"CHIP LINK"
A NOC ROUTER FOR NEXT GENERATION FPGA

A DISSERTION
SUBMITTED TO THE DEPARTMENT OF ELECTRONICS
AND ELECTRICAL COMMUNICATIONS
ENGINEERING
OF CAIRO UNIVERSITY
IN PARIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR

MARWAN A. THABET

ISLAM I. GAD

KHALED M. TAREK

MAHMOUD M. MOHAMMED

OMAR M. ABDEL KARIM

AMR M. AMR

JULY 2014

Abstract

As Modern digital integrated circuits go more complex and the numbers of SoCs on a small electronic chip are rapidly increased, the need for a more sophisticated communication infrastructure between different IP cores and system on-chips is raised. NoCs is like a trivial solution to the restrictions created by the limited bandwidth and flexibility y of current cross-bars and buses. It helps to overcome the clock skew problem on large chips.

The interconnection networks are relatively simple and it is easy to design an inter-connection network that replaces and solves regular crossbars and buses problems. Unfortunately, if the basic principles are not understood it is also easy to design an intercon-nection network that works poorly if at all.

In this thesis we present one of a simple NoCs for the purpose of working on FPGAs platforms, we start our thesis with couple of introductory chapters gathered from reliable and rich resources in field that enable the reader to begin his first steps and find his path in inter-connection design.

In the following chapters we introduce our simple router "CHIP-LINK" from the very beginning of FPGAs and passing through all the design flow starting with the behavioural modelling down to the lower levels ending with implementation a floorplan of the proposed design and simulation results.

Acknowledgements

All thanks to God, the **Almighty**, for helping us by giving us the patience and courage to introduce this work.

With deep appreciation we would like to thank Dr/ Hassan Mustafa for his aid and support all over the year.

Table of Contents

1 Overview on FPGA

1.1 What is FPGA???

FPGA which is an abbreviation to Field –Programmable Gate Array, is a pre-fabricated silicon device that can be programmed to be any digital circuit or digital system that the designer need.

FPGA consists of array of logic blocks of different types such as memories, multipliers, processors and other logic blocks. These blocks placed and interconnected by routing fabric.

These arrays of blocks are surrounded by input-output blocks which are considered to be the interface between FPGA and the outside world.

The input-output blocks and the routing fabric connected the logic blocks are programmed to make FPGA add certain function by a certain programmable technique.

The programming process change the connection between the blocks and choose the input-output blocks that the designer need to do the function which means that it makes a change in the behavior of pre-fabricated chip after fabrication.

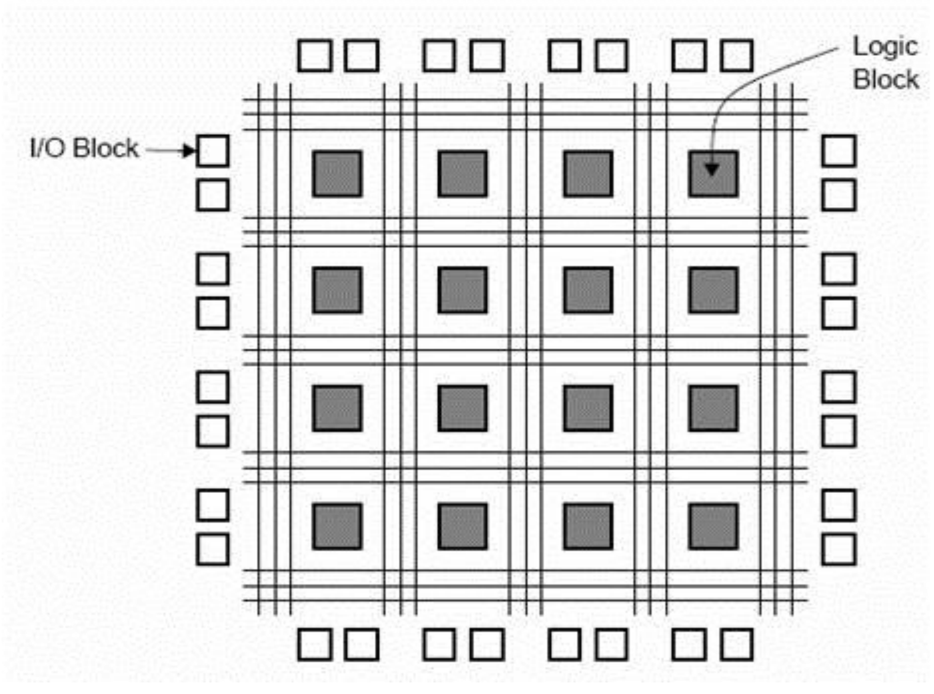


Figure 1

1.2 The difference between FPGA and micro-controllers.

In the field of digital electronics and communications, micro-controllers are known as simple computers that can be placed in a single chip and programmed to do simple tasks for other hardware.

Micro-controllers have some peripherals embedded in it such as memory and timers. They can be designed to do its certain task as a dedicated device .

FPGA contains millions of logic blocks that can be electrically programmed to perform a certain task ,its more flexible than controllers as the micro-controllers has its own circuitry and instruction set which makes a restriction on the programming to do a certain task .But FPGA have a bigger usage area.

FPGA consumes power than controllers making them un suitable for applications where power drain is an issue.

FPGA takes a longer time to set up than micro-controller as you would have to write all the code from scratch and convert it to machine language; also building devices with FPGA are more costly than micro-controllers, that are why FPGA's are usually seen in products that have a high degree of complexity but with low demand.

In the market level, when the demand rises and mass-production becomes necessary, the circuit is moved to AISC.

1.3 ASIC technology

ASIC which is an abbreviation to application-specific integrated circuit, is a micro-chip designed for a specific application.

ASIC take months to fabricate and cost hundreds of thousands up to millions of dollars during fabrication process. It is much more than FPGA fabrication cost.

The draw-backs of FPGA over ASIC appears in area, speed and power consumption in which FPGA occupies area 20 to 30 times more than the area occupied by ASIC. Speed performance in FPGA is 3 to 4 times slower than ASIC speed chip. Also FPGA consumes 10 times dynamic power than ASIC power consumption.

These disadvantages of FPGA in area, speed and power consumption arises from the programmable routing fabric in FPGA.

ASIC design is more difficult and expensive. Also the investment required to produce a perfect useful design consists of more items in terms of money and time such as CAD tools for verification, simulation and timing analysis. So due to these high cost and the need for higher return on investment most digital design forward towards FPGA implementation. But in the market field the most importance thing is to have a perfect product to be acceptable for users so the companies have the capability to go to ASIC technology and handle high cost in order to make a perfect product due to the disadvantages of FPGA implementation. So the optimized design in FPGA to get rid of its draw-backs in area speed and power consumption will spared high cost and makes the market field more comfortable.

"Figure 2" shows that the cost to build ASIC starts higher than FPGA but in a certain level in manufacturing the cost of FPGA reaches the cost of ASIC at a critical point.

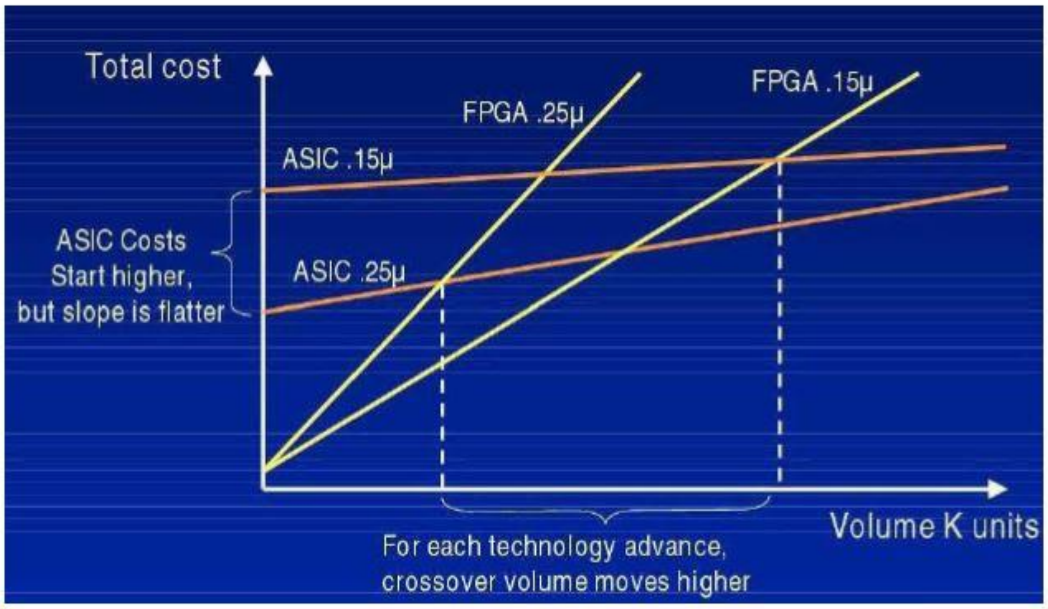


Figure 2

In deep volume in k units the cost of FPGA becomes more than that in ASIC and continue to be more high which makes FPGA has no value in the market field due to the higher cost in the future.

The optimized design will introduce new technology in the world of digital design like nanotechnology [4].

Figure 3 shows that the routing power becomes more significant percentage of the total power and expected to exceed the logic power with technology scaling. The increased inter-connect and routing complexity reduces FPGA speed and scalability. This is the reason of why designers seek to optimize the routing techniques in FPGA.

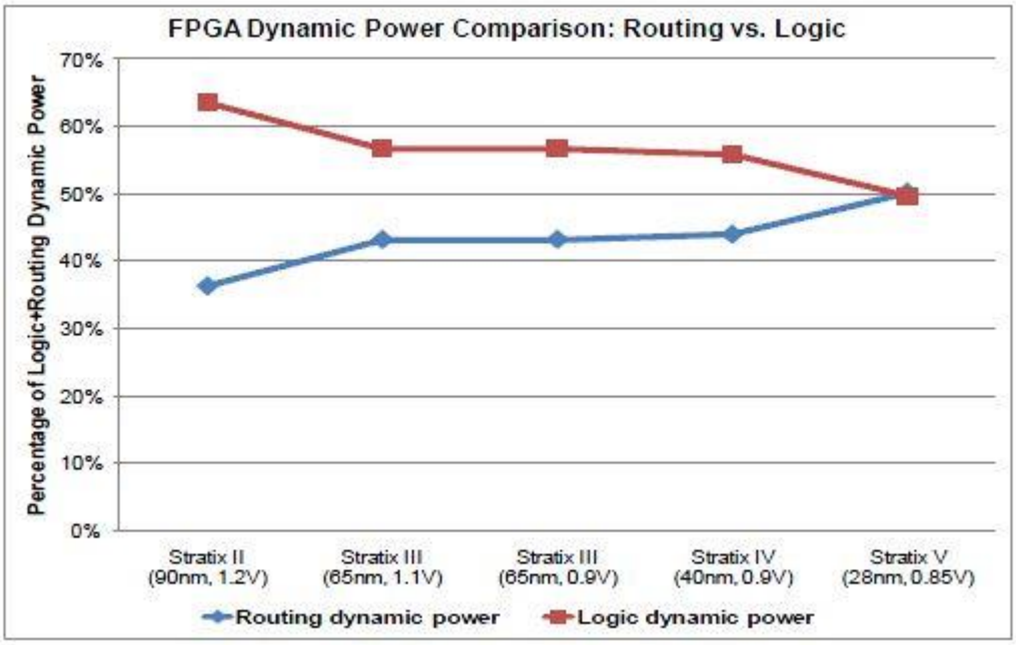


Figure 3 Different Graphs for both routing and logic Dynamic power

2 Network on chips

2.1 Overview

Due to the draw-backs of FPGA in area, speed and power consumption, electronics engineers must be able to solve these problems by optimizing the design in FPGA to make FPGA more reliable and usable in human life.

Engineers introduce the solution by making a full-network fabricated in FPGA chip which called network on chip technology in FPGA system design and from here we are able to get rid of the dis-advantages.

Due to the increasing of logic blocks and the need to integrate millions of blocks in a single system on chip like FPGA, communication management becomes more critical as more functions that supported latency and bandwidth requirements in an interconnect fabric must be introduced.

Designers call network on chip technology a front-end solution to a back-end problem. It is like to make more bridges between two nodes to facilitate the traffic of data between these two nodes.

We can summarize why we use network on chip in digital systems on chip due to four reasons:

1. Reduce wire routing congestion.
2. Ease timing closure.
3. Higher operating frequencies.
4. Changing IP Protocol easily.

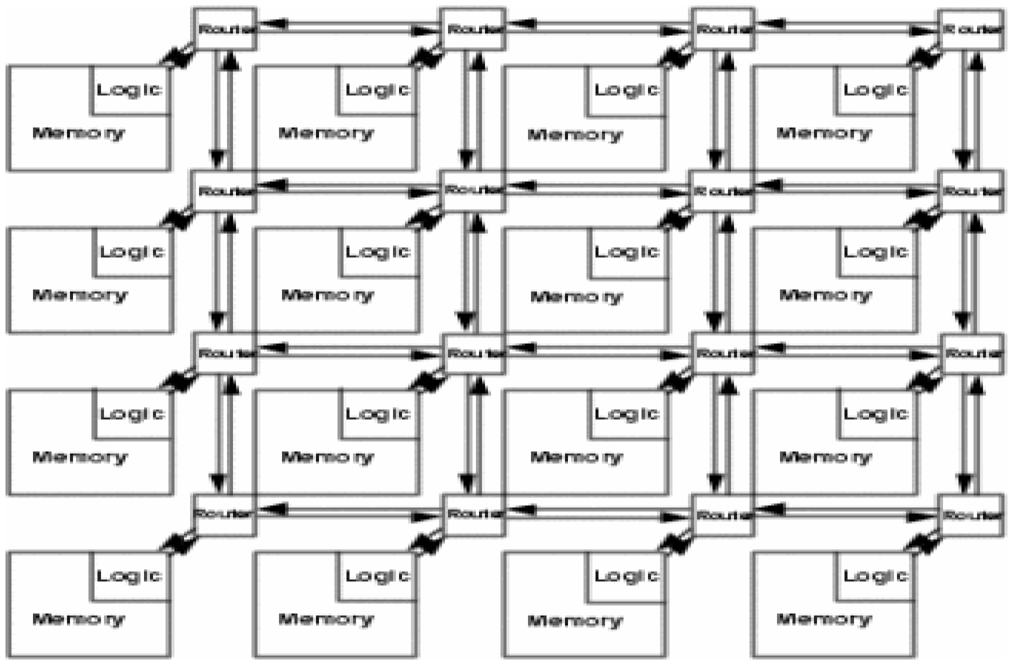


Figure 4

In the previous figure, there is an example of network in system on chip which is a block contains a large number of logic blocks connecting with each other by routers that introduces multi paths come and go between the logic blocks.

By looking to these figure, you will be able to understand that network on chip is like building bridges and paved roads in an empty area.

We will discuss the requirements of why we use network on chip in the following points:

2.2 First routing congestion

Network on chip avoids routing congestion by reducing number of connected wires in a single fabricated chip.

Routing congestion becomes more significant factor in digital systems when number of IP blocks increased in it.

Wire routing congestion occurs in digital systems when a lot of wires are routed in a narrow space which leads to errors in routing process and introduces more delay.

Since the wires that carry the data are routed after power supply and clock wires as the signal wires are smaller and shorter than clock wires and power supply wires so the signal wires are constrained because they must be routed in a manner that accommodates the existing power supply at clock wires.

As system on chip designs becomes more complex, routing congestion increases system on chip complexity due to shrinking sizes of transistors which lead to:

1. More transistors per area make the system design more complex.
2. Large system on chip enables the opportunity to integrate more IP blocks on one chip. The number of wires required for system on chip grows proportional to the square of number of transistors in one chip.
3. The cross-sectional size of wires connecting IP blocks shrinks less than size of transistors.

2.3 Second timing closure

Network on chip introduces solution of the problem of timing closure in which system on chip with old generations suffers from the problem of delay and not managing the time accurately but now in new generation SoC, network on chip introduces a design in pipelining concept to overcome the timing closure issue.

Pipelining concept is to make some operations at the same time and it will be discussed later in this book.

2.4 Third operating frequency

NoC technology simplifies the hardware requires for switching and routing functions so it makes the design to reach higher operating frequency.

By applying the pipelining concept, more data transferred in a less time and the bandwidth will be managed better so the system will be operated at higher frequencies.

Also network on chip follows GALS technology which is synchronous modules with locally generated clocks with asynchronous connections between them.

GALS is an abbreviation to globally asynchronization locally synchronization.

2.5 Fourth changing IP

NoC deals with packets in which there is a block that splitting the data into packets which are small versions of the original data. These blocks are placed before the router and called network interface cores NIC.

Network on chip change the formation of data from original data into packets. In network engineering, we call this issue changing from IP to transport layer.

3 Basics of networking

3.1 Introduction

Because the demand for interconnection networks that have grown more rapidly than the capability of the underlying wires, interconnection networks have become a critical bottle-neck in most systems.

The main function of the network is that introducing the connection between the cores like processors, multipliers and logic blocks to perform the transferring of data between them accurately.

The transferring process of the data must be accurate which means that translation of data between two points' source and destination without losses and without losses in band-width.

Inter-connection networks are used in digital systems which have many components to connect. In computer systems, networks are used to connect processors with memories and I/O ports. Also in FPGA, networks are used to connect the logic blocks with each other and con-nect with the input-output blocks to make all available paths to make the using of FPGA more simple and reliable to the designers and users.

Networks are more important in FPGA as FPGA contains millions of blocks which need to be connected with each other.

Also inter-connected networks are used to connect many FPGA's with each other fabricated in a single chip.

Inter-connection network play an important role in the performance of the system in which the inter-connection networks between the cores determines the latency and bandwidth of the system which are two key performances in the digital systems.

There are some things that we must interest when we deal with networks in which these things are considered to be the basics of networking [1], they are:

1. Network Topology.
2. Routing Algorithms.
3. Flow Control.
4. Router Architecture.
5. Performance of Network.

3.2 Network Topology

In communication networks, network topology is a description of the arrangement of nodes and lines in the network.

There are two types to describe a specific network physical topology and logical topology.

Physical topology of a network is the geometry lay-out description of the network while logical topology refers to how signals in the paths go in a certain direction.

There are many types of physical topology like:

1. Bus Topology in which all the nodes in the network connect with the main bus as in Fig 5.
2. Star Topology in which all the nodes connect with the main node as in Fig 6.
3. Ring Topology in which all the nodes connect with each other in a circle way as in
4. Fig 7.
5. Token Ring Topology is the same as ring but the signals go in only one direction.
6. Mesh Topology in which all the nodes connect with each other as in Fig 8.
7. Tree Topology which is a combination of much architecture in one architecture as in fig 9.

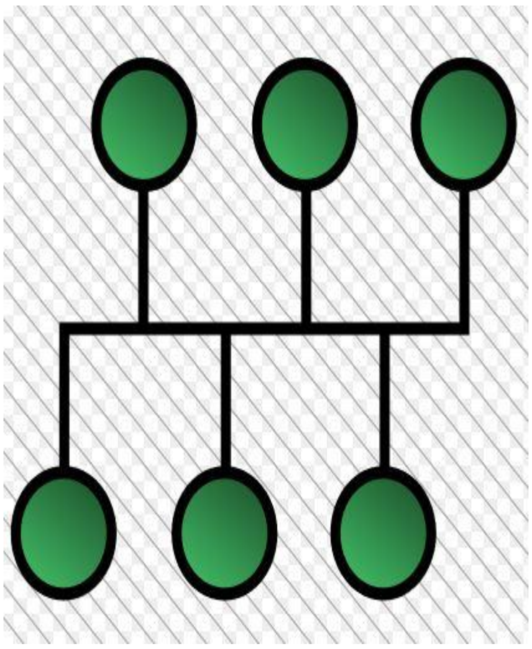


Figure 5 Bus Network Topology

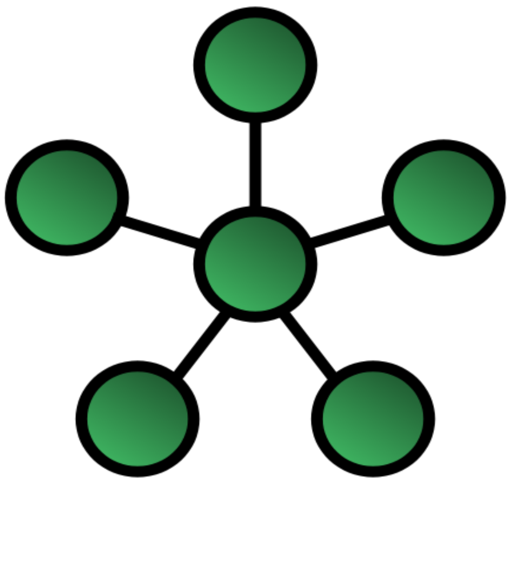
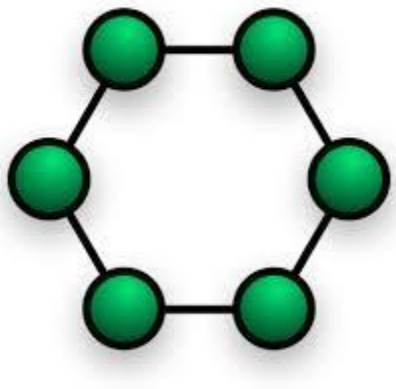


Figure 6 Star Network Topology

Figure 7 Mesh Network topology



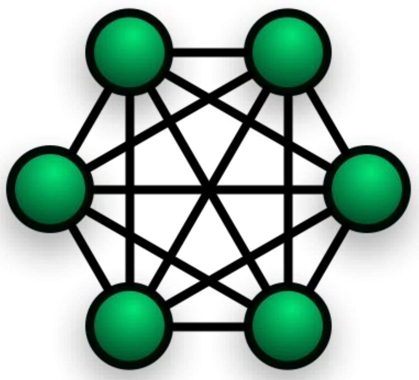


Figure 8 **Ring Network Topology**

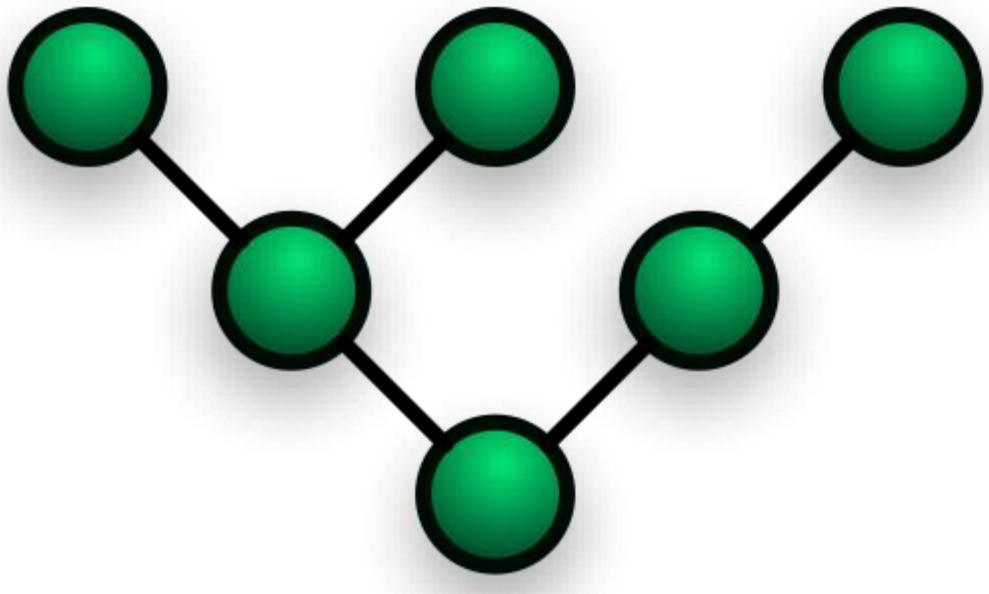


Figure 9 **Tree Network Topology**

1. Torus topology in which all nodes connected with each other like mesh topology but in more than one dimension like 2-D in fig 3.6 and 3-D in fig 3.7.

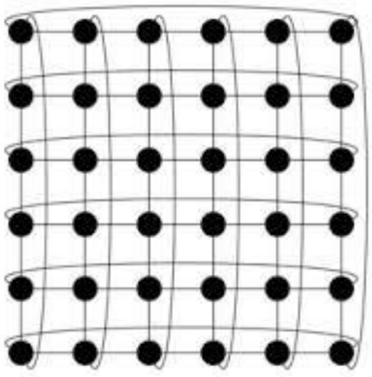


Figure 10

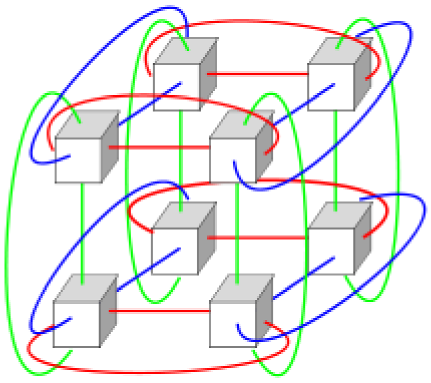


Figure 11

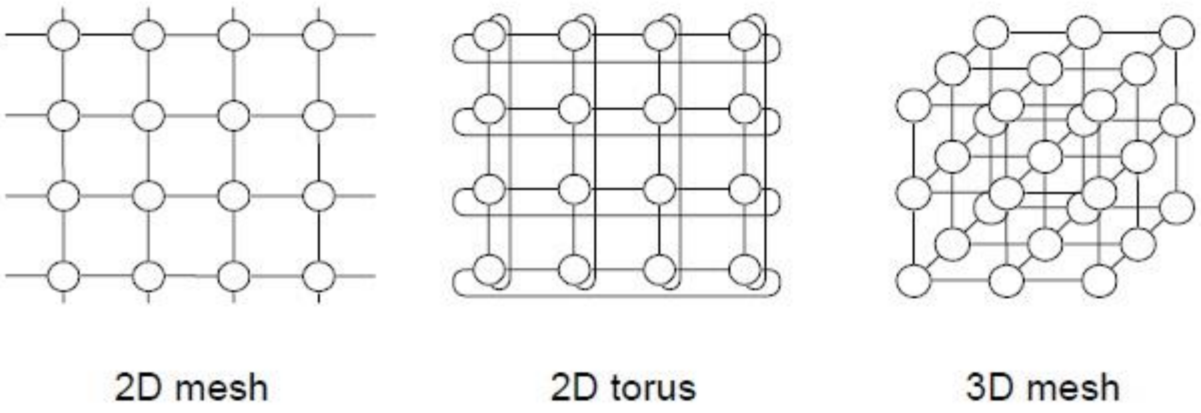


Figure 12 Different Network Connection topologies

3.3 Routing algorithms

Routing is the main work done by a network in which the router is responsible to manage the traffic in the network and find the best route for the data to go on.

The most important criterion for routing is *where* and *when* the routing function is determined.

The router can perform its role in the network by getting some information about the network status in order to make the decisions for choosing the best path for the data to transfer between the source and the destination.

There are many routing algorithms that the router follows to collect its information about the network or to choose the perfect path for the data between the two nodes:

1. Deterministic routing algorithm in which it generates the same single routing path for given pair of source and destination addresses which is the shortest one.
2. Oblivious routing algorithm in which it don't take into consideration any other information except the addresses, equally to deterministic routing. The routing decision is *oblivious to the status* of network traffic. Any deterministic routing is oblivious, but oblivious routing is not necessarily deterministic.

3. Adaptive routing algorithm in which it uses information about network traffic and channel status to avoid congested or faulty regions of the network. Adaptive routing do two main tasks, firstly the routing function as it delivers to a set of output channels if exists. Secondly the output selection function as it selects one of free output channels if exists using local status information.

Routing algorithms are implemented using look up tables as the source nodes and the router have routing tables which includes all information about the network like the number of nodes in the network and the status of paths. The router uses these tables to maintain its algorithm [3][1].

3.4 Core network interface

CNI or core network interface is a digital circuit placed between the network and the IP cores which are the source of the data.

The function of the network is to transfer the data from core to another core but the network doesn't deal with the data, it deals with small versions of these data.

The core network interface split the data into packets before entering the data to the network to realize packet switching concept.

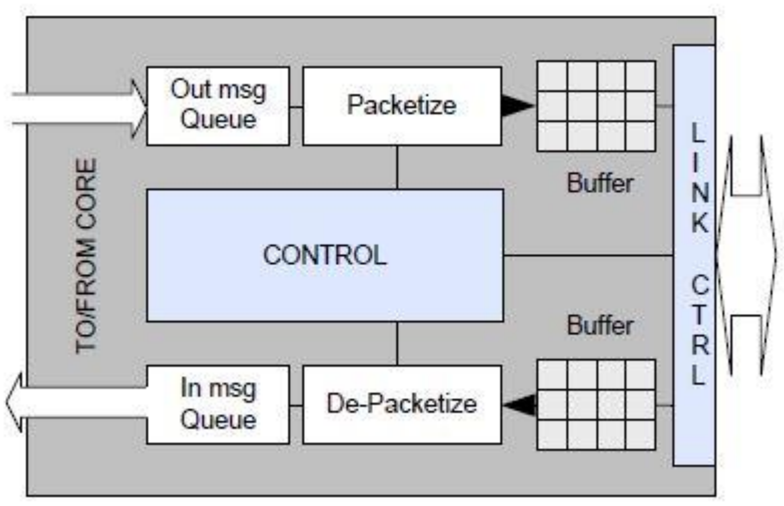


Figure 13 Shows the implementation of core interface network and its components.

The CNI architecture assumed in this research provides simple traffic translation to enable inter-core communication across the network-on-chip. For more complex systems, other functionality may be added to this base model as necessary. Other researchers present a thorough examination of additional functionality that may be added [5].

4 Router Architecture

Router is the block that performs routing process and manages the traffic in the network. By considering the router as a closed box, it has inputs and outputs. The inputs are called input channels and outputs are called output channels in which each input or output channel represent a node in the network and the router connect between these nodes and transfer data between them.

The router consists of two main blocks which responsible of do two main functions:

1. Data-path block.
2. Control unit block.

The data path block is the block that responsible of transfer the data from input channels to output channels while the control unit block is the block that controls the routing process in which it is makes the decisions on the path and knows when the data transfer and when the router is stall.

The data path block consists of input and output buffers and switches. Control unit block consists of arbiters, switch allocators and virtual channel allocators. All these parts will be discussed later in detail in chapter 5.

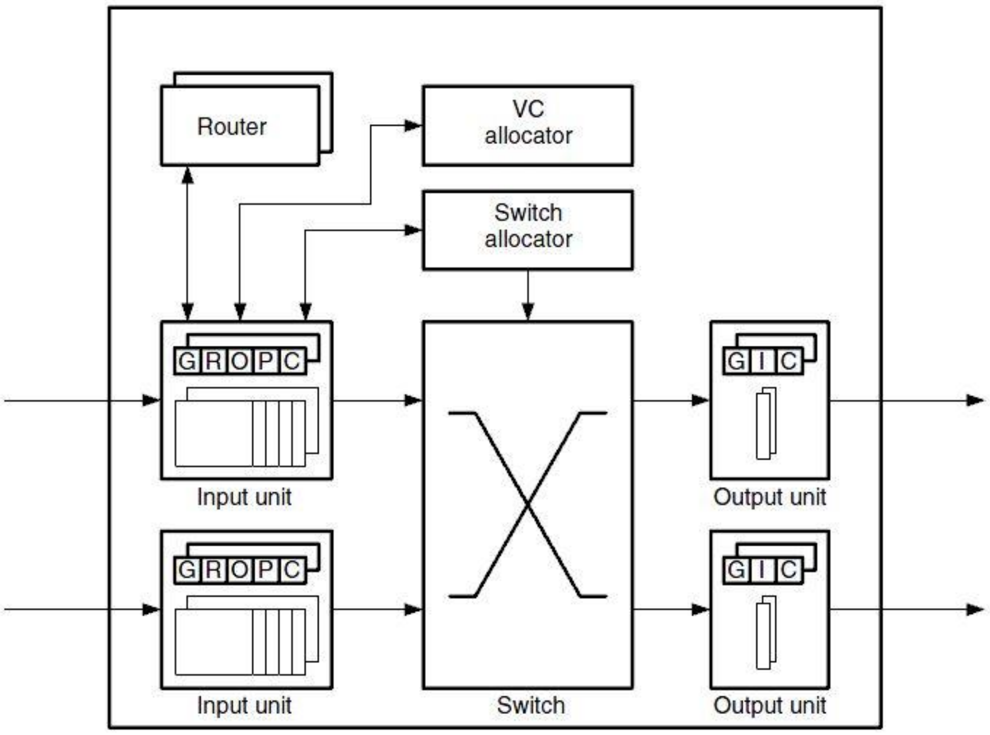


Figure 14 Router Data path

The router deals with the data at different levels in which we have:

1. **Data** or **Information**, this is the original data that the router wants to transfer between two nodes.
2. **Packets**, this is the data after splitting in which the network doesn't deal with the original data but the data is splitting into small versions of it called packets. The data was splitting by the network interface core.
3. **Flits**, this is small versions of packets in which the router deals with the data at flit level when it works pipelined.
4. **Credits**, this is the flits that move from output channels to input channels to tell the control blocks that the data received and the output buffer is allocated by these data. In other way, we can describe the credit that the number which describes the number of occupied locations in the buffer.
5. For example, the output buffer has 4 locations in which each location can carry one flit. If the output buffer has 3 flits so the credit field at the output will have 3.

The router consists of some components in which each component do its role in the routing process as follows:

- **Buffers** are the blocks that responsible of storing the data in it. This process is very important as the router store the data at its input until the data is received correctly at the output and with no errors to guaranteed accurate transmitting and receiving. Also the importance of storing the data at the input appears when all the paths and routes in the router are allocated so the data must be maintained in the input buffer until a path become free to avoid router congestion. Storing the data at the output buffer is very important as the data must be stalled until the next node or the next router in the network become free to receipt it.

- **The switch** is the block that responsible of connecting the input and output channels of the router. It is considered to be the heart of the router. There is much architecture that can implement the switch with them such as a single bus connects the input and output buffers or can take cross-bar architecture. Architectures of switches will be discussed in chapter 5 in detail.
- **Switch allocator** is the block which controls the switch and tell it which input will be connected to which output.

- **The physical channel** is the link or the wire that carry the data. Now, in the new generation of digital systems the data is needed to reach more than one destination and the routers becomes very complex so designers split the physical channel to group of channels which called virtual channels. **Each virtual channel** can carry different data or the same data which transfer to different destinations. With the use of virtual channels concept, the router can do its function by following pipelining concept.
- **Virtual channel allocator** is the block that allocates the virtual channels to the flits. It does its task to complete the path between input and output.

4.1 Router Pipelining

Pipelining is the exploitation of all resources in all times by making all the re-sources work at the same time.

When all the blocks in the router such as buffer, switches and allocators work in the same times and in all cycles, we say that this router is router-based pipelined.

The great advantage of following the pipelined concept is managing the time and overcoming the delay.

For example, a router has 4 cycles pipelining to get the flit from input to output in which:

- First cycle is route computation, in this cycle the router do its calculations to know the destination of the data based on the information that it takes from the head flit. The head flit is the first flit in the packet.
- Second cycle is virtual channel allocation, in this cycle the allocator allocates a virtual channel for the flit.
- Third cycle is switch allocation, in this cycle the allocator allocates the switch for the flit.
- Fourth cycle is switch traversal, in this cycle the process of transferring the flit is done and the flit reaches the output buffer waiting for entering the next node in the network.

If the data consists of 4 flits so in case of non-pipelined router each flit will take 4 cycles to reach to the output port so the 4 flits will take 16 cycles to reach their destination. But in case of pipelined-router when a flit is in a stage from four stages, another flit will be in another stage at the same time so the 4 flits will take 7 cycles to reach their destination as shown in figure.

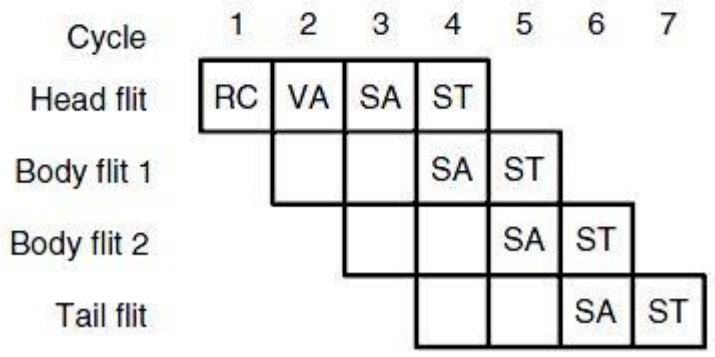


Figure 15

Note that this is ideal case in which this description of the stages of the pipelined router hasn't take in consideration the happen of the stalls.

Note that the head flit only will make route computation and virtual channel allocation because these two stages must be made at packet level and there is no need for all flits in the same packet to enter these stages.

Now, in modern routers the designers optimize the pipeline process to make the system more speed by many methods such as speculation method which is a method depend on a technique that predict the next step with high success percentage and this done by using either hardware or software. Now, the design will be more complex but more optimized into the best.

Also designers can play in hardware and make it smarter to get an advantage such as making virtual channel allocation and switch allocation in the same cycle.

Stalls means something happen makes the router stop in a cycle or more for a certain flit. There are packet stall and flit stall. When the router fails in virtual channel allocation so this is called packet stall and when it fails in switch allocation so it is called flit stall.

Packet stall occurs due to virtual channel is busy. For example, the head flit of one packet is arrived before tail flit of the previous packet so the virtual channel is allocated to the first packet while the second packet is stalled. Also, routing cannot be completed due to some errors such as the router fails to allocate a virtual channel so it repeats the allocation in the next cycle.

Flit stall occurs when there is a fail occurs in switch allocation so the router need to repeat the allocation in the next cycle. Also, if the output buffer is full so there is no credit available so the flit is unable to do switch allocation until the output buffer free a location. Also, flit stall occur when input buffer is empty so there is no flits available to do switch allocation.

5 Theory of Allocation

5.1 Overview

Concept of allocation rose as many NoC's are using packet switching technique to make the best use of network limited resources and to distribute these resources fairly to requesting agents, unlike circuit switching where each resource is dedicated to a specific agent.

The resources to be managed in a NoC are the output ports as multiple packets may compete to request a specific port and to share a specific route, this conflict will be settled by a *Channel allocator* (or a *Virtual channel allocator* for more advanced allocation). Other possible resource is the transmission bus (or crossbar) that is limited by a small size to reduce total area on chip so it has to be managed to multiple agents, a *switch allocator* will be used to distribute this resource to multiple packets traversal on a single bus.

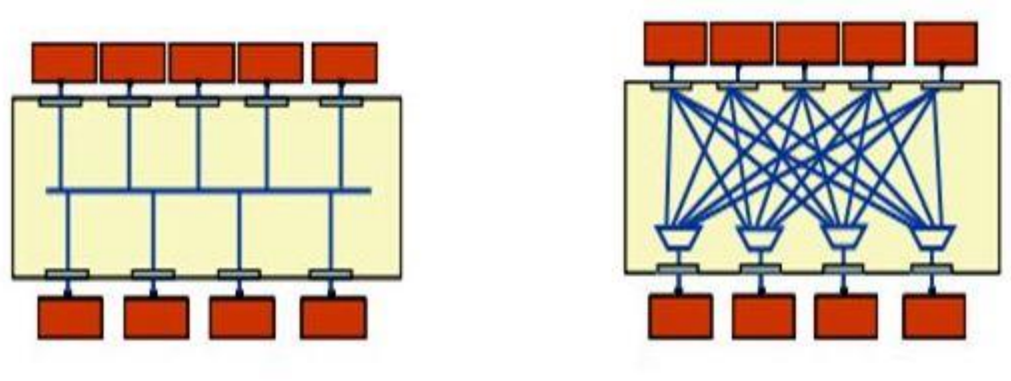


Figure 16 (A) Shared-bus between agents (B) Shared output ports between agents

5.2 Arbiters

The control paths of routers are largely composed of allocators which depend mainly on arbiters to distribute a single resource to a multiple agents.

The process of arbiters is to generate grants *G* to a set of requests *R* under two conditions:

- Grants only generated to agents issued their requests.
- Single grant to each request.

5.2.1 Fixed priority arbiter

This kind of arbiters distributes its resource on a linear fixed order, either ascending or descending order and it has the simplest design among all arbiters.
the idea of his arbiters is to grant access to the first agent issued its request and this can be implemented by inverting that issued request as carry for the other agents to block the from passing their request through . The basic implementing cell is shown in figure 6.3 and in figure 6.4 is a 4 agents fixed priority arbiter.

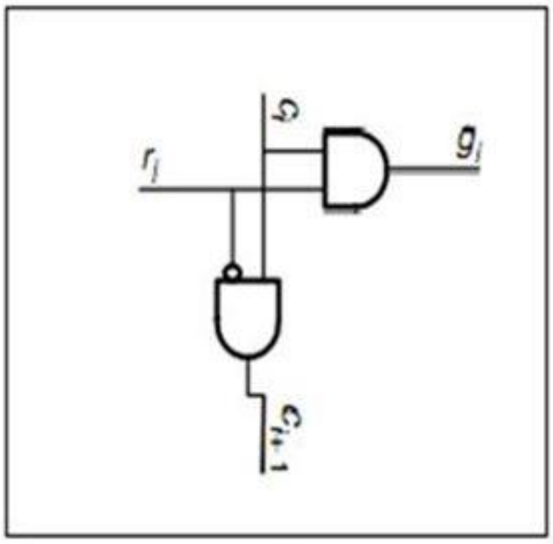


Figure 17 Basic cell of fixed priority arbiter

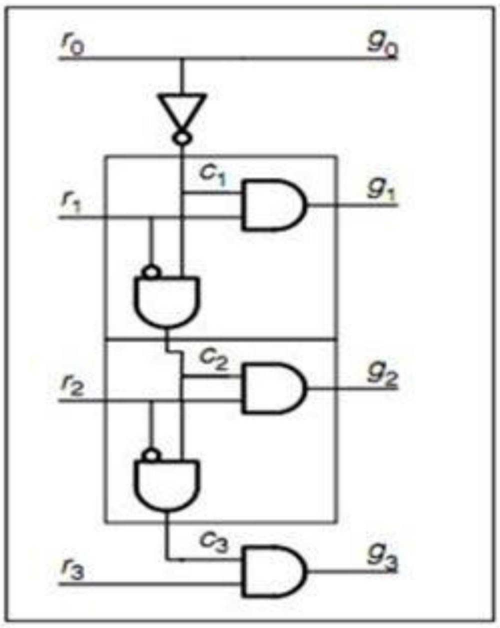
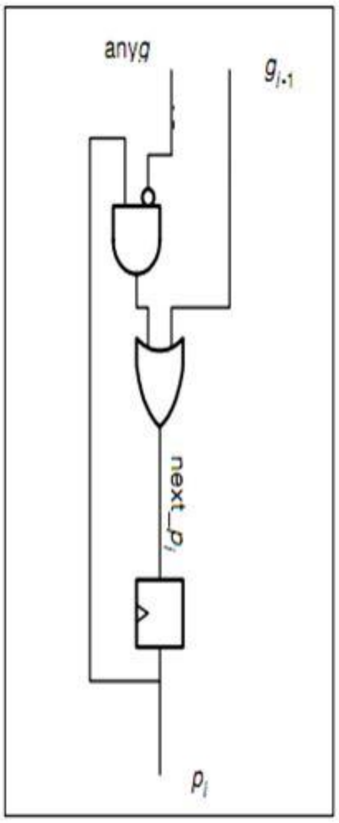


Figure 18 Implementation of 4 agent fixed-priority

5.2.2 Round Robin arbiter

It's a sort and an extension of priority arbiters as fixed priority arbiters, priority is granted to a specific agent by setting a signal p, when the arbitration is done this agent becomes the lowest priority and the next agent becomes the highest priority.

But some applications require unfair distribution of grants among agents so a weighted Round Robin is a good solution for such applications. So a weighted agent grants the resource or a time equal is weight divided by the total weights of the requests.



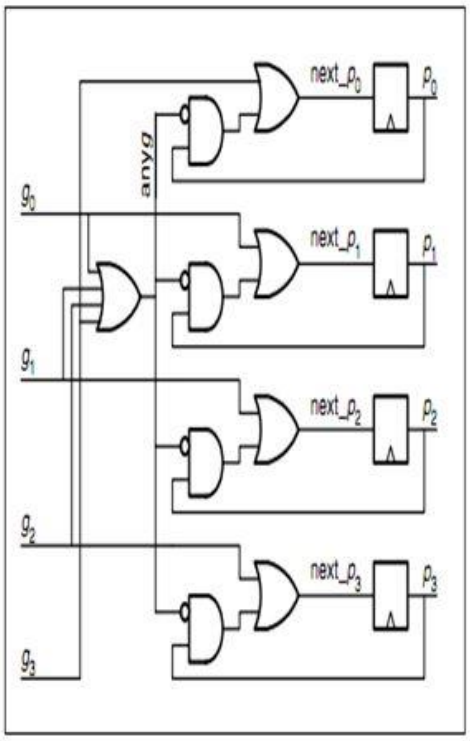


Figure 19 (A) Round-robin arbiter (B) 4-agents round-robin

5.2.3 Matrix arbiter

This kind of arbiters implements a least recently served order of priority, which means that for a pair of requests *A,B* there is a state bit *I*A,B that tells whether *A* is higher in priority than *B* or not. If one request received its grants it clears all its state bits to become lowest in priority.

That state bit is stored in a register, and for every pair of inputs there is a register to store their state bit(s) that gives the Matrix scheme figure 6.7.

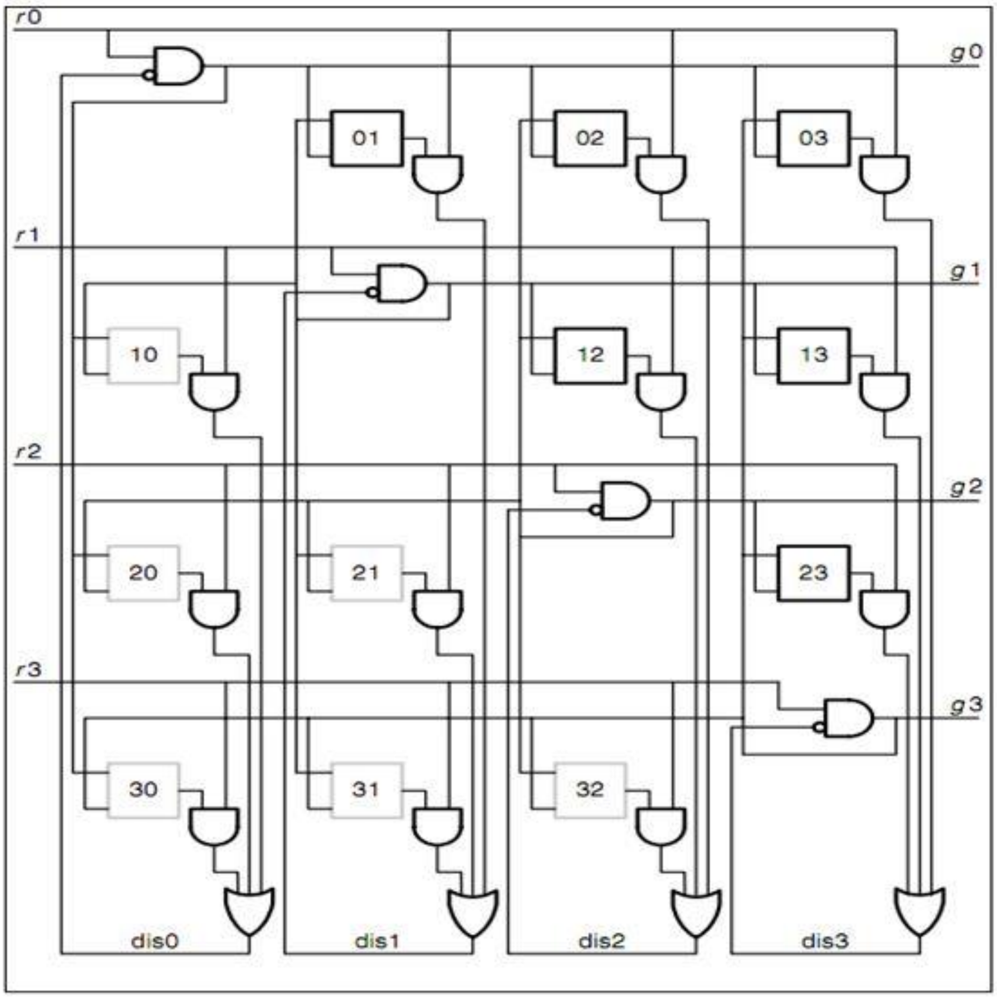


Figure 20 Matrix arbiter

5.2.4 6.2.4 Queuing arbiter

As it may appear of the name, that kind of arbiter relays on serving the first issued request "first come first serve" or "FIFO" policy.

When agent issues a request it also sets a counter and a time indicator and a set of comparators that help to determine which agents issued its request first.

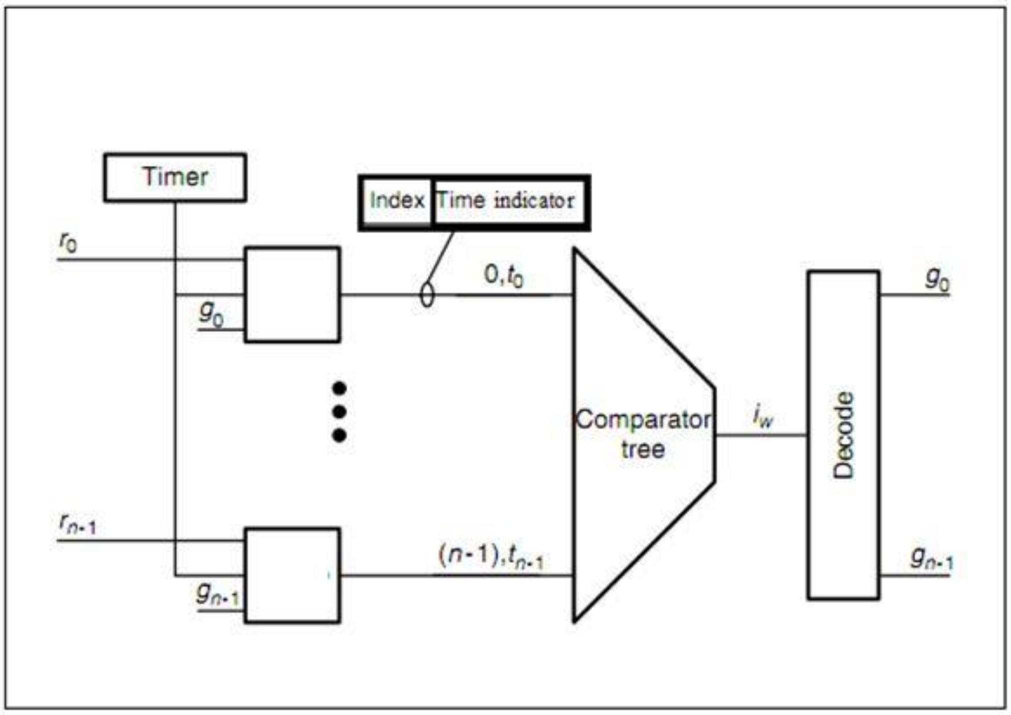


Figure 21 Queuing arbiter

5.2.5 Tree arbiter

To avoid large sizes of arbiters in large applications tree arbiters divides the requests issued by agents into groups. The arbitration process is done over two or more stages
first selecting between groups and selecting agent among all agents in each group.

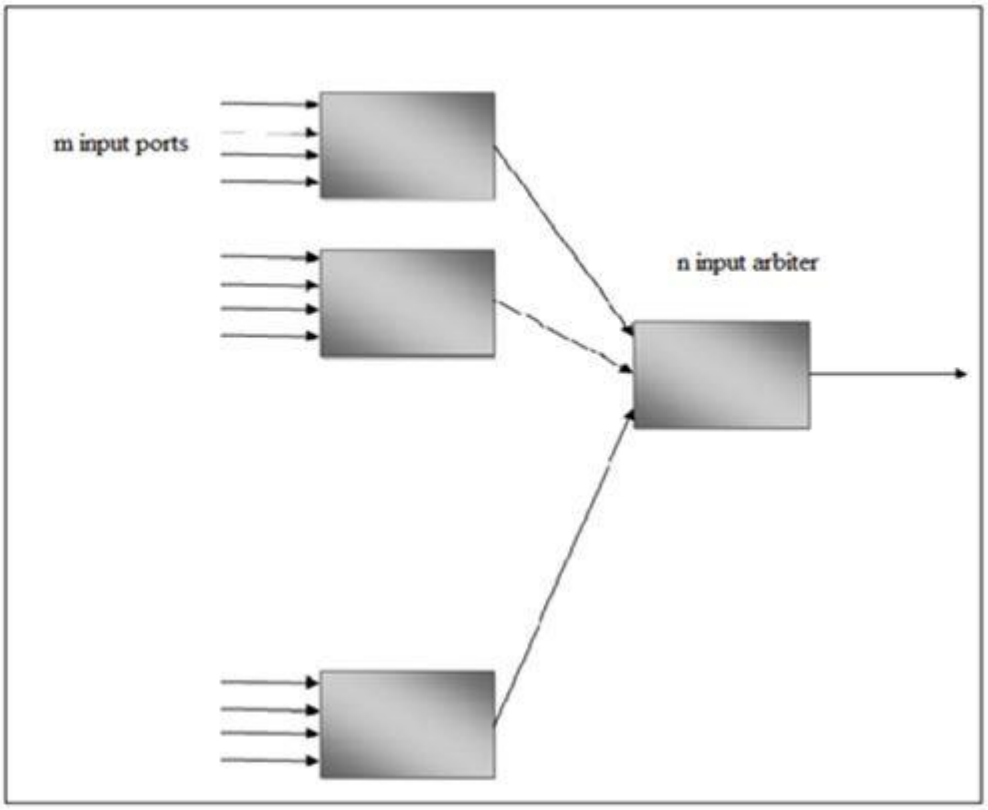


Figure 22 Tree arbiter

5.2.6 6.2.6 Dynamic-Priority arbiter

The kind of arbiters prioritizes grants to agents that satisfy certain criteria. That can be used in NoC which has more dynamic algorithms like adaptive routing these arbiters can give high priority for low traffic ports for example.

5.3 6.3 Allocation

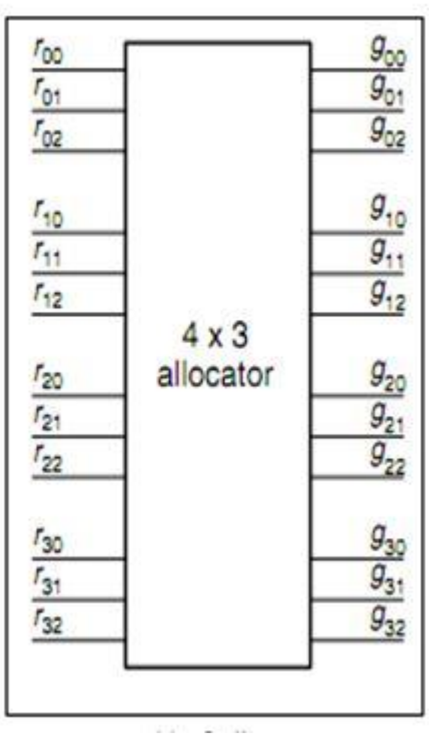


Figure23 4X3 allocator

The purpose of allocation is to match a set of resources to a set of agents which may request one or more resource in a way that each at most one destined output port is assigned to one requesting input port.

The process of allocation is subjected to two main constraints:

- Resources are only granted to agents requests, non-destined resources aren't granted.
- Each single agent grants at most a single resource and vice versa.

Allocators can mathematically modelled by its number of requests and agents for example a*I X K* allocator means that we have a set of *I* agents are requesting a group of *K* resources .

The allocator is to generate a vector of grants equals the number of resources and provide this grant to each winning agent.

5.3.1 Separable Allocation

In a separable allocator we perform two arbitrations one arbitration is done at input ports and the other is done at the output ports.

In input first allocator arbitration is done firstly at the input ports selecting a single request at each port then arbitration is done at the output ports to select a request for each output ports.

That kind of arbitration may not get maximum matching as a winning agent may lock the only request from other agents.

A separable allocator can also be realized by performing the output arbitration first and then the input arbitration where the first rank of three 4-input arbiters selects the winning request for each output port. Only one of the resulting signals will be asserted for each output port. The input arbiters then take as input, pick the winning request for each input, and output this result on grant signals ensuring that at most one of grants s asserted for each input.

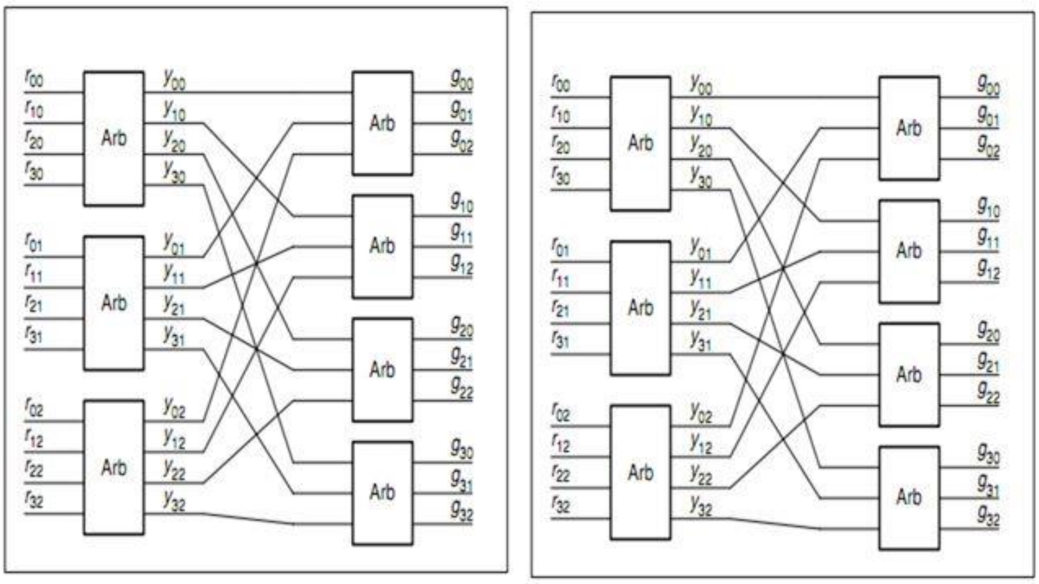
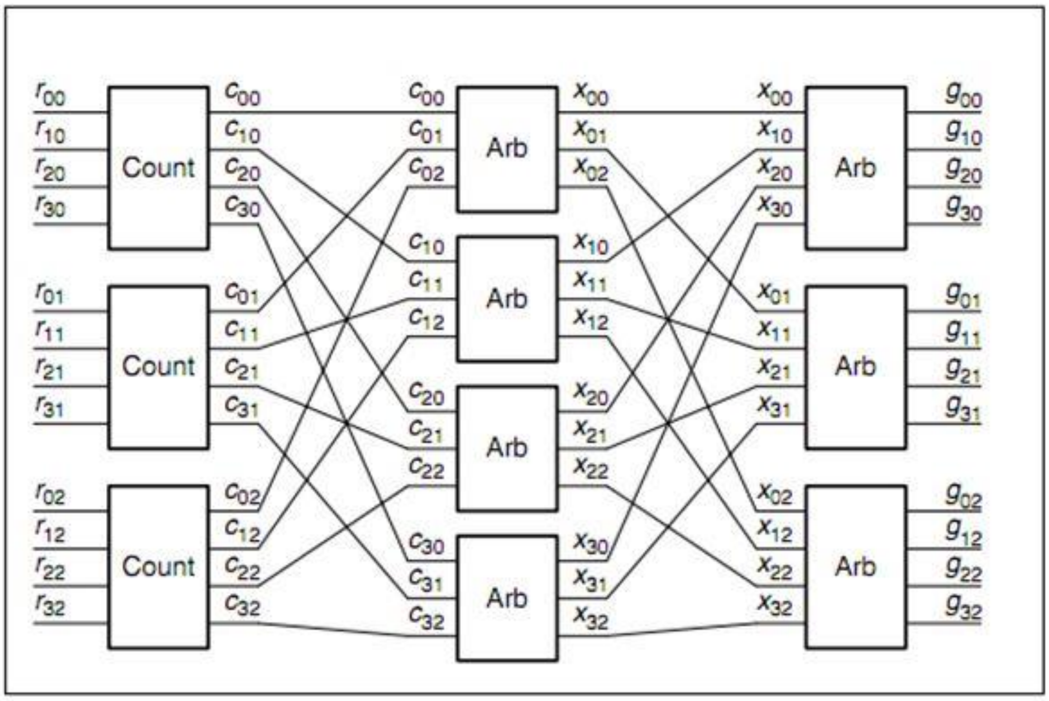

r_{00} r_{10} r_{20} r_{30}
r_{01} r_{11} r_{21} r_{31}
r_{02} r_{12} r_{22} r_{32}
Arb
y_{00} y_{10} y_{20} y_{30}
y_{01} y_{11} y_{21} y_{31}
y_{02} y_{12} y_{22} y_{32}
g_{00} g_{01} g_{02}
g_{10} g_{11} g_{12}
g_{20} g_{21} g_{22}
g_{30} g_{31} g_{32}

Figure 24 (B) Output first separable allocator (A) Input first separable allocator

5.3.2 Lonely output Allocation

To solve the problem of locking input with one request in separable input first allocators, lonely output allocator was introduced to overcome this problem by adding a stage before the input arbiters that counts the number of requests for each output. The input arbiters then give priority to requests for outputs that have low request counts the lonely outputs. This reduces the number of conflicts at the output stage, resulting in a better matching.



r_{00} r_{10} r_{20} r_{30}
Count
c_{00} c_{10} c_{20} c_{30}
r_{01} r_{11} r_{21} r_{31}
Count
c_{01} c_{11} c_{21} c_{31}
r_{02} r_{12} r_{22} r_{32}
Count
c_{02} c_{12} c_{22} c_{32}
Arb
x_{00} x_{01} x_{02} … x_{32}
g_{00} g_{10} g_{20} g_{30}
g_{01} g_{11} g_{21} g_{31}
g_{02} g_{12} g_{22} g_{32}

Figure 25 Lonely output allocator

5.3.3 Parallel iterative Allocation

It's a version of separable allocators but it depends on a series of probabilistic separable arbitration.

Randomizing the arbitrations acts to probabilistically stagger the input arbiters, this makes it unlikely that they will all pick the same input, and it also can eliminate that input is repeatedly locked out due to deterministic priority adjustments.

5.3.4 ISLIP Allocation

ISLIP is another version of separable allocators. it uses round-robin arbiters and changes priority when that arbiter generates a winning grant. That algorithm helped to reduce conflicts at the output stage. Priorities are updated so that a winning request has the lowest priority in the next round.

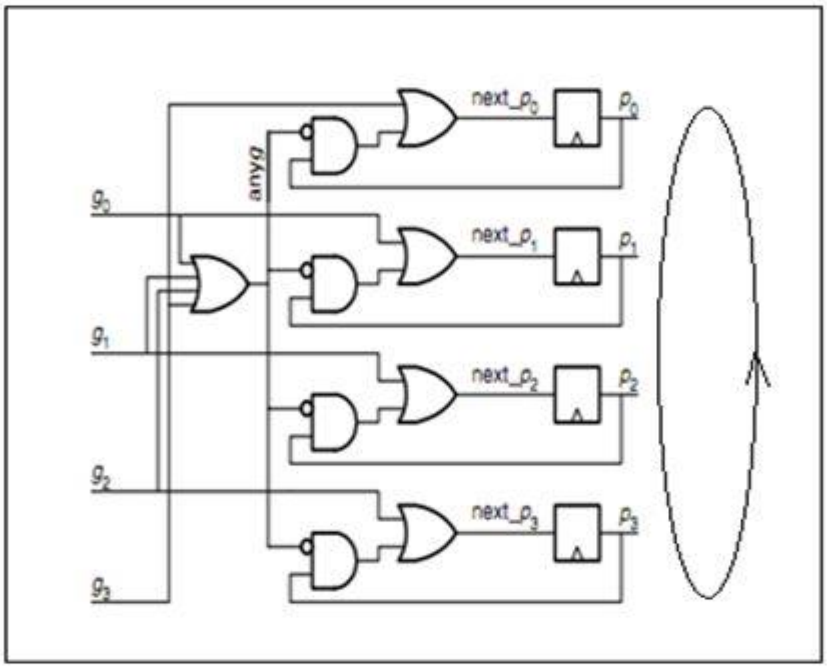


Figure 26 Rotating priority of ISLIP Arbiters

5.3.5 6.3.5 Wave front Allocation

Unlike the separable allocators this wave front allocation makes he both the input stage and output stage arbitration at same time.

The wave front allocator works by granting row and column tokens broken to a diagonal group of cells, in effect giving this group priority.

A cell with a row(column) token that is unable to use the token passes the token to the right (down),wrapping around at the end of the array. These tokens propagate in a wave front from the priority diagonal group, hence the name of the allocator. If a cell with a request receives both a row and a column token, either because it was part of the original priority group or because the tokens were passed around the array, it grants the re-quest and stops the propagation of tokens. To improve fairness, the diagonal group receiving tokens are rotated each cycle. However, this only ensures weak fairness.

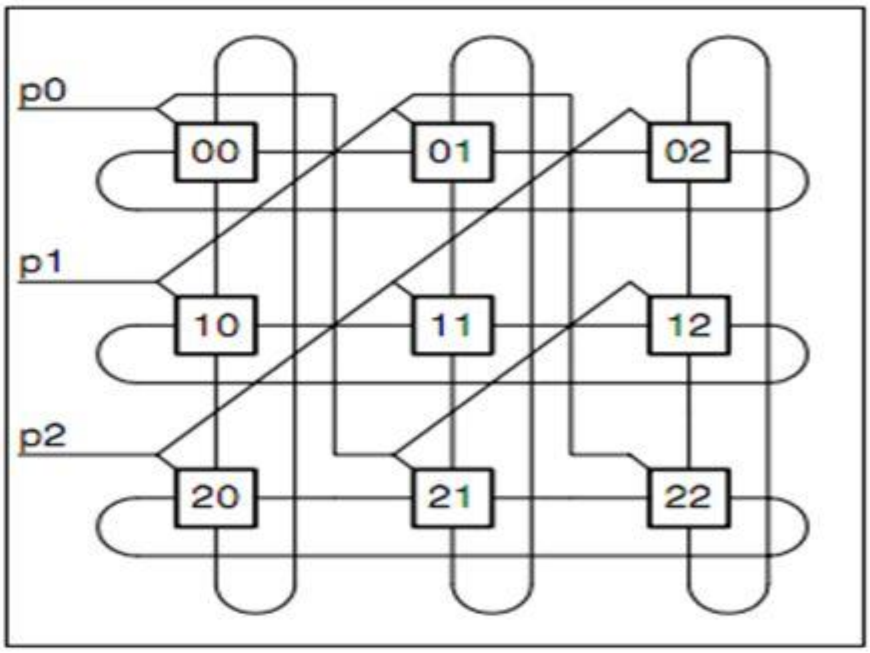


Figure 27 Wave-front allocator

5.3.6 Multi stage Allocation

Some applications require multiple stages of allocation. For example, in a router, we might want to grant requests to high-priority packets first and then allocate any remaining resources to lower-priority packets. In another application, we might want to grant requests for multicast traffic first and then allocate remaining ports to unicast traffic.

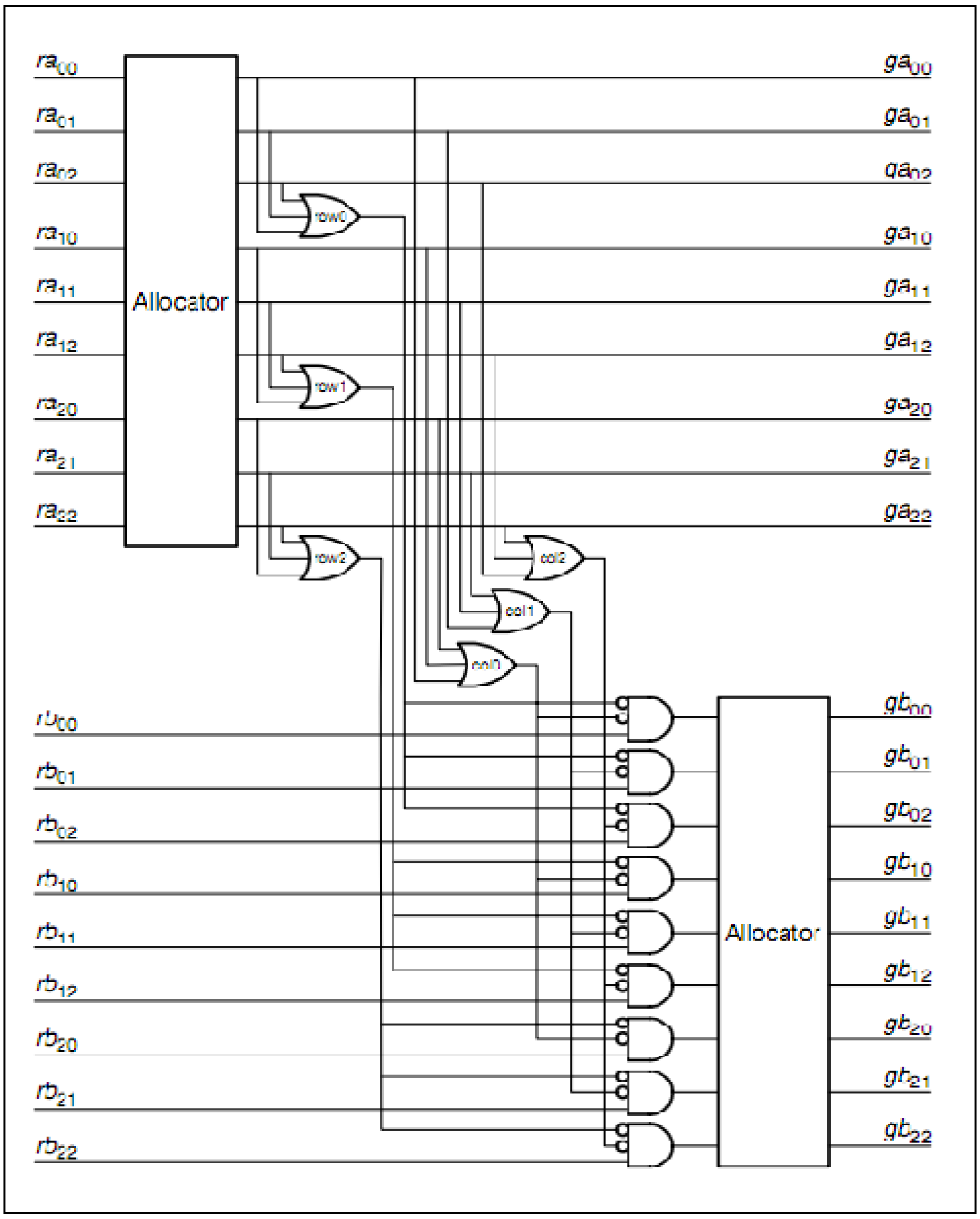


Figure 28 Multistage Allocator

6 Buffers & switches

6.1 Buffers

Input Buffers hold the flits waiting for virtual channels and the switch bandwidth also output buffers hold the flits to enter the next node.

The flow control protocol allocates space in these buffers to hold flits. The buffers provide space for arriving flits so that the incoming channel needn't be slowed when a packet is de-layed due to a pipeline stall or a contention for a virtual channel or physical channel.

There are many techniques to design the buffer and partitioning it as follows:

1. Using of central memory for all input ports in which all the ports hold their flits in the same memory as in Fig 6.1.1
2. Using of a memory per physical channel as in Fig 6.1.2.
3. Using of a memory per virtual channel as in Fig 6.1.3.

Note that the third technique is the best because it is the fastest one but it is complex so we choose our design depending on what we need and what we prefer in our circumstances and in our project.

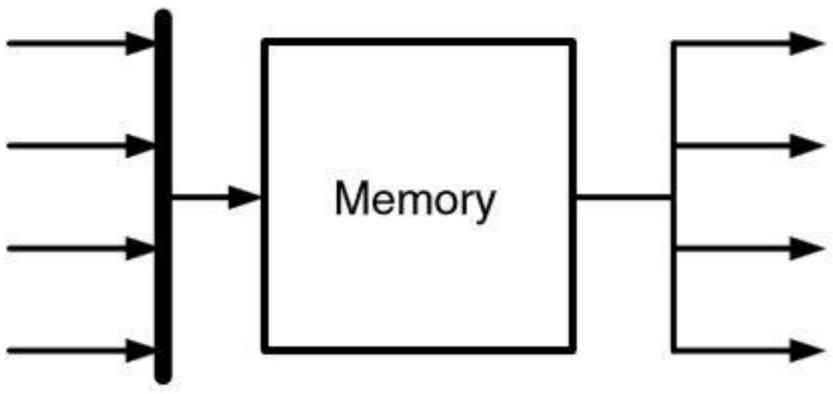


Figure 29 One Large Main memory for all ports

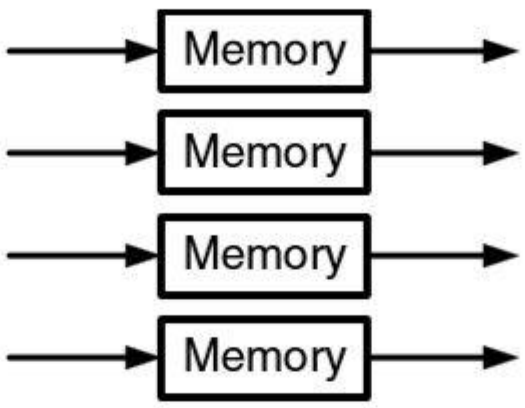


Figure 30 Single memory for each port

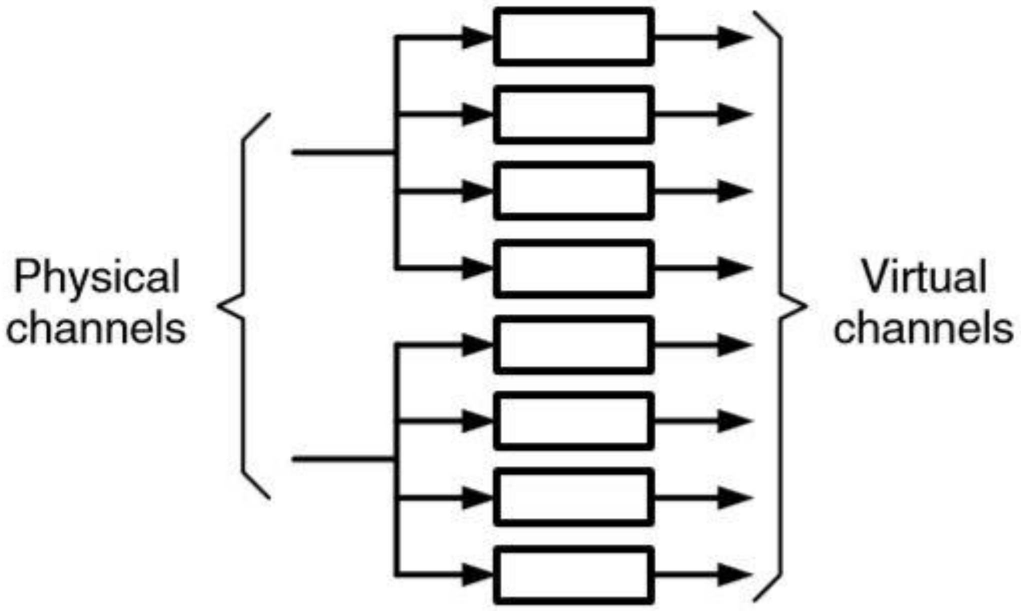


Figure 31 Virtual channel explanation

The lost ring in the design of buffers that how we manage the data inside it, how data enter and leave the buffer. There are two famous techniques in these issue:

1. Circular buffer technique.
2. Linked list technique.

These techniques are similar to C++ programming language in managing the data inside the memory in which the circular buffer technique is similar to mnanging the data inside an array in C++ and linked list is similar to managing the data inside dynamically allocated memory inC++ [1].

6.2 Switches

The switch is the main part of router. It is where that packets and flits are actually routed to their wanted output port. The most important design parameter with a switch is its "Speedup" -the ratio of the switch bandwidth to the minimum switch bandwidth needed to support full throughput on all i/ps & o/ps.

6.2.1 Bus Switches

When a flit time consists of more internal clock cycles (phit times) than the number of ports, a bus may be used as a switch, as illustrated in Figure 17.5. The timing of this type of bus switch is shown in Figure 17.6. Each input port of the switch accepts phits of a flit and accumulates them until it has at least P phits, where P is the number of input switch ports. Once an input port has P phits, it arbitrates for the bus. When it acquires the bus, it transmits the P phits, possibly an entire flit, broadside to any combination of output units. The receiving output units then DE serialize them and transmit them one at a time to the downstream logic.

The Timing diagram in Fig. 6.2.1 shows a 4 × 4 bus switch in which each flit consists of four phits. Flits *a,f,k*, and*p* arrive on the four input ports, one phit at a time, during the first flit time; flits *b, g, l*, and *q* arrive during the second flit time; and flits *c, h, m*, and *r* arrive during the third flit time. While the ith flits are arriving, the (i−1)th flits are transmitted broadside across the bus, one flit being transmitted each phit time, and the (i − 2)th flits are being serialized on the output ports.

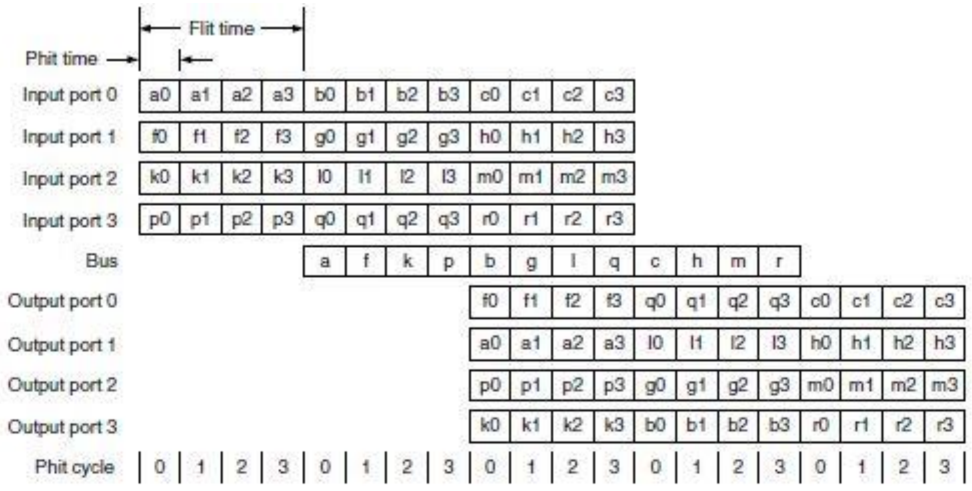

Flit time
Phit time
Input port 0: a0 a1 a2 a3 b0 b1 b2 b3 c0 c1 c2 c3
Input port 1: f0 f1 f2 f3 g0 g1 g2 g3 h0 h1 h2 h3
Input port 2: k0 k1 k2 k3 l0 l1 l2 l3 m0 m1 m2 m3
Input port 3: p0 p1 p2 p3 q0 q1 q2 q3 r0 r1 r2 r3
Bus: a f k p b g l q c h m r
Output port 0: f0 f1 f2 f3 q0 q1 q2 q3 c0 c1 c2 c3
Output port 1: a0 a1 a2 a3 l0 l1 l2 l3 h0 h1 h2 h3
Output port 2: p0 p1 p2 p3 g0 g1 g2 g3 m0 m1 m2 m3
Output port 3: k0 k1 k2 k3 b0 b1 b2 b3 r0 r1 r2 r3
Phit cycle: 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3

Figure 32 Bus Switch Timing diagram

6.2.2 Crossbar switches

n x m crossbar switch can connect from n inputs to m outputs.

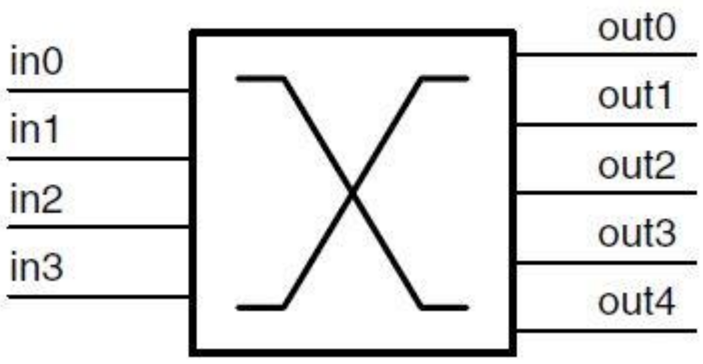


Figure 33 **Symbol for a 4 × 5 crossbar switch.**

Speed up: it is the ratio of provided bandwidth to required bandwidth.

We can provide speedup on the input of the crossbar, the output of the crossbar or on both sides.
This speedup can be provided in space (additional inputs) or time (higher bandwidth inputs).

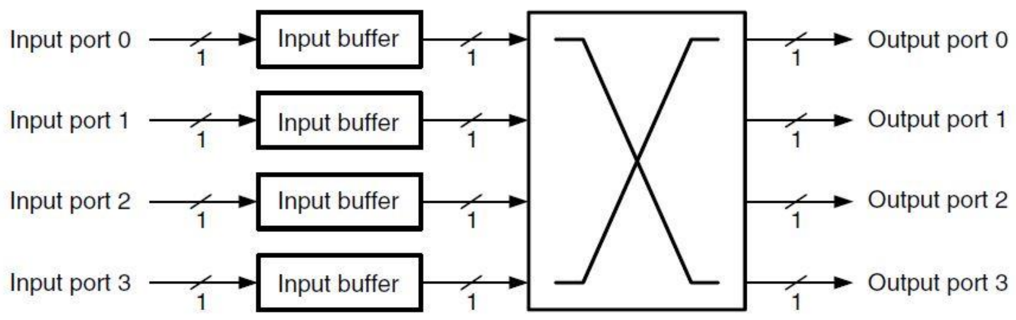


Figure 34 Original crossbar with no speed up

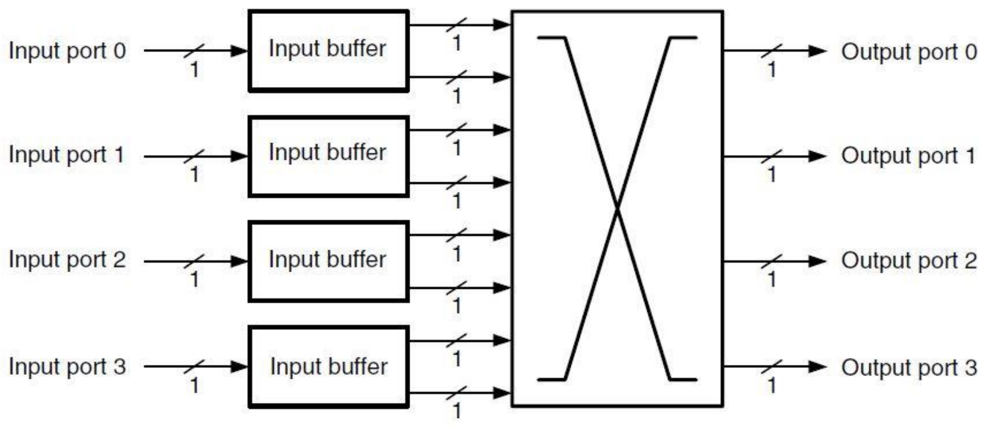


Figure 35 Input speedup

This switch has input speed up of two resulting in a simpler allocation problem.

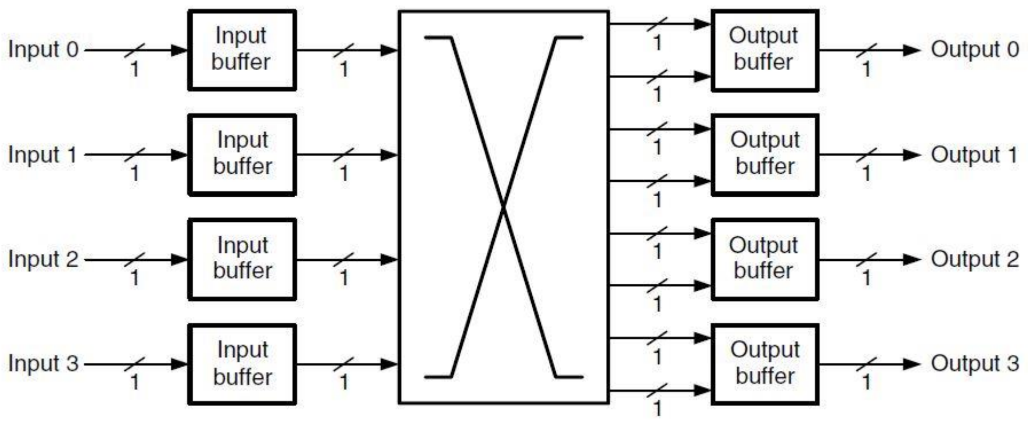


Figure 36 Output speed up

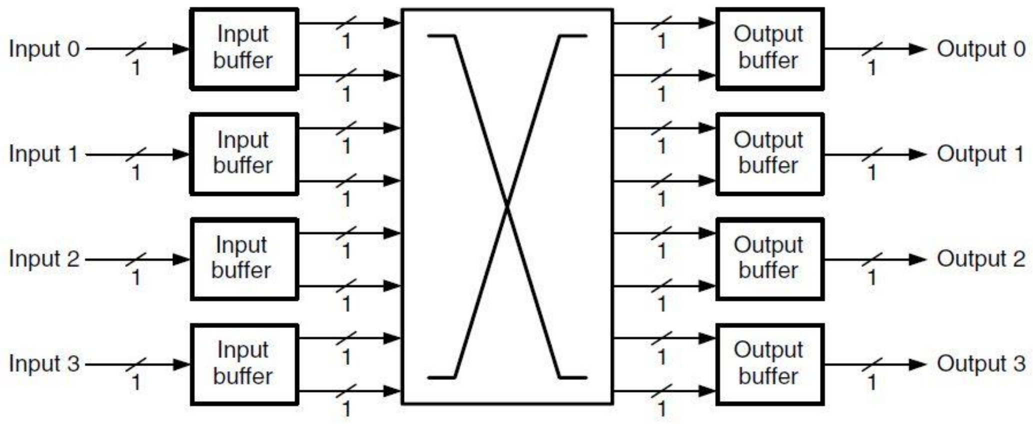


Figure 37 Switch speed up of two

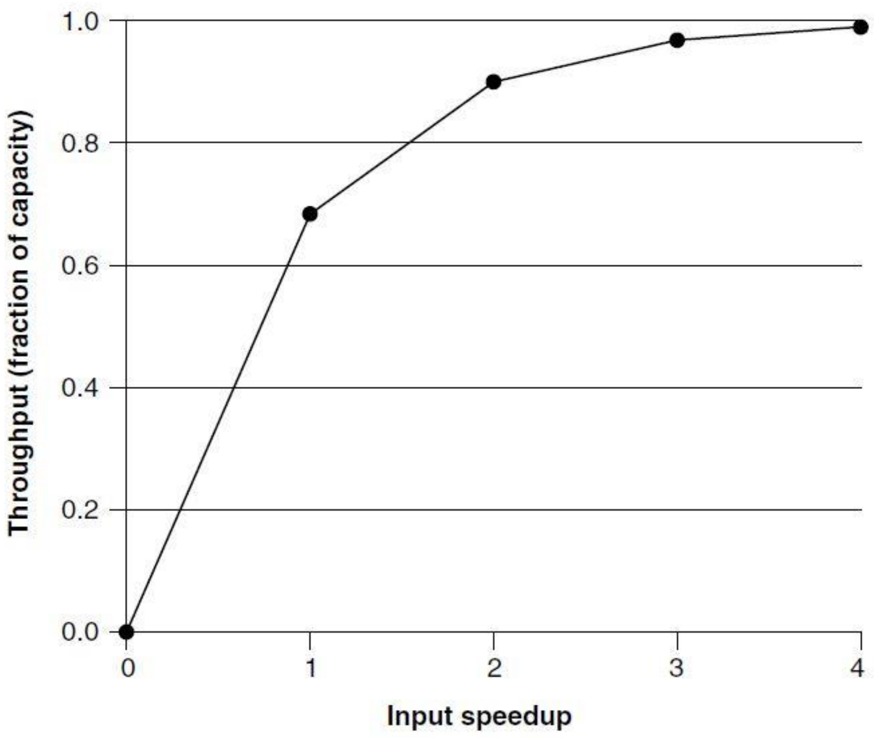


Figure 38

This graph shows the relation between throughput and the input speedup .

7 Chapter 7 chip-link router

7.1 First Simple Router

After explaining each part in the router designed for NoC purpose we started our first step by designing a simple router without buffers - just single reg at each port.

7.1.1 Allocator:

We needed just to check that we can achieve the function of routing so we used fixed priority arbiter in implementing our Allocator [1].

The code:

```
module arbiter(clk, thisPort, r0, r1, r2, r3, select, shift) ;
input clk ;                               // chip clock
input [1:0] thisPort ;                    // identifies this output port
input [3:0] r0,r1,r2,r3 ;                 // top four bits of each input phit
output [3:0] select ;                     // radial select to multiplexer
output shift ;                            // directs shifter to discard upper two bits
wire [3:0] grant, select, head, payload, match, request, hold ;
wire [2:0] pass ;
reg [3:0] last ;
wire avail ;
assign head = {r3[3:2]==3,r2[3:2]==3,r1[3:2]==3,r0[3:2]==3} ;
assign payload = {r3[3:2]==2,r2[3:2]==2,r1[3:2]==2,r0[3:2]==2} ;
assign match =
{r3[1:0]==thisPort,r2[1:0]==thisPort,r1[1:0]==thisPort,r0[1:0]==thisPort} ;
assign request = head & match ;
assign pass = {pass[1:0],avail} &~ request[2:0] ;
assign grant = request & {pass,avail} ;
assign hold = last & payload ;
assign select = grant | hold ;
assign avail = ~(|hold) ;
assign shift = |grant ;
always @(posedge clk) last = select ;
endmodule
```

Here you can see that we use a combinational circuit as there is no dependency on previous or current state of grants just only dependency on the input requests.

If you track the code you will find it implements the following circuit "see Fig. 33".

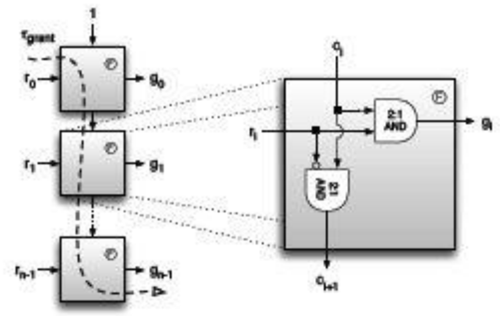


Figure 39 Fixed priority Arbiter

7.1.2 Rest of the Datapath:

Then we used this arbiter block with other registers and multiplexers to make the full simple router implementation.

<u>The code:</u>

```
// simple four-input four output router with dropping flow control
module simple_router(clk,i0,i1,i2,i3,o0,o1,o2,o3) ;
input clk ;                              // chip clock
input [17:0] i0,i1,i2,i3 ;               // input phits
output [17:0] o0,o1,o2,o3 ;              // output phits
reg [17:0] r0,r1,r2,r3 ;                 // outputs of input registers
reg [17:0] o0,o1,o2,o3 ;                 // output registers
wire [17:0] s0,s1,s2,s3 ;                // output of shifters
wire [17:0] m0,m1,m2,m3 ;                // output of multiplexers
wire [3:0] sel0, sel1, sel2, sel3 ;      // multiplexer control
wire shift0, shift1, shift2, shift3 ;    // shifter control
// the four allocators
alloc a0(clk, 2'b00, r0[17:14], r1[17:14], r2[17:14], r3[17:14], sel0, shift0) ;
alloc a1(clk, 2'b01, r0[17:14], r1[17:14], r2[17:14], r3[17:14], sel1, shift1) ;
alloc a2(clk, 2'b10, r0[17:14], r1[17:14], r2[17:14], r3[17:14], sel2, shift2) ;
alloc a3(clk, 2'b11, r0[17:14], r1[17:14], r2[17:14], r3[17:14], sel3, shift3) ;
// multiplexers
mux4_18 mx0(sel0, r0, r1, r2, r3, m0) ;
mux4_18 mx1(sel1, r0, r1, r2, r3, m1) ;
mux4_18 mx2(sel2, r0, r1, r2, r3, m2) ;
mux4_18 mx3(sel3, r0, r1, r2, r3, m3) ;
// shifters
shiftp sh0(shift0, m0, s0) ;
shiftp sh1(shift1, m1, s1) ;
shiftp sh2(shift2, m2, s2) ;
shiftp sh3(shift3, m3, s3) ;
// flip flops
always @(posedge clk)
begin
r0=i0 ; r1=i1 ; r2=i2 ; r3=i3 ;
o0=s0 ; o1=s1 ; o2=s2 ; o3=s3 ;
end
endmodule
```

Then we simulated it and run net list drawing to give the following RTL.

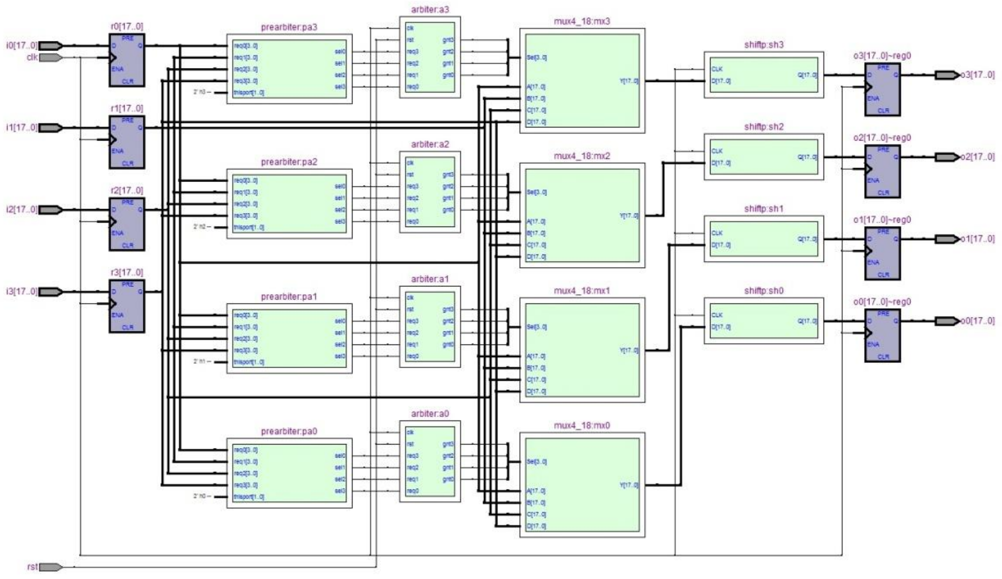


Figure 40 Netlist simple router

7.2 Second Full Router

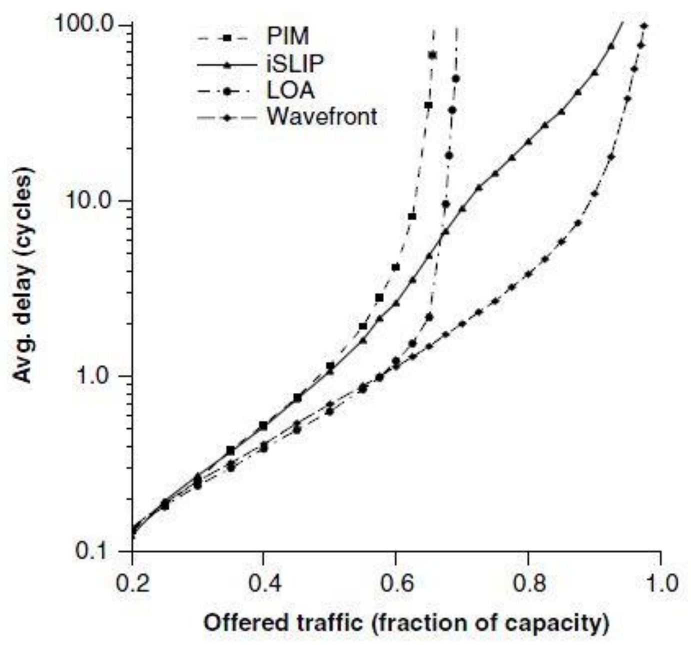
Our Full router passed by some steps to be in its final form which consists of fifo input buffers, Routing table "address translation block", Allocator, Cross bar and fifo output buffers.

we can simplify these steps for illustration as following.

7.2.1 Allocator:

As we know the allocator is one of the most important blocks that the whole routing and resources allocation depends on - good allocation technique give the most optimum technique.

We choose the Islip allocation technique which have the advantage over other allocation techniques as we illustrated before and as this figure shows fig 7.3 .



Performance of single iteration allocators. Each curve shows the average delay versus offered traffic for an 8×8 crossbar under uniform traffic.

Figure 41 Islip graph versus other allocations

The Islip allocation technique uses 3 way hand shake for granting a given resource to a given agent which may introduce delay to our router so we needed to make it more simple so all what we need from Islip technique is that it achieves the fair round-robin plus granting that most of the output ports which have conflict are granted to one input port only, which we achieved by the following allocator code with just only one cycle not 3 cycles as the iSLP allocation technique.

<u>Arbiter The code:</u>

```
/*
**************************************************
* Round-robin arbiter with variable Priority vector
**************************************************
*/
`timescale 1ns/1ps
module arbiter(clk,
rst,
req,
grant,
anyGrant);
parameter N = 4;
parameter S = 2; // ceil of log_2 of N - put manually
parameter CHOISE = 0;  // 0 blind round-robin and 1 true round robin
// I/O interface
input       clk;
input       rst;
input  [N-1:0]  req;
output [N-1:0]  grant;
output          anyGrant;
// internal pointers
reg [N-1:0] Priority; // one-hot Priority vector
//reg [N-1:0] trig;
// Outputs of combinational logic - real wires - declared as regs for use in a
always block
// Better to change to wires and use generate statements in the future
 reg [N-1:0]  g[S:0]; // S levels of Priority generate
reg [N-1:0]  p[S-1:0]; // S-1 levels of Priority propagate
//reg [N-1:0]  gf[s:0]; //
// internal synonym wires of true outputs anyGrant and grant
wire anyGnt;
wire [N-1:0] gnt;
//wire [N-1:0] gntf;
assign anyGrant = anyGnt;
assign grant = gnt;
//assign trig = req & clk;
//assign gntf = gnt;
////////////////////////////////////////////////
// Parallel prefix arbitration phase
////////////////////////////////////////////////
integer i,j;
// arbitration phase
//always@(posedge clk) begin
always@(req  or Priority)
begin
// transfer request vector to the first propagate positions
p[0] = {~req[N-2:0], ~req[N-1]};
// transfer Priority vector to the first generate positions
g[0] = Priority;
// first log_2n - 1 prefix levels
for (i=1; i < S; i = i + 1) begin
for (j = 0; j < N ; j = j + 1) begin
if (j-2**(i-1) < 0) begin
g[i][j] = g[i-1][j] | (p[i-1][j] & g[i-1][N+j-2**(i-1)]);
p[i][j] = p[i-1][j] & p[i-1][N+j-2**(i-1)];
end else begin
g[i][j] = g[i-1][j] | (p[i-1][j] & g[i-1][j-2**(i-1)]);
p[i][j] = p[i-1][j] & p[i-1][j-2**(i-1)];
end
end
end
// last prefix level
for (j = 0; j < N; j = j + 1) begin
if (j-2**(S-1) < 0)
g[S][j] = g[S-1][j] | (p[S-1][j] & g[S-1][N+j-2**(S-1)]);
else
g[S][j] = g[S-1][j] | (p[S-1][j] & g[S-1][j-2**(S-1)]);
end
end
//end
// any grant generation at last prefix level
assign anyGnt = ~(p[S-1][N-1] & p[S-1][N/2-1]);
```

```
// output stage logic
assign gnt = req & g[S];
/////////////////////////////////////////////////
// Pointer update logic
// ------------------------
// Version 1 - blind round robin CHOISE = 0
// Priority visits each input in a circural manner irrespective the granted
output
// ------------------------
// Version 2 - true round robin CHOISE = 1
// Priority moves next to the granted output
// ------------------------
// Priority moves only when a grant was given, i.e., at least one active request
/////////////////////////////////////////////////
always@(posedge clk)
begin
if (rst == 1'b1) begin
Priority <= 1;
end else begin
// update pointers only if at leas one match exists
if (anyGnt == 1'b1) begin
if (CHOISE == 0) begin // blind circular round robin
// shift left one-hot Priority vector
Priority[N-1:1] <= Priority[N-2:0];
Priority[0] <= Priority[N-1];
end else begin // true round robin
// shift left one-hot grant vector
Priority[N-1:1] <= grant[N-2:0];
Priority[0] <= grant[N-1];
end
end
end
end
endmodule
// The code in compination between the code written by G. Dimitrakopoulos
((Nov. 2008)) and some updates we writes them.
```

The following code gives us a very similar results to the Islip allocation technique but with less delay and one clock cycle granting. Arbiter, Allocator Net list and wave Form simulation.

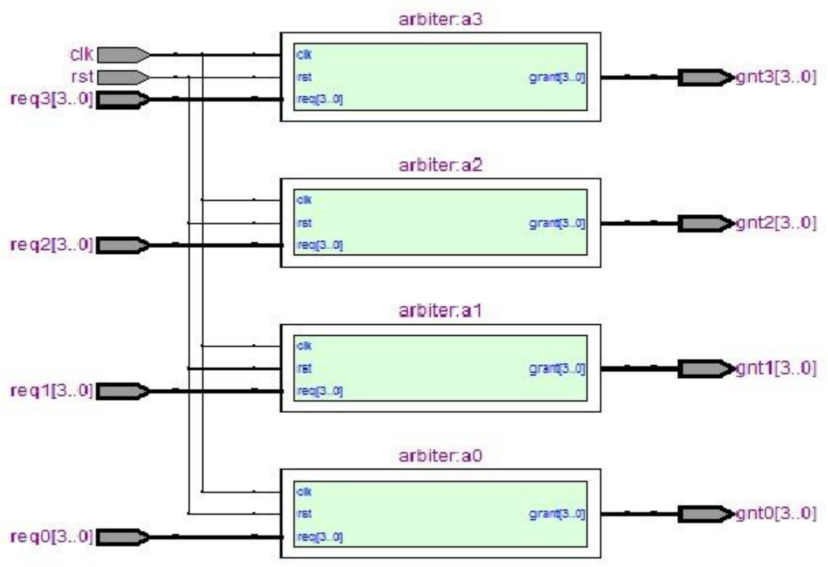


Figure 42 Allocator Netlist

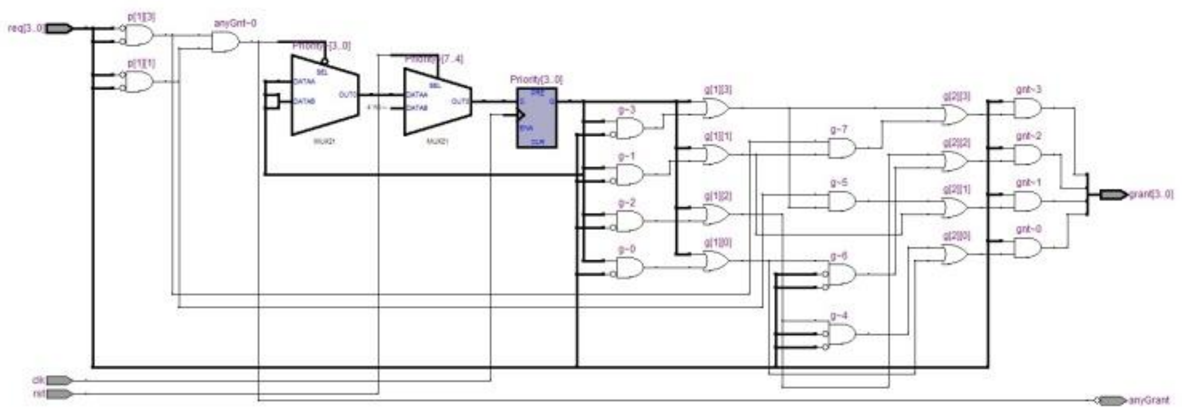


Figure 43 Arbiter Netlist

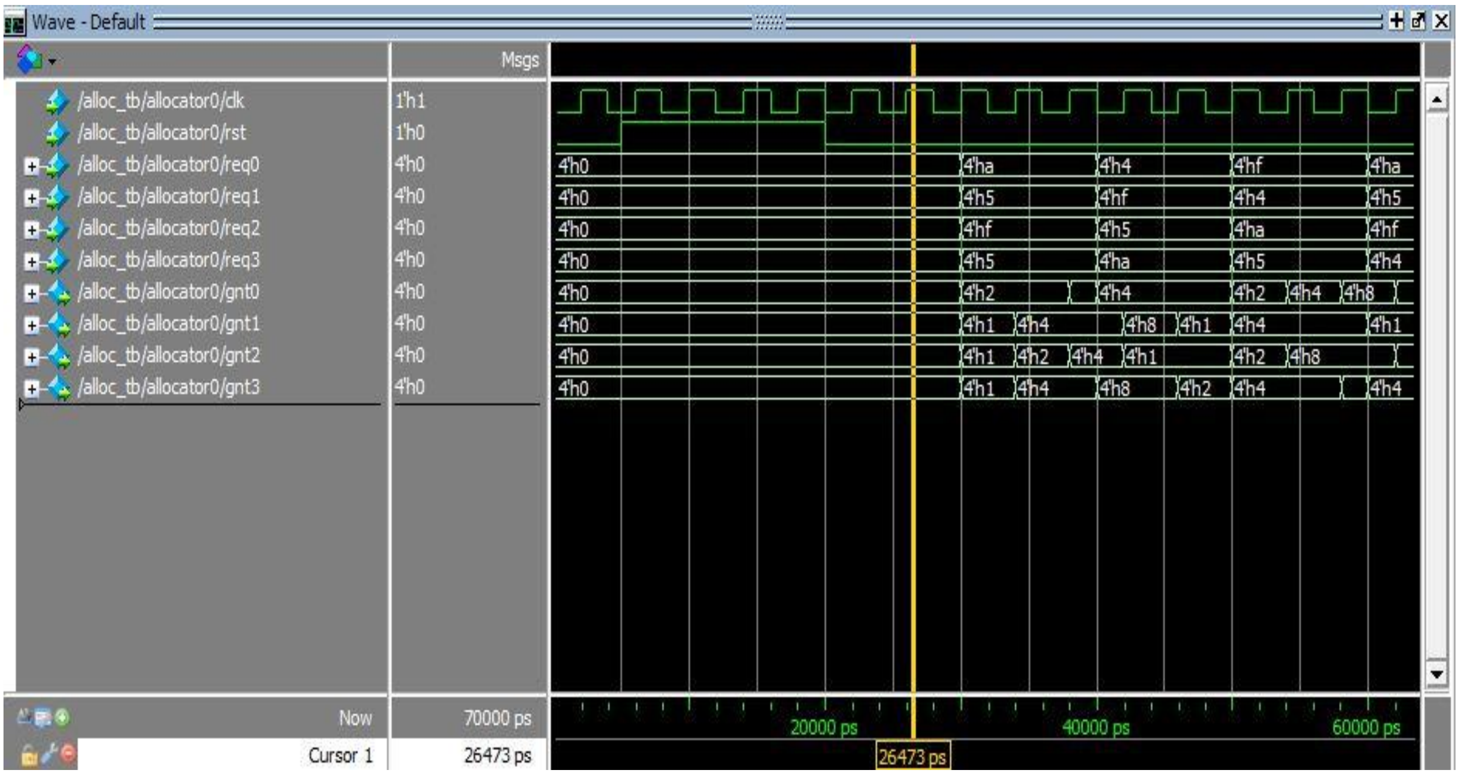

Wave - Default
Msgs
/alloc_tb/allocator0/clk 1'h1
/alloc_tb/allocator0/rst 1'h0
/alloc_tb/allocator0/req0 4'h0
/alloc_tb/allocator0/req1 4'h0
/alloc_tb/allocator0/req2 4'h0
/alloc_tb/allocator0/req3 4'h0
/alloc_tb/allocator0/gnt0 4'h0
/alloc_tb/allocator0/gnt1 4'h0
/alloc_tb/allocator0/gnt2 4'h0
/alloc_tb/allocator0/gnt3 4'h0
Now 70000 ps
Cursor 1 26473 ps
20000 ps
40000 ps
60000 ps
26473 ps

Figure 44 Arbiter Waveform

This arbiter is a simple form of Islip arbiter with only one cycle grant generation where the priority vector is not incremented unless a grant to that port is generated "non-blind round-robin".

7.2.2 The FIFO Buffers:

The specifications of our buffers are somehow matches perfect to a dual port memory implementing FIFO with generic size and width.

The code:

```
`define BUF_WIDTH 3    // BUF_SIZE = 16 -> BUF_WIDTH = 4, no. of bits to
be used in pointer
`define BUF_SIZE ( 1<<`BUF_WIDTH )
module fifo( clk, rst, buf_in, buf_out, wr_en, rd_en, buf_empty, buf_full,
fifo_counter );
input                rst, clk, wr_en, rd_en;
// reset, system clock, write enable and read enable.
input [7:0]          buf_in;
```

```
// data input to be pushed to buffer
output[7:0]          buf_out;
// port to output the data using pop.
output               buf_empty, buf_full;
// buffer empty and full indication
output[`BUF_WIDTH :0] fifo_counter;
// number of data pushed in to buffer
reg[7:0]             buf_out;
reg                  buf_empty, buf_full;
reg[`BUF_WIDTH :0]   fifo_counter;
reg[`BUF_WIDTH -1:0]  rd_ptr, wr_ptr;          // pointer to read and write
addresses
reg[7:0]             buf_mem[`BUF_SIZE -1 : 0]; //
always @(fifo_counter)
begin
buf_empty = (fifo_counter==0);
buf_full = (fifo_counter== `BUF_SIZE);
end
always @(posedge clk or posedge rst)
begin
if( rst )
fifo_counter <= 0;
else if( (!buf_full && wr_en) && ( !buf_empty && rd_en ) )
fifo_counter <= fifo_counter;
else if( !buf_full && wr_en )
fifo_counter <= fifo_counter + 1;
else if( !buf_empty && rd_en )
fifo_counter <= fifo_counter - 1;
else
fifo_counter <= fifo_counter;
end
always @( posedge clk or posedge rst)
begin
if( rst )
buf_out <= 0;
else
begin
if( rd_en && !buf_empty )
buf_out <= buf_mem[rd_ptr];
else
buf_out <= buf_out;
end
end
always @(posedge clk)
begin
if( wr_en && !buf_full )
buf_mem[ wr_ptr ] <= buf_in;
else
buf_mem[ wr_ptr ] <= buf_mem[ wr_ptr ];
end
always@(posedge clk or posedge rst)
begin
if( rst )
begin
wr_ptr <= 0;
```

```
rd_ptr <= 0;
end
else
begin
if( !buf_full && wr_en )    wr_ptr <= wr_ptr + 1;
else  wr_ptr <= wr_ptr;
if( !buf_empty && rd_en )   rd_ptr <= rd_ptr + 1;
else rd_ptr <= rd_ptr;
end
end
endmodule
```

This FIFO gives us more signals that will be used in future progress such as buffer count where we can use in Adaptive Routing and Adaptive allocation.

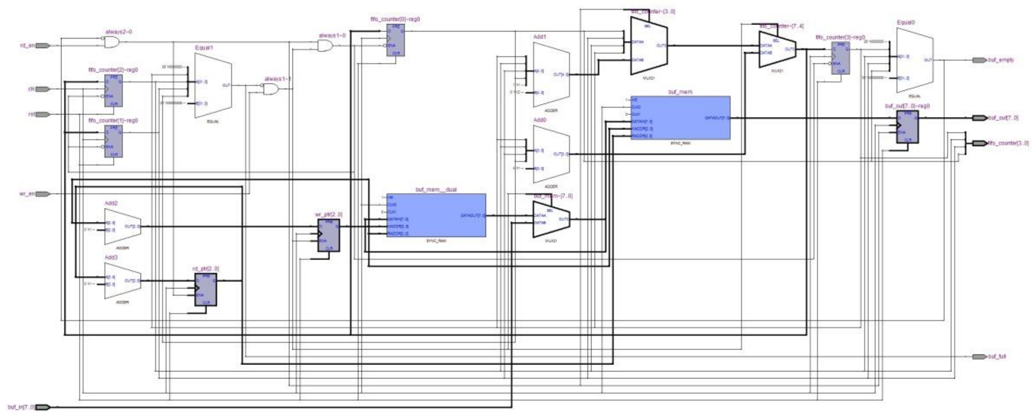


Figure 45 FIFO Netlist

7.2.3 Control

The Control Block which is responsible for delete granted flits form input buffers and open the cross bar to be saved in the output buffers.

The code:

```
module control(flitin0,
flitin1,
flitin2,
flitin3,
flitin4,
flitin5,
flitin6,
flitin7,
grantin0,
grantin1,
grantin2,
grantin3,
portout0,
portout1,
portout2,
portout3,
erase,
enable);
input[7:0] flitin0;
input[7:0] flitin1;
input[7:0] flitin2;
input[7:0] flitin3;
input[3:0] grantin0;
input[3:0] grantin1;
input[3:0] grantin2;
input[3:0] grantin3;
output[7:0] portout0;
output[7:0] portout1;
output[7:0] portout2;
output[7:0] portout3;
output [3:0] erase,enable;
reg [3:0] en;
//Code starts here
always @(grantin0,grantin1,grantin2,grantin3)
begin
begin
        if(grantin0 !==(4'd0))
        en[0]=1'b1;
        else
        en[0]=1'b0;
end
begin
        if(grantin1 !== (4'd0))
        en[1]=1'b1;
        else
        en[1]=1'b0;
end
begin
        if(grantin2 !== (4'd0))
        en[2]=1'b1;
        else
        en[2]=1'b0;
end
begin
        if(grantin3 !== (4'd0))
en[3]=1'b1;
else
en[3]=1'b0;
end
end
assign erase =(grantin0 | grantin1 | grantin2 | grantin3);
assign enable =en;
//module mux4_18(Sel, A, B, C, D, Y);
mux4_18 m0 (grantin0,flitin0,flitin1,flitin2,flitin3,portout0);
mux4_18 m1 (grantin1,flitin0,flitin1,flitin2,flitin3,portout1);
mux4_18 m2 (grantin2,flitin0,flitin1,flitin2,flitin3,portout2);
mux4_18 m3 (grantin3,flitin0,flitin1,flitin2,flitin3,portout3);
endmodule
```

7.2.4 Routing Table

We used a technique which is similar to packet switching where each flit of data carries its destination Address and a payload.

The routing table or Address translation block function can be summarized in translating the Address of the Network to port address in the router.

The routing table is configurable in our deterministic routing Network.

7.2.5 Router as a whole block

Here we gathered all the blocks and accurately signalled them to give us this router Fig 7.7.

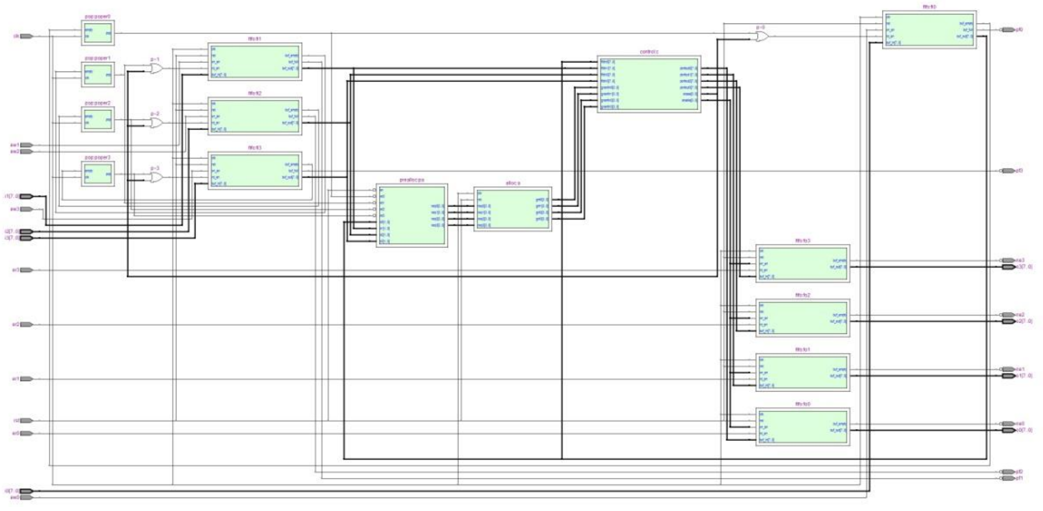


Figure 46 Router Netlist

Router signals

1- clk: Clock signal

Synchronous clock for the whole router or Network.

2- rst: reset signal

Restets the router or the network and all its internal blocks.

3- iN: input data port n

Input Data ports.

4- oN: output port N

Output Data Ports.

5- ewN: enable write N

This signal is set high when there is valid data at the input data port.

6- erN: enable read N

This input signal is set high indicating that blocks are ready to read the flits.

7- pfN: port free N

This output signal is set high when the input buffers is not full.

8- neN: not empty N

This output signal is set high when the output buffers is not empty.

<u>Simulation wave form</u>

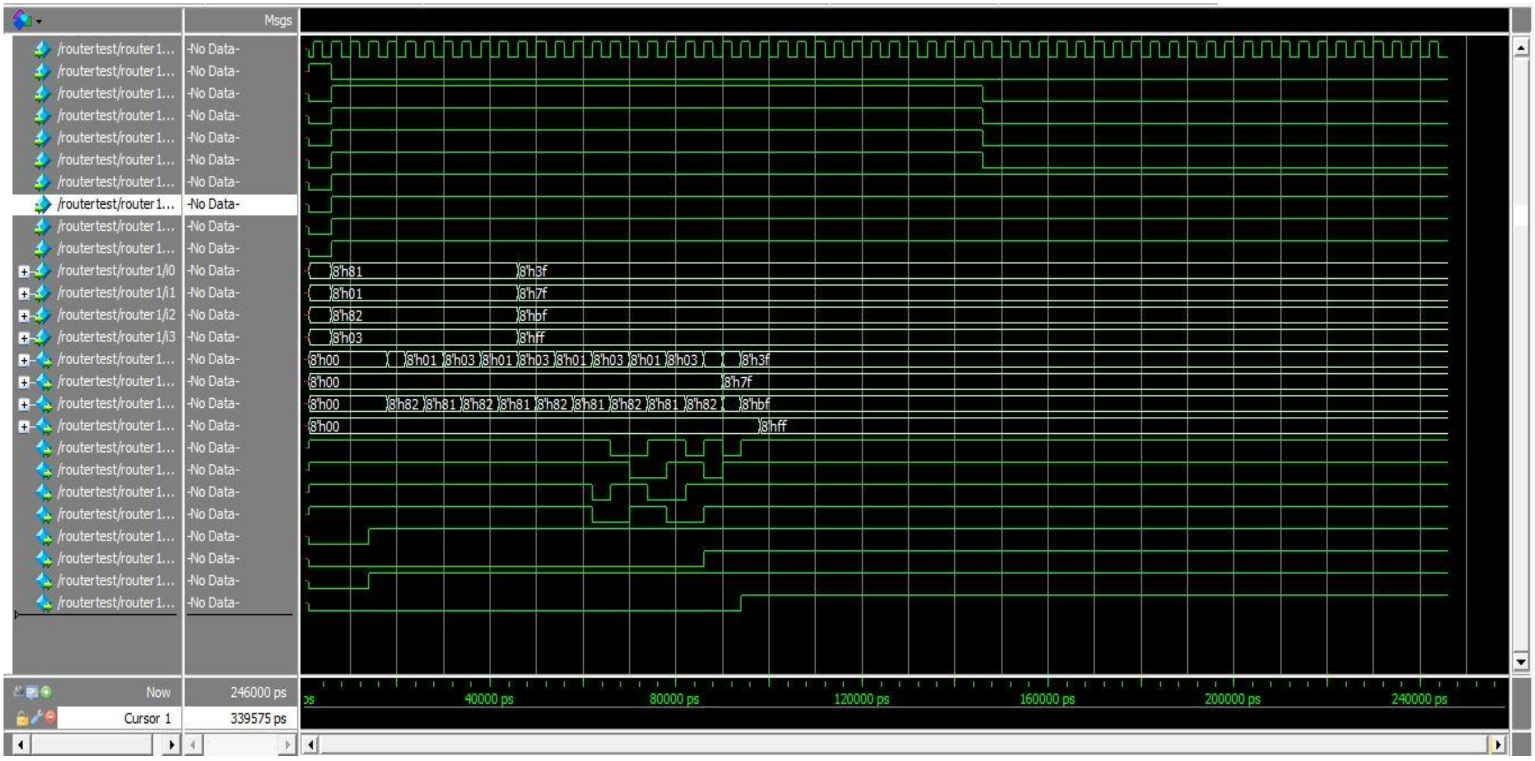


Figure 47 The router simulation waveform

7.3 Simulation and layout results

7.3.1 Simulation results

Chip link was designed and implemented using VERILOG HDL , compilation and functionality verification were made using Modelsim EDA program .

The figure below shows router operation while sending and routing stream of packets through the different input ports .

packets routed to the destined output ports are shown in figure , delay can be estimated knowing the data input and output instants .

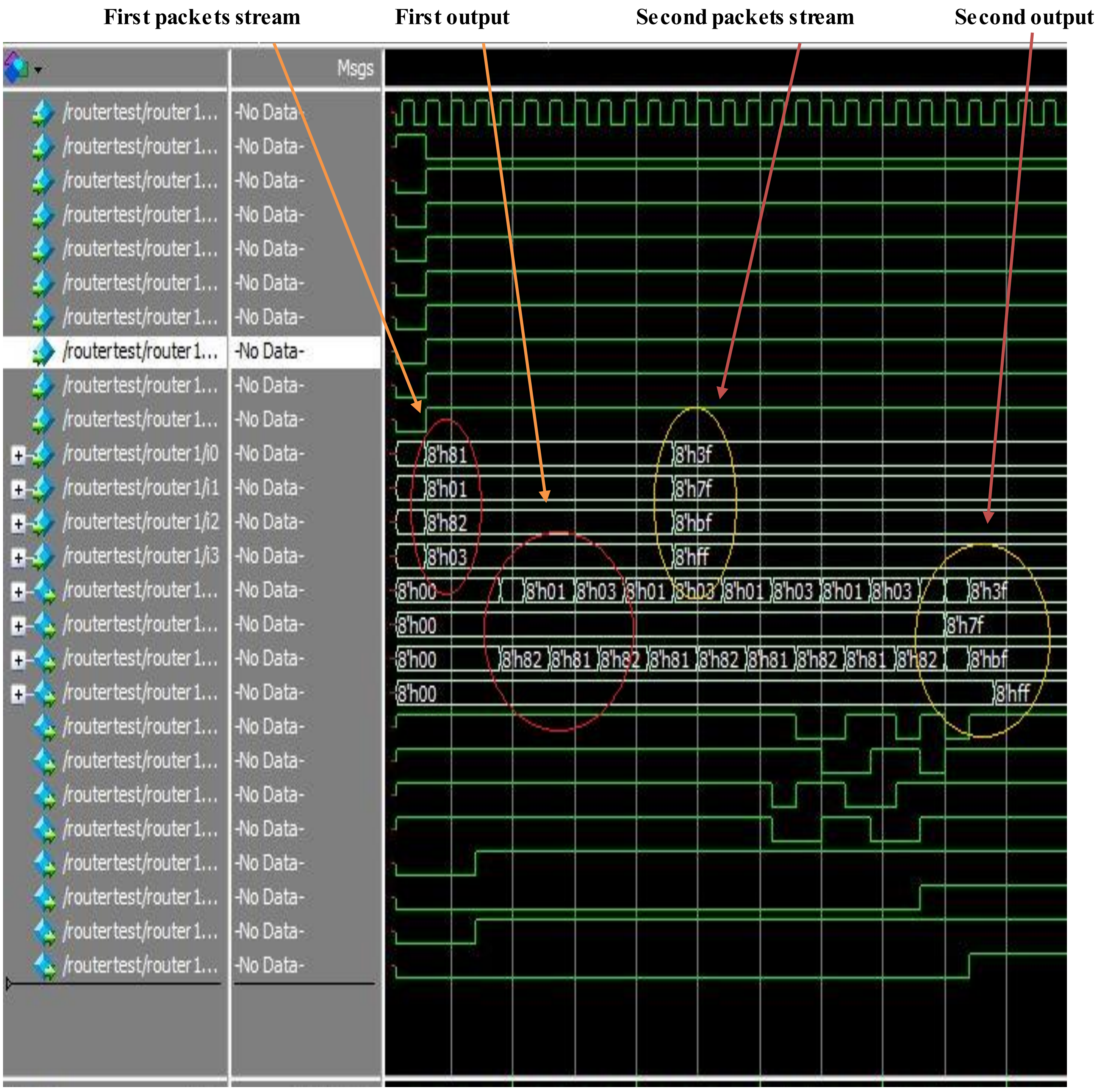


Figure 48

After functionality verification the design was imported for logic synthesises, and initial delay and area estimation.

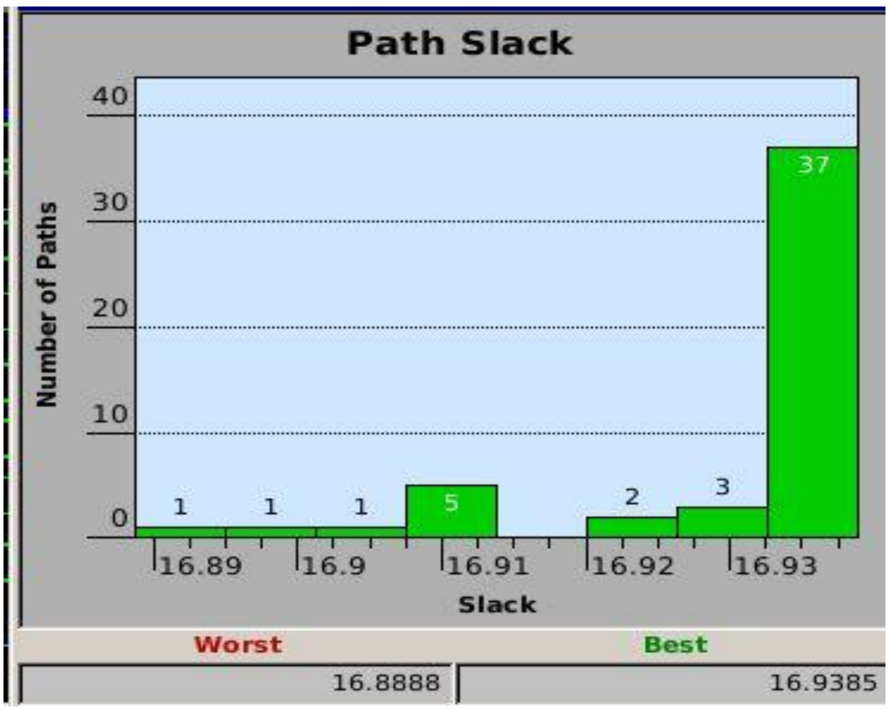


Figure 49 slack histogram

Best path (largest slack)

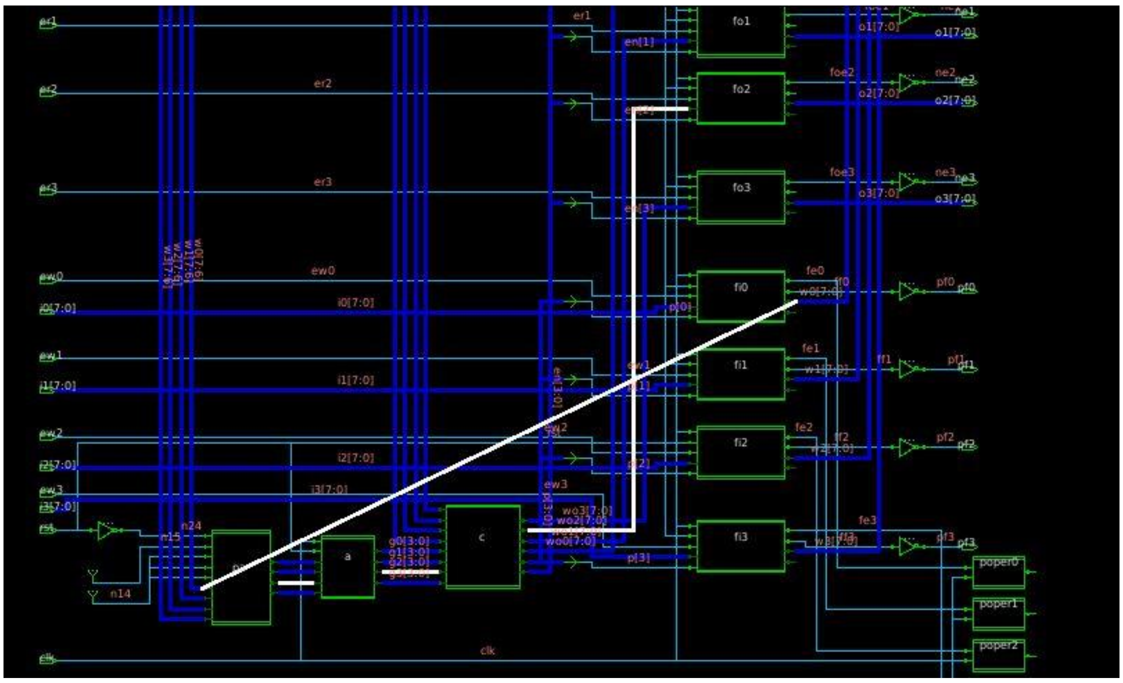


Figure 50 largest slack

Worst path (smallest slack)

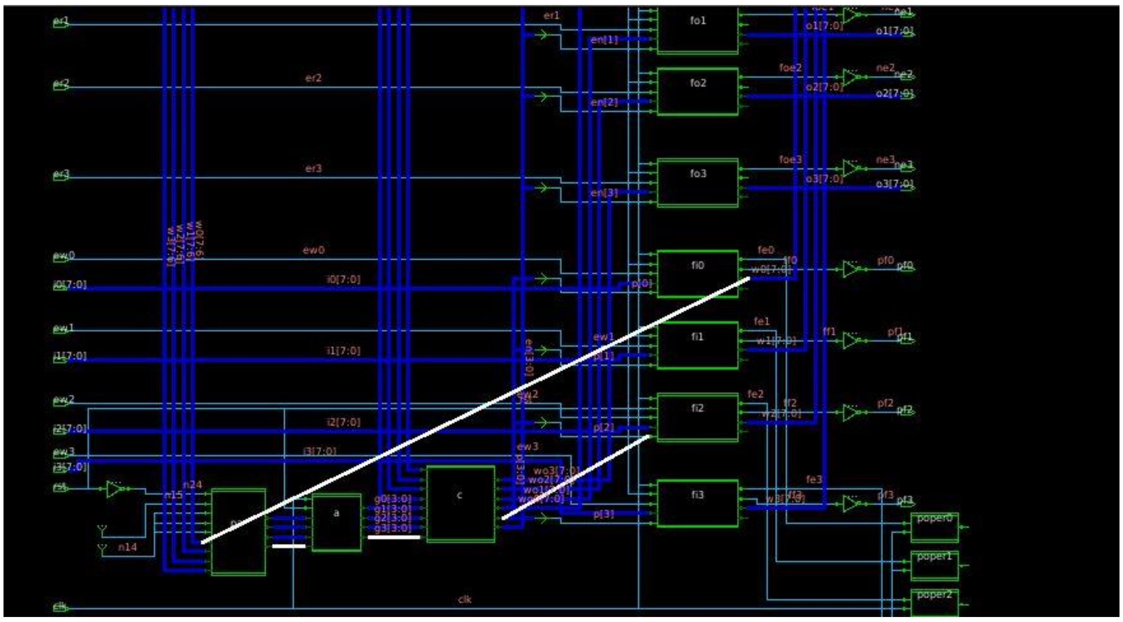


Figure 51 Smallest slack

Using SoC encounter EDA to generate a floor plan of CHIP Link router.
obtained reports shows :

geometric violations	N/A
Max cap violations	N/A
Fan out load	N/A
placement violations	N/A
total logic gates	9598
Total standard cells	1742
total area	10366.2 μm^2
total delay for data	4 ns
operational frequency	250 MHZ

Obtained floorplan

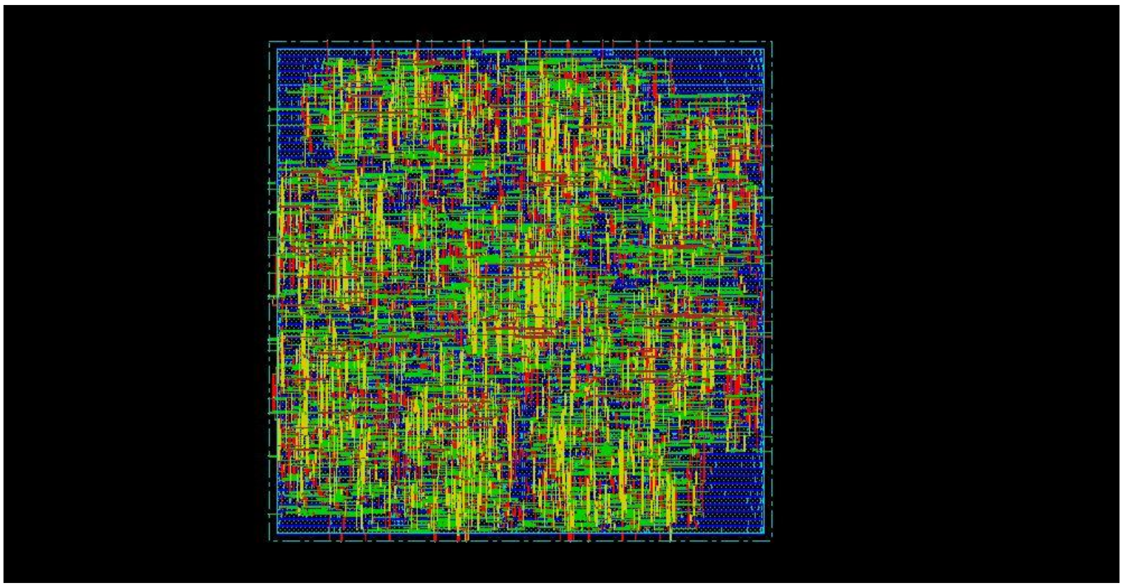


Figure 52 Full floorplan

A closer look in Figure 50

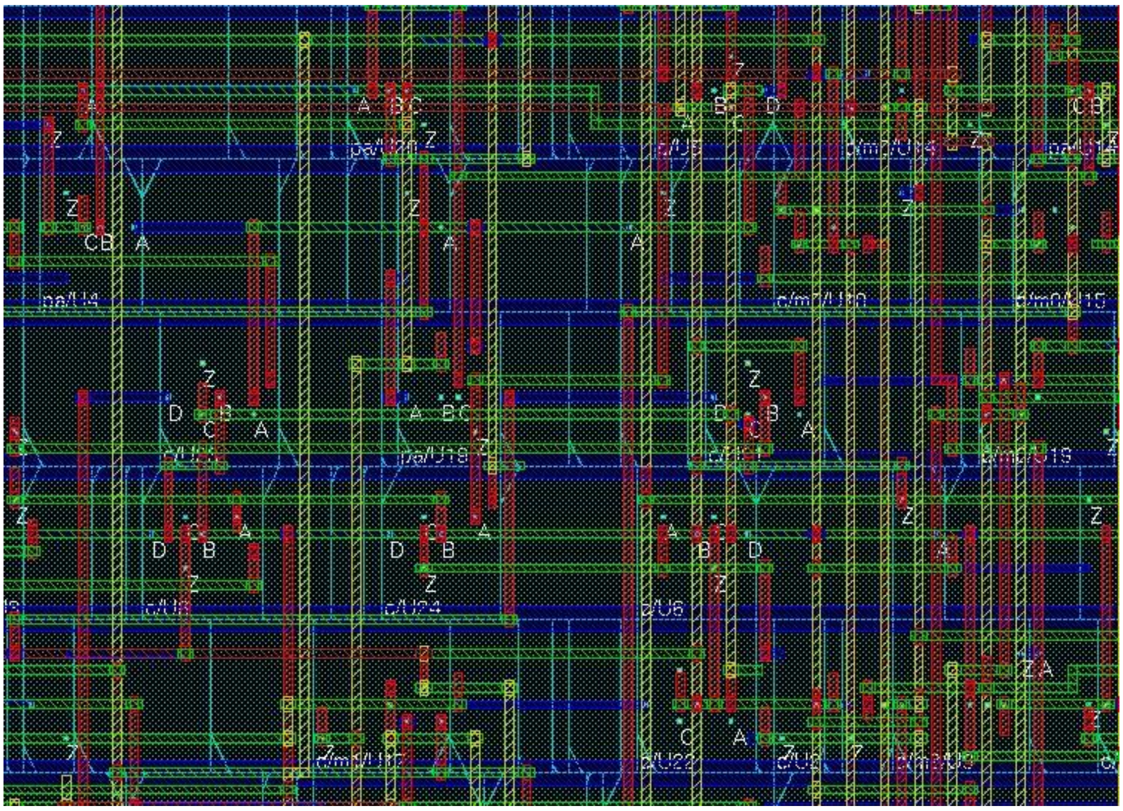


Figure 53 Closer look to the layout

8 Conclusion

Network on chip technology is the solution that makes digital systems like FPGA has more meaning and more useful in human life. It makes the performance of the digital systems better.

Now, system on chip become more complex and contains millions of IP cores which makes stresses on its performance to decrease step by step in area, speed and power consumption.

In order to beat the draw-backs, designers are able to optimize the programmable routing fabric inside digital systems by introducing NoC technology in which building a complete network embedded in the digital chip to manage the data traffic between IP cores and make the system do its role faster and better.

Chip-Link router is a simple router based-on the next generations in FPGA's chips. The router consists of 9598 logic gates and 1742 standard cells.

Chip-Link arbiter is a simple form of Islip arbiter with only one cycle grant generation where the priority vector is not incremented unless a grant to that port is generated "non-blind round-robin".

The operating frequency is 250 MHz and the maximum delay for the router to get the data from its input to its output is 4 n sec.

The design can be optimized in the future to have best results and best performance and makes FPGA grows in the human life and market field so that the investment will be high.

References:

[1] William J. Dally and Brian Towles ,”*Principles and Practices of Interconnection Networks*”.

[2] Daniel U. Becker,” *Efficient Microarchitecture for Network on chip routers*”.

[3] Nick McKeown,” *The iSLIP Scheduling Algorithm for Input-Queued Switches*”.

[4] P. Bhojwani and R. Mahapatra, "*Core network interface architecture and latency constrained on-chip communication*".

[5] Ian Kuon, Russell Tessier, and Jonathan Rose,” *FPGA Architecture: Survey and Challenges*” .

[6] Suman K. Mandal, Nikhil Gupta, Ayan Mandal, Javier Malave, Jason D. Lee and Rabi N. Mahapatra,” *NoCBench: A Benchmarking Platform for Network on Chip*”.

[7] SoC/ASIC/SoC-FPGA/S-ASIC Design and Verification, Intelop Corporation.

9 Appendix 1

Arbiter	arbiter variable priority round robin arbiter
Signal in/out	Description
Clk	Input clock
Rst	Input rest signal
Req	N bits Input defines the requests for a given resource output port.
Grant	N bits Output defines the grant signal for certain agent "input port"
Any grant	Output is high when any agent is granted the resource.

Allocator	Allocator iSLP single granting cycle.
Signal in/out	Description
Clk	Input clock
Rst	Input rest signal
Req0,Req1,Req2,Req3	N bits Input define the request for each output port.
Gnt0,Gnt1,Gnt2,Gnt3	N bits output defines the grant signal for each input port.

Control	Control and bus switch generates enable write and erase signals for output and input buffers respectively.
Signal in/out	Description
flitin0,flitin1,flitin2,flitin3	M bits Input Data.
grantin0,grantin1,grantin2,grantin3	N bits Input grant signals form allocator.
portout0,portout1,portout2,portout3	M bits Output Data Bus.
Enable	N bits output signal for enable write for output buffers.
Erase	N bits output signal for erase data from input buffers.

Fifo	FIFO input and output buffers.
Signal in/out	Description
Clk	Input clock.
Rst	Input reset.
Wr_en	Input enable write signal to write in the buffers.
Rd_en	Input enable read signal to read form the buffers.
Buf_in	M bits Input for input data.
Buf_out	M bits Output for output data.
Buf_empty	Output signal is high when the buffer is empty.
Buf_full	Output signal is high when the buffer is full.
Fifo_counter	Buffer_width Output signal gives the count of the buffer occupation.