Shown below is a system using a bus architecture versus a NoC system. [1]

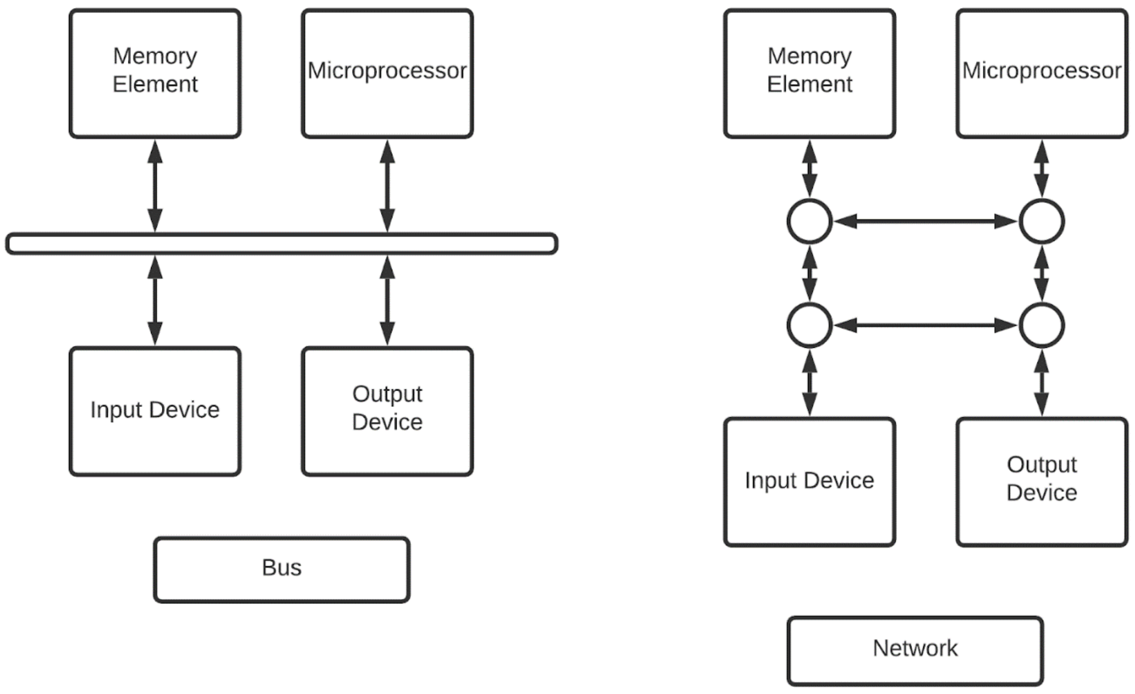


Figure 6 shows a bus system and a NoC system

When it comes to NoC design three criteria are of great importance; these are how switching is done, how routing is done and finally the topology of the system. There are two major switching techniques: circuit switching and packet switching, packet switching is more frequently used as it can potentially simultaneously service multiple nodes. Topology is defined as the way the nodes within the system are placed and connected, multiple topologies have been used over the years but the most prevalent being mesh, torus and octagon with 2-D mesh being the most used topology among them as it fits the 2-D nature of the chip onto which the system is implemented. Finally, the routing algorithm is the method by which a packet moves between the source and the destination and depending on this method the routing algorithm can be described as either deterministic or adaptive. With deterministic algorithms, the path of the packet is determined purely on the source and target addresses while adaptive algorithms also take into consideration the state of the network to determine the optimal path. [1]

The most important part of a NoC system is the router as it directs the traffic from the source to the destination and as such should be designed with utmost attention. [1] To that end, many different implementations of a NoC router have been introduced, in this part of the report, some of these implementations will be reviewed.

The first implementation to be reviewed is the implementation provided by Chandravanshi, R., & Tiwari, V. in their article “VHDL Based NOC Router Architecture” found in the International Journal of Innovative Trends. This implementation proposes an architecture consisting of several 8-Bit registers that store the input data, an equal number of demultiplexers that move the input data to their appropriate output ports, a FIFO (first-in-first-out) unit which is made up of a RAM (Random Access Memory) component and a FIFO control component. The FIFO control regulates the write and read operations to the RAM component by checking whether read and write requests are valid or not (an invalid read request is one that attempts to read from RAM while the FIFO is empty, while an invalid write request is one that attempts to write while the FIFO is full, and finally a scheduler unit that follows the round-robin algorithm to select which active packet-carrying data flow is allowed to transfer its packets, then as per the round-robin algorithm the source of the packets that was serviced will be at the end of next round of scheduling. [1]

This implementation shows a simple way of implementing a NoC router, however, the scheduling algorithm is somewhat primitive thus decreasing the overall performance of the NoC. Additionally, this implementation causes the output to be delayed before it becomes error-free by an amount of time proportional to the number of incoming packets.

The next implementation to be examined is the one provided by Wanjari, M. M., Agrawal, P., and Kshirsagar, R. V. from the International Journal of Computer Applications in their article Design of NoC Router Architecture using VHDL. This implementation’s architecture consists of three major components: a virtual channel, arbiter, and a crossbar switch (each will be explained in further detail later). [2]

This implementation describes a router with several input ports and output ports, a local port for accessing a processing element, as well as a logical block that determines how the packets will be routed through the NoC. When a packet is to be routed by this router it is first stored in a buffer till the routing algorithm sorts the path by which this packet will travel, once the path is sorted the crossbar switch physically connects the input port to the intended output port described to it by the arbiter which determines said output port depending on the priority level assigned to each input port, additionally the previously described buffer on each of the input ports is a virtual channel this type of buffer improves network throughput and provides flexibility while decreasing how impactful blocking is. [2]

In conclusion, this router provides lower latency and can be implemented within a smaller area than the previously discussed router, however, due to the use of the fixed priority arbiter the number of requests is somewhat limited and additionally lower priority requests can await processing for very long periods of time thus this implementation's performance would benefit from implementing an arbiter which enables the quick processing of even low priority requests such as round-robin. [2]

The next implementation to be analyzed is one provided by Bahmani, M., Sheibanyrad, A., Petrot, F., Dubois, F., and Durante, P. in their article "A 3D-NoC Router implementation Exploiting Vertically-Partially-Connected Topologies" from the IEEE Computer Society Annual Symposium. Unlike the previously described 2D NoC systems, this article outlines the implementation of a 3D NoC router (i.e., NoC with several tiers of 2D topologies placed on top of each other, within this implementation each tier follows the mesh topology, but each tier can vary in size and technology). These vertical layers can communicate with each other as they are vertically-partially connected via having some of the routers within each tier be dedicated to routing packets to upper and lower tiers while others simply route within the tier itself. This vertical separation can cause any packet routing to be separated into two categories, a category where the source and destination belong on the same vertical tier, and a category where the source and destination exist on different vertical tiers. [3]

For routing within the same vertical tier any normal routing algorithm can be used to perform the routing. However, when the routing is done between different vertical tiers, then a more sophisticated algorithm is needed, as there is no guarantee that the router that a packet arrives at is one capable of moving the packet upwards or downwards. And as such this article proposes an algorithm called "Elevator-First", in this algorithm a router without up/down ports first routes the packet to a router that does have up/down ports (called an elevator), the elevator then sends the packet to the desired tier where it is then routed to its destination. This routing is done by having the first add a temporary header to the packet which dictates the address of the elevator that the packet first needs to be routed to, and has two flags, the first flag indicates that the current header is a temporary one and the second dictates which direction the packet should travel to (Upwards or downwards). After the elevator routes the packet to the upper or lower tier, an intermediate router then checks to see if the destination of the packet lies on its tier, if not then the previous step of adding a temporary header to the packet is repeated until the intended tier is reached. Once the tier is reached a traditional routing algorithm is used to route the packet to its destination within the tier. [3]

In summation, this implementation provides a way of expanding a NoC without increasing the cross-sectional area of the chip by increasing the size in the Z-axis instead. However, this expansion does not come free of cost as the "Elevator-First" routing algorithm causes the average path travelled by the packet to be longer than the physical distance between the source and destination, as packets will sometimes need to be routed to intermediary elevators to be moved to the destined tier, and then routing within the tier itself must also be conducted. This longer path can cause latencies for the packets as well as consuming additional energy. Furthermore, the smaller the number of elevators per tier the higher the power consumption of the NoC as on average the packets will have to be routed to further away elevators before being sent to the destination. [3]

For this implementation, the router can have a varying number of ports of various types. For example, the router can have only 2D ports (i.e., ports to route to the east, west, north, and south) or could have 2D ports in addition to one 3D port such as the up and down ports or could have 2D ports and both 3D ports.

4.0 Design Implementation

Module	Functionality
Register	Synchronize input and output data with clock.
Demux	Send input data to chosen address destination.
Block Ram	Memory element used to read and write data.
Gray Counter	Generate read and write addresses.
Gray to Binary Converter	Convert output of grey counter to binary.
FIFO-Controller	Validate read and write request before being send to ram.
FIFO	store data and read data in First-in/First-out order.
Round Robin Schedular	Map output register to different queue every rising edge clock.
Router	Read data entered and assign to required destination port.

4.1 Register

Register will hold the input data until rising edge of the clock, then it will map the input data from input port to output port. In order to synchronize only data change at rising edge, so no data will be lost due to being overwritten by new data. See section 9.1 for implementation using VHDL.

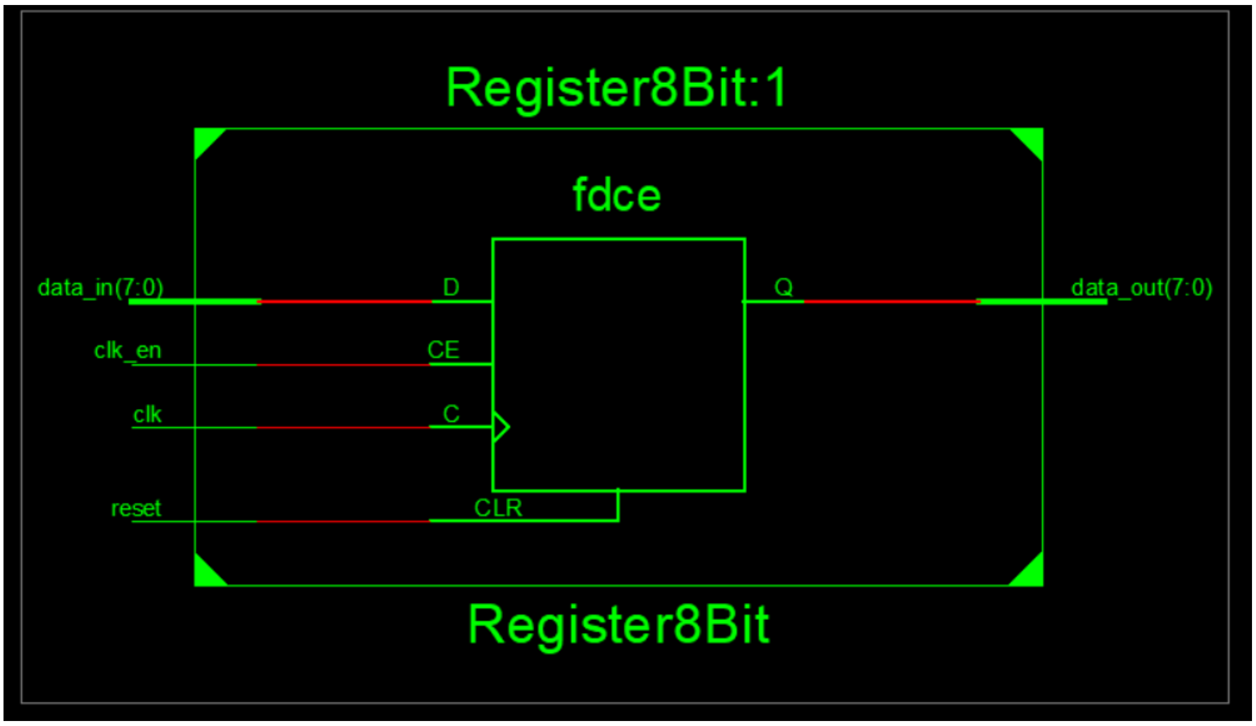


Figure 7 shows the block diagram of 8-Bit register

4.2 Demux

Data packets will have the least two significant bits as destination address. We have four output ports, then the Demux will connect with four different queues (FIFO) each FIFO is mapped with different output port from the Demux. Then we use the least two significant bits as a selection to Demux to map data into the right queue. See section 9.2 for implementation using VHDL.

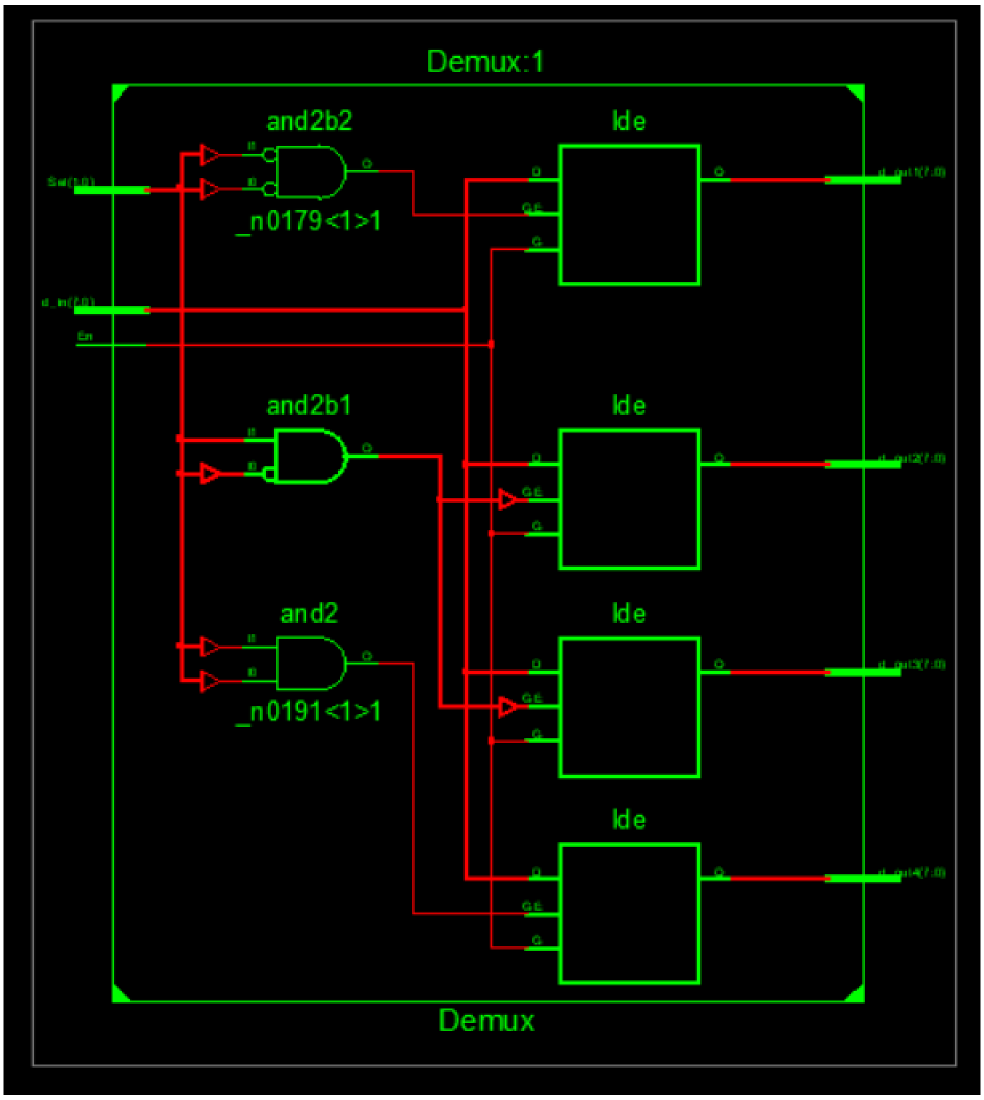


Figure 8 shows the block diagram of Demux

4.3 Gray counter

Grey counter is used to generate read and write addresses, so we can keep track of addresses that have been written into ram, so we will not overwrite data, and keep track of addresses that have been read from ram to give chance to ram to rewrite new data on same place. Also, it helps to keep track when ram will be full or empty.

Then we need to use two grey counters inside FIFO controller, one for read address and one for write address. See section 9.4 for implementation using VHDL.

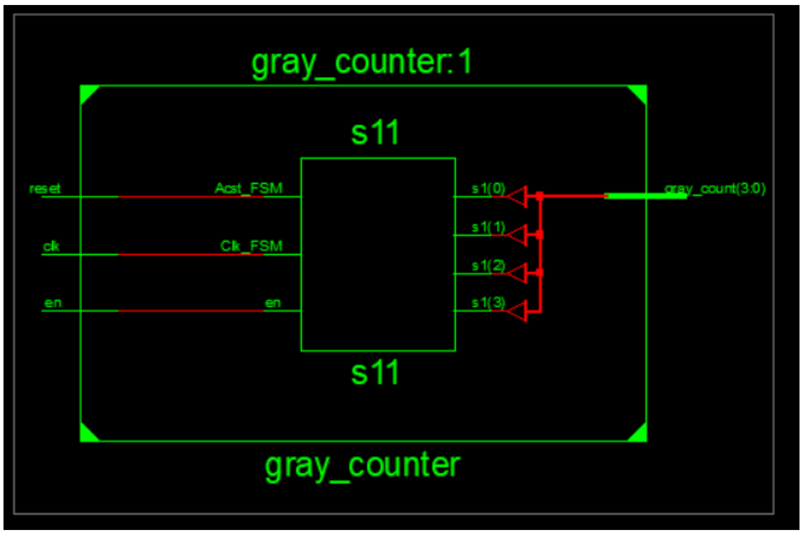


Figure 9 shows the block diagram of Gray Counter

4.4 Gray to Binary Convertor

Grey counter will generate numbers of sequence (0,1,3,2,6,7… in binary) so we need to use grey to binary convertor to convert this sequence to (0,1,2,3,4,5….) As we use to generate output to specify addresses of read and write in ram. Ram should write and read sequentially, as a result, we need to use two Gray to Binary Convertors, one for write grey counter and one for read grey counter. we use one for read as we should read in same sequence as we write to apply the idea of First in, Fist out. See section 9.5 for implementation using VHDL.

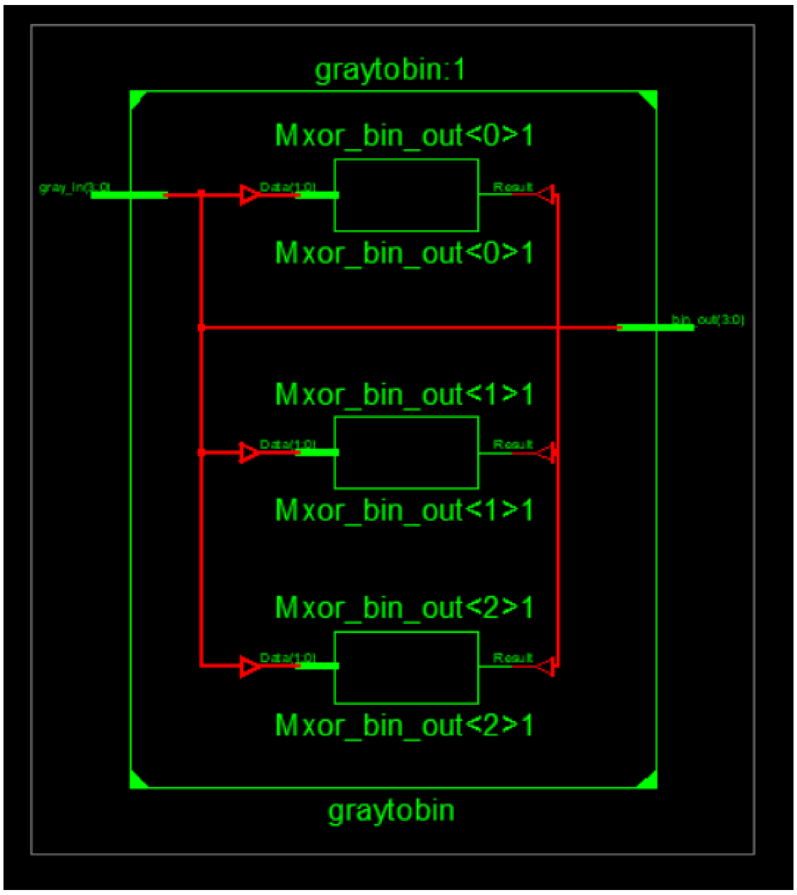


Figure 10 shows the block diagram of the gray to binary converter

4.5 Ram

Ram is used to store data. On rising edge of read clock, ram will check if it has a read request, it will output data from requested address, if read request is false, then the output will be high impudence. Also, when rising edge of write clock, ram will check if it has write request, ram will write data in requested address. See section 9.3 for implementation using VHDL.

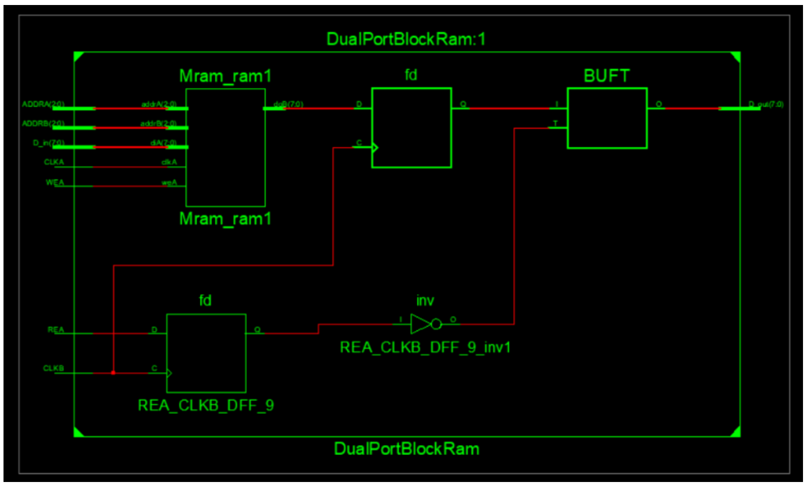


Figure 11 shows the block diagram for RAM

4.6 FIFO-Controller

FIFO-controller is responsible for to validate any read and write request coming before it sends a request to the ram, and then generates an address for read or write. It also checks if the ram is full or empty. We used grey counter connected with grey to binary convertor to generate addresses for read or write. FIFO-controller uses two separate counters each one is associated with different convertor one for read addresses and one for write addresses, counter will be enabled only if a request is valid to generate new address.

With each address generated, it checks if the ram became empty or full, by comparing the first three bits, if read address is the same as write address then it checks the Most significant bit If they are the same then it is empty, else It is full. Then it checks the request validity, write request can only be valid if the ram is not empty and read request can only be valid if the ram is not empty, both working on falling edges of the corresponding clocks. See section 9.6 for implementation using VHDL.

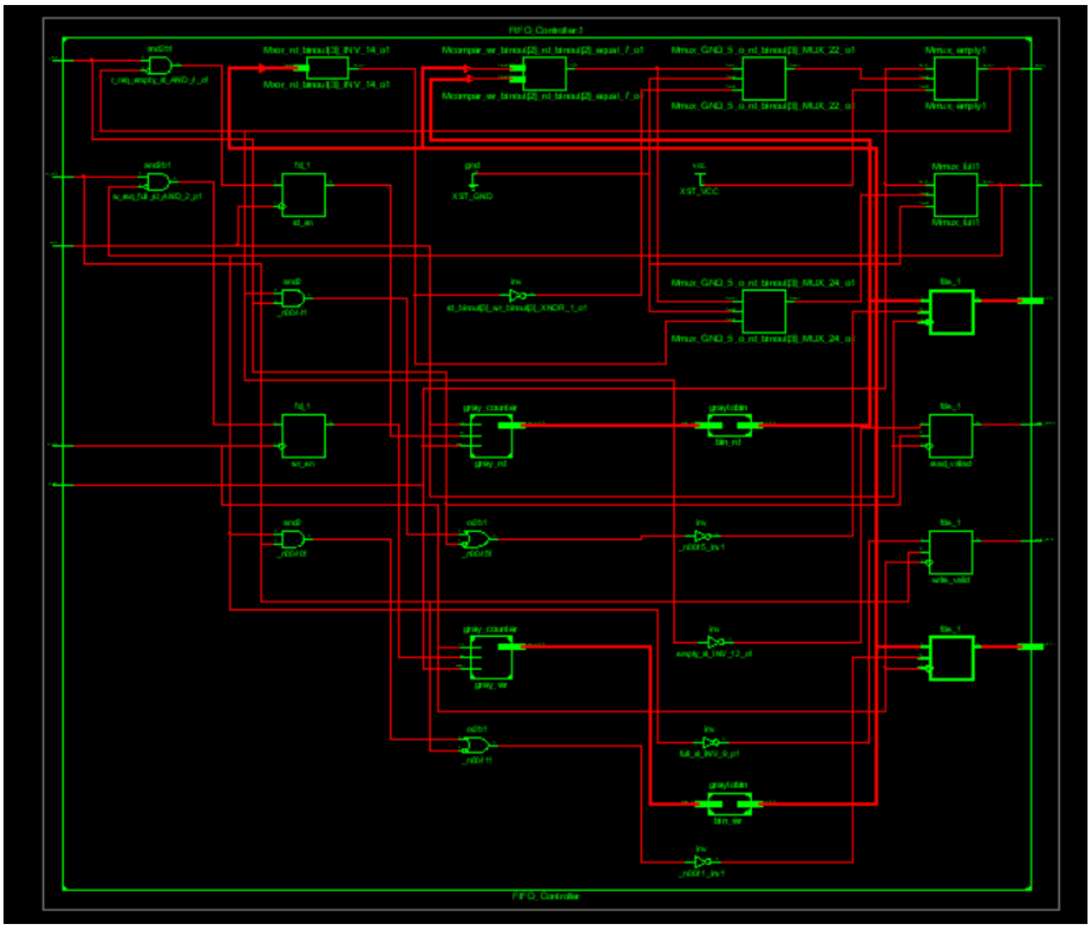


Figure 12 shows the block diagram of the FIFO controller

4.7 FIFO

FIFO connects FIFO-controller and ram together. To synchronize between FIFO-controller and Ram, we let FIFO-controller working on falling edge and ram working on rising edge, as a result, we will avoid delays as FIFO will only take one clock cycle. See section 9.7 for implementation using VHDL.

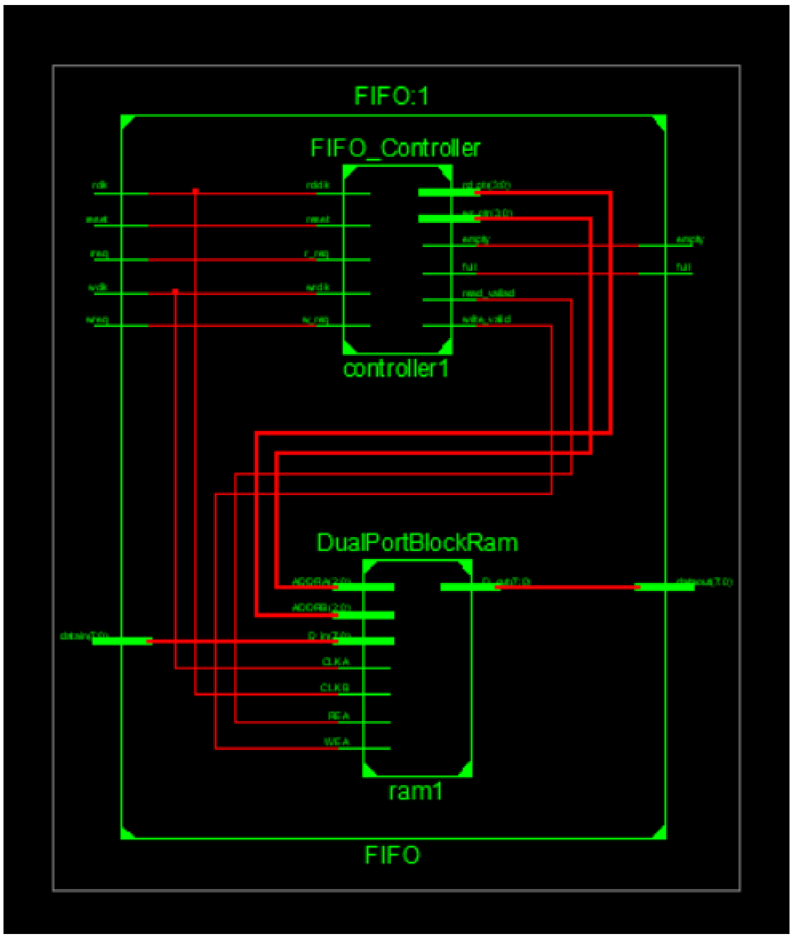


Figure 13 shows the block diagram of the FIFO unit

4.8 Round Robin

Round robin is implemented by using FSM. It is connected to four different FIFOs each one is connected to different input port. Round robin will change current state to next state synchronized with rising edge, then the output will be mapped to different queue at every rising edge. See section 9.8 for implementation using VHDL.

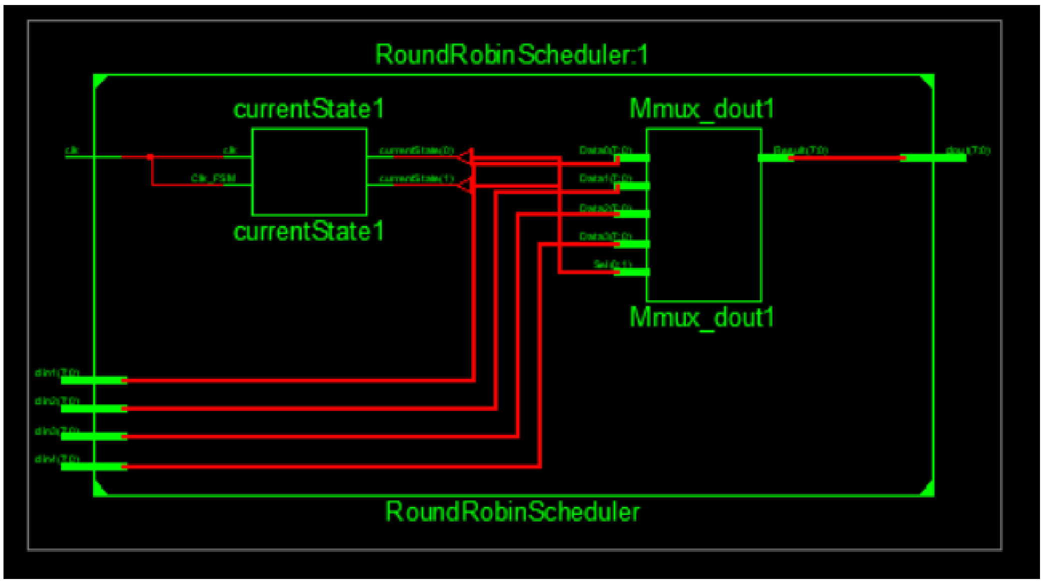


Figure 14 shows the block diagram of the Round Robin Scheduler

4.9 Router

Router has four inputs ports, then it needs four registers. Register clock is connected to write clock and write request is connected to router write enable. Then we connect each register output port with a Demux, each output port of a Demux is connected to FIFO. data in and write request checks on the least two significant bits if they are matched, then queue destination will be true. Use read signal connect with FIFO read request each FIFO will have read request = 1 only one time every three clock cycles. output port of each FIFO is connected to round robin. round robin clock is connected to read clock also output ports of round robin are connected to registers, in order to synchronize output of router with rising edge read clock. See section 9.9 for implementation using VHDL.

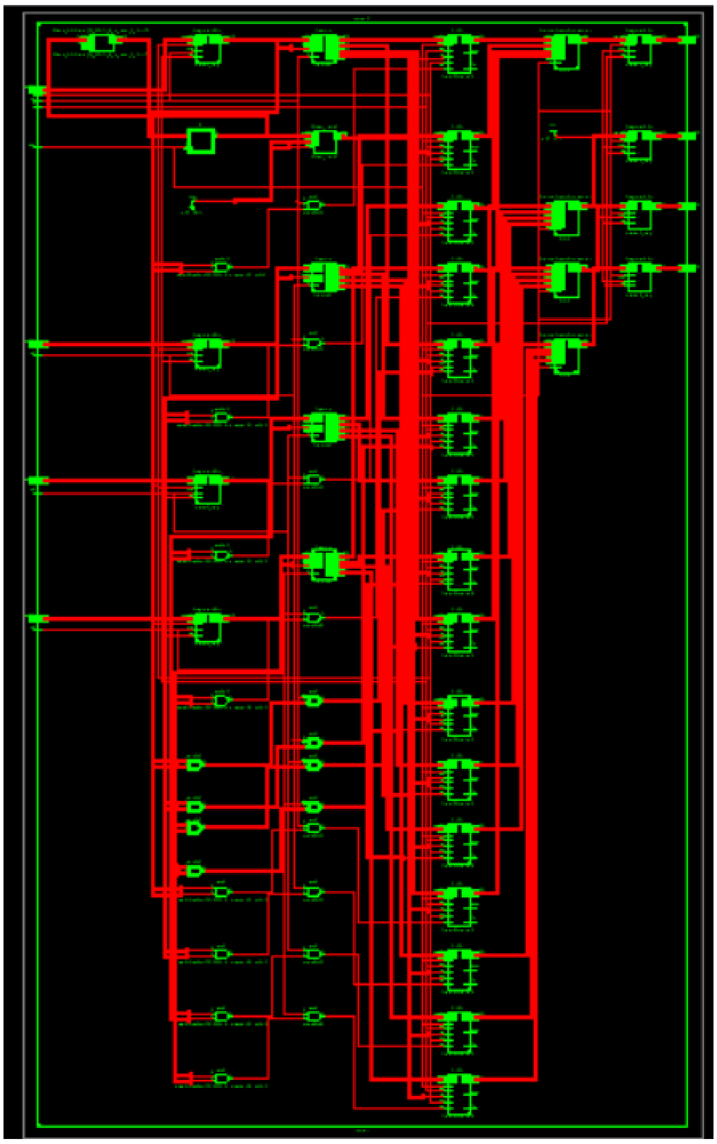


Figure 15 shows the block diagram of the router

5.0 Schedular design and FSM implementation

5.1 Schedular Design:

The Scheduler is implemented as a finite state machine with four states: A, B, C, D. It shall change its state to the consecutive state at rising edges of clock. It has four inputs: din1, din2, din3, din4, and an output: dout. It should map an input to an output once its state arrives.

So dout should be equal to

- din1 if state is A.
- din2 if state is B.
- din3 if state is C.
- din4 if state is D.

states change at rising edges of clock so if a rising edge came:

- if current state is A then next state would be B
- if current state is B then next state would be C
- if current state is C then next state would be D
- if current state is D then next state would be A

If no rising edge happens, then neither el dout nor the current state change.

5.2 FSM Diagram:

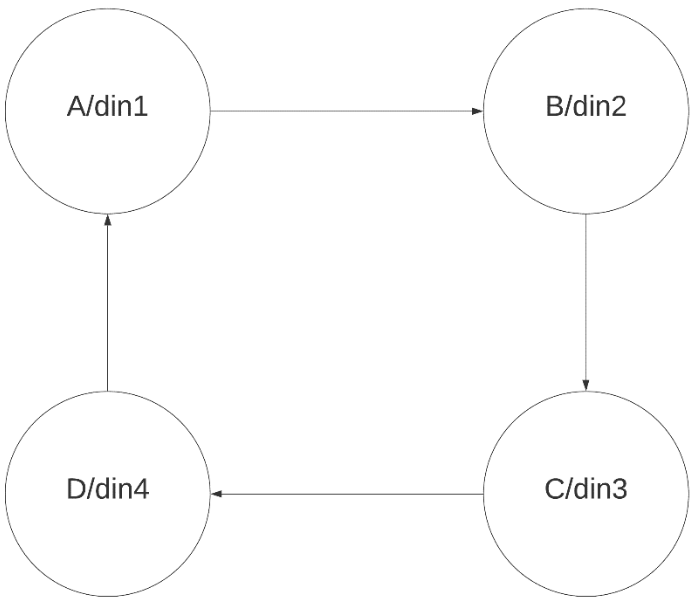
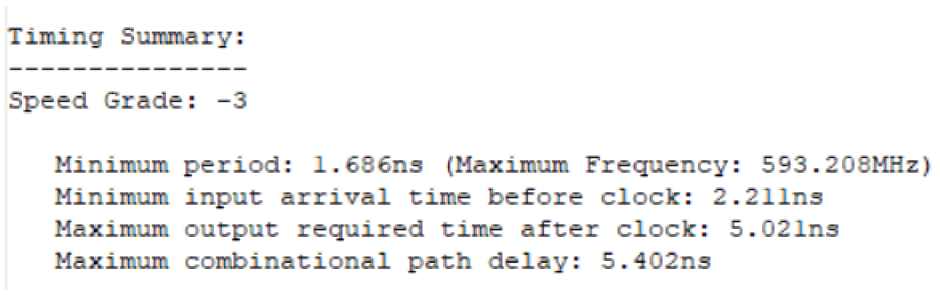


Figure 16 shows the FSM diagram

5.3 FSM Type:

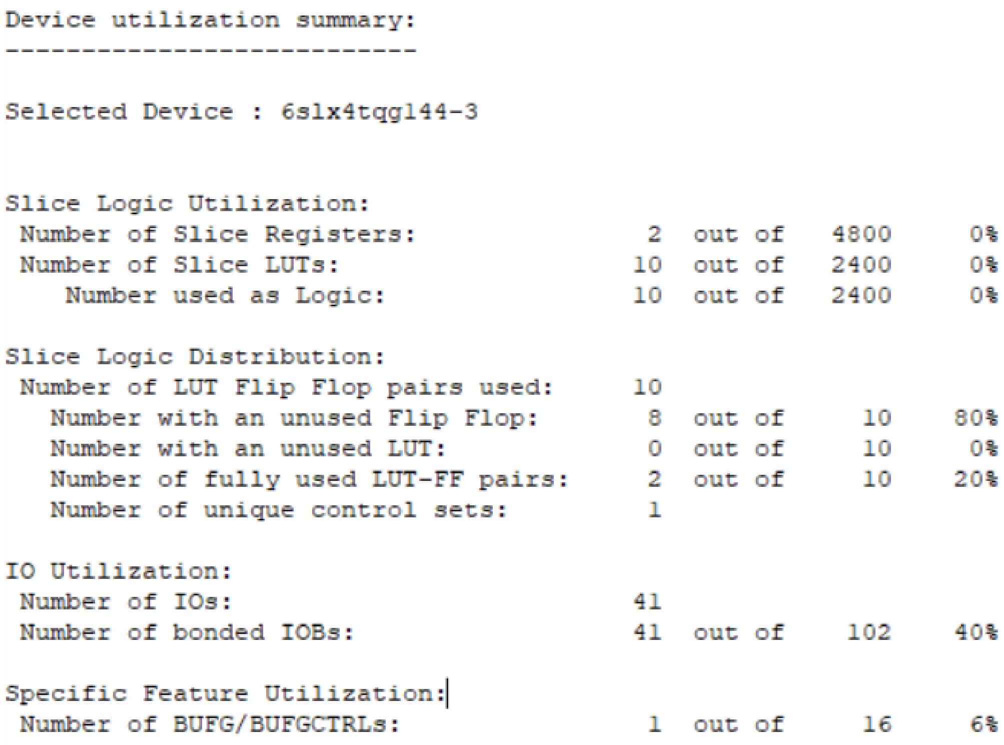
The FSM is implemented as a Moore type because the output only depends on current state and not the inputs. If input changes output will not unless a change in state happens upon rising edge of clock.

5.4 Timing Analysis:



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

   Minimum period: 1.686ns (Maximum Frequency: 593.208MHz)
   Minimum input arrival time before clock: 2.211ns
   Maximum output required time after clock: 5.021ns
   Maximum combinational path delay: 5.402ns
```

5.5 Synthesis:



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:               2  out of   4800     0%
 Number of Slice LUTs:                   10  out of   2400     0%
    Number used as Logic:                10  out of   2400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     10
   Number with an unused Flip Flop:       8  out of     10    80%
   Number with an unused LUT:             0  out of     10     0%
   Number of fully used LUT-FF pairs:     2  out of     10    20%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                          41
 Number of bonded IOBs:                  41  out of    102    40%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
```

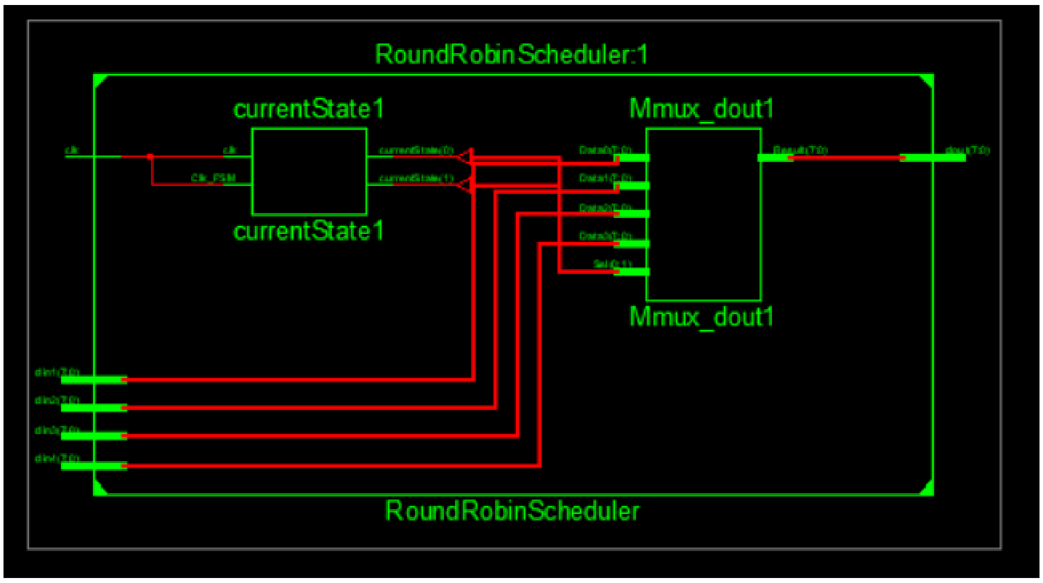
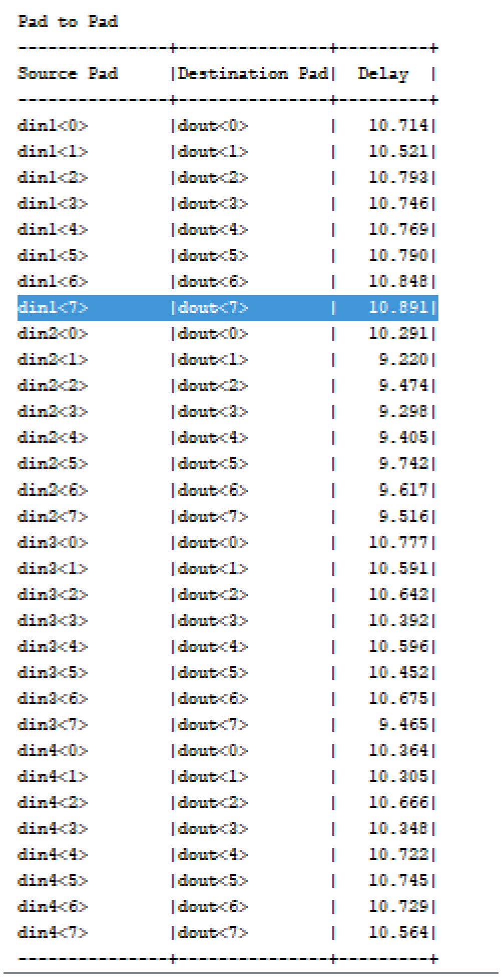


Figure 17 shows the scheduler's block diagram

```
=========================================================================
*                          Low Level Synthesis                          *
=========================================================================
Analyzing FSM <MFsm> for best encoding.
Optimizing FSM <FSM_0> on signal <currentState[1:2]> with user encoding.
-------------------
 State | Encoding
-------------------
 a     | 00
 b     | 01
 c     | 10
 d     | 11
-------------------
```

5.6 Critical Path:



```
Pad to Pad
---------------+---------------+---------+
Source Pad     |Destination Pad|  Delay  |
---------------+---------------+---------+
din1<0>        |dout<0>        |   10.714|
din1<1>        |dout<1>        |   10.521|
din1<2>        |dout<2>        |   10.793|
din1<3>        |dout<3>        |   10.746|
din1<4>        |dout<4>        |   10.769|
din1<5>        |dout<5>        |   10.790|
din1<6>        |dout<6>        |   10.848|
din1<7>        |dout<7>        |   10.891|
din2<0>        |dout<0>        |   10.291|
din2<1>        |dout<1>        |    9.220|
din2<2>        |dout<2>        |    9.474|
din2<3>        |dout<3>        |    9.298|
din2<4>        |dout<4>        |    9.405|
din2<5>        |dout<5>        |    9.742|
din2<6>        |dout<6>        |    9.617|
din2<7>        |dout<7>        |    9.516|
din3<0>        |dout<0>        |   10.777|
din3<1>        |dout<1>        |   10.591|
din3<2>        |dout<2>        |   10.642|
din3<3>        |dout<3>        |   10.392|
din3<4>        |dout<4>        |   10.596|
din3<5>        |dout<5>        |   10.452|
din3<6>        |dout<6>        |   10.675|
din3<7>        |dout<7>        |    9.465|
din4<0>        |dout<0>        |   10.364|
din4<1>        |dout<1>        |   10.305|
din4<2>        |dout<2>        |   10.666|
din4<3>        |dout<3>        |   10.348|
din4<4>        |dout<4>        |   10.722|
din4<5>        |dout<5>        |   10.745|
din4<6>        |dout<6>        |   10.729|
din4<7>        |dout<7>        |   10.564|
---------------+---------------+---------+
```

5.7 FSM Implementation Styles

To implement a finite-state machine (FSM) using VHDL the tasks associated with finite-state machines are done within processes. These tasks are

1. Determining whether asynchronous reset is active and taking the appropriate actions for reset
2. Updating current state register
3. Determining the next state
4. Determining outputs which is dependent on the current state only in Moore style, and dependent on the current state and inputs in the Mealy style.

Depending on the designer, these tasks can be assigned to one, two, or three processes. In a professional setting, two or three processes FSM styles are preferred over one process. This is since during simulation, single process FSM implementations are typically harder to detect errors in due to the intermediate "next state" signal not being available. Additionally, during synthesis one process FSM implementations typically result in higher gate equivalents, this is due to it being at a relatively higher level of disconnect from physical hardware. [4]

5.7.1 One Process

In one-process FSM, all the tasks are done within one process this is done by having the clock, asynchronous reset, and current state signals in the sensitivity list of the process in Moore FSM, while in Mealy FSM the inputs are also added to the sensitivity list as a change within the inputs will change the output of the FSM in Mealy style. The previously mentioned tasks are then done sequentially within the process. Within this style, the intermediate "next state" signal is not present instead the order of manipulation of the current state signal substitutes the need for it.

5.7.2 Two Processes

For two-processes FSM, the tasks are divided between two processes, with the first process carrying out the "asynchronous reset" and "current state register updating" tasks, while the second process carries out determination of the next state and outputs. In Mealy and Moore style FSM the first process is sensitive to the reset and clock signals so that any changes in the reset can be detected and acted upon within this process and to update the current state register when the clock is on the rising edge. The second process in both Mealy and Moore styles has the current state and input signals in their sensitivity lists, this is done so that in Mealy style the outputs and next state are determined using the current state as well as the inputs, but in Moore style, the code within the process will ensure that only the current state determines the output while the inputs determine the next state while taking the current state into consideration.

5.7.3 Three Processes

For three-processes FSM, the first process from the two-process style remains functionally identical while the second process from the two-process style is split into two processes. This is done so that the second process will only carry out the determination of the next state task, while the third process will only carry out the determination of outputs task. In Mealy style FSM, both the second and third processes will be sensitive to both the current state and inputs, as in Mealy style the output and next state are determined by the current state and the inputs. While in

Moore style FSM, the third process (responsible for output determination) will be sensitive to only the current state as in a Moore FSM the output is decided only by the current state. The second process (responsible for the determination of the next state) will remain the same as the one found in Mealy style as the next state is determined by both the current state and the inputs.

6.0 Testing and Simulation Results

Tested feature	Inputs									Delay after	Expected output			
	Reset	D_in1	D_in2	D_in3	D_in4	wr_1	wr_2	wr_3	wr_4		D_out1	D_out2	D_out3	D_out4
Check if reset is working	1	XXXXXXX	XXXXXXX	XXXXXXX	XXXXXXX	X	X	X	X	0 ns	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
Enters all the values from same clock cycle in that ends with same bit and extract them in right order	0	11001000	10010000	01010100	10001000	1	1	1	1	25 ns and 10 ns between each queue and each following queue and 10 ns between each output within the same queue	10001000	10100001	10110010	01110011
		10100001	01110001	10100101	11100001						11001000	01110001	10110110	01100111
		10111010	10110010	10110110	10100010						10010000	10100101	10100010	10101111
		10101111	01100111	01110011	01100111						01010100	11100001	10111010	01100111
Checks if there will be outputs, if write enable is disabled	0	11001000	10010000	01010100	10001000	0	0	0	0	25 ns	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
		10100001	01110001	10100101	11100001									
		10111010	10110010	10110110	10100010									
		10101111	01100111	01110011	01100111									
Write enable changes to 1, last output is stored	0	10101111	01100111	01110011	01100111	1	1	1	1	25 ns and 10 ns between each queue and each following queue and 10 ns between each output within the same queue	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ	01110011
		ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ									01100111
														10101111
														01100111
Enters all the values from same clock cycle in that ends with varying bit and extract them in right order	0	10000000	10000001	10000010	10000011	1	1	1	1	25 ns and 10 ns between each queue and each following queue and 10 ns between each output within the same queue	10000000	11110001	11100010	10100011
		10100011	10100000	10100001	10100010						10100000	10000001	11110010	11100011
		11100010	11100011	11100000	11100001						11100000	10100001	10000010	11110011
		11110001	11110010	11110011	11110000						11110000	11100001	10100010	10000011
Ensures that the output is according to queue which depends on	0	10000000	10010000	11000000	10000100	1	1	1	1	25 ns and 10 ns between each queue and each following queue and 10 ns	10000000	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ
		10100000	10100000	10100000	10100000						10010000			
		11100000	11100000	11100000	11100000						11000000			

the clock cycles		11110000	11110000	11110000	11110000					between each output within the same queue	10000100			
		ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ	ZZZZZZZZ						10100000			
											11100000			
											11110000			

The modules have been implemented individually and tested by their own test benches. Then Router integrates all the modules into 1 module that performs all the functionality. To test the router, we have developed different test cases. However, the router should always be tested that the data output 1 is for the queue "00", data output 2 is for the queue "01", ...etc. Therefore, we have made a process that works concurrently beside the test bench to always perform a check on the output of the queue if it is in the right queue. (Warning process code)

Before test case 1, we have set the reset to '1' in order to test its functionality. Test case 1 is a test case where we have write enabled for all queues and, we initialize the registers with bits that ends with "00" to check if the router enqueues all the 4 values at start in the first queue and dequeues them in the right order. Then we set the registers with bits that ends with "01" to check if the router enqueues all the 4 values at start in the second queue and dequeues them in right order. The test case also implements the same check for queue 3 (ends with "10"), and queue 4 (ends with "11"). The waveform allows us to track the outputs and determine if outputs are same as expected. (Test case 1 code) (Test case 1 waveform)

Test case 2 is a test case, where we initialize the registers with values as test case 1; however, write enable is only on after last initialization. This allows us to test the write enable and determine if it works as intended. The main goal is to determine the write enable functionality and to check if the queue writes last value in the registers or not. This test case has been successful by tracking the waveform, we can see there was no output until write was enabled and the output is the data stored in the register. (Test case 2 waveform) (Test case 2 code)

Test case 3 is a test case, where unlike test case 1 and 2, we set each register of the four registers ending with different bits. This allows to test whether the queue of data outputs works as intended in different clock cycles with varying bits in each register. This test focuses on the demultiplexer testing between the queues and ensures the demultiplexer selects right queue each time. Also, this ensures the order of queue is working as intended on the input priority not register priority. The simulation can be tracked in the waveform that shows the output is what intended. (Test case 3 waveform) (Test case 3 code)

Test case 4 is a test case, where we stress the testing on the first queue. The goal of this test case is to ensure there is no issue in the first queue, which will ensure that there is no issue in other queues. The test case has the inputs all ends with bits "00" in all clock cycles, which means all the 16 elements must be written correctly in order of the state and clock cycles. The simulation shows us a waveform that correctly outputs the state of each cycle then the states of next cycle. (Test case 4 waveform) (Test case 4 code)

The testbench covers the demultiplexer testing, register testing, state order testing, data output queue testing and ensures that the output is what intended. However, there is intended delay in the outputs. The testbench needs 700 ns to run and it will show each test case waveform as shown in Appendix C. The testbench code can be found in Appendix B.

7.0 Conclusion

Routing technique for network on Chip design is performed where it is utilized for networking and transmitting data among 2 or more packet-switched processor networks. The synthesis and the simulation of the router are accomplished using the ModelSim simulator, and for creating a timing report we used the Xilinx tool.

There are several challenges to the implementation of the router. It includes synchronization between the sequential and combinational logic various elements as well as it depends on all additional modules in the project.

The best implementation style for FSM is two or three processes FSM styles are preferred over one process. This is since during simulation, single process FSM implementations are typically harder to detect errors in due to the intermediate “next state” signal not being available as mentioned in 5.7. As well as the timing results is mentioned in the Appendix A in 9.0.

Furthermore, there were nine modules for every module that is qualified for a specific function. The Demultiplexer is accountable for the adoption of implemented data through Switch-case. The register is responsible for transporting the implemented data by applying IF-condition. The Ram-Block is the physical and actual memory component in the router. The Gray-Counter is responsible for calculating each clock cycle in grey code while the change from gray to binary is known as Gray to Binary Converter adopts the XOR gate to change the gray system to Binary. The Frist in First out controller is known as FIFO controller that is performed relying on IF-conditions to establish values to blank, complete, and demands legality of the Read and Write, as well as it maps the controller with the Ram-Block in unity component to control the movement by mapping a comparable Port between these modules. The round-robin is executed in three processes and two Switch cases. The first Switch is to define the next state, and the second Switch is to define the output. Conclusively, the router is executed by classifying and referring the previously mentioned modules and mapping them together directly.

In the future, we look forward to checking if the data is correct or not before implementing, developing the design by creating more universal code and update the scheduling algorithm for more reliable execution and modify the router to be higher than 8-bits.

8.0 Task Distribution

The project workload is distributed equally among all team members.

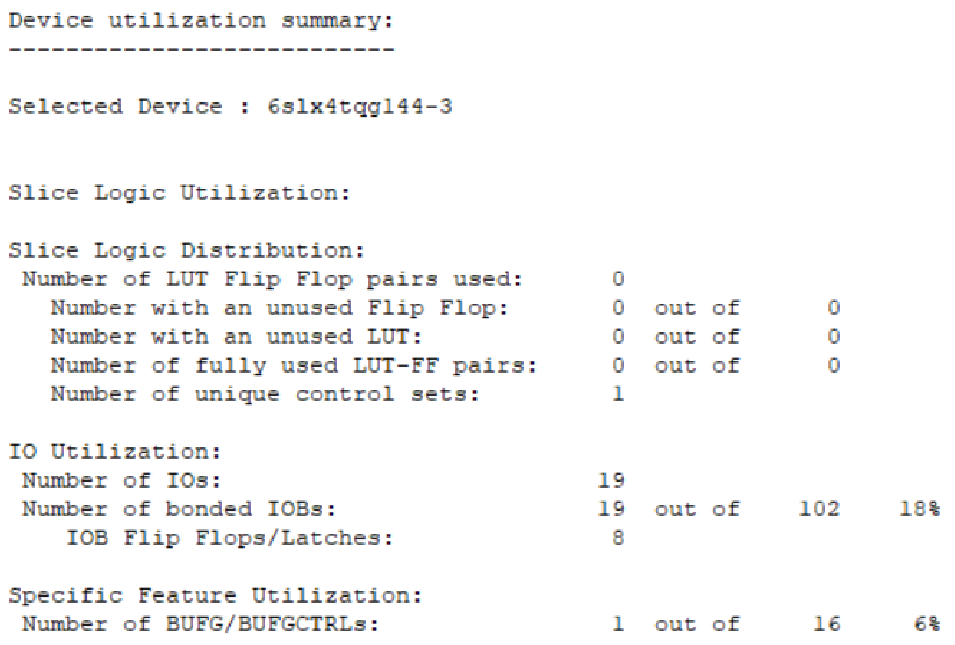
9.0 Appendix A

9.1 Register

9.1.1 VHDL source code

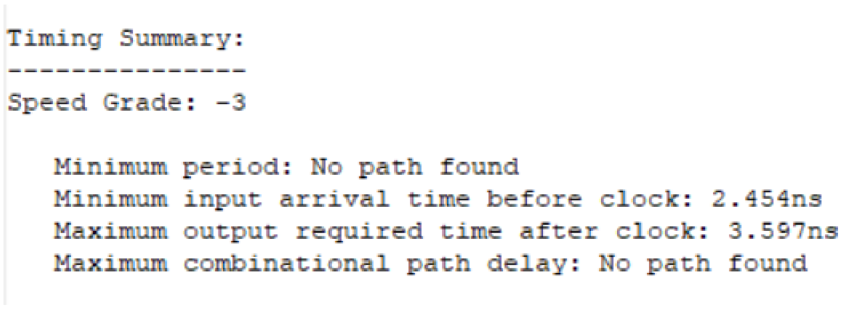
```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Register8Bit IS
        PORT (  reset    : IN    std_logic;
                clk      : IN    std_logic;
                clk_en   : IN    std_logic;
                data_in  : IN    std_logic_vector (7 downto 0);
                data_out : OUT   std_logic_vector (7 downto 0)
        );
END ENTITY Register8Bit;

ARCHITECTURE Register8BitArch OF Register8Bit IS BEGIN
        PROCESS (clk, reset,clk_en) IS BEGIN
                IF reset = '1' THEN
                        data_out <= (OTHERS=>'0');
                ELSIF clk_en = '1' AND rising_edge(clk) THEN
                        data_out <= data_in;
                END IF;
        END PROCESS;
END ARCHITECTURE;
```

9.1.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      0
   Number with an unused Flip Flop:       0  out of      0
   Number with an unused LUT:             0  out of      0
   Number of fully used LUT-FF pairs:     0  out of      0
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                          19
 Number of bonded IOBs:                  19  out of    102    18%
    IOB Flip Flops/Latches:               8

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                1  out of     16     6%
```

9.1.3 Timing analysis



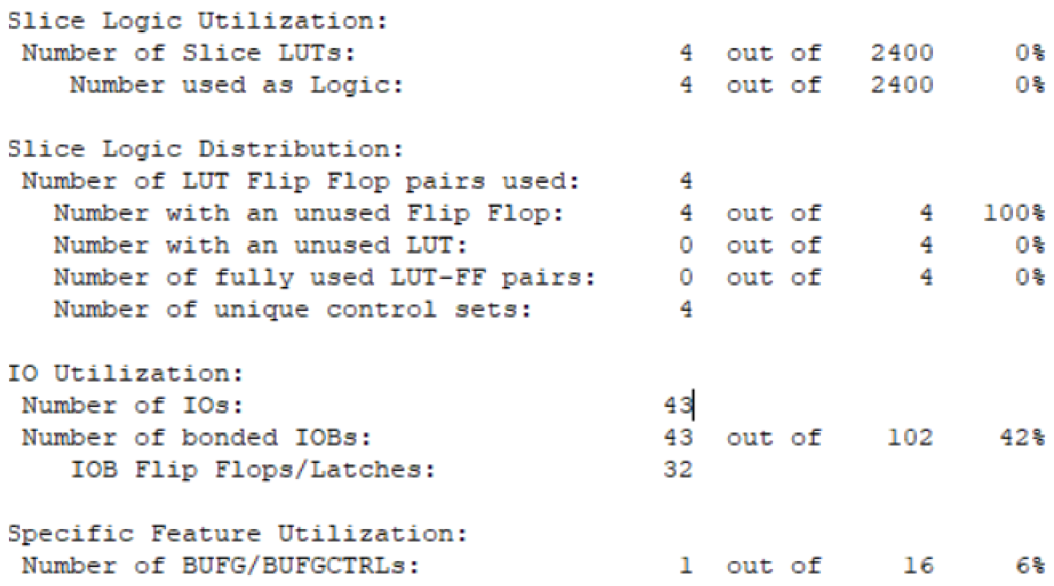
```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: 2.454ns
   Maximum output required time after clock: 3.597ns
   Maximum combinational path delay: No path found
```

9.2 Demux

9.2.1 VHDL source code

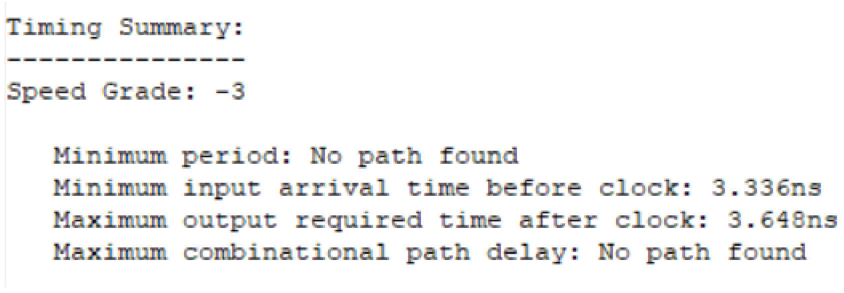
```
Library IEEE;
use ieee.std_logic_1164.all;

Entity Demux IS
        PORT (  En      : IN     std_logic;
                Sel     : IN     std_logic_vector (1 downto 0);
                d_in    : IN     std_logic_vector (7 downto 0);
                d_out1  : OUT    std_logic_vector (7 downto 0);
                d_out2  : OUT    std_logic_vector (7 downto 0);
                d_out3  : OUT    std_logic_vector (7 downto 0);
                d_out4  : OUT    std_logic_vector (7 downto 0)
        );
END ENTITY Demux;

ARCHITECTURE Demux_arch of Demux is

Begin
        start: PROCESS (En,Sel,d_in) is Begin
                IF (En = '1') THEN
                        Case Sel IS

                                WHEN "00" =>
                                d_out1 <= d_in;


                                WHEN "01" =>
                                d_out2 <= d_in;

                                WHEN "10" =>
                                d_out3 <= d_in;

                                WHEN "11" =>
                                d_out4 <= d_in;

                                WHEN OTHERS =>

                        END CASE;
                END IF;
        END PROCESS;
END ARCHITECTURE Demux_arch;
```

9.2.2 Total equivalent gate count



```
Slice Logic Utilization:
 Number of Slice LUTs:                      4  out of   2400     0%
    Number used as Logic:                   4  out of   2400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:        4
   Number with an unused Flip Flop:         4  out of      4   100%
   Number with an unused LUT:               0  out of      4     0%
   Number of fully used LUT-FF pairs:       0  out of      4     0%
   Number of unique control sets:           4

IO Utilization:
 Number of IOs:                            43
 Number of bonded IOBs:                    43  out of    102    42%
    IOB Flip Flops/Latches:                32

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                  1  out of     16     6%
```

9.2.3 Timing analysis



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

   Minimum period: No path found
   Minimum input arrival time before clock: 3.336ns
   Maximum output required time after clock: 3.648ns
   Maximum combinational path delay: No path found
```

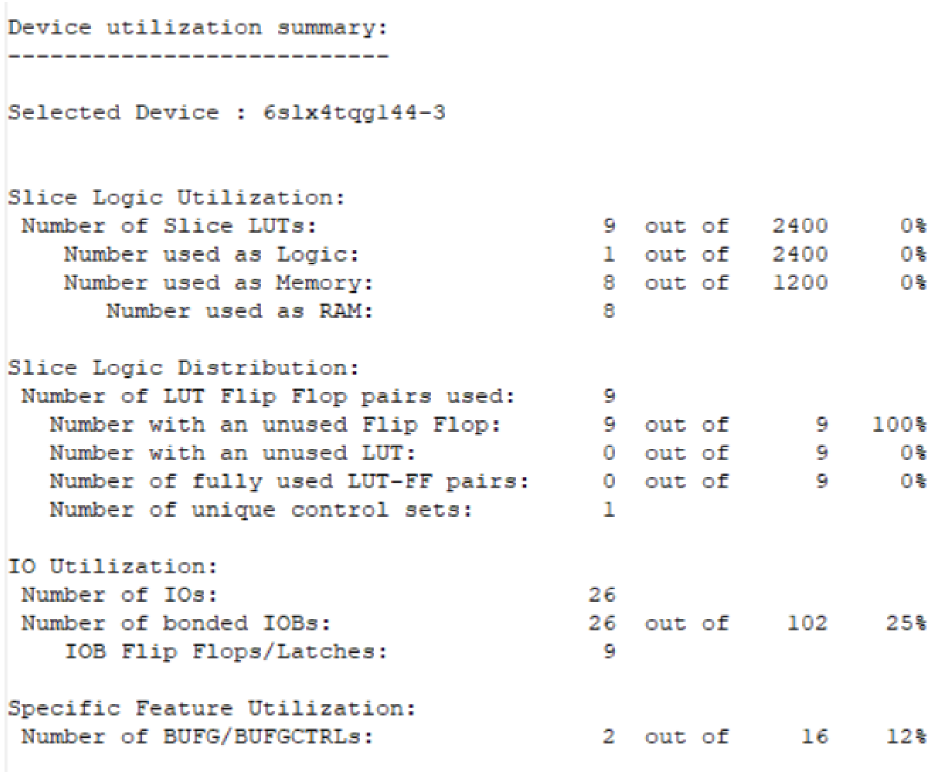
9.3 Ram

9.3.1 VHDL source code

```
Library ieee;
Use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

Entity DualPortBlockRam is
        port(
                D_in:   in  std_logic_vector (7 downto 0);
                D_out:  out std_logic_vector (7 downto 0);
                WEA:    in  std_logic;
                REA:    in  std_logic;
                ADDRA:  in  std_logic_vector (2 downto 0);
                ADDRB:  in  std_logic_vector (2 downto 0);
                CLKA:   in  std_logic;
                CLKB:   in  std_logic
        );
End entity;

Architecture behavioural of DualPortBlockRam is

type ram_type is array (7 downto 0) of std_logic_vector (7 downto 0);
```

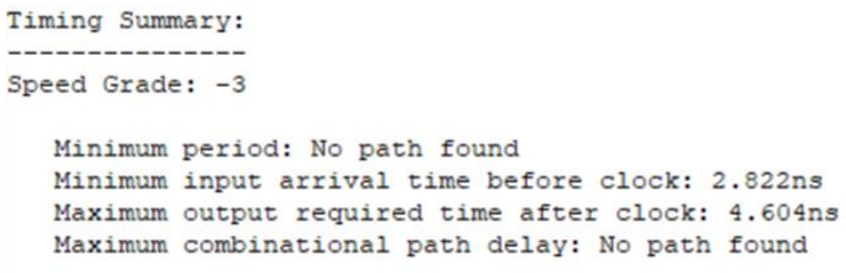
```
SIGNAL ram : ram_type;

begin
        process (CLKA) is begin
                if rising_edge(CLKA) then
                        if WEA = '1' then
                                ram(to_integer(unsigned(ADDRA)))<=D_in;
                        end if;
                end if;

        end process;

        process (CLKB) is begin
                if rising_edge(CLKB) then
                        if          REA          =          '1'          AND
NOT(ram(to_integer(unsigned(ADDRB)))="XXXXXXXX")then
                                D_out<=ram(to_integer(unsigned(ADDRB)));
                else
                                D_out <= "ZZZZZZZZ";
                        end if;
                end if;
        end process;
end architecture;
```

9.3.2 Total equivalate gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice LUTs:                    9  out of   2400     0%
    Number used as Logic:                 1  out of   2400     0%
    Number used as Memory:                8  out of   1200     0%
       Number used as RAM:                8

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      9
   Number with an unused Flip Flop:       9  out of      9   100%
   Number with an unused LUT:             0  out of      9     0%
   Number of fully used LUT-FF pairs:     0  out of      9     0%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                          26
 Number of bonded IOBs:                  26  out of    102    25%
    IOB Flip Flops/Latches:               9

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                2  out of     16    12%
```

9.3.3 Timing analysis



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

   Minimum period: No path found
   Minimum input arrival time before clock: 2.822ns
   Maximum output required time after clock: 4.604ns
   Maximum combinational path delay: No path found
```

9.4 Gray Counter

9.4.1 VHDL source code

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

entity gray_counter is
port(
        clk: in std_logic;
        reset: in std_logic;
        en: in std_logic;
        gray_count: out std_logic_vector(3 downto 0));

end entity;

architecture counter_arch of gray_counter is

signal s1: std_logic_vector (3 downto 0);
begin

p1: process(clk, reset, en) is begin

--Asynchronous reset
if(reset = '1') then
s1 <= "0000";

elsif(clk'event and clk ='1' and en='1') then

case s1 is
when "0000" => s1 <= "0001";
when "0001" => s1 <= "0011";
when "0011" => s1 <= "0010";
when "0010" => s1 <= "0110";
when "0110" => s1 <= "0111";
when "0111" => s1 <= "0101";
when "0101" => s1 <= "0100";
when "0100" => s1 <= "1100";
when "1100" => s1 <= "1101";
when "1101" => s1 <= "1111";
when "1111" => s1 <= "1110";
when "1110" => s1 <= "1010";
when "1010" => s1 <= "1011";
when "1011" => s1 <= "1001";
when "1001" => s1 <= "1000";
when "1000" => s1 <= "0000";
when others => null;
end case;
end if;

end process p1;

gray_count <= s1;
end architecture counter_arch;
```

9.4.2 Total equivalent gate count

```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:                4  out of   4800     0%
 Number of Slice LUTs:                     4  out of   2400     0%
    Number used as Logic:                  4  out of   2400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:       8
   Number with an unused Flip Flop:        4  out of      8    50%
   Number with an unused LUT:              4  out of      8    50%
   Number of fully used LUT-FF pairs:      0  out of      8     0%
   Number of unique control sets:          1

IO Utilization:
 Number of IOs:                            7
 Number of bonded IOBs:                    7  out of    102     6%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 1  out of     16     6%
```

9.4.3 Timing analysis

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

   Minimum period: 1.714ns (Maximum Frequency: 583.431MHz)
   Minimum input arrival time before clock: 2.555ns
   Maximum output required time after clock: 3.732ns
   Maximum combinational path delay: No path found
```

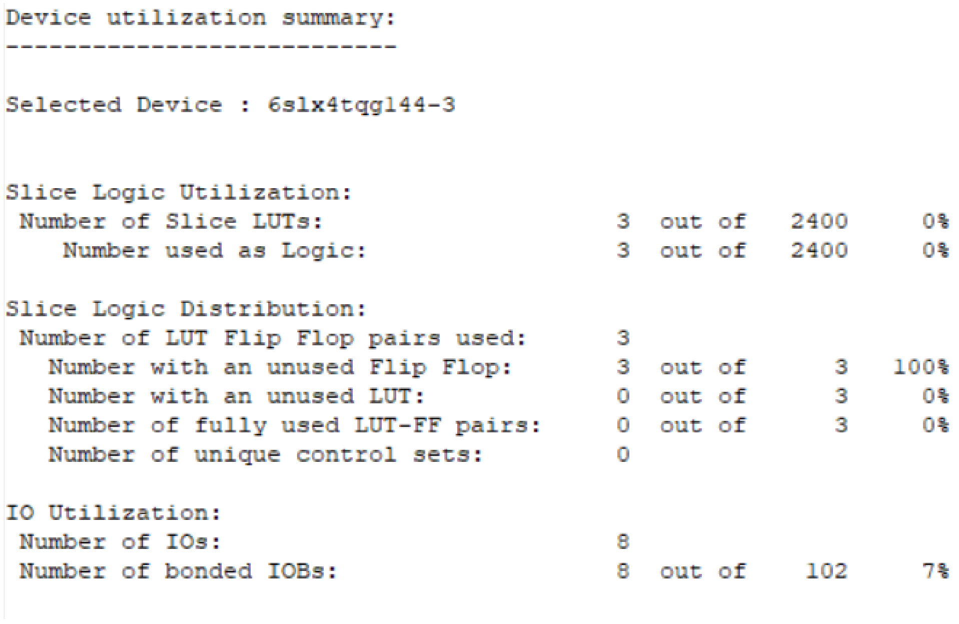
9.5 Grey to binary convertor

9.5.1 VHDL source code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity graytobin is
port ( gray_in : in std_logic_vector (3 downto 0);
       bin_out : out std_logic_vector (3 downto 0));
end entity graytobin;

Architecture behave of graytobin is
```

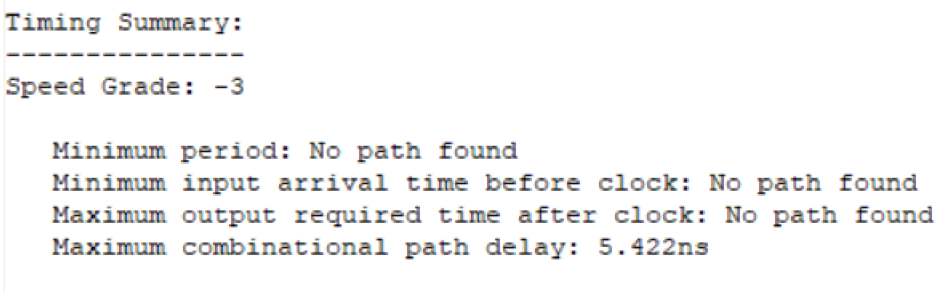
```
begin

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 Architecture behave;
```

9.5.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice LUTs:                    3  out of   2400     0%
    Number used as Logic:                 3  out of   2400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      3
   Number with an unused Flip Flop:       3  out of      3   100%
   Number with an unused LUT:             0  out of      3     0%
   Number of fully used LUT-FF pairs:     0  out of      3     0%
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                           8
 Number of bonded IOBs:                   8  out of    102     7%
```

9.5.3 Timing analysis



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

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 5.422ns
```

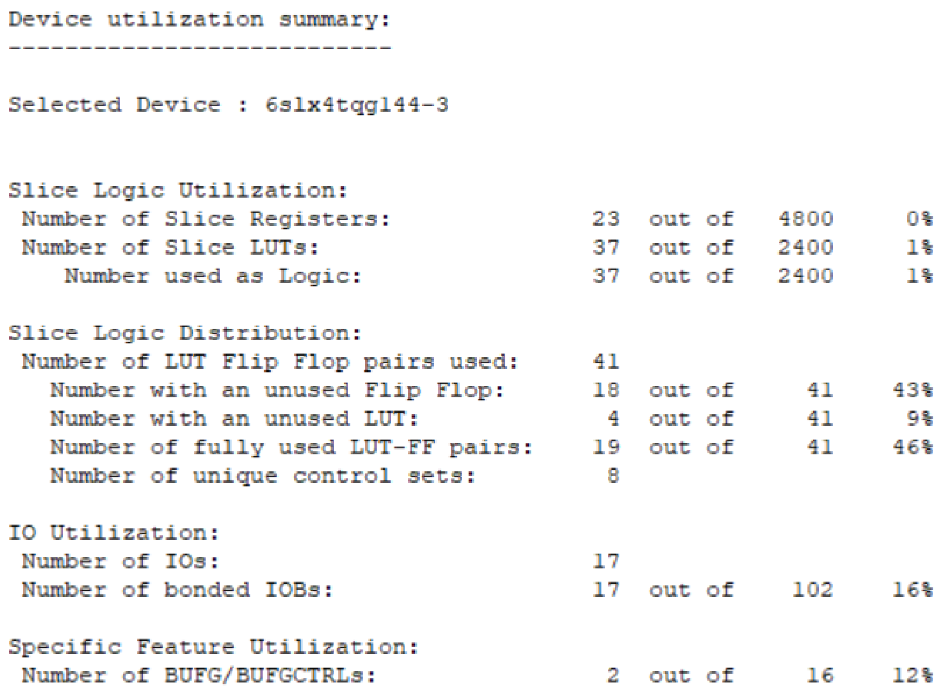
9.6 FIFO-Controller

9.6.1 VHDL source code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity FIFO_Controller is
port ( reset ,rdclk,wrclk,r_req,w_req : in std_logic;
      write_valid,read_valiad,empty,full : out std_logic;
      wr_ptr , rd_ptr : out std_logic_vector (3 downto 0));
end entity FIFO_Controller;
Architecture behave of FIFO_Controller is
component gray_counter is
```

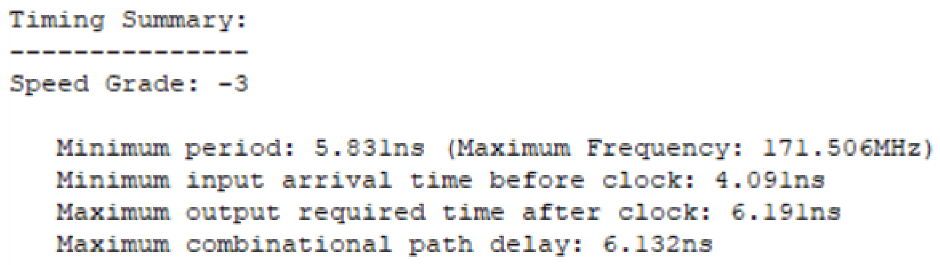
```
port(
        clk: in std_logic;
        reset: in std_logic;
        en: in std_logic;
        gray_count: out std_logic_vector(3 downto 0));
end component ;
 for all:  gray_counter use ENTITY work.gray_counter;
component graytobin is
port (
       gray_in : in std_logic_vector (3 downto 0);
       bin_out : out std_logic_vector (3 downto 0));
end component ;
 for all : graytobin use entity work.graytobin;
signal empty_sl,full_sl,wr_en,rd_en:std_logic;
signal gray_rdout,gray_wrout,rd_binout,wr_binout :std_logic_vector(3 downto
0);
BEGIN

grey_wr: gray_counter port map(wrclk,reset,wr_en,gray_wrout);
bin_wr: graytobin port map(gray_wrout,wr_binout);

grey_rd: gray_counter  port map(rdclk,reset,rd_en,gray_rdout);
bin_rd: graytobin port map(gray_rdout,rd_binout);

wr_en<= w_req and (not full_sl) when falling_edge(wrclk);
rd_en<= r_req and (not empty_sl) when falling_edge(rdclk);

wr:process (wrclk,w_req)
begin
if falling_edge(wrclk) and w_req ='1' then
if full_sl='0' then
write_valid <='1';
wr_ptr<=wr_binout;
else
write_valid <='0';
end if;
end if;
end process wr;

rd: process (rdclk,r_req)
begin
if falling_edge(rdclk) AND r_req ='1'  then
if empty_sl='0'  then
read_valiad <='1';
rd_ptr <= rd_binout;
else
read_valiad <='0';
end if;
end if;
end process rd;

update: process(rd_binout,wr_binout,reset)
begin
if reset ='1' then
   empty_sl<='1';
   full_sl<='0';
elsif rd_binout(2 downto 0)= wr_binout(2 downto 0) then
     if (rd_binout(3) XNOR wr_binout(3))='1' then
     empty_sl<='1';
     full_sl<='0';
     else
     full_sl<='1';
     empty_sl<='0';
     end if;
else
      empty_sl<='0';
      full_sl<='0';
end if;
end process update;

empty<=empty_sl;
full<=full_sl;
end architecture behave;
```

9.6.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:              23  out of   4800     0%
 Number of Slice LUTs:                   37  out of   2400     1%
    Number used as Logic:                37  out of   2400     1%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     41
   Number with an unused Flip Flop:      18  out of     41    43%
   Number with an unused LUT:             4  out of     41     9%
   Number of fully used LUT-FF pairs:    19  out of     41    46%
   Number of unique control sets:         8

IO Utilization:
 Number of IOs:                          17
 Number of bonded IOBs:                  17  out of    102    16%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                2  out of     16    12%
```

9.6.3 Timing analysis



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

   Minimum period: 5.831ns (Maximum Frequency: 171.506MHz)
   Minimum input arrival time before clock: 4.091ns
   Maximum output required time after clock: 6.191ns
   Maximum combinational path delay: 6.132ns
```

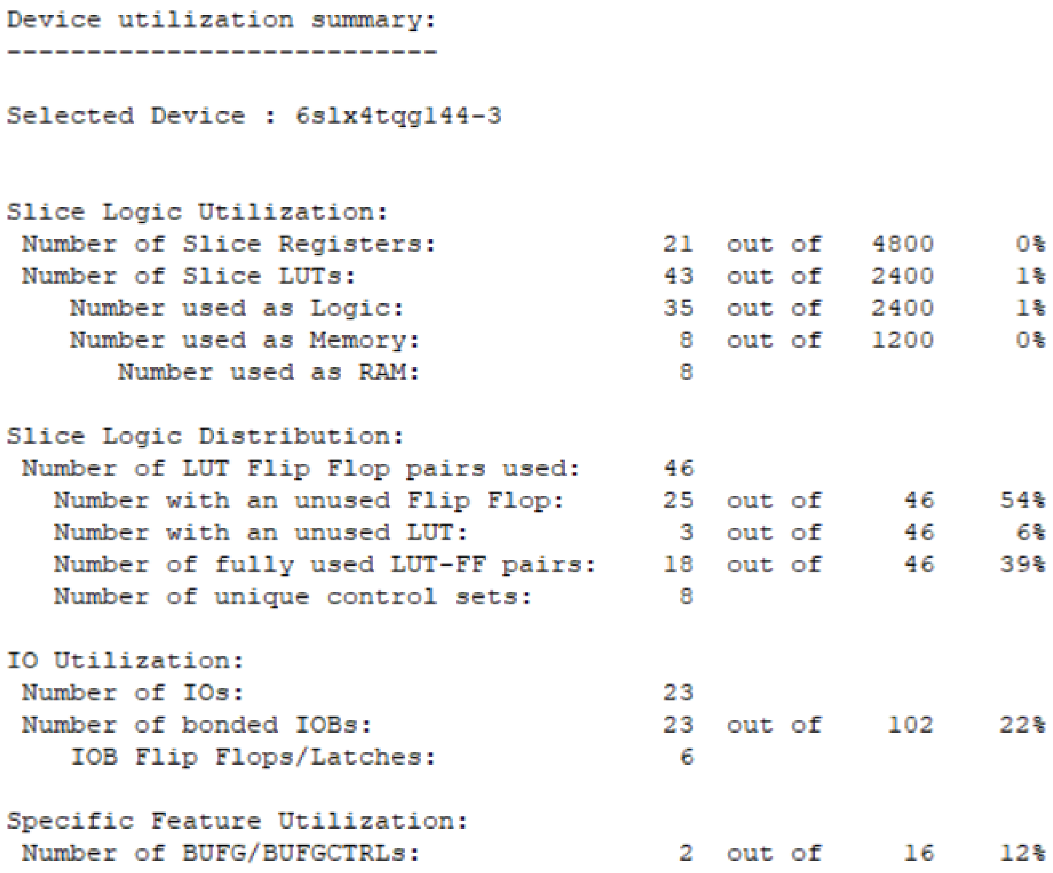
9.7 FIFO

9.7.1 VHDL source code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

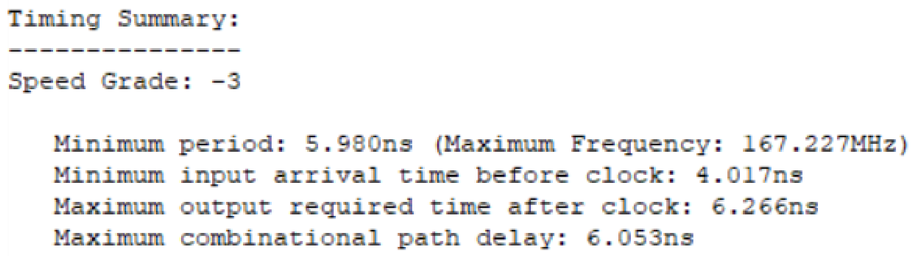
```
entity FIFO is
port (  reset ,rclk,wclk,rreq,wreq : in std_logic;
        datain: in std_logic_vector (7 downto 0);
        dataout: out std_logic_vector (7 downto 0);
       empty, full : out std_logic);
end entity FIFO;

Architecture structural of FIFO is


component fifocontroller is
port (  reset ,rdclk,wrclk,r_req,w_req : in std_logic;
        write_valid,read_valiad, empty,full : out std_logic;
        wr_ptr , rd_ptr : out std_logic_vector (3 downto 0));
end component fifocontroller;
for controller1:  fifocontroller use ENTITY work.FIFO_Controller(behave);

component DualPortBlockRam is
        port(
                D_in:   in  std_logic_vector (7 downto 0);
                D_out:  out std_logic_vector (7 downto 0);
                WEA:    in  std_logic;
                REA:    in  std_logic;
                ADDRA:  in  std_logic_vector (2 downto 0);
                ADDRB:  in  std_logic_vector (2 downto 0);
                CLKA:   in  std_logic;
                CLKB:   in  std_logic);
End component;
for ram1:  DualPortBlockRam use ENTITY work.DualPortBlockRam(behavioural);

signal write_valid_sl,read_valiad_sl : std_logic;
signal wr_ptr_sl , rd_ptr_sl : std_logic_vector (3 downto 0);

begin

controller1:                                 fifocontroller                port
map(reset,rclk,wclk,rreq,wreq,write_valid_sl,read_valiad_sl,
empty,full,wr_ptr_sl , rd_ptr_sl);
ram1:    DualPortBlockRam     port    map(datain,dataout,    write_valid_sl,
read_valiad_sl,wr_ptr_sl(2 downto 0),rd_ptr_sl(2 downto 0),wclk, rclk);
end architecture structural;
```

9.7.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:              21  out of   4800     0%
 Number of Slice LUTs:                   43  out of   2400     1%
    Number used as Logic:                35  out of   2400     1%
    Number used as Memory:                8  out of   1200     0%
       Number used as RAM:                8

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     46
   Number with an unused Flip Flop:      25  out of     46    54%
   Number with an unused LUT:             3  out of     46     6%
   Number of fully used LUT-FF pairs:    18  out of     46    39%
   Number of unique control sets:         8

IO Utilization:
 Number of IOs:                          23
 Number of bonded IOBs:                  23  out of    102    22%
    IOB Flip Flops/Latches:               6

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                2  out of     16    12%
```

9.7.3 Timing analysis



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

   Minimum period: 5.980ns (Maximum Frequency: 167.227MHz)
   Minimum input arrival time before clock: 4.017ns
   Maximum output required time after clock: 6.266ns
   Maximum combinational path delay: 6.053ns
```

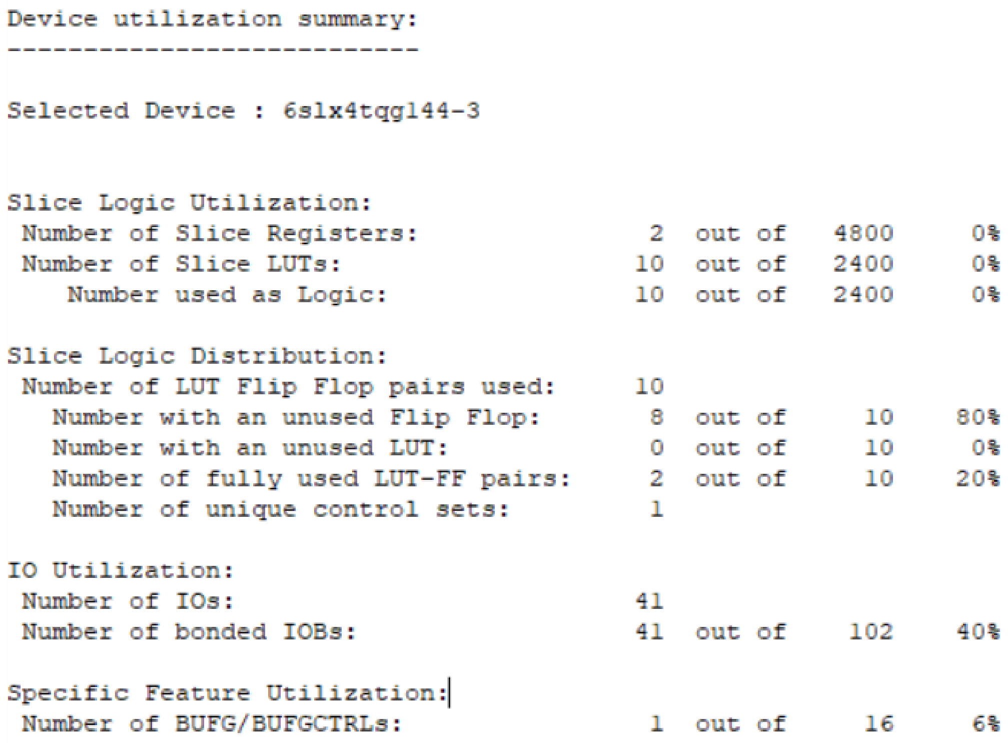
9.8 Round Robin

9.8.1 VHDL source code

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY RoundRobinScheduler IS
        PORT (  clk     : IN     std_logic;
                din1    : IN     std_logic_vector (7 downto 0);
                din2    : IN     std_logic_vector (7 downto 0);
                din3    : IN     std_logic_vector (7 downto 0);
                din4    : IN     std_logic_vector (7 downto 0);
                dout    : OUT    std_logic_vector (7 downto 0)
        );
END ENTITY RoundRobinScheduler;
```

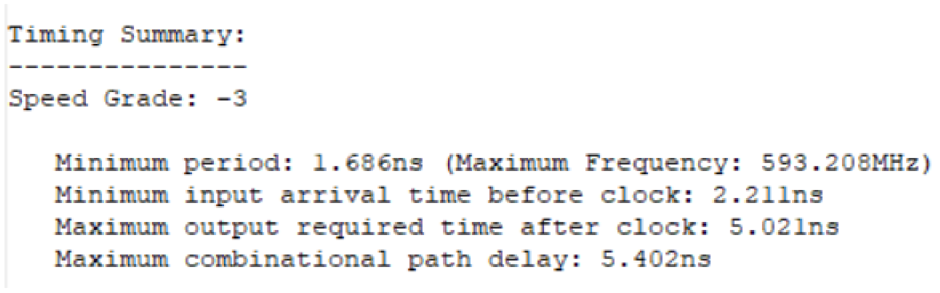
```
ARCHITECTURE RoundRobinScheduledrArch OF RoundRobinScheduler IS
        TYPE state IS (A,B,C,D);
        SIGNAL currentState : state:=D;
        SIGNAL nextState : state:=A;
BEGIN
        PROCESS(clk) IS
        BEGIN
                IF rising_edge(clk) THEN
                        currentState <= nextState;
                END IF;
        END PROCESS;

        calcNextState : PROCESS (currentState) IS BEGIN
                case currentState is
                when A =>
                        nextState <= B;

                when B =>

                        nextState <= C;

                when C =>
                        nextState <= D;
                when D =>
                        nextState <= A;

        END CASE;
        END PROCESS;


        calcOutput : PROCESS (currentState) IS BEGIN
                CASE currentState IS
                        WHEN A => dout <= din1;
                        WHEN B => dout <= din2;
                        WHEN C => dout <= din3;
                        WHEN D => dout <= din4;
                END CASE;
        END PROCESS;
END ARCHITECTURE;
```

9.8.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:                2  out of   4800     0%
 Number of Slice LUTs:                    10  out of   2400     0%
    Number used as Logic:                 10  out of   2400     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      10
   Number with an unused Flip Flop:        8  out of     10    80%
   Number with an unused LUT:              0  out of     10     0%
   Number of fully used LUT-FF pairs:      2  out of     10    20%
   Number of unique control sets:          1

IO Utilization:
 Number of IOs:                           41
 Number of bonded IOBs:                   41  out of    102    40%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 1  out of     16     6%
```

9.8.3 Timing analysis



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

   Minimum period: 1.686ns (Maximum Frequency: 593.208MHz)
   Minimum input arrival time before clock: 2.211ns
   Maximum output required time after clock: 5.021ns
   Maximum combinational path delay: 5.402ns
```

9.9 Router

9.9.1 VHDL source code

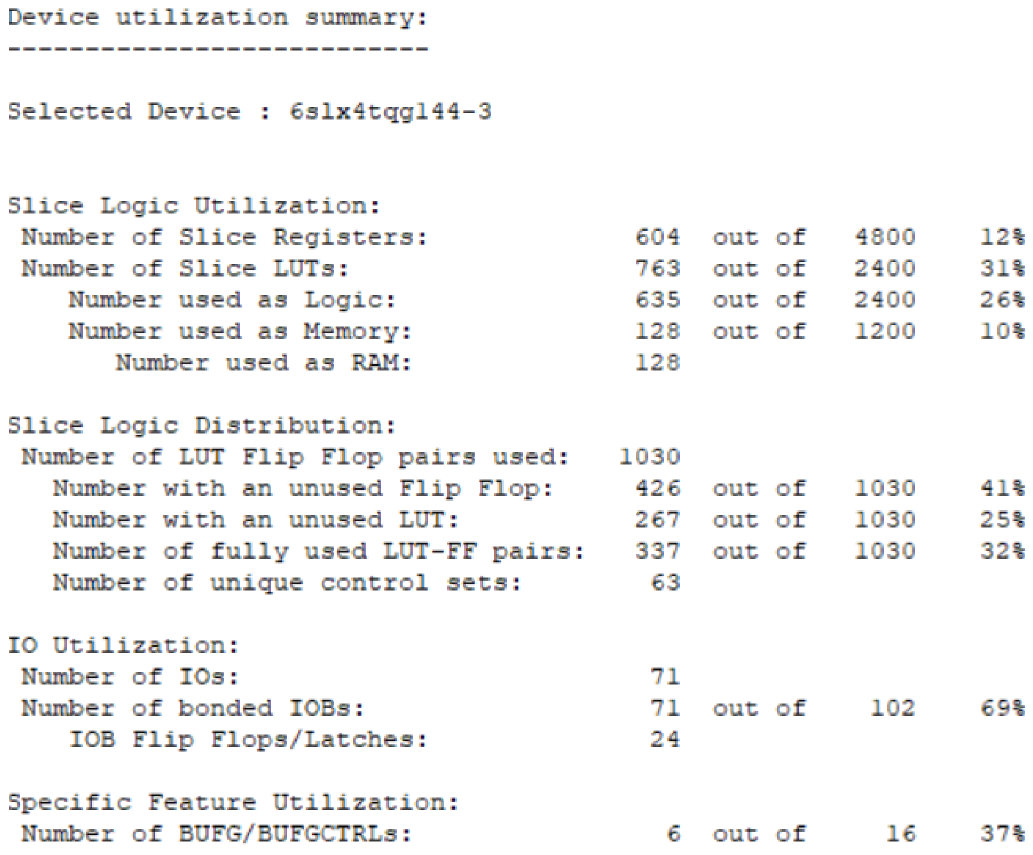
```
Library IEEE;
use ieee.std_logic_1164.all;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;

Entity router is
        PORT ( rst     : IN     std_logic;
               wclock  : IN     std_logic;
               rclock  : IN     std_logic;
               wr1     : IN     std_logic;
               wr2     : IN     std_logic;
               wr3     : IN     std_logic;
               wr4     : IN     std_logic;
               datai1  : IN     std_logic_vector (7 downto 0);
               datai2  : IN     std_logic_vector (7 downto 0);
               datai3  : IN     std_logic_vector (7 downto 0);
               datai4  : IN     std_logic_vector (7 downto 0);
               datao1  : OUT    std_logic_vector (7 downto 0);
               datao2  : OUT    std_logic_vector (7 downto 0);
               datao3  : OUT    std_logic_vector (7 downto 0);
               datao4  : OUT    std_logic_vector (7 downto 0)
       );
End router;

Architecture router_Arch of router is
       Signal  dataINReg1out,  dataINReg2out,  dataINReg3out  ,dataINReg4out
:std_logic_vector (7 downto 0);

       Signal  Demux1Q1 ,Demux1Q2 , Demux1Q3 , Demux1Q4,
               Demux2Q1 ,Demux2Q2 , Demux2Q3 , Demux2Q4,
               Demux3Q1 ,Demux3Q2 , Demux3Q3 , Demux3Q4,
               Demux4Q1 ,Demux4Q2 , Demux4Q3 , Demux4Q4 :std_logic_vector (7
downto 0);

       Signal  Q1O1out ,Q1O2out , Q1O3out , Q1O4out,
               Q2O1out ,Q2O2out , Q2O3out , Q2O4out,
               Q3O1out ,Q3O2out , Q3O3out , Q3O4out,
               Q4O1out  ,Q4O2out  ,  Q4O3out  ,  Q4O4out  :std_logic_vector  (7
downto 0);

       Signal  wreq1Q1 ,wreq1Q2 , wreq1Q3 , wreq1Q4,
               wreq2Q1 ,wreq2Q2 , wreq2Q3 , wreq2Q4,
               wreq3Q1 ,wreq3Q2 , wreq3Q3 , wreq3Q4,
               wreq4Q1 ,wreq4Q2 , wreq4Q3 , wreq4Q4 :std_logic;

       SIGNAL RRState : std_logic_vector(1 downto 0)  := "00";
       Signal  rreq :std_logic_vector (3 downto 0);

       Signal  dataOUTReg1IN,  dataOUTReg2IN,  dataOUTReg3IN  ,dataOUTReg4IN
:std_logic_vector (7 downto 0);
       Component Register8Bit IS
               PORT (  reset    : IN    std_logic;
                       clk      : IN    std_logic;
                       clk_en   : IN    std_logic;
                       data_in  : IN    std_logic_vector (7 downto 0);
                       data_out : OUT   std_logic_vector (7 downto 0)
               );
       END Component Register8Bit;
       FOR     ALL:     Register8Bit     USE     ENTITY     WORK.Register8Bit
(Register8BitArch);

       Component Demux IS
               PORT (  En      : IN     std_logic;
                       Sel     : IN     std_logic_vector (1 downto 0);
                       d_in    : IN     std_logic_vector (7 downto 0);
                       d_out1  : OUT    std_logic_vector (7 downto 0);
                       d_out2  : OUT    std_logic_vector (7 downto 0);
                       d_out3  : OUT    std_logic_vector (7 downto 0);
                       d_out4  : OUT    std_logic_vector (7 downto 0)
               );
       END Component Demux;
       For ALL: Demux USE ENTITY work.Demux(Demux_arch);

       Component FIFO is
       Port (  reset ,rclk,wclk,rreq,wreq : in std_logic;
                       datain: in std_logic_vector (7 downto 0);
                       dataout: out std_logic_vector (7 downto 0);
```

```
                empty, full : out std_logic);
        end component FIFO;
        FOR ALL: fifo use entity work.FIFO (structural);

        COMPONENT RoundRobinScheduler IS
                        PORT (  clk      : IN     std_logic;
                                din1     : IN     std_logic_vector   (7   downto
0);
                                din2     : IN     std_logic_vector   (7   downto
0);
                                din3     : IN     std_logic_vector   (7   downto
0);
                                din4     : IN     std_logic_vector   (7   downto
0);
                                dout     : OUT    std_logic_vector (7 downto 0)
                        );
        END COMPONENT RoundRobinScheduler;
        FOR  ALL:  RoundRobinScheduler  USE  ENTITY  WORK.RoundRobinScheduler
(RoundRobinSchedulerArch);
Begin
        RRState <= RRState + "01" when rising_edge(rclock);
        rreq <=         "0001" when RRState = "00" else
                        "0010" when RRState = "01" else
                        "0100" when RRState = "10" else
                        "1000" when RRState = "11" else
                        "XXXX";

        wreq1Q1 <= '1' when ((dataINReg1out(1 downto 0) = "00") and wr1 ='1'
) else '0';
        wreq1Q2 <= '1' when ((dataINReg2out(1 downto 0) = "00") and wr2 ='1'
) else '0';
        wreq1Q3 <= '1' when ((dataINReg3out(1 downto 0) = "00") and wr3 ='1'
) else '0';
        wreq1Q4 <= '1' when ((dataINReg4out(1 downto 0) = "00") and wr4 ='1'
) else '0';

        wreq2Q1 <= '1' when ((dataINReg1out(1 downto 0) = "01") and wr1 ='1'
) else '0';
        wreq2Q2 <= '1' when ((dataINReg2out(1 downto 0) = "01") and wr2 ='1'
) else '0';
        wreq2Q3 <= '1' when ((dataINReg3out(1 downto 0) = "01") and wr3 ='1'
) else '0';
        wreq2Q4 <= '1' when ((dataINReg4out(1 downto 0) = "01") and wr4 ='1'
) else '0';

        wreq3Q1 <= '1' when ((dataINReg1out(1 downto 0) = "10") and wr1 ='1'
) else '0';
        wreq3Q2 <= '1' when ((dataINReg2out(1 downto 0) = "10") and wr2 ='1'
) else '0';
        wreq3Q3 <= '1' when ((dataINReg3out(1 downto 0) = "10") and wr3 ='1'
) else '0';
        wreq3Q4 <= '1' when ((dataINReg4out(1 downto 0) = "10") and wr4 ='1'
) else '0';

        wreq4Q1 <= '1' when ((dataINReg1out(1 downto 0) = "11") and wr1 ='1'
) else '0';
        wreq4Q2 <= '1' when ((dataINReg2out(1 downto 0) = "11") and wr2 ='1'
) else '0';
        wreq4Q3 <= '1' when ((dataINReg3out(1 downto 0) = "11") and wr3 ='1'
) else '0';
```

```
        wreq4Q4 <= '1' when ((dataINReg4out(1 downto 0) = "11") and wr4 ='1'
) else '0';

        datai1_reg:  Register8Bit   PORT   MAP   (rst,wclock,   wr1,   datai1,
dataINReg1out);
        datai2_reg:  Register8Bit   PORT   MAP   (rst,wclock,   wr2,   datai2,
dataINReg2out);
        datai3_reg:  Register8Bit   PORT   MAP   (rst,wclock,   wr3,   datai3,
dataINReg3out);
        datai4_reg:  Register8Bit   PORT   MAP   (rst,wclock,   wr4,   datai4,
dataINReg4out);

        Demux1: Demux PORT MAP (wr1 ,dataINReg1out(1 downto 0),dataINReg1out
,Demux1Q1 ,Demux1Q2 , Demux1Q3 , Demux1Q4);
        Demux2: Demux PORT MAP (wr2 ,dataINReg2out(1 downto 0),dataINReg2out
,Demux2Q1 ,Demux2Q2 , Demux2Q3 , Demux2Q4);
        Demux3: Demux PORT MAP (wr3 ,dataINReg3out(1 downto 0),dataINReg3out
,Demux3Q1 ,Demux3Q2 , Demux3Q3 , Demux3Q4);
        Demux4: Demux PORT MAP (wr4 ,dataINReg4out(1 downto 0),dataINReg4out
,Demux4Q1 ,Demux4Q2 , Demux4Q3 , Demux4Q4);

        Data1Queue1:fifo              PORT             MAP             (rst
,rclock,wclock,rreq(0),wreq1Q1,Demux1Q1, Q1O1out);
        Data1Queue2:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(1),wreq1Q2,Demux2Q1, Q1O2out);
        Data1Queue3:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(2),wreq1Q3,Demux3Q1, Q1O3out);
        Data1Queue4:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(3),wreq1Q4,Demux4Q1, Q1O4out);

        Data2Queue1:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(0),wreq2Q1,Demux1Q2, Q2O1out);
        Data2Queue2:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(1),wreq2Q2,Demux2Q2, Q2O2out);
        Data2Queue3:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(2),wreq2Q3,Demux3Q2, Q2O3out);
        Data2Queue4:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(3),wreq2Q4,Demux4Q2, Q2O4out);

        Data3Queue1:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(0),wreq3Q1,Demux1Q3, Q3O1out);
        Data3Queue2:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(1),wreq3Q2,Demux2Q3, Q3O2out);
        Data3Queue3:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(2),wreq3Q3,Demux3Q3, Q3O3out);
        Data3Queue4:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(3),wreq3Q4,Demux4Q3, Q3O4out);

        Data4Queue1:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(0),wreq4Q1,Demux1Q4, Q4O1out);
        Data4Queue2:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(1),wreq4Q2,Demux2Q4, Q4O2out);
        Data4Queue3:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(2),wreq4Q3,Demux3Q4, Q4O3out);
        Data4Queue4:          fifo          PORT          MAP          (rst
,rclock,wclock,rreq(3),wreq4Q4,Demux4Q4, Q4O4out);

        RR1: RoundRobinScheduler PORT MAP (rclock, Q1O1out, Q1O2out, Q1O3out,
Q1O4out, dataOUTReg1IN);
        RR2: RoundRobinScheduler PORT MAP (rclock, Q2O1out, Q2O2out, Q2O3out,
Q2O4out, dataOUTReg2IN);
```

```
        RR3: RoundRobinScheduler PORT MAP (rclock, Q3O1out, Q3O2out, Q3O3out,
Q3O4out, dataOUTReg3IN);
        RR4: RoundRobinScheduler PORT MAP (rclock, Q4O1out, Q4O2out, Q4O3out,
Q4O4out, dataOUTReg4IN);

        datao1_reg: Register8Bit PORT MAP (rst,rclock, '1', dataOUTReg1IN,
datao1);
        datao2_reg: Register8Bit PORT MAP (rst,rclock, '1', dataOUTReg2IN,
datao2);
        datao3_reg: Register8Bit PORT MAP (rst,rclock, '1', dataOUTReg3IN,
datao3);
        datao4_reg: Register8Bit PORT MAP (rst,rclock, '1', dataOUTReg4IN,
datao4);
End Architecture router_Arch;
```

9.9.2 Total equivalent gate count



```
Device utilization summary:
---------------------------

Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:            604  out of   4800    12%
 Number of Slice LUTs:                 763  out of   2400    31%
    Number used as Logic:              635  out of   2400    26%
    Number used as Memory:             128  out of   1200    10%
       Number used as RAM:             128

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  1030
   Number with an unused Flip Flop:    426  out of   1030    41%
   Number with an unused LUT:          267  out of   1030    25%
   Number of fully used LUT-FF pairs:  337  out of   1030    32%
   Number of unique control sets:       63

IO Utilization:
 Number of IOs:                         71
 Number of bonded IOBs:                 71  out of    102    69%
    IOB Flip Flops/Latches:             24

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:               6  out of     16    37%
```

9.9.3 Timing analysis

All modules will work on same clock, then for all modules to work correctly, we should choose maximum clock period of all modules. FIFO have maximum clock period with 5.980ns. Then minimum clock router can work on is 5.980ns.

10.0 Appendix B

10.1 Router Testbench

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

Entity RouterTb IS
END ENTITY RouterTb;

ARCHITECTURE RouterTbArch OF RouterTb IS
    COMPONENT router is
        PORT ( rst      : IN     std_logic;
            wclock     : IN     std_logic;
            rclock     : IN     std_logic;
            wr1        : IN     std_logic;
            wr2        : IN     std_logic;
            wr3        : IN     std_logic;
            wr4       : IN     std_logic;
            datai1     : IN     std_logic_vector (7 downto 0);
            datai2     : IN     std_logic_vector (7 downto 0);
            datai3     : IN     std_logic_vector (7 downto 0);
            datai4     : IN     std_logic_vector (7 downto 0);
            datao1     : OUT    std_logic_vector (7 downto 0);
            datao2     : OUT    std_logic_vector (7 downto 0);
            datao3     : OUT    std_logic_vector (7 downto 0);
            datao4     : OUT    std_logic_vector (7 downto 0)
        );
    END COMPONENT router;

    FOR DUT: router USE ENTITY WORK.router (router_Arch);

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

BEGIN

    DUT: router PORT MAP (rst, wclock, rclock, wr1, wr2, wr3, wr4, datai1,
datai2, datai3, datai4, datao1, datao2, datao3, datao4);

        wclk: PROCESS IS BEGIN
                wclock <= '0', '1' after 5 ns;
                wait for 10 ns;
        END PROCESS;

         rclk: PROCESS IS BEGIN
                rclock <= '0', '1' after 5 ns;
                wait for 10 ns;
        END PROCESS;

        warnings: PROCESS IS BEGIN
                wait for 10 ns;
                ----Checks if Data output 1 ends with "00" ----
                Assert (datao1(1 downto 0) = "00" or datao1 = "ZZZZZZZZ" or
datao1 = "XXXXXXXX")
                Report "Data output 1 should end with 00"
                Severity error;
                ----Checks if Data output 2 ends with "01" ----
                Assert (datao2(1 downto 0) = "01" or datao2 = "ZZZZZZZZ" or
datao2 = "XXXXXXXX")
                Report "Data output 2 should end with 01"
                Severity error;

                ----Checks if Data output 3 ends with "10" ----
                Assert (datao3(1 downto 0) = "10" or datao3 = "ZZZZZZZZ" or
datao3 = "XXXXXXXX")
                Report "Data output 3 should end with 10"
                Severity error;

                ----Checks if Data output 4 ends with "11" ----
                Assert (datao4(1 downto 0) = "11" or datao4 = "ZZZZZZZZ" or
datao4 = "XXXXXXXX")
                Report "Data output 4 should end with 11"
                Severity error;

        END PROCESS;

        tb : PROCESS IS BEGIN
                rst <= '1';
                wait for 10 ns;

                --Test 1, Setting Values
                ----Select --> 00 ----
                datai1   <=   "11001000";datai2   <=   "10010000";datai3   <=
"01010100";datai4 <= "10001000";
                rst <= '0';wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
                wait for 10 ns;

                ----Select --> 01 ----
                datai1   <=   "10100001";datai2   <=   "01110001";datai3   <=
"10100101";datai4 <= "11100001";
                wait for 10 ns;

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

                ----Select --> 11 ----
                datai1   <=   "10101111";datai2   <=   "01100111";datai3   <=
"01110011";datai4 <= "01100111";
                wait for 10 ns;

                ----Removing Items ----
                datai1   <=   "ZZZZZZZZ";datai2   <=   "ZZZZZZZZ";datai3   <=
"ZZZZZZZZ";datai4 <= "ZZZZZZZZ";
                wait for 10 ns;

                wr1 <= '0';wr2 <= '0';wr3 <= '0';wr4 <= '0';
                wait for 100 ns;

                --Test 2, Setting values while wr ='0'
                datai1   <=   "11001000";datai2   <=   "10010000";datai3   <=
"01010100";datai4 <= "10001000";
                wait for 10 ns;

                datai1   <=   "10100001";datai2   <=   "01110001";datai3   <=
"10100101";datai4 <= "11100001";
```

```
                wait for 10 ns;

                datai1   <=   "10111010";datai2   <=   "10110010";datai3   <=
"10110110";datai4 <= "10100010";
                wait for 10 ns;

                datai1   <=   "10101111";datai2   <=   "01100111";datai3   <=
"01110011";datai4 <= "01100111";
                wait for 10 ns;

                wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
                wait for 10 ns;

                ----Removing Items ----
                datai1   <=   "ZZZZZZZZ";datai2   <=   "ZZZZZZZZ";datai3   <=
"ZZZZZZZZ";datai4 <= "ZZZZZZZZ";
                wait for 10 ns;

                wr1 <= '0';wr2 <= '0';wr3 <= '0';wr4 <= '0';
                wait for 100 ns;

           --Test 3, Setting values with varying queues each clock

                wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
                datai1   <=   "10000000";datai2   <=   "10000001";datai3   <=
"10000010";datai4 <= "10000011";
                wait for 10 ns;

                datai1   <=   "10100011";datai2   <=   "10100000";datai3   <=
"10100001";datai4 <= "10100010";
                wait for 10 ns;

                datai1   <=   "11100010";datai2   <=   "11100011";datai3   <=
"11100000";datai4 <= "11100001";
                wait for 10 ns;

                datai1   <=   "11110001";datai2   <=   "11110010";datai3   <=
"11110011";datai4 <= "11110000";
                wait for 10 ns;

                ----Removing Items ----
                datai1   <=   "ZZZZZZZZ";datai2   <=   "ZZZZZZZZ";datai3   <=
"ZZZZZZZZ";datai4 <= "ZZZZZZZZ";
                wait for 100 ns;

          --Test 4 , Stress testing on first queue

                wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
                datai1   <=   "10000000";datai2   <=   "10010000";datai3   <=
"11000000";datai4 <= "10000100";
                wait for 10 ns;

                datai1   <=   "10100000";datai2   <=   "10100000";datai3   <=
"10100000";datai4 <= "10100000";
                wait for 10 ns;

                datai1   <=   "11100000";datai2   <=   "11100000";datai3   <=
"11100000";datai4 <= "11100000";
                wait for 10 ns;
                datai1   <=   "11110000";datai2   <=   "11110000";datai3   <=
"11110000";datai4 <= "11110000";
                wait for 10 ns;

                ----Removing Items ----
                datai1   <=   "ZZZZZZZZ";datai2   <=   "ZZZZZZZZ";datai3   <=
"ZZZZZZZZ";datai4 <= "ZZZZZZZZ";
                wait for 10 ns;
                wait;

        END PROCESS;
END ARCHITECTURE;
```

10.2 Warning process code

```
warnings: PROCESS IS BEGIN             wait          for          10          ns;
                ----Checks   if   Data   output   1   ends   with   "00"   ----
                Assert  (datao1(1 downto 0)  = "00" or datao1 = "ZZZZZZZZ" or
datao1                                     =                      "XXXXXXXX")
                Report "Data output 1 should end with 00"
        Severity                                                          error;
                                ----Checks if Data output 2 ends with "01" --
--
                Assert  (datao2(1 downto 0)  = "01" or datao2 = "ZZZZZZZZ" or
datao2                                     =                      "XXXXXXXX")
                Report "Data output 2 should end with 01"
        Severity                                                          error;

                ----Checks   if   Data   output   3   ends   with   "10"   ----
                Assert  (datao3(1 downto 0)  = "10" or datao3 = "ZZZZZZZZ" or
datao3                                     =                      "XXXXXXXX")
                Report "Data output 3 should end with 10"
        Severity                                                          error;

                ----Checks   if   Data   output   4   ends   with   "11"   ----
                Assert  (datao4(1 downto 0)  = "11" or datao4 = "ZZZZZZZZ" or
datao4                                     =                      "XXXXXXXX")
                Report "Data output 4 should end with 11"
        Severity                                                          error;

        END PROCESS;
```

10.3 Test case 1 code

```
        --Test 1, Setting Values
                ----Select --> 00 ----
                datai1   <=   "11001000";datai2   <=   "10010000";datai3   <=
"01010100";datai4 <= "10001000";
                rst <= '0';wr1 <= '1';wr2 <= '1';wr3 <= '1';wr4 <= '1';
                wait for 10 ns;

                ----Select --> 01 ----
                datai1   <=   "10100001";datai2   <=   "01110001";datai3   <=
"10100101";datai4 <= "11100001";
                wait for 10 ns;

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

                ----Select --> 11 ----
                datai1   <=   "10101111";datai2   <=   "01100111";datai3   <=
"01110011";datai4 <= "01100111";
                wait for 10 ns;

                ----Removing Items ----
                datai1   <=   "ZZZZZZZZ";datai2   <=   "ZZZZZZZZ";datai3   <=
"ZZZZZZZZ";datai4 <= "ZZZZZZZZ";
                wait for 10 ns;

                wr1 <= '0';wr2 <= '0';wr3 <= '0';wr4 <= '0';
                wait for 100 ns;
```

10.4 Test case 2 code

```
                  --Test 2, Setting values while wr ='0'            datai1
<=  "11001000";  datai2  <=  "10010000";  datai3  <=  "01010100";  datai4  <=
"10001000";
                wait                 for                 10                 ns;

                datai1  <=  "10100001";  datai2  <=  "01110001";  datai3  <=
"10100101";             datai4               <=                 "11100001";
                wait                 for                 10                 ns;
                        datai1 <= "10111010"; datai2 <= "10110010"; datai3 <=
"10110110";             datai4               <=                 "10100010";
                wait                 for                 10                 ns;
                        datai1 <= "10101111"; datai2 <= "01100111"; datai3 <=
"01110011";             datai4               <=                 "01100111";
                wait                 for                 10                 ns;
                        wr1   <=   '1';wr2   <=   '1';wr3   <=   '1';wr4   <=   '1';
                wait                 for                 10                 ns;
                                ----Removing            Items            ----
                datai1  <=  "ZZZZZZZZ";  datai2  <=  "ZZZZZZZZ";  datai3  <=
"ZZZZZZZZ";             datai4               <=                 "ZZZZZZZZ";
                wait                 for                 10                 ns;
                        wr1   <=   '0';wr2   <=   '0';wr3   <=   '0';wr4   <=   '0';
                wait for 100 ns;
```

10.5 Test case 3 code

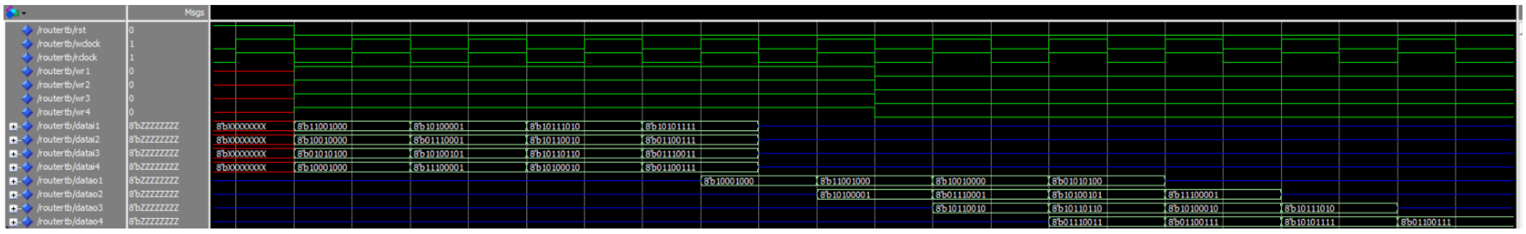
```
--Test    3,    Setting    values    with    varying    queues    each    clock

                wr1   <=   '1';wr2   <=   '1';wr3   <=   '1';wr4   <=   '1';
                datai1  <=  "10000000";  datai2  <=  "10000001";  datai3  <=
"10000010";          datai4                <=                "10000011";
                wait                for                10                ns;
                        datai1 <= "10100011"; datai2 <= "10100000"; datai3 <=
"10100001";          datai4                <=                "10100010";
                wait                for                10                ns;
                        datai1 <= "11100010"; datai2 <= "11100011"; datai3 <=
"11100000";          datai4                <=                "11100001";
                wait                for                10                ns;
                                datai1  <=  "11110001";  datai2  <=  "11110010";
datai3       <=        "11110011";        datai4        <=        "11110000";
                wait                for                10                ns;
        ----Removing                       Items                        ----
                datai1  <=  "ZZZZZZZZ";  datai2  <=  "ZZZZZZZZ";  datai3  <=
"ZZZZZZZZ";          datai4                <=                "ZZZZZZZZ";
                wait for 100 ns;
```

10.6 Test case 4 code

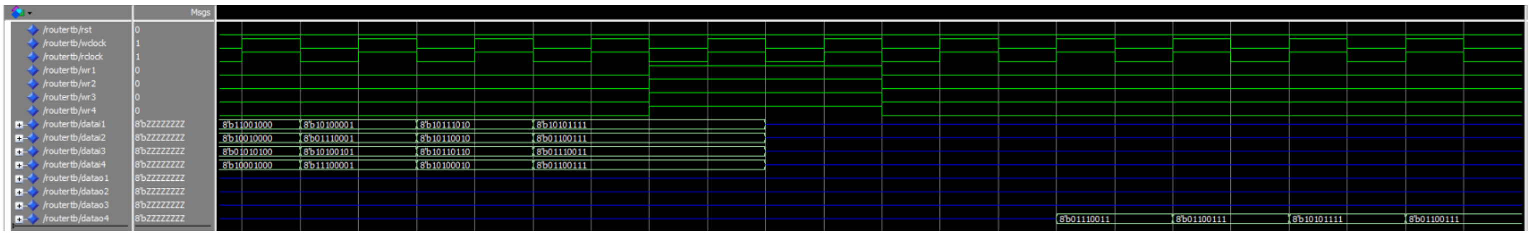
```
          --Test     4     ,     Stress     testing     on     first     queue

                wr1   <=   '1';wr2   <=   '1';wr3   <=   '1';wr4   <=   '1';
                datai1  <=  "10000000";  datai2  <=  "10010000";  datai3  <=
"11000000";          datai4                <=                "10000100";
                wait                for                10                ns;
                        datai1 <= "10100000"; datai2 <= "10100000"; datai3 <=
"10100000";          datai4                <=                "10100000";
                wait                for                10                ns;

                datai1  <=  "11100000";  datai2  <=  "11100000";  datai3  <=
"11100000";          datai4                <=                "11100000";
                wait                for                10                ns;
                        datai1 <= "11110000"; datai2 <= "11110000"; datai3 <=
"11110000";          datai4                <=                "11110000";
                wait                for                10                ns;
                 ----Removing                   Items                   ----
                datai1  <=  "ZZZZZZZZ";  datai2  <=  "ZZZZZZZZ";  datai3  <=
"ZZZZZZZZ";          datai4                <=                "ZZZZZZZZ";
                wait for 10 n;s
```

11.0 Appendix C

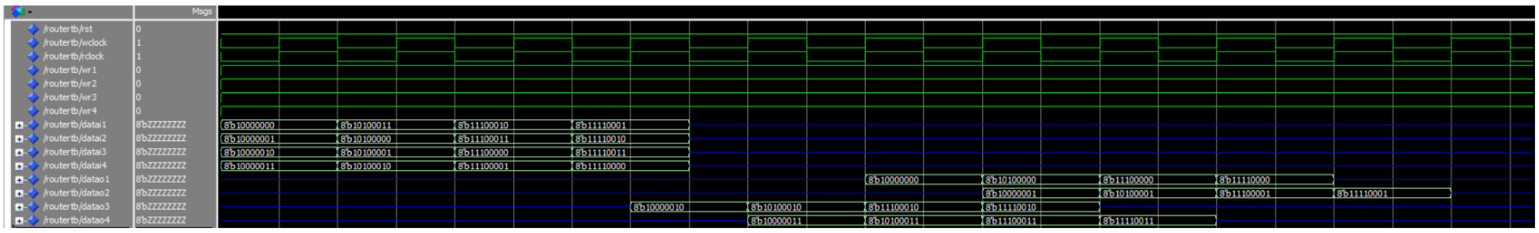
11.1 Test case 1 waveform



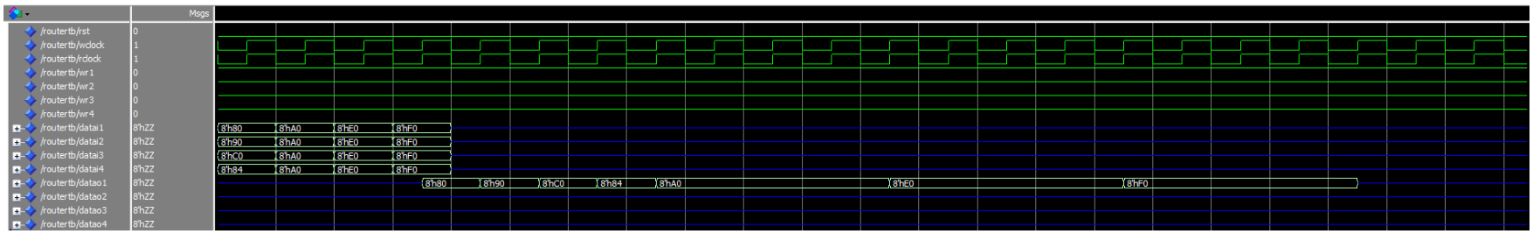
11.2 Test case 2 waveform



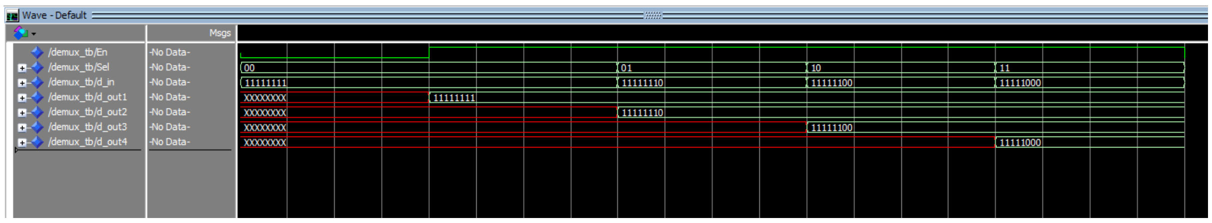
11.3 Test case 3 waveform



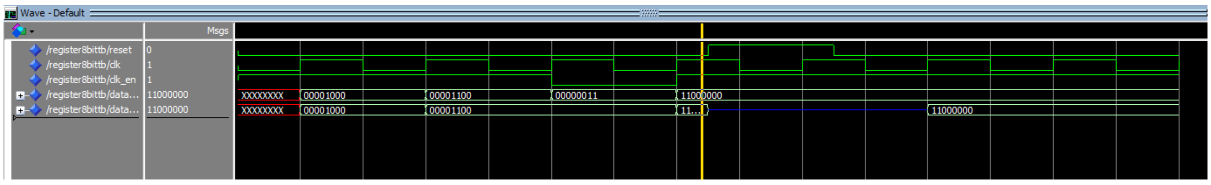
11.4 Test case 4 waveform



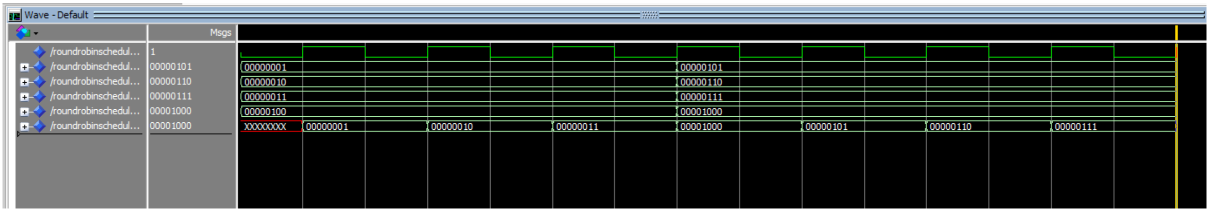
11.5 Demux:



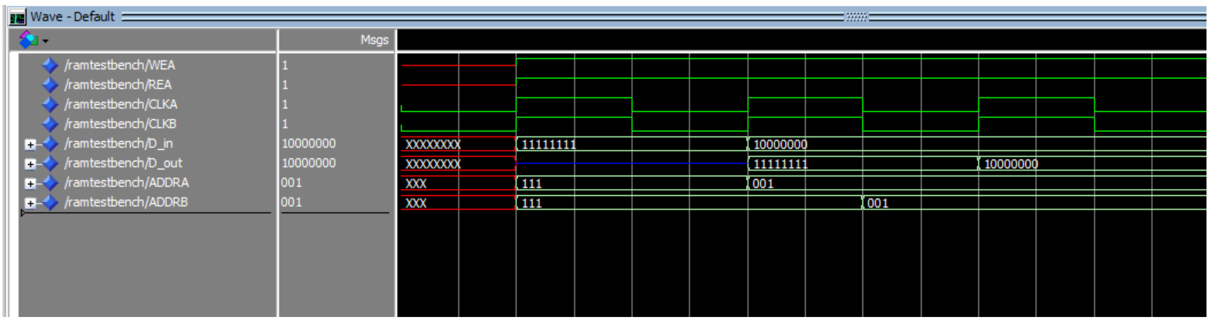
11.6 Register:



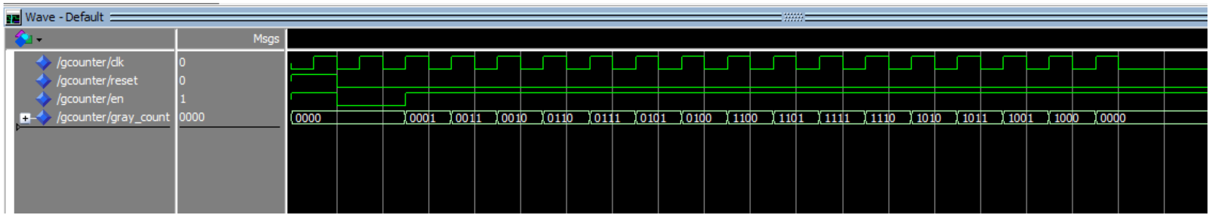
11.7 Round Robin:


Wave - Default
Msgs
/roundrobinschedul... 1
/roundrobinschedul... 00000101
/roundrobinschedul... 00000110
/roundrobinschedul... 00000111
/roundrobinschedul... 00001000
/roundrobinschedul... 00001000
00000001
00000010
00000011
00000100
00000101
00000110
00000111
00001000
XXXXXXXX

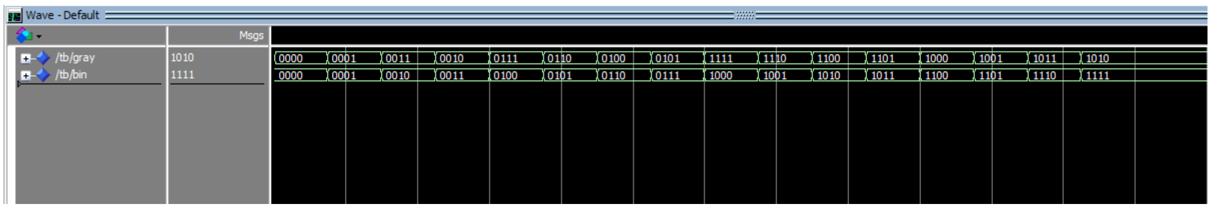
11.8 Ram:



11.9 Gray Counter:

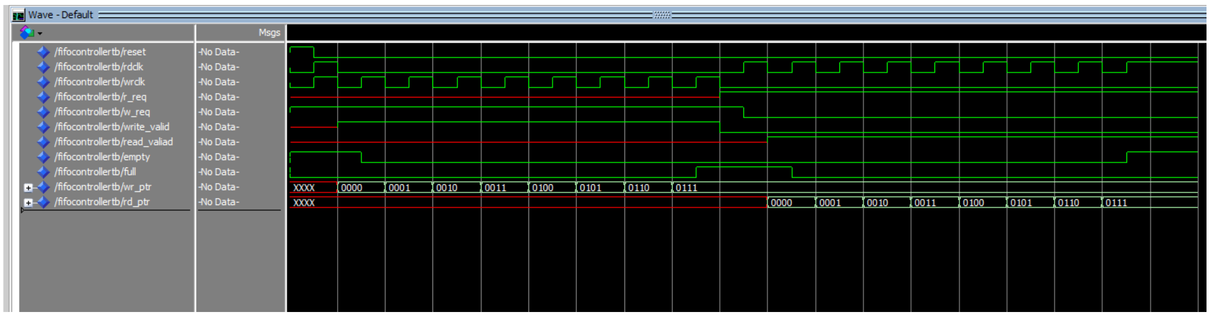


11.10 Converter:



```
Wave - Default
              Msgs
/tb/gray      1010
/tb/bin       1111
```

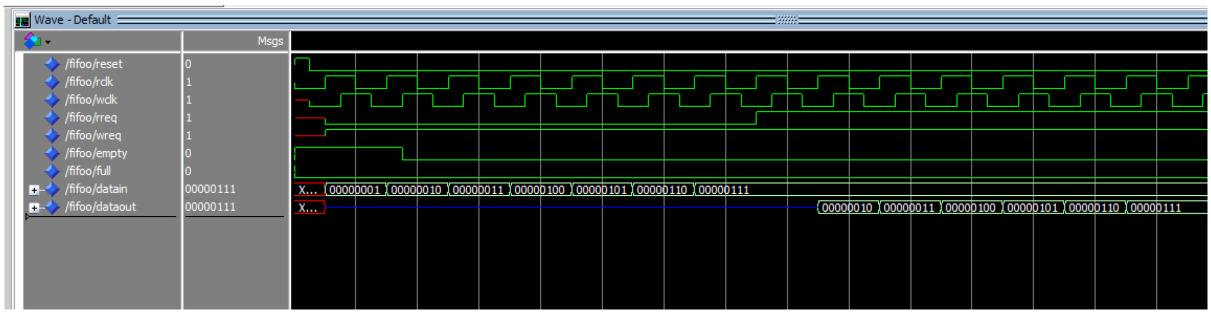
11.11 Controller:



```
Wave - Default
                                Msgs
/fifocontrollertb/reset         -No Data-
/fifocontrollertb/rdclk         -No Data-
/fifocontrollertb/wrclk         -No Data-
/fifocontrollertb/r_req         -No Data-
/fifocontrollertb/w_req         -No Data-
/fifocontrollertb/write_valid   -No Data-
/fifocontrollertb/read_valiad   -No Data-
/fifocontrollertb/empty         -No Data-
/fifocontrollertb/full          -No Data-
/fifocontrollertb/wr_ptr        -No Data-
/fifocontrollertb/rd_ptr        -No Data-
```

XXXX 0000 0001 0010 0011 0100 0101 0110 0111
XXXX 0000 0001 0010 0011 0100 0101 0110 0111

11.12 FIFO:



```
Wave - Default
                  Msgs
/fifoo/reset      0
/fifoo/rclk       1
/fifoo/wclk       1
/fifoo/rreq       1
/fifoo/wreq       1
/fifoo/empty      0
/fifoo/full       0
/fifoo/datain     00000111
/fifoo/dataout    00000111
```

X... 00000001 00000010 00000011 00000100 00000101 00000110 00000111
X... 00000010 00000011 00000100 00000101 00000110 00000111

12.0 References

[1] Chandravanshi, R., & Tiwari, V. (n.d.). Review: VHDL Based NOC Router Architecture. International Journal of Innovative Trends in Engineering, 14(2395-2946).

[2] Wanjari, M. M., Agrawal, P., & Kshirsagar, R. V. (2015). Design of NoC Router Architecture using VHDL. *International Journal of Computer Applications, 115*.

[3] Bahmani, M., Sheibanyrad, A., Petrot, F., Dubois, F., & Durante, P. (2012). A 3D-NoC Router Implementation Exploiting Vertically-Partially-Connected Topologies. *2012 IEEE Computer Society Annual Symposium on VLSI*. doi:10.1109/isvlsi.2012.19

[4] State Processes. (n.d.). Retrieved January 05, 2021, from https://www.vhdl-online.de/courses/system_design/synthesis/finite_state_machines_and_vhdl/state_processes